Obsah

Úvod

V dubnu roku 1974 firma Intel učinila jeden z nejdůležitějších kroků v historii počítačové techniky. Ohlášením mikroprocesoru 8080, následovníka 4004 a 8008, začala nová etapa ve vývoji počítačů, etapa v duchu mikropočítačů.

Mikroprocesor 8080 vznikl u Federico Faggina, tým návrhářů vedl mladý inženýr firmy Intel, Masatoshi Shima. Svou koncepcí se stal okamžitě středem zájmu a už v lednu 1975 uvedl Popular Electronics první ze série konstrukcí založených na 8080 : Altair kit (procesor, napěťový zdroj, přední panel s množstvím kontrolek a 256 bajtů paměti) za $395.

Na přelomu roku 1975 a 1976 opouští Federico Faggin firmu Intel a zakládá svou vlastní firmu Zilog. Současně s ním odchází i Masatoshi Shima. Jejich cíl je zcela jasný, vytvořit 'super 8080'. Roku 1976 byl obvod Z-80 představen světu, ale ovací se rozhodně nedočkal. I odborníci se na něj, stejně jako na mikroprocesor 6502, dívali se skepsí. Tyto první obavy byly brzo překonány a Z-80 se stal šlágrem své doby. Po stránce programové i obvodové byl nesrovnatelně schopnější než mikroprocesor 8080. Výrobci počítačů jej velice brzy použily pro nové typy.

V únoru roku 1980 představila svůj stroj firma Sinclair, model ZX-80 (Z80A, 1kB RAM, 4kB interpretru integer BASICu v ROM a membránová klávesnice) za $199, následovaným ZX-81 (real BASIC v 8kB ROM, obrazovka stále jen v textové módu) a konečně v roce 1982 přichází s modelem ZX-Spectrum. Uvnitř počítače se skrývá procesor Z-80, 16kB RAM, 16kB ROM BASIC a obvod ULA řídící grafický barevný výstup na obrazovku, reproduktor a komunikaci s magnetofonem. Membrána zůstává, pouze je překryta maticí gumových tlačítek. Tato, ale hlavně další verse s pamětí 48kB RAM se staly velice populární na celém světě. V některých zemích světa se těší této popularitě dodnes.

Počítač je vybaven jazykem BASIC (pochopitelně nekompatibilní s ostatními počítači), který není pro programování velké části úloh nejvhodnější. Kompilátory jazyku BASIC nesplnily očekávané požadavky. Určitá změna nastala při příchodu kompilátorů jazyka Pascal a poté C. Oba nabídla firma Hisoft a v oblasti kompilátorů nenašla konkurenci. Oba překladače mají ale jeden velký handicap a tím je jejich délka. Pokud se od celkové délky 48kB odečte videoram, systémové proměnné a překladač jazyka, zbývá přibližně 15kB paměti na zdrojové texty a přeložený kód.

Nyní už zbývá poslední možnost a ta je z hlediska mikroprocesoru nejpřirozenější. Promluvíme k němu jeho vlastní řečí, která bývá nazývána "strojový kód". Strojový kód je posloupnost instrukcí (ve formě čísel), kterou programátor zaznamená do paměti, a kterou mikroprocesor čte a vykonává. Programování pouze ve strojovém kódu je velice pracné (každý pokyn procesoru musí být přeložen do odpovídajícího kódu) a nepřehledné (řada čísel). Proto byl vytvořen jazyk symbolických instrukcí (JSI), který jednotlivé kódy pojmenovává a tím přibližuje lidskému myšlení. Program, který provádí překlad z jazyka symbolických instrukcí do strojového kódu se nazývá 'assembler' (slovem 'assembler' bývá někdy označován i jazyk symbolických instrukcí).

Vytváření programů ve strojovém kódu má řadu výhod. Neexistuje nic, co leží mezi ním a mikroprocesorem, jak je to v případě vyšších jazyků, odpadá tedy mnoho omezení: programátor sám určí postupy řešení elementárních operací, má možnost provést optimalizace rychlosti nebo délky, v neposlední řadě jsou programy psané ve strojovém kódu velice rychlé.

Masm Z-80 je rychlý assembler, jeden ze série tří assemblerů: Masm 8080, Masm Z-80 a Masm 8048. Oplývá velkým komfortem jak při psaní zdrojových textů, tak i ve vlastním pseudojazyce. Povoleny jsou lokální bloky, makroinstrukce s parametry, křížové reference, tajné instrukce atd. A hlavně je plně kompatibilní s programem Gens, který je ve světě i u nás velice populární. Od Gensu se však liší již zmíněnými vlastnostmi, které jej posunují opět o něco dále. Pro uživatele programu Gens bude přechod k programu Masm otázkou několika okamžiků. Zkuste si to, prosím.

Zapněte počítač, začínáme.

Začínáme

Ze vše[illegible] nahrajte program MASM Z-80 (dále jen MASM) do [illegible]. Po zapnutí počítače se na obrazovce objeví :

(c [illegible] Logic Air Research Ltd

Pro nahr[illegible] programu do počítače vložte příkaz:

LO[illegible] a stiskněte ENTER

do magne[illegible] vložte kazetu s programem (nastaven na začátek) a [illegible]. Po nahrání programu do počítače se objeví hlav[illegible] [illegible]něte magnetofon.

Pro vy[illegible] assembleru zkusíme přeložit krátký program, který je [illegible] demonstraci assembleru :

- stiskně[illegible] písmeno G (Get)
- progra[illegible] na jméno zdrojového textu, stiskněte ENTER a pusťte magnetofon, do paměti se nahrává demonstrační program: [illegible] magnetofon vypněte
- stiskně[illegible] písmeno A (Assembly), na otázky 'Table size' a 'Optio[illegible] stiskněte ENTER a program se začne překládat
- stiskně[illegible] písmeno R (Run) a program se rozběhne; jeho obsluha je [illegible] : 1=nahoru, 0=dolů, O=vlevo, P=vpravo X=konec
- až si [illegible] vyzkoušíte, budeme pokračovat.

Budete-li chtít používat program Masm z microdrivu, můžete jej po [illegible] zavádějícího BASICu přenést na microdrive pomocí prostředků Sinclair Express. Úprava BASICu spočívá v náhradě příkazu

LOAD "" CODE

příkazem

LOAD "m";1;"c" CODE

Do zavá[illegible] BASICového programu se dostanete velice jednoduše [illegible] inicializaci počítače příkazem RANDOMIZE USR 0 začnete [illegible] zavádějící program pomocí příkazu MERGE "" místo [illegible] příkaz MERGE způsobí nahrání programu, k autostar[illegible] nedojde. Nyní můžete provést potřebné změny.

V tom[illegible] případě bude program Masm nahráván vždy z microdrivu [illegible]. Chcete-li mít zaváděcí program úplně obecný, tzn., [illegible] strojový kód ze stejného microdrivu jako zavádějící BASIC, [illegible] tento zavádějící program:

```
10 RANDOMIZE USR 25000:
20 CLS
30 POKE 23631,PEEK 23766:
40 MERGE "":REM 9999:
50 LOAD *"m";PEEK 23681;"c" CODE:
60 RUN
```

Tento program bude startován na řádce 20. V paměťové buňce na adrese 23681 se uschovává číslo mikrodrivu, ze kterého byl nahráván zaváděcí program.

1. Masm po spuštění

Při spuštění programu se dostáváte do hlavního menu. To je zobrazeno na druhé řádce. První řádka je zde použita pro informování uživatele o současném stavu programu:

```
Capitals          - je-li zapnutý režim velkých písmen
Overwrite/Insert  - aktuální textový režim
<filename>        - aktuální jméno zdrojového textu
```

Levá část první řádky je použita pro zobrazení názvů vkládaných parametrů.

Hlavní menu obsahuje tyto položky:

```
Ass  - assemblování zdrojového textu v paměti počítače
Bas  - návrat z programu (do Basicu)
Code - nahrání přeloženého kódu
Del  - smazání zdrojového textu
Edit - přechod do editoru
Get  - nahrání zdrojového textu do paměti počítače
Incl - nahrání zdrojového textu pro následnou inklusi
Mem  - změna velikosti bufferů
Name - změna jména zdrojového textu
Put  - nahrání zdrojového textu z počítače
Run  - spuštění přeloženého kódu
Text - informace o zdrojovém textu
Ver  - verifikace nahrávky
Wrt  - výpis zdrojového textu na tiskárně
Xref - záznam návěští pro křížové reference
```

Počáteční písmeno jednoznačně určuje volbu. Pokud je program v hlavním menu, očekává tlačítko. Odpovídá-li některé možnosti, přechází se na její vykonávání. Všechny parametry jsou očekávány a jejich vkládání se provádí v druhé řádce. Vkládání je jednotné a možné příkazy pro editaci jsou popsány v kapitole 2.3.

První dva řádky na obrazovce jsou trvale rezervovány pro zobrazování stavu a vkládání parametrů Masm, 22 spodních řádek se používá pro editor nebo pro výstup jednotlivých příkazů. Při spuštění vlastního programu (R příkaz) je pochopitelně k dispozici celá obrazovka.

2. Editor

Program Masm vyžaduje pro svou hlavní funkci, kterou je překlad, zdrojový text. Tento zdrojový text může být připravován i jiným textovým editorem, celková rychlost takového systému nebude ale nejvyšší. Z tohoto důvodu je součástí assembleru textový editor, který celou tvorbu programů značně urychluje.

Textový editor je typu 'full screen', tzn. umožňuje pohyb kurzoru po celé obrazovce. Maximální délka řádku, kterou dovoluje editovat je 80 znaků. Kromě standartních příkazů známých od ostatních výrobců programů, je editor Masm specializován pro práci v jazyku symbolických instrukcí. To se projevuje především v tabelaci. Ve spojení s full-screen verzí tak vznikl velice přehledný editor s vysokým komfortem.

Protože program Masm pracuje během celé své činnosti v režimu 64 znaků na řádek (víc to opravdu nejde), je na každé řádce vždy 16 znaků neviditelných. Pokud se uživatel přiblíží k neviditelné oblasti, řádka se začne posunovat a znaky z této oblasti se stanou viditelnými.

V dalším textu bude SS chápáno jako Symbol Shift, CS jako Caps Shift.

2.1 Stavová řádka

Stavová řádka má stejný formát jako při aktivaci hlavního menu. Zleva jsou zobrazeny tyto informace:

```
Line  llll  číslo aktuálního řádku (1..x)
Col   cc    aktuální pozice kursoru (1..80)
Error ee    poslední nalezená chyba při překladu (jinak 0)
```

2.2 Formát textu

V textovém editoru jsou tabelátory nastaveny tak, aby jejich pozice neodporovaly žádnému požadavku, a aby byla snížena zbytečná redundance na jednotlivých řádcich. Formát je tedy:

```
1       8     13              26
Label   Ld    a,10            ; komentář

^       ^     ^               ^
:       :     :               :
:       :     :               * zde začíná komentář (od ";")
:       :     * parametry instrukce (pokud existují)
:       * jméno instrukce
* návěští (nemusí existovat)
```

Jsou povoleny také formáty:

```
; pouze komentář
Label   ; pouze návěští s komentářem
        ; komentář místo instrukce
             ; komentář místo parametru
                          ; komentář od 26. pozice
```

nebo kombinace:

```
                          ; komentované návěští
Label
Label         ; komentář
```

atd.

Tabelátor je ve zdrojovém textu reprezentován pouze jedním bajtem, proto je celková délka textu kratší než při vyjádření pomocí mezer. Rychlejší je pochopitelně i překlad do strojového kódu.

Jednou z hlavních výhod editoru je tzv. "autotabelace", která zrušila hlavní nevýhodu full-screen editoru při psaní zdrojových textů pro assembler. Neustálý pohyb kursoru nebo tabelace znepříjemňovaly uživatelům život do té míry, že se nakonec vrátili do obyčejných řádkových editorů, zdaleka ne tak přehledných. Rychlost psaní programů zde ale byla vyšší.

Princip autotabelace je v tom, že editoru (stejně jako u řádkových editorů) pouze naznačujeme jak text tabelovat a zarovnávat. Po opuštění editované řádky se na ni vrhne zarovnávací rutina a zajistí zarovnání.

Např.

```
Lab Ld a,10 ; Komentář
```

bude zarovnán (na tabelátory):

```
Lab    Ld    a,10          ; Komentář
```

Mezerami jsme tedy oddělili jednotlivé části řádky a podle těchto mezer bylo toto zarovnání provedeno. Z příkladu je jasné, že autotabelace zarovnává na nejbližší vyšší tabelátor. Pokud budete chtít umístit text až na konkrétní tabelátor (a autotabelace nemá potřebný počet bloků), musíte přesunout text až na požadovaný tabelátor.

Budeme-li chtít vložit řádku:

```
Lab    Ret                    ; Komentář
```

nemůžeme to provést:

```
Lab Ret ; Komentář
```

ale takto:

```
Lab Ret                       ; Komentář
```

Instrukce Ret bude zarovnána na svou pozici, Komentář zůstává kde je. Pro posun kursoru je pochopitelně nejvhodnější funkce tabelátoru (viz dále).

V komentářích a v parametrech povelů assembleru autotabelace nepracuje. Řádky :

```
*F 1:filename
```

nebo

```
; toto je pouze zkouška komentáře
```

zůstávají proto nezměněny.

2.3 Klávesnice

Klávesnice na počítači ZX-Spectrum nepatří jistě k tomu nejlepšímu, co Clive Sinclair roku 1982 světu nabídl. Jistě se ale i ona podílela na extrémně nízké ceně. 40 tlačítek na gumovém Spectru a o něco více na Spectru Plus, z toho dva shifty, neposkytují příliš mnoho možností, jak rozmístění vyřešit.

Znaky, které jsou uvedeny na tlačítkách, se mohou vkládat přímo (buď se symbol-shiftem nebo bez něj). Aby zbylo potřebné množství tlačítek pro povely editoru, byl nutný následující kompromis : pro vkládání znaků [,],{ , },~ , ¦ , \ musí být použita editorová funkce SS-f, kombinace jako SS-f, SS-g atd. jsou použity pro příkazy editoru. Kursory jsou na svém místě a musí být doprovázeny Caps-Shiftem. Příkazy byly rozmístěny tak, aby co možná nejvíce připomínaly původní řádkový editor Ing. Adámka. Extended příkazy, které tento editor umí, pracují v Masm podobně, pouze se aktivují jinými (snad lépe zapamatovatelnými) tlačítky.

2.4 Řádkové příkazy

V rámci jedné řádky jsou k dispozici tyto příkazy:

CS-2	Zapínání/vypínání režimu velkých písmen
CS-5	Kursor vlevo
CS-8	Kursor vpravo
CS-9	Tabelátor
CS-0	Mazání znaku
SS-D	Smazání řádky
SS-E	Skok na konec/začátek řádky
SS-F	Vkládání extended kódu
SS-I	Změna režimu

Caps lock On/Off , CS-2

Kombinací CS-2 může uživatel v kterémkoliv okamžiku změnit režim psaní velkých písmen na zapnutý nebo vypnutý. Stav je neustále zobrazen ve stavové řádce. Masm nevyžaduje velká písmena, jak je tomu u ostatních assemblerů z-80, čtěte tedy pozorně kapitolu 4.

Kursor vlevo , CS-5

CS-5 způsobí posun kursoru o jednu pozici vlevo. Pokud je na první pozici řádky, bude příkaz ignorován. Pokud se tímto způsobem vracíte ze zadní neviditelné oblasti, 10 znaků před koncem se začne řádka posunovat doprava a dostává se do své normální polohy.

Kursor vpravo , CS-8

CS-6 způsobí posun kursoru o jednu pozici vpravo. Pokud je na poslední pozici řádky, bude příkaz ignorován. Jestliže se takto přesunujete do zadní neviditelné oblasti, 4 znaky před koncem se začne řádka posunovat doleva a nakonec zůstane zobrazena celá neviditelná část.

Tabelátor , CS-9

CS-9 způsobuje skok na další tabelátor. Jak již bylo uvedeno, tabelátory jsou rozmístěny na pevných pozicích, přesně tak, jak to bylo uvedeno v předcházejícím textu. Vždy se provádí skok na vyšší pozici. Pokud je kursor umístěn na posledním tabelátoru nebo za ním, posune se kursor opět na první pozici.

Smazání znaku , CS-O

Kombinace CS-O (DELETE) způsobí smazání znaku, který leží před kursorem. Pokud je kursor na začátku řádky, bude příkaz ignorován. Pro posun řádku zde platí stejná pravidla jako pro CS-5.

Smazání řádku , SS-D

Kombinace SS-D způsobí, že řádek na kterém je umístěn kursor,bude vymazán. Část pod řádkou se posune vzhůru a kursor se přesune na začátek následující řádky. Pokud byla smazána poslední řádka v textu, bude na jeho konci vytvořena nová. Je pochopitelně prázdná.

Skok na konec/začátek , SS-E

Pokud se potřebujete rychle dostat na konec řádky, můžete použít kombinaci SS-E. Kursor se okamžitě přesune za poslední pozici a je-li třeba, bude řádka odsunuta vlevo. Výjimku tvoří situace, kdy je řádka zaplněna i na posledním místě. Pak kursoru nezbývá nic jiného, než se usadit právě na této pozici. Žádná další totiž neexistuje.

Tato funkce má ale ještě svou druhou část. Stisknete-li kombinaci SS-E ještě jednou, přesune se kursor na začátek řádky (na pozici 1). Na začátek se SS-E přesunuje rovněž v případě, že už kursor leží na poslední pozici.

Vkládání extended kódu , SS-F

Jak bylo vysvětleno v kapitole o klávesnici, byla řada kombinací na klávesnici vyhrazena editoru i na úkor několika znaků. Tyto znaky se musí vkládat přes příkaz SS-F. Po stisknutí SS-F zmizí kursor a editor čeká na vložené tlačítko. Nyní stiskněte tlačítko, pod kterým je vámi požadovaný znak.

Např. chceme vložit znak '['. Stiskneme SS-F pak 'y'. Na obrazovce se objeví znak '['.

Změna režimu , SS-I

Textový editor umí pracovat ve dvou režimech, stav je zobrazen ve stavové řádce. Jsou to režimy Overwrite (přepiš) a Insert (vkládej, vlož). Liší se následovně:

Overwrite - jakoukoliv pozici v textu můžeme přepsat.
Insert - ať jsme na jakékoliv pozici, můžeme na tuto pozici vložit znak a kursor se posune o pozici doprava. Posune se rovněž zbytek řádky. Je proto logické, že do řádky s obsazenou poslední pozicí nelze nic vkládat.

Režim editoru se uplatňuje rovněž při funkci CR (viz. dále).

2.5 Příkazy pro pohyb kursoru

Pro změnu pozice kursoru v textu je připraveno v editoru několik příkazů. Kromě již zmíněných řádkových příkazů jsou to:

CS-3	skok o stránku dolů (směrem ke konci textu)
CS-4	skok o stránku nahoru
CS-6	posun o řádek dolů
CS-7	posun o řádek nahoru
ENTER	skok na začátek další řádky
SS-Q	skok na začátek textu
SS-W	skok na konec textu
SS-Y	vložení prázdné řádky

Skok o stránku dolů , CS-3

Tento příkaz přesune kursor o jednu stranu textu níž. Pozice na řádku zůstane zachována a nenarazíme-li na konec textu, zůstane zachována i y-ová souřadnice kursoru na obrazovce.

Skok o stránku nahoru , CS-4

Ekvivalentní příkaz příkazu CS-3 je skok o stránku textu nahoru (CS-4). Pozice na řádku zůstane zachována a nenarazíme-li na začátek textu zůstává zachována i y-ová souřadnice kursoru na obrazovce.

Posun o řádek dolů , CS-6

Příkaz CS-6 posunuje kursor o jednu pozici dolů. To ale pouze v případě, že další řádka existuje (na rozdíl např. od programu Tasword 2, kde je textový buffer k dispozici celý).

Posun o řádek nahoru , CS-7

Příkaz CS-7 posunuje kursor o jednu pozici nahoru. To ale pouze v případě, že nejsme na první řádce textu. Pak je příkaz CS-7 ignorován.

Skok na další řádku , ENTER

Klávesou ENTER přechází kursor na začátek další řádky. Opuštěná řádka je zarovnána a v závislosti na režimu se:

Overwrite: skočí na 1. pozici další existující řádky. Pokud je kursor na konci textu, vytvoří se nová prázdná řádka.

Insert: pod řádkou, kterou opouštíme, se vytvoří nová prázdná řádka. Pokud jsme na konci, platí totéž co v režimu Overwrite.

Bližší informace o vnitřním tvaru řádky jsou uvedeny v kapitole 6.1.

Skok na začátek textu , SS-Q

Příkaz SS-Q přesune kursor na první znak prvního řádku zdrojového textu.

Skok na konec textu , SS-W

Příkaz SS-W přesune kursor na první znak posledního řádku zdrojového textu.

Vložení prázdné řádky , SS-Y

Pokud budete potřebovat vložit prázdnou řádku před 1.řádku textu nebo budete chtít vložit prázdou řádku aniž by se změnila pozice korsoru, můžete použít příkaz SS-Y. Ten řádku, na které je umístěn kursor, odsune dolů a do vytvořeného prostoru umístí řádku novou. Pochopitelně prázdnou.

2.6 Extended příkazy

Jak již bylo řečeno, není možné umístit všechny příkazy editoru na klávesnici. Některé z nich jsou přístupné pouze z Extended menu, které se zobrazí ve druhé řádce obrazovky. Volba Extended menu se provádí kombinací SS+CS. Jednotlivé položky mají následující význam:

Start - nastavení začátku bloku
End - nastavení konce bloku
Copy - zkopírování bloku
Delete - smazání bloku
Move - přenos bloku
Put - nahrání bloku
Gost - skok na začátek bloku (GO at STart)
Wrt - výpis bloku
Label - hledání návěští
Find - hledání řetězce
Repl - hledání a nahrazení řetězce
Quit - návrat do hlavního menu

I v tomto menu se volba provádí stisknutím klávesy shodné s počátečním písmenem voleného příkazu. Při špatné volbě se program vrací do editoru.

2.6.1 Blokové příkazy

Blokové příkazy se staly nutností u každého pořádného editoru a ani v Masm nejsou vynechány. Pro zjednodušení práce jsou v něm příkazy pro kopírování, smazání a přenos bloku. Před těmito příkazy musíte ale blok nejprve nadefinovat. Není důležité, zda určíte nejprve začátek nebo konec bloku. Musí ale platit pravidlo, že End>=Start. Informace o začátku a konci bloku nezanikají ani při nesplnění této podmínky, blok není ale zobrazen. Část textu, která je blokem, je zobrazena na podkladu s velkým jasem. Tím je blok jednoznačně určen.

V bloku můžete používat všechny příkazy editoru. Můžete řádky bloku odmazávat, vkládat nebo modifikovat. Řádky, kterými byl blok specifikován, nesou tuto informaci stále.

Pokud budete chtít zrušit definici bloku, stačí porušit podmínku, že End>=Start. O něco výhodnější je umístit End před Start, protože Start může být využíván jako cílový bod pro skokový příkaz EXT+G (viz. dále).

Definice bloku , EXT+Start a EXT+End

Blok se označuje pomocí příkazů Start a End. Kursor přesunete na řádku kde chcete mít začátek nebo konec bloku, zvolíte extended režim a příkaz Start nebo End podle potřeby. Pozice kursoru na řádce zůstane zachována. V textu neproběhne žádná změna. Teprve až po určení druhé (zbývající) hranice je blok nadefinován. To se ihned projeví zvýšeným jasem v oblasti bloku.

Kopie bloku , EXT+Copy

Tento příkaz zkopíruje blok před řádku, na které stojíme. Není povoleno kopírovat blok do sebe. V takovém případě editor požadavek ignoruje a vrací se zpět. Po provedení příkazu je kursor umístěn na stejné pozici v řádku. Dojde-li při vykonávání tohoto příkazu k přeplnění paměti, bude ohlášena chyba a Masm se vrátí do hlavního menu.

Smazání bloku , EXT+Delete

Příkaz smaže celý blok tak, že řádka ležící nyní za řádkou End bude ležet v místě, kde nyní leží Start. U příkazu Delete může být při jeho volání kursor uvnitř bloku. Pokud blok není nadefinován, bude příkaz ignorován.

Pohyb bloku , EXT+Move

Příkazem Move se přenese nadefinovaný blok před řádku, na které je umístěn kursor. Při volání musí být kursor mimo blok, jinak je příkaz ignorován. Dojde-li při vykonávání tohoto příkazu k přeplnění paměti, bude ohlášena chyba a Masm se vrátí do hlavního menu.

Skok na začátek , EXT+Gost

Na rozdíl od příkazu SS-Q skáče příkaz Gost na začátek bloku, přesněji 'skáče na řádku Start'. Není tedy nutné, aby byl nadefinován konec bloku. Kursor se přesune na řádku Start i v případě, že End<Start.
Tento příkaz patří současně do příkazů pro pohyb kursoru.

2.6.2 Hledání a zaměňování

Pro zrychlení a částečně i zautomatizování tvorby zdrojových textů je editor doplněn příkazy pro hledání a zaměňování textu. V těchto příkazech se pracuje s jednotkou zvanou 'řetězec',který můžeme definovat jako posloupnost znaků s omezenou délkou v závislosti na určení.

Tyto řetězce se vkládají jako parametry. Vložíte-li je při jednom zavolání příkazu, budou vám nabídnuty rovněž při příštím zavolání tohoto příkazu. Počáteční stav jsou prázdné řetězce.

Řetězce jsou tři (v závorce max. délka):

Label - (6) - jméno hledaného návěští
Search - (20) - hledaný řetězec
Replace - (20) - řetězec pro záměnu

Důležité je připomenout, že Search a Replace pracují od aktuální pozice kursoru (rovněž od aktuální pozice kursoru na řádce) do konce textu. Label pracuje od začátku řádky, na které je právě kursor. U všech tří příkazů se v případě nezdaru hlásí chyba 'Not Found' a kursor zůstává na konci textu. Velká a malá písmena jsou u všech příkazů rozlišována. Hledané řetězce nesmí mít nulovou délku. U řetězce, kterým nahrazujeme, toto omezení neplatí. Zde může být délka nulová.

Hledání návěští , EXT+Label

Tento příkaz hledá od začátku aktuální řádky návěští, které vložíte. Pokud je nalezeno, je kursor na tuto řádku přemístěn. Návěštím se rozumí prvních šest znaků řádky v délce max. 6 (komentáře a povely assembleru se nepočítají). Podrobný popis je v kapitole 4.1.1.

Hledání řetězce , EXT+Find

Hledá vložený řetězec od aktuální pozice kursoru. Jakmile je řetězec nalezen, zůstává kursor za ním. Tím se umožňuje opakované hledání.

Pravidla pro vkládání řetězce jsou následující:

- rutina pro vkládání řetězce umožňuje použít všechny řádkové příkazy
- pro smazání vstupní řádky slouží opět příkaz SS-D
- pokud požaduje nalezení řetězce, jehož části jsou mezery, vložte je rovněž do řádku a kursor umístěte za řetězec; např. budeme chtít nalézt 'asd ', musíme vložit celý řetězec a kursor umístit až za mezery, jinak by editor hledal pouze řetězec 'asd'
- při opuštění rutiny hledá editor konec řetězce; pokud je kursor umístěn v řetězci, prohlásí koncem poslední znak řetězce; pokud je umístěn v prostoru za řetězcem, zahrnují se do řetězce i mezery před kursorem
- přestože je rutina pro vkládání řetězce velice podobná rutině pro vkládání řádky, při opuštění se autotabelace neprovádí

Hledání a zaměňování , EXT+Repl

Tento příkaz je nadstavbou příkazu EXT+F. Pro vkládání řetězců platí stejná pravidla jako jako u příkazu EXT+F. Po nalezení řetězce (pokud existuje) nabídne editor tři možnosti:

```
                y/n/q
```

ve smyslu:

```
                y - yes,  ano proveď záměnu
                n - no,   neprováděj záměnu
                q - quit, konec příkazu Replace
```

Nyní čeká program na tlačítko. Pokud stisknete 'q', příkaz končí a vracíte se do hlavní části editoru. Při stisknutí 'y' je provedena záměna a příkaz pokračuje v dalším hledání. Při volbě 'n' se rovnou pokračuje v prohledávání. Pokud neexistuje další výskyt, hledání se ukončí na konci textu s chybovým hlášením 'Not Found'.

Jestliže je při nahrazování nový řetězec delší než původní, může dojít k situaci, že je nová délka řádky větší než 80 znaků. V takovém případě je text od 81. znaku dále vypuštěn.

Pro následující příklad budeme chtít například zaměnit instrukce 'jr' instrukcí 'djnz'.

```
Start   loc
        ld    e,0
:1      inc   e
        jr    :1            ; zde bude první náhrada
:2      nop
        jr    :2            ; zde je druhá náhrada
        endl
        ret
```

V editoru zvolíme režim Extended. Poté klávesou 'r' příkaz Replace. Vložíme hledaný řetězec 'jr ' (dvě mezery v řetězci jsou nutné, aby byla délka shodná s instrukcí 'djnz') a stiskneme ENTER. Dále vložíme řetězec, který má nahrazovat. Tedy 'djnz' a opět stiskneme ENTER. Editor nám zobrazí nalezený řetězec a nabídne možnosti jak s ním naložit. Volíme klávesu 'y' a editor nám okamžitě nabízí tyto možnosti pro další výskyt 'jr'. Opět volíme 'y'. Další výskyt editor nenalezl a končí na poslední řádce s hlášením 'Not Found'. Nyní si můžeme text prohlédnout. Obě instrukce 'jr' byly skutečně nahrazeny 'djnz'.

2.6.3 Blokové vstupy a výstupy

Aby mohlo být při výstupech upuštěno od únavného zadávání intervalu řádek, kterého se výstup týká, jsou příkazy v hlavním menu určeny pro práci s celým zdrojovým textem. Pokud budete chtít pracovat pouze s částí textu, musíte to řešit pomocí blokových funkcí. Existují dvě. Výpis na tiskárnu a nahrávání zdrojového textu na magnetofon nebo microdrive.

Výpis bloku na tiskárnu , EXT+Wrt

Po označení bloku stačí v Extended menu zvolit příkaz 'w' a celý blok bude vypsán na tiskárně. Pochopitelně ve stejném formátu jako na obrazovce. U tiskáren je šířka válce většinou alespoň 80 znaků na řádku a tak budou řádky zobrazeny celé. Pokud vlastníte tiskárnu s menším počtem znaků na řádku, bude zbytek tištěn na další řádce. Vlastní tiskovou rutinou můžete popřípadě formát změnit. Technické detaily týkající se tiskárny jsou popsány v kapitole 6.2.

Nahrání bloku na vnější médium , EXT+Put

Při volbě extended příkazu 'p' (Put) budete dotázáni na jméno, pod kterým chcete blok nahrát. Pokud použijete tuto funkci v Masm více než jedenkrát, bude vám při dalším použití nabídnuto jméno posledního nahrávaného bloku. Opět můžete použít všechny řádkové příkazy editoru a pro potvrzení ENTER. Na mikropočítači ZX-Spectrum se připouští jméno s maximální délkou deset znaků. Pro microdrive musíte před jméno vložit předponu skládající se z čísla microdrive (1..8) a znaku ':'. Prázdné jméno je nepřípustné a hlásí se okamžitě chyba.

Např.: 'Text 001' jméno pro magnetofon
'3:TextXY' jméno pro microdrive č.3

Na microdrive se zdrojový kód nahrává jako printfile. Není tedy možné nahrát jej do paměti počítače pomocí BASICového příkazu LOAD*"m";x;"jméno"CODE. Na druhé straně je možný přístup přes sekvenční soubory. Čtěte také podrobně příručku dodanou k interfejsu 1, kde jsou přístupy k souborům popsány.

2.6.4 Návrat do hlavního menu , EXT+Quit

Příkaz 'Quit' způsobí, že program opustí editor a vrací se do hlavního menu. Editovaný text zůstává v paměti neporušen a v hlavním menu jej můžeme dále zpracovat.

2.7 Souhrn příkazů editoru

Řádkové příkazy:

CS-2	Zapínání/vypínání režimu velkých písmen
CS-5	Kursor vlevo
CS-8	Kursor vpravo
CS-9	Tabelátor
CS-0	Mazání znaku
SS-D	Smazání řádku
SS-E	Skok na konec/začátek
SS-F	Vkládání extended kódu
SS-I	Změna režimu

Skokové příkazy:

CS-3	skok o stránku dolů (směrem ke konci textu)
CS-4	skok o stránku nahoru
CS-6	posun o řádek dolů
CS-7	posun o řádek nahoru
ENTER	skok na začátek další řádky
SS-Q	skok na začátek textu
SS-W	skok na konec textu
SS-Y	vložení prázdné řádky

Extended příkazy:

Start	nastavení začátku bloku
End	nastavení konce bloku
Copy	zkopírování bloku
Delete	smazání bloku
Move	pohyb bloku
Put	nahrání bloku
Gost	skok na start bloku (GO at STart)
Wrt	vypis bloku
Label	hledání návěští
Find	hledání řetězce
Repl	hledání a nalezení řetězce
Quit	návrat do hlavního menu

3. Hlavní menu

V této kapitole bude kromě příkazu Ass probráno vše, co nabízí hlavní menu.

3.1 Vstupy a výstupy

Těžko rozhodnout, zda popsat nejprve assemblování bez upřesnění vstupů a výstupů nebo naopak. Snad je jednodušší vyložit nejprve vstupní a výstupní příkazy, pak se teprve vrhnout na překlad.

Vstupy a výstupy jsou nejdůležitější části nejen počítačů, ale i jednotlivých programů. Masm vám nabízí tyto možnosti, které jsou uvedeny (a abecedně seřazeny) v hlavním menu:

Code - nahrání přeloženého kódu na vnější médium
Get - nahrání zdrojového textu z vnějšího média
Incl - nahrání zdrojového textu pro inklusi na magnetofon (Include)
Name - změna jména zdrojového textu
Put - nahrání zdrojového textu na vnější médium
Ver - kontrola nahrávky zdrojového textu (Verify)
Wrt - výpis zdrojového textu na tiskárně (Write)
Xref - nahrání informace o návěštích pro křížovou referenci

3.1.1 Výstup na tiskárnu , Wrt

Tento příkaz je určen pro vypsání zdrojového textu na tiskárně. Text je vypisován celý, naformátován je přesně jako na obrazovce, jsou ale zobrazeny celé řádky. Pokud používáte tiskárnu, která má méně než 80 znaků na řádku, budou dlouhé řádky rozděleny. Takové tiskárny se dnes dnes téměř nepoužívají, neoficiálním standartem u mikropočítačů je 80 znaků na řádek. Pokud budete chtít vyřešit výstupní rútinu na tiskárnu tak, aby dlouhé komentáře odřezávala, čtěte pozorně rovněž kapitolu 6.2.

Při tisku je na konci každé řádky testován stav tlačítka SPACE. Jestliže je stisknuto, výstup na tiskárnu je pozastaven až do stisku dalšího tlačítka. Pokud je tímto tlačítkem 'e', vrací se program do hlavního menu. Ostatní tlačítka způsobí pokračování tisku.

Důležité je připomenout, že dnešní tiskárny mají velký paměťový buffer. Je tedy velice pravděpodobné, že ještě několik minut po vašem přerušení bude tiskárna pracovat. Zbavit se dat v bufferu nelze u většiny tiskáren programově, dokonce ani z jejich klávesnice. Buď tiskárnu vypnete a zapnete (to však Spectrum vždy nevydrží a často se zhroutí) nebo si z tiskárny vyvedete signál RESET, který provede její inicializaci.

3.1.2 Vnější médium

Program Masm je připraven pro komunikaci s magnetofonem a rovněž s microdivem. To je ostatně uvedeno ve versi ..TM.. . Firma Miles Gordon Technology nabízí svůj Disciple, který by podle údajů výrobce měl při stejné programové obsluze pracovat s pružnými disky. Interfejs Beta od Technology Research, který se na světě objevil o něco dříve a je tudíž rozšířenější, nemá s programovou obsluhou microdrivu nic společného a s programem Masm pracovat nebude. Informace o interfejsu Beta jsou v kap. 6.3.

K dispozici je následujících 7 příkazů.

Nahrání zdrojového textu do počítače , Get

Příkaz Get je určen k nahrání zdrojového textu z vnějšího média do počítače. Po volbě tohoto příkazu jste dotázáni na jméno nahrávky a současně vám Masm nabízí poslední platné jméno, které bylo vloženo během minulého příkazu Get nebo Name (viz. dále). Toto jméno můžete modifikovat pomocí řádkového editoru a jakmile jste přesvědčeni o jeho bezchybnosti, můžete stisknout klávesu ENTER pro potvrzení. Jestli chcete nahrávat zdrojový text z microdrivu, musí před jménem předcházet předpona skládající se z čísla microdrivu (1..8) a znaku ':'.

Zde existuje malá výjimka a tou je nahrávání zdrojového textu z magnetofonu. Není totiž nutné vkládat jeho jméno. Pokud stisknete klávesu ENTER a potvrdíte tím prázdné jméno, bude do počítače nahráván první zdrojový text, který bude na magnetofonové pásce nalezen. Masm používá z důvodů kompatibility s programem Gens klasický formát nahrávání kódu, jak jej znáte s BASICu a je proto možné, že bude do počítače nahráván soubor typu code, ne však zdrojový text. Tomuto nebezpečí můžete čelit vhodnou volbou jmen (např. používat příponu '.a' nebo '.ass'). Pro přeložené kódy si můžete vybrat z jiné skupiny přípon. Je velice příjemné, když lze u jména přímo rozlišit typ nahrávaného souboru. Pak je jasné, zda se do počítače bude přenášet zdrojový text nebo kód, který se zdrojovými texty nemá nic společného (např. nahraná obrazovka, část hry, text z Taswordu).

Je-li již před nahráváním zdrojového textu v paměti nějaký text, bude nahrávaný text umístěn za jeho konec.

U příkazu Get je rovněž kontrolováno, vejde-li se nahrávaný text do volné paměti. Pokud ne, příkaz končí a je zobrazeno chybové hlášení 'Full memory'.

Po nahrání zdrojového textu se můžete volbou příkazu Edit přesunout do editoru a prohlédnout si zdrojový text.

Nahrání zdrojového textu z počítače , Put

Jak už název napovídá, je příkaz Put určen k nahrávání zdrojových textů z paměti počítače na vnější médium. Po volbě jste požádáni o vložení jména pro nahrávku. Pokud jste ale už za nějaké situace jméno zdrojového textu vložili, bude vám nabídnuto. Může se např. stát, že přehráváte zdro-

jové texty z pásky na microdrive. V příkazu Get vložíte jméno nahrávky na pásku, nahrajete zdrojový text a poté volíte Put. Masm vám nabízí jméno nahrávky. Stačí pouze skočit na začátek jména a v režimu 'Insert' vložit číslo microdrivu (např. 2:) a jméno potvrdit klávesou ENTER. Jméno pro nahrání textu z počítače nesmí být nulové délky, maximální délka je 10 znaků (kromě čísla microdrivu a znaku ':'). Podrobnější informace jsou v kapitole 6.4.

Verifikace zdrojového textu , Ver

Pokud hrozí nebezpečí, že by právě zaznamenaný zdrojový text mohl být na vnějším médiu porušen, ale i pro klid duše, si můžete správnost nahrávky ověřit. Po zavolání tohoto příkazu jste žádáni o vložení jména nahrávky. Masm vám nabídne poslední jméno, které bylo pro zdrojový text použito.

Příkaz Verify umí zkontrolovat na microdrivu jakýkoliv zdrojový text a na pásce zdrojový text, který je umístěný v paměti. Nelze verifikovat nahrání přeloženého kódu. Pokud tento příkaz detekuje chybu, je zobrazeno chybové hlášení 'File Error'. V případě verifikace zdrojového textu z microdrivu je zobrazováno číslo právě kontrolovaného záznamu.

Nahrání zdrojového textu pro inklusi , Incl

Příkaz Incl je určitou výjimkou mezi příkazy týkajících se vnějšího média. Jako jediný je určen pouze pro magnetofon. Jak je popsáno v kap 4., umí Masm provádět tzv. 'inkluse', pomocí nich můžeme překládat zdrojové textv mnohem větší délky než je kapacita paměti. Takový překlad funguje tak, že se assembleru postupně dodává po částech celý zdrojový text. V pausách mezi těmito částmi je překládán. Na rozdíl od microdrivu, kde si můžeme přenos na chvíli zastavit, u pásky přerušení přenosu dat neexistuje. Proto musel být Masm doplněn dalším příkazem pro nahrání zdrojového textu na magnetofon v rozkouskované podobě.

Po zavolání příkazu Incl budete opět dotázáni na jméno. Toto jméno je určeno výhradně pro pásku, nelze použít předpony pro nahrání na microdrive. Při dalším zavolání tohoto příkazu vám bude pochopitelně nabídnuto poslední vložené jméno. Po potvrzení platnosti jména se začne nahrávat zdrojový text, který je umístěn v paměti.

Přejmenování zdrojového textu , Name

Příkaz Put umožňuje sice, aby byl zdrojový text jednoho jména nahrán pod jménem jiným, ale původní jméno pro další práci zůstává. Budete-li nahrávat zdrojový text příkazem Put znovu, bude vám opět nabídnuto původní jméno. Pro trvalou změnu jména souboru, který je právě umístěn v paměti, použijte proto příkaz Name.

Po jeho zavolání můžete v řádkovém editoru opravit nové jméno a klávesou ENTER změnu potvrdit. Toto jméno je trvalé a bude vám nabízeno vždy, když požádáte o příkaz Put.

Nahrání přeloženého kódu na vnější médium , Code

Příkaz Code byl do programu umístěn, aby uživatel mohl přeložený kód nejen spustit, ale i zaznamenat. Bylo by nešikovné provádět tuto činnost v BASICu příkazem SAVE, proto je příkaz v hlavním menu. Stejně jako u nahrávání zdrojového textu, budete dotázáni na jméno pro nahrávání. Pokud příkaz vyvoláte podruhé, bude vám nabídnuto jméno, které jste vložily při prvním zavolání. A tak dál. Opět platí, že jména nahrávek určených pro microdrive začínají číslem microdrivu a znakem ':'.

Příkaz Code funguje ale za určitých okolností. Stručně by se dalo říci, že nahrává, když má co. To znamená:

- byl přeložen zdrojový text a nedošlo k žádné chybě
- nebyl zvolen Option 2 (negenerování kódu) viz. kapitola 4.

Přeložený strojový kód může být nahráván na pásku i microdrive. Na microdrivu bude typu kód (code), tzn. bude přístupný tudíž BASICovým příkazem LOAD *"m";x;"name" CODE .

Příkaz Code pracuje tak, že si v systémových proměnných najde adresy začátku a konce přeloženého strojového kódu a ty předá nahrávací rutině. Pokud ale volíte Option 16 (umístění kódu za tabulku symbolů), bude kód správně nahrán, ale v hlavičce (nebo direktoráři) nahrávky bude startovní adresa kde kód leží po překladu a ne odkud je adresován (=kam bude nahrán). Při nahrávání do počítače musíte u takového kódu specifikovat startovní adresu.

Např.:

Budeme potřebovat přeložit krátkou rutinu, která musí být umístěna od adresy 30000. Na této adrese leží ale ještě program Masm, překlad do tohoto prostoru je nemožný. Jedinou možností je překlad s volbou 16 (umístění kódu za tabulku symbolů). Strojový kód bude překládán přesně jako na původním místě, pouze bude po překladu ležet za tabulkou symbolů. Z tohoto místa si jej můžeme nahrát příkazem Code třeba na magnetofon pod jménem 'kod'. Nyní Masm opustíme příkazem Basic a provedeme:

```
LOAD "kod" CODE 30000
```

Přeložený kód bude nahrán do paměti na správné místo a může být pochopitelně spuštěn. Do Masm se ale vrátit nemůžeme, protože jsme jeho část přehráli programem 'kod'.

Příkaz Code nahrává pouze poslední přeložený blok. Blokem se rozumí ta část přeloženého strojového kódu, které vznikla z té části zdrojového textu, kde je maximálně jedna pseudoinstrukce ORG. Pokud tato existuje, bude začátkem bloku strojový kód, který vznikl přeložením zdrojového textu od ORG dále. Shrneme-li tuto 'definici', platí tyto dvě pravidla:

- pokud není v programu jediná pseudoinstrukce ORG, je situ-

ace nejjednodušší; bude nahráván celý kód
- v ostatních případech bude nahráván kód od adresy specifikované poslední pseudoinstrukcí ORG dále

Např. v programu :

```
        Org   25000
Test    Ld    a,10
        Jp    Test2

        Org   26000
Test2   Ld    a,20
        Ret
```

bude po překladu s volbou 16 nahrána příkazem Code pouze část strojového kódu pocházející ze zdrojového textu od pseudoinstrukce ORG 26000 dál. Tedy 'Ld a,20' a 'Ret'. Budete-li chtít změnit čítač adres při překladu, použijte raději pseudoinstrukci DEFS. V minulém příkladě by vypadala opravená verse s pseudoinstrukcí DEFS takto:

```
        Org   25000
Test    Ld    a,10
        Jp    Test2

        Defs  26000-$       ; zde je oprava
Test2   Ld    a,20
        Ret
```

Pro správné pochopení příkazu Code čtěte kapitolu 4.

Nahrání návěští pro křížovou referenci , Xrf

Jak používat křížové reference je popsáno v kapitole 4. Zde se pouze zmíníme o jedné jejich části, která patří do této kapitoly. Snad by bylo rozumnější přečíst nejprve kapitolu 4. a pak se k příkazu Xref vrátit. Správné pochopení jeho činnosti bude pak mnohem jednoduší.

Pokud přeložíme zdrojový text a nastavenou volbou 64 a nebyla nastavena volba 8 ani 16 pro křížové reference (a překlad proběhl bez chyb), je ve speciálním bufferu vygenerován zdrojový text určený pro křížové reference. Ten můžeme příkazem Xref nahrát na magnetofon nebo microdrive. I zde platí všechny zásady o psaní jmen pro vnější média. Stane-li se, že po překladu bude Xref-buffer prázdný, příkaz nebude vůbec vykonáván. Nebudete tedy dotázáni ani na jméno.

Toto nahrání je vhodné udělat ihned po překladu. Řada příkazů může totiž data v bufferu zničit. Pokud Masm zjistí, že jste po překladu použily některé 'pro Xref buffer' destruktivní příkazy, Xref nebude vůbec vykonáván.

Destruktivní příkazy jsou : Basic (Bas)
Delete (Del)
Edit
Get
Memory (Mem)
Run

Po těchto příkazech Xref nefunguje. Další překlad, při kterém nevznikne Xref buffer, bude také chápán jako destruktivní.

3.2 Co zbylo v hlavním menu

Do této kapitoly byly zahrnuty příkazy, které nabízí hlavní menu a nebyly probrány v dosavadních kapitolách (kromě příkazu Assembly, na který si necháme samostatnou kapitolu). Jsou to příkazy:

Bas - návrat do BASICu
Del - smazání zdrojového textu
Mem - nastavování velikosti bufferů
Run - spuštění strojového kódu
Text - informace o textu

Návrat do Basicu , Bas

Snad by bylo vhodnější psát 'návrat do operačního systému' Spectra, uživatelé jsou ale připraveni reagovat na slovo 'Basic'. Tak je tu.

Tímto příkazem se tedy vrátíte do Basicu. Program obnoví, vše co změnil a co je pro interpret jazyka Basic tak nutné, povolí přerušení a už jsme v Basicu. Co bude dál už nezáleží na Masm, ale na vás.

Smazání zdrojového textu , Del

Příkazem Delete žádáte Masm o smazání zdrojového textu v paměti. Nemusíte se bát, že omylem stisknete tlačítko 'D' a vše bude zničeno. Opak je pravdou. Po zvolení tohoto příkazu budete dotázáni, zda jste si skutečně svou volbou jisti. Pokud ano, stiskněte klávesu 'Y', všechny ostatní tlačítka způsobí návrat do hlavního menu. Pokud budete chtít smazat třeba jen část zdrojového textu, přejděte do editoru a proveďte smazání pomocí blokových příkazů (kap. 2.6.1).

Změna velikosti bufferů , Mem

Tento příkaz je určen pro změnu velikosti makrobufferu a bufferu pro inklusi. Po zavolání tohoto příkazu jste dotázáni na velikost bufferu pro inklusi ('Include buffer :)' z pásky nebo microdrivu. Velikost se zadává v bajtech. Pokud stisknete ENTER, zůstane zachována původní hodnota. Minimální hodnota je 256 bajtů, menší hodnoty nelze volit. Co je buffer pro inklusi a k čemu se používá se dočtete v kapitole 4.6.

Při překladu zdrojového textu, v němž se vyskytují definice makroinstrukcí, je potřeba tyto definice zaznamenat a v případě potřeby použít. Tato věc nelze řešit bez bufferu,

který je v tomto případě nazýván makrobuffer. Po nahrání programu Masm je jeho velikost 0. Budete-li chtít tuto velikost změnit, příkaz Mem vám v tom rovněž pomůže. Poté, co jste vložili velikost bufferu pro inklusi, budete dotázáni na velikost makrobufferu ('Macro size :'). Pro optimální volbu velikosti je potřeba trochu zkušenosti s prací v assembleru. Je důležité si uvědomit, že se do makrobufferu kopirují celé definice makroinstrukcí. Jejich velikost lze s trochou zkušenosti odhadnout. Přečtěte si také kapitolu 6.1 o vnitřní reprezentaci textu, pak budete mít o jeho velikosti jasnější představu.

Spuštění přeloženého kódu , Run

Přeložený program může být spuštěn pokud platí:
- zdrojový text byl přeložen bez chyb
- nebyla zvolena volba 2 ani 16
- byl určen vstup do strojového programu pseudoinstrukcí ENT

Při splnění těchto podmínek jsme na konci překladu informováni o adrese, od které bude program spuštěn. Na tuto adresu skáče příkaz Run. Rovněž pro tento příkaz existuje skupina destruktivních příkazů. Po nich nebude přeložený program spuštěn. Je to tato skupina příkazů:

Ass - překlad
Bas - návrat do BASICu
Del - smazání zdrojového textu
Mem - nastavování velikostí bufferů
Text - informace o textu

Při skoku do přeloženého programu je situace v počítači následující:

- obrazovka je smazána
- kanálové informace mají standartní hodnoty
- registry IY,HL' mají hodnoty stejné jako v okamžiku startu programu Masm
- registr I má hodnotu stejnou jako v okamžiku startu programu Masm
- přerušení je povoleno (je zapnut IM1)

Během běhu vašeho programu nesmí být změněn zásobník od SP výše a po návratu musí SP ukazovat na stejné místo jako při skoku do vaší rutiny. Po návratu musíte zajistit IM1 a původní hodnotu registru I. Vše ostatní si Masm doplní.

Na závěr pouze krátkou poznámku : než použijete příkaz Run, nahrajte si raději zdrojový text. Hlavně pokud se strojovým kódem začínáte.

Informace o zdrojovém textu , Text

Může se stát, že budete provádět nějaké změny uvnitř Masm nebo v datech nebo vás bude prostě zajímat délka a umístění zdrojového textu. Je nepohodlné prohledávat systémové pro-

měnné (ještě ke všemu v BASICu Spectra) a proto je k dispozici příkaz Text, který poskytuje informace o textu. Jeho startovní adresu, cílovou adresu a navíc je zde informace o versi programu Masm. Od startovní adresy zdrojového textu jsou odvozovány všechny ostatní důležité adresy. Paměťová mapa je v kapitole 6.5.

4. Příkaz Assembly , Ass

Jediným příkazem, který nebyl dosud probrán, je Ass. S jeho pomocí se dostaneme až na samý vrchol při práci v programu Masm. Po nahrávání, psaní a zase nahrávání můžete nyní zdrojový text přeložit.

4.1 Formát textu

Znovu si uvedeme příklad, jaký je formát při psaní v jazyku symbolických instrukcí :

```
1       8     13              24
Label   Ld    a,10            ; komentář

                              ^ zde začíná komentář (od ";")
              ^ parametry instrukce (pokud existují)
        ^ jméno instrukce
^ návěští
```

Jsou přípustné určité modifikace tohoto formátu, ty byly popsány v kap. 2.2. Zde se budeme zabývat jednotlivými částmi řádky. Na začátku řádky může být umístěn Label (návěští), může zde být ';', ale i znak '*', který signalizuje příchod povelu assembleru (popsány v kapitole 4.6). Nejprve tedy návěští.

4.1.1 Návěští

Návěští je symbol, který v programu representuje osmi nebo šestnácti-bitovou informaci. Může být být použit k určení adresy určité instrukce nebo datového pole, rovněž může být použit jako konstanta definovaná pomocí EQU nebo DEFL. Assembler hlídá jeho bitovou délku. Pokud budete chtít použít 16-ti bitovou hodnotu v místě, kde je povoleno 8 bitů, bude hlášena chyba. Např.:

```
    Label   equ  40000
            db   Label
```

způsobí vznik chyby 10. Návěští může obsahovat libovolný počet znaků, ale pouze prvních šest je použito. Následující návěští

```
    Label_one     a     Label_two
```

jsou z pohledu assembleru totožná. Jako návěští nesmí být použito reservované slovo (jsou uvedena v příloze E). Návěští v Masm jsou dělena do třech skupin:

- normální
- vybraná
- lokální

Probereme si je tedy podrobněji.

Normální návěští

Tato návěští jsou pro normální používání. Na rozdíl od vybraných a lokálních nemají žádnou speciální vlastnost. Musí být tvořena posloupností znaků z množiny '0'..'9', '$' a 'A'..'z'. Návěští nesmí začínat číslicí ani znakem '_' (znakem "_" by vzniklo tzv. vybrané návěští). Masm rozlišuje u návěští velká a malá písmena. Uvedeme si několik příkladů možných návěští:

platná:	neplatná:
Loop	X*Y
A1	hl
a1	a
XY[23]	XY(23)
we\re	we/re
DJNZ	x1

Použití návěští např.:

```
; rutina pro tisk řetězce

String ld   hl,Data      ; vezmi začátek dat
       ld   a,(Length)   ; přečti délku
       ld   b,a
Cycle  ld   a,(hl)       ; čti řetězec
       call out          ; zavolej výstupní rutinu
       inc  hl           ; zvětši ukazatel
       djnz Cycle        ; konec?
       ret               ; ano

Data   defm "Hello"
Length defb 6
```

Symbol 'Data' representuje adresu, od které začíná v paměti řetězec 'Hello'. Symbol 'Length' ukazuje na paměťovou buňku, kde leží konstanta 6.

Návěští vybraná

Vybraná návěští se mohou používat stejně jako normální návěští. Jejich speciální význam, který se týká křížových referencí, bude probrán později. Zde se pouze zaměříme na způsob jejich vytváření. Pro vybraná návěští platí stejná pravidla jako pro normální návěští s tím rozdílem, že první znak musí být '_'. Ten se u normálního návěští nesmí na této pozici vyskytovat.

Příklady vybraných návěští: _label
_123
_x_w

Znak '_' se zahrnuje do celkové délky návěští.

Návěští lokální

Význam lokálních návěští bude probrán v kapitole 4.5, zde popíšeme pouze pravidla pro jejich psaní. Platí naprosto vše co bylo popsáno u normálních návěští. Používání těchto návěští má ale daleko přísnější pravidla. Aby byly od vybraných a normálních návěští odlišena, musí být prvním znakem ':'.

Příklady lokálních návěští: :label
:123
:x_w

Pouze ':' se jako návěští nepřipouští a je hlášena chyba. Znak ':' se zahrnuje do celkové délky návěští.

4.1.2 Instrukce a registry

Instrukce bychom mohli rozdělit na standartní a tajné. Mezi ty standartní patří všechny ty, které zveřejnila firma Zilog. Tajné instrukce byly objeveny později. Funkce některých logicky vyplývají z návrhu mikroprocesoru, jiné byly zjištěny experimentálně. Na světě existuje několik výrobců Z-80, není však potvrzeno, že všechny tyto instrukce budou na jejich versi Z-80 pracovat správně. Existuje ale určitá skupina instrukcí, u kterých je zaručena správná funkce na všech mikroprocesorech typu Z-80. Jedná se o instrukce s polovinami indexových registrů IX a IY. Jejich poloviny jsou v Masm nazvány:

XL spodních 8 bitů registru IX
XH horních 8 bitů registru IX
YL spodních 8 bitů registru IY
YH horních 8 bitů registru IY

To jsou tedy další čtyři registry, které se uplatní v několika instrukcích.

U všech instrukcí a registrů Masm nerozlišuje velká a malá písmena (na rozdíl od návěští). Je tedy úplně jedno, zda napíšete Ldir, LDIR, ldir nebo lDiR.

Standartní instrukce byly mnohokrát popsány, zaměříme se tedy na část tajných instrukcích. Registry XH,XL,YH a YL mohou být použity v instrukcích:

- aritmetických jednobajtových

ADD, ADC, SUB, SBC, AND, XOR, OR, CP, INC, DEC

např.:
```
add  a,xl
sub  yl
dec  yh
cp   xh
```

- přenosu jednobajtových

LD

```
např.:      ld   a,xl
            ld   yh,e
            ld   c,xh
            ld   xh,123
```

Protože jsou instrukce pracující s IX a IY vlastně instrukce pracující s registrem HL doplněné o vhodný prefix, nesmí být použita kombinace, kde se vyskytují oba registry nebo kde dochází ke kolizi prefixů:

```
            ld   h,yl          ; nelze
            ld   xl,(ix+10)    ; nelze
            ld   xl,yl         ; nelze
            ld   yl,(ix+10)    ; nelze
```

Registry XH,XL,YH a YL se nesmí vyskytovat také v těchto instrukcích (přestože se také nabízejí):

BIT, SET, RES, OUT, IN, RR, RL, RRC, RLC, SRA, SLA.

Určité možnosti 'navíc' poskytuje Masm v podmíněných skocích a návratech. Standartní mnemonika připouští pouze podmíněné skoky v závislosti na stavu vlajek a také to v instrukci vyjadřuje. V některých chvílích není ale toto řešení nejlogičtější. Např.:

```
            cp   10
            jp   z,lab
```

Skok se provede, pouze když bude mít akumulátor hodnotu 10. Tedy musí být vlajka Zero (nula) nastavena. Aby byly podmínky přirozenější, umožňuje Masm použít rovněž druhou skupinu podmínek. Jsou to:

význam	původní podmínka	nová podmínka	význam
=	Z	EQ	equal
<>	NZ	NE	not-equal
<	C	LT	little
<=	nelze	nelze	
>=	NC	GE	greater or equal
>	nelze	nelze	

. Předchozí příklad může být tedy přepsán do logičtějšího tvaru:

```
            cp   10
            jp   eq,lab
```

Řešení je o jeden znak delší, ale zato přehlednější.

4.1.3 Výrazy

Výraz je vstupní operand obsahující jeden člen nebo více členů oddělených operátory. Umožňují programátorovi vkládat hodnoty, které by musel zvlášť spočítat nebo hodnoty, které je možné určit až při překladu.

Zde jsou definovány členy a operátory:

Členy:

dekadické konstanta	např. 1029
hexadecimální konstanta	např. #405
binární konstanta	např. %10000000101
znaková konstanta	např. "1","c","_",atd.
návěští	např. Lab, _lab, :0
parametr makroinstrukce	např. =0, =c
čítač adres	$

Operátory:

sčítání	+
odečítání	-
násobení	*
dělení celočíselné	/
dělení modulo	?
logický OR	@
logický AND	&
logický XOR	!

Členy

Hexadecimální a binární konstanty nejsou definovány jako ve většině assemblerů (123H, 0A2H, 1010B, ..). Masm používá stejný tvar jako Gens, kde vychází délka zápisu takových konstant někdy kratší a zobrazení je přehlednější. Pro jednotlivé hexadecimální cifry jsou použity znaky '0'..'9' a za nimi následují 'A'..'F' nebo 'a'..'f'. Ani zde Masm nerozlišuje velká a malá písmena.

Znakové konstanty jsou uzavírány do znaku '"'. Pokud definujeme znakovou konstantu délky 1, může mít tvar "a", "A", "@", ale i """. Pro znakovou konstantu délky větší než 1 se znak '"' nesmí v konstantě vyskytovat ("123as"dfd56" nelze).

U návěští existuje jediné omezení,o kterém se psalo již v kapitole o návěštích. Do místa, kde je očekávaná osmibitová konstanta, se nesmí přiřadit konstanta šestnáctibitová. Pokud ano, je hlášena chyba. Pouze u relativních skoků můžeme použít skok na návěští, přestože je parametr těchto instrukcí pouze osmibitový. Masm sám vytvoří osmibitový doplněk.

U parametru makroinstrukce platí jediné omezení. Stejně jako u návěští musí být hlídáno jak velký parametr přiřazujeme.

Čítač adres '$' representuje adresu, na kterou se bude ukládat strojový kód (ať už instrukce nebo data).

Operátory

Při řešení výrazu postupuje kalkulátor zleva doprava a nerozlišuje priority jednotlivých operátorů. Napíšete-li výraz:

```
1+2*3
```

bude výsledek 9 a nikoliv 7. Ve zdrojových textech se málokdy používá složitější výraz, proto byl kalkulátor řešen tak jednoduše. Pochopitelně to má vliv i na rychlost překladu. Je důležité připomenout, že se ve výrazech nesmí vyskytovat mezery. Kromě narušení funkce autotabelace může dojít k tomu, že zapomenete operátor a kalkulátor ohlásí špatné výsledky.

Dále je nutno poznamenat, že celá aritmetika v Masm pracuje šestnáctibitově. Všechny vstupní hodnoty jsou převedeny na šestnáctibitová čísla v intervalu 0..65535. Čísla od 32768 výše nejsou po převedení chápána jako záporná, ale jako velká kladná čísla. To je rozdíl oproti programu Gens, který z tohoto důvodu hlásil v mnoha případech zcela nesmyslné výsledky. Vkládat záporná čísla je pochopitelně povoleno stále. Vložíte-li například -1, bude převedena na 65535 (#ffff). Při přiřazení na místo, kde je očekávána osmibitová hodnota, bude použita pouze spodní polovina slova.

V kalkulátoru není prováděna žádná typová kontrola, jak ji znáte z vyšších jazyků. Můžete si dovolit například tento výraz:

```
123+"x"-#a2+%111
```

Kalkulátor výraz vyhodnotí, aniž by se mu zdálo něco divného. U assemblerů se totiž typová kontrola nepoužívá. Jako ukázku si zkusíme několik příkladů řešení výrazů:

```
%10101010&%00001111     = %00001010
50000?256               = 80
40000+30000             = 4464
#9c40+#7530             = #1170
1-3                     = -2 = #fffe
```

Rovněž si na rozdíl od Gensu můžete vypočítat výrazy:

```
400*150                 = 60000
60000/150               = 400
```

Jestliže budete postaveni před úkol vyřešit výraz, který nelze pomocí kalkulátoru řešit v jednom výrazu, musíte si pomoci jinak. Např.:

požadujeme výraz AA^2+BB^2 neboli AA*AA+BB*BB. S pomocí pseudoinstrukce DEFL si nadefinujeme pomocnou proměnnou, kterou použijeme takto:

```
Pom     defl BB*BB
Vyraz   equ  AA*AA+Pom
```

Až si přečtete informace o DEFL uvidíte, že takto definovanou proměnnou můžete používat i několikrát.

4.2 Pseudoinstrukce

Pseudoinstrukce jsou řídící povely assembleru. Nejsou překládány do strojového kódu, řídí pouze assembler při překladu.

Pseudoinstrukce jsou zapisovány ve stejném formátu jako výkonné instrukce. Může jim předcházet návěští (při EQU a DEFL jim předcházet musí) a mohou být následované komentářem.

Pseudoinstrukce ORG

Pokud nebude určeno, kam uložit výsledný strojový kód, bude jej Masm ukládat za tabulku symbolů. V mnoha případech potřebujeme, aby byl ukládán od jiné adresy. Pak můžeme využít pseudoinstrukci ORG, která mění hodnotu čítače adres. Syntaxe je:

```
ORG  numerický_výraz
```

Jestliže se změní čítač adres na hodnotu, která leží v rozsahu programu Masm a jeho dat, přitom není zvolena ani volba 2,16 ani 32 pro překlad, pak bude hlášena chyba assembleru číslo 19.

Je-li při překladu zvolena volba 16, bude kód ukládán za tabulku symbolů, ale spustitelný bude až po přehrání na místo určené pseudoinstrukcí ORG. Vyskytuje-li se ORG v programu vícekrát, bude se při této instrukci měnit čítač adres, ale ukládání do paměti bude prováděno stále za sebou. Podrobnější popis je v kapitole 4.7. Masm nevyžaduje, aby měly parametry pseudoinstrukce ORG vzrůstající tendenci. Při volbě 16 vznikne ale v paměti pěkný chaos. Následující program demonstruje tuto situaci:

```
Start  ORG  #fefe        ; začátek prvního bloku
       .    .
       .    .
       jp   Start2       ; skok na druhý blok

Start2 ORG  #fe00        ; začátek druhého bloku
       .    .
       .    .
```

Při překladu s nastavenou volbou 16 je vhodnější použít místo dalších pseudoinstrukcí DEFS (to je ale použitelné pouze pro zvyšování čítače adres).

Pseudoinstrukce EQU

Této pseudoinstrukci musí předcházet návěští. Hodnota návěští bude nastavena na hodnotu výrazu za pseudoinstrukcí

EQU. Formát:

```
Label  EQU  numerický_výraz
```

Všechny členy v numerickém výrazu musí být v okamžiku definování návěští známy. V opačném případě bude hlášena chyba číslo 13.

Při generování tzv. EQU-souborů, které jsou určeny pro křížovou referenci, se jevila pseudoinstrukce EQU jako příliš dlouhá a často nemohl být EQU-soubor vytvořen. Proto byl vytvořen ještě jeden kratší formát pro přiřazení výrazu a tím je znak '='. Lze použít místo kterékoli pseudoinstrukce EQU ve formátu:

```
Label  =  numerický_výraz
```

Zdrojové texty psané pomocí '=' jsou pochopitelně kratší a dokonce se jeví jako přehlednější. Napíšete-li definici konstant takto:

```
cr   =   #0d
lf   =   #0a
bs   =   #08
sp   =   #20
```

je vše okamžitě jasné.

Pseudoinstrukce DEFB

Tato pseudoinstrukce slouží k uložení osmibitových hodnot do paměti od adresy, která je zaznamenána v čítači adres. Po každém uložení se čítač adres zvětší o 1. Formát:

```
DEFB num_výraz,num_výraz,.....
```

Výsledkem všech numerických výrazů musí být číslo s hodnotou max. 255 (#ff, %11111111). Jednotlivé hodnoty výrazů jsou postupně ukládány za sebe. Jejich počet je limitován délkou řádky. Při hodně výrazech na řádce se stává zdrojový text nepřehledný a proto se většinou používá 3 až 5 výrazů.

Např. :

```
DEFB 10,#0b,"c","a",3+2
```

umístí do paměti sekvenci bajtů 10,11,99,97,5 dekadicky.

Pseudoinstrukce DEFW

Tato pseudoinstrukce slouží k uložení šestnáctibitových hodnot do paměti od adresy, která je zaznamenána v čítači adres. Po každém uložení se čítač adres zvětší o 2. Formát:

```
DEFW num_výraz,num_výraz,.....
```

Výsledkem všech numerických výrazů musí být číslo s hodno-

tou max. 65535 (#ffff, %1111111111111111). Jednotlivé hodnoty výrazů jsou postupně ukládány za sebe. Jejich počet je limitován délkou řádky.

Např. :

```
DEFW 1024,#0b,"c","a",3+300
```

umístí do paměti sekvenci bajtů 0,4,11,0,99,0,97,0,47,1.

Pseudoinstrukce DEFS

Pseudoinstrukce DEFS je určena ke změně hodnoty čítače adres nebo k vytvoření vektoru stejných bajtů. Používá se ve dvou formátech:

- verse kompatibilní s Gens. V tomto formátu se určuje pouze o kolik se má zvýšit čítač adres. Např.:

  ```
  DEFS numerický_výraz
  ```

  Hodnota čítače adres bude zvětšena o hodnotu numerického výrazu a prostor, který je takto 'přeskočen', zůstává neporušený.

- v druhé versi formátu je pseudoinstrukce doplněna ještě o jeden parametr a tím je hodnota:

  ```
  DEFS numerický_výraz,hodnota
  ```

  Tuto pseudoinstrukci zpracuje Masm tak, že zvětší čítač adres o hodnotu numerický_výraz a prostor, který přeskakuje, bude vyplněn posloupností bajtů 'hodnota'.

Jednou z výhod DEFS oproti ORG je i to, že budou prostory vzniklé změnami čítače adres zachovány i při překladu s nastavenou volbou 16. Např. strojový kód vzniklý z tohoto programu :

```
        Org   25000
Test    Ld    a,10
        Jp    Test2

        Org   26000
Test2   Ld    a,20
        Ret
```

bude po překladu s volbou 16 'nahuštěn' za sebe aniž by byla mezi jeho dva bloky vložena mezera v délce téměř 1000 bajtů.

Mnohem vhodnější je změnit čítač adres pomocí pseudoinstrukce DEFS.

```
        Org   25000
Test    Ld    a,10
        Jp    Test2
```

```
        DEFS 26000-$      ; zde je oprava
Test2   Ld   a,20
        Ret
```

Po překladu s nastavenou volbou 16 bude prostor vzniklý použitím pseudoinstrukce DEFS zachován.

Pseudoinstrukce DEFM

Pseudoinstrukce DEFM definuje obsah n-tice po sobě jdoucích bajtů od adresy uložené v čítači adres, kde n je délka vkládaného řetězce. Za pseudoinstrukcí smí být umístěn pouze jeden řetězec. Syntaxe:

```
DEFM "řetězec"
```

V řetězci se nesmí vyskytovat znak '"'. Délka řetězce je limitována délkou řádky. Pokud za řetězcem nenásleduje komentář a řádka tedy končí, není ukončovací znak řetězce '"' nutný. Konec řádky je v takovém případě chápan jako ukončovací symbol řetězce. Do řetězce můžete uložit všechny znaky, které vám editor programu dovolí (kromě '"'), jsou to tedy znaky z intervalu 32..126. Kód 127 má v normě ASCII řídící charakter, proto jej ani Masm nechápe jako znak.

Pseudoinstrukce DEFL

Pseudoinstrukce DEFL má podobný význam jako EQU. I před ní musí předcházet návěští, kterému je přidělena hodnota numerického výrazu. Syntaxe je:

```
Label  DEFL numerický_výraz
```

Všechny členy v numerickém výrazu musí být v okamžiku definování návěští známy. V opačném případě bude hlášena chyba číslo 13.

Návěští definované pseudoinstrukcí DEFL má ale jednu důležitou vlastnost, kterou si udržuje od definice až do konce překladu. Může být totiž kdykoli změněno. Změna může být provedena jak použitím před výkonou instrukcí, tak pseudoinstrukcí EQU nebo samotnou DEFL. Příklad:

```
Label  DEFL 100          ; definice návěští
       ld   a,Label      ; použití
Label  inc  c            ; změna hodnoty návěští
       djnz Label        ; požití při skoku
Label  equ  200          ; změna hodnoty pomocí equ
       defb Label        ; použití v defb
Label  DEFL 300          ; změna hodnoty pomocí DEFL
```

Pseudoinstrukce DEFL může být tedy použita ve smyslu proměnné. Její hodnota může být kdykoli změněna. To je ale důvod pro zvýšenou opatrnost. Může se stát, že návěští nebude

mít hodnotu, jakou u něj očekáváte. Např. v programu:

```
Open    jp   Label

Label   DEFL #1601

Label   equ  200

Set     ld   a,Label
        ret
```

Zde souvisí náš problém s tím, že je Masm dvouprůchodový assembler. Při prvním průchodu zjistí hodnoty návěští a v průchodu druhém generuje cílový program. Náš program projde prvním průchodem a jeho opuštění má Label hodnotu 200. V druhém průchodu bude tedy generována instrukce 'jp 200' místo 'jp #1601'. Buďte proto při používání pseudoinstrukce DEFL velice opatrní.

Pseudoinstrukce ENT

Pseudoinstrukce ENT určuje, na jakou adresu má skákat příkaz Run (hlavní menu). Její formát je:

```
ENT  numerický_výraz
```

Numerický výraz adresu jednoznačně určuje. Příkaz Run bude pracovat pouze po bezchybném překladu. Všechny podmínky jsou popsány v kapitole 3.2. Budete-li chtít definovat skok na určitou pozici programu a nechce se vám zjišťovat její adresu, můžete použít symbol '$', který reprezentuje čítač adres:

```
ENT  $
```

Tento příkaz způsobí, že příkaz Run skočí do místa, kde jste pseudoinstrukci použili.

Pseudoinstrukce IF

Tato pseudoinstrukce je stejně jako ELSE a END určena pro podmíněný překlad. Používá se v syntaxi:

```
IF   numerický_výraz
```

Část zdrojového textu za IF bude překládána, pokud bude hodnota numerického výrazu pravdivá (zde se jako pravdivá hodnota chápe 'nenulová' hodnota). Jestliže je hodnota nulová, bude zdrojový text text ignorován až do pseudoinstrukce ELSE nebo END.

Pseudoinstrukce ELSE

Pseudoinstrukce ELSE se používá jako doplňující nepovinná instrukce k IF. Zdrojový text za ELSE bude překládán pouze v případě, že numerický výraz za IF je nepravdivý (tedy nulový). Pokud byl zdrojový text před ELSE překládán, bude překlad vypnut a naopak. Při vypnutí překladu bude pokračovat až od pseudoinstrukce END. Syntaxe ELSE je jednoduchá:

```
ELSE
```

Pseudoinstrukce ELSE není po IF povinná. Lze použít i konstrukci IF-END.

Pseudoinstrukce END

Pseudoinstrukce END se používá jako doplňující instrukce k IF. Uzavírá konstrukci podmíněného překladu IF-ELSE-END nebo IF-END. Po pseudoinstrukci END bude překlad zapnut. Syntaxe je opět jednoduchá:

```
END
```

! Pseudoinstrukce pro podmíněný překlad nesmí být vkládány do sebe jako třeba ve vyšších jazycích. Konstrukce:

```
IF    numerický_výraz_1
.
.
IF    numerický_výraz_2   ; začátek vnořené části
.
.
END                       ; konec vnořené části
.
.
ELSE
.
.
END
```

je nepřípustná. Jakmile bude numerický_výraz_1 neplatný, bude ignorováno vše až do pseudoinstrukce END, u ELSE bude překlad opět zakázán a za další pseudoinstrukcí END opět povolen.

Uživatelé assemblerů často neumí podmíněný překlad využít. Přitom je to jeden z nejmocnějších prostředků, které jsou k dispozici. Jako příklad uvedeme situaci, kdy potřebujeme napsat obslužné rutiny pro zatím neexistující hardware. Není možné, aby se programátor pustil do programování až poté, co bude hardware dokončen (někde to možné bohužel je). Řešení je jednoduché. Pokud je známo, jak se bude zařízení chovat (i to se někdy stává), stačí jeho chování nasimulovat programově a pracovat zatím na programové obsluze o jednu úroveň výše. Vezmeme například komunikaci s modemem. Přesto, že jej ještě dlouho neuvidíme na pultech prodejen, již dnes můžeme odladit zdrojové programy, které s ním budou někdy

pracovat. V opačném případě bychom museli celou část týkající se modemu vypustit a někdy v budoucnu, kdy už o programu budeme sotva něco vědět, ji zase dopisovat. Zkusme to třeba takto:

```
; na začátku definuji konstantu vyjadřující, zda se bude
; překládat program pro skutečný modem nebo pouze simu-
; laci

Modem  =    0            ; stále pouze simulace

; zde je část používající modem
; zatím si vystačíme s rutinou Write

        .
        .
      call Write         ; vyšli znak
        .
        .

; zde jsou rutiny pro obsluhu

Write  IF  Modem
        .
; rutina pro skutečný zápis
        .
       ELSE
        .
; rutina pro simulaci zápisu
        .
       END
```

A až někdy připojíte ke svému stroji skutečné zařízení, nebudete muset prohlížet celý program a modifikovat jej. Pouze na začátku programu zaměníte konstantu 0 za 1, provedete překlad a program bude pracovat celý.

Pseudoinstrukce MAC

Pseudoinstrukce MAC je určena k definici makroinstrukcí. Její podrobný popis je 4.3.

Pseudoinstrukce ENDM

Pseudoinstrukce ENDM je určena k definici makroinstrukcí. Její podrobný popis je 4.3.

Pseudoinstrukce LOC

Pseudoinstrukce LOC je určena k definici lokálních bloků. Její podrobný popis je 4.5.

Pseudoinstrukce ENDL

Pseudoinstrukce ENDL je určena k definici lokálních bloků. Její podrobný popis je 4.5.

Zkrácené verse pseudoinstrukcí

Protože paměť mikropočítače napatří k těm největším a velice rychle se plní, obsahuje Masm zkrácenou versi pseudomnemoniky. Všechny zkrácené pseudoinstrukce pracují přesně tak, jako jejich delší ekvivalenty.

Pseudoinstrukce	Zkrácená verse
equ	=
defb	db
defw	dw
defs	ds
defl	dl
defm	dm

4.3 Makroinstrukce

Program Masm umožňuje definovat a používat makroinstrukce. Makroinstrukcí rozumíme posloupnost instrukcí, jejichž parametry mohou být měněny. Programátor pak nemusí znovu a znovu psát celou sekvenci, ale stačí když napíše jméno makroinstrukce.

Makroinstrukce se definuje pomocí pseudoinstrukcí MAC a ENDM:

```
Jméno  MAC              ; začátek makroinstrukce
       .
       .
       definice makroinstrukce
       .
       .
       ENDM             ; konec makroinstrukce
```

Při použití píšeme pouze jméno a parametry:

```
       .
       .
       Jméno parametr1,parametr2,...
       .
       .
```

Parametry jsou numerické výrazy, které mohou být použity v těle makroinstrukce. Parametrů může být maximálně 16. Při použití v těle makroinstrukce jsou parametry značeny znakem '=' a hexadecimálním číslem od 0 až do F. Nejsou rozlišována velká a malá písmena.

Zkusíme tedy nadefinovat makroinstrukci, která obsahuje instrukce pro přenos paměťového bloku. Parametry budou 'od-

kud', 'kam' a 'kolik'. Makroinstrukci nazvemne třeba Move.

```
; definice makroinstrukce Move

Move    MAC
        ld    hl,=0       ; odkud
        ld    de,=1       ; kam
        ld    bc,=2       ; kolik
        ldir              ; presun
        ENDM
```

Budeme-li chtít Move použít, např. pro přenos obsahu paměti od adresy 0 do videoram (i do atributů je celková délka 6912), budou parametry 0,16384,6912.

```
; použití makroinstrukce v programu

        .
        .
        Move 0,16384,6912 ; přenos
        .
        .
```

Velice často se používají makroinstrukce k doplnění instrukčního soubouru o chybějící instrukce. Například je potřeba přenést obsah registru HL do DE nebo naopak. K tomu si můžeme nadefinovat makroinstrukci. Pro směr z HL do DE ji nazveme třeba LdDeHl.

```
LdDeHl  MAC
        ld    e,l
        ld    d,h
        ENDM
```

Použití takto definované makroinstrukce je mnohem přehlednější než dvě instrukce v jejím těle. Pokud zkrátíte její jméno, dojde ještě ke značnému ušetření paměti při psaní zdrojového textu.

Při překladu těla makroinstrukce musí být celé k dispozici. Z tohoto důvodu se definice makroinstrukcí ukládají do tzv. makrobufferu. Jeho velikost je po nahrání Masm nulová a lze změnit pomocí příkazu Mem v hlavním menu. Jestliže dojde při překladu k naplnění makrobufferu, bude hlášena chyba.
Při práci s makroinstrukcemi musí být splněny následující podmínky:

- každá makroinstrukce musí mít jméno, každá jiné
- v těle makroinstrukce nesmí být definována jiná makroinstrukce
- v těle makroinstrukce nesmí být volána žádná makroinstrukce
- použití makroinstrukce musí být až po její definici

Při překladu těla makroinstrukce je zobrazováno pouze její jméno a parametry, nikoliv tělo. Budete-li chtít zobrazit celý rozvoj makroinstrukce, použijte povel assembleru '*M+' (popsán v kap. 4.6).

4.4 Předdefinované makroinstrukce

Přestože se většina makroinstrukcí může velice lehce nadefinovat přímo v programu, nabízí Masm čtyři předdefinované makroinstrukce, které můžete použít aniž byste museli definovat makrobuffer. Jsou to (první dvě byly zařazeny z důvodu kompatibility s programem Gens, další dvě jsou nové):

Brk (Break)

Tato makroinstrukce generuje instrukci rst #38, tedy skok na obsluhu maskovaného přerušení. Je určena pro počítače, u kterých je možné změnit tuto rutinu. To v případě, že je v této oblasti ROM, nelze.

Rcal (Relative Call)

Makroinstrukce RCal je určena pro generování relativních instrukcí Call. Syntaxe je jako u relativních skoků:

```
        RCAL návěští        ; resp. numerická_hodnota
```

nebo

```
        RCAL podm,návěští   ; podm. jako u relativních skoků
```

V případě RCAL návěští je definována posloupnost:

```
        RST  #28
        DEFB relativní_hodnota_skoku
```

U relativních skoků s podmínkou používá Gens dvě posloupnosti:

```
        RST  #28
        DEFB relativní_hodnota_skoku
```

pro podmínky Z a NZ, ale:

```
        RST  #38
        DEFB relativní_hodnota_skoku
```

pro C a NC.

To byly dvě makroinstrukce, které byly dodány jako součást programu Gens. Jak bylo uvedeno, nemají zase tak velký praktický význam, jsou zde přítomny z důvodů kompatibility. Jejich chování také není právě nejvhodnější.

Err (Error)

Tato makroinstrukce se může použít pro skok na chybovou rutinu, resp. volání 'hook' kódu z interfejsu 1. Používá se v syntaxi:

```
        Err  číslo_chyby
```

Instrukce generuje posloupnost:

```
RST  #08
DEFB číslo_chyby     : nebo číslo hook - 42du
```

Tato makroinstrukce jistě zjednoduší orientaci ve zdrojovém textu.

Main

Čtvtá, poslední předdefinovaná makroinstrukce, je vhodná pro psaní programů, které běží v režimu aktivní stínové ROM Interfejsu 1. Její syntaxe je:

```
MAIN adresa_v_hlavní_ROM
```

a generuje posloupnost:

```
RST  #10
DEFW adresa_v_hlavní_ROM
```

Např.: jsme postaveni před problém otevření kanálu pro tiskárnu, ale při při připojené stínové ROM. Nemůžeme tedy použít program

```
ld   a,3          ; kanál 3 = tiskárna
call #1601        ; otevři kanál
```

ale třeba tento

```
ld   a,3
MAIN #1601        ; připoj hlavní ROM a skoč do ru-
                  ; tiny na #1601
```

Na první pohled je zřejmé, že je verse s makroinstrukcí Main přehlednější.

4.5 Lokální bloky

Lokálním blokem rozumíme část zdrojového textu mezi pseudoinstrukcemi LOC a ENDL. V takto definovaném bloku můžeme používat tzv. lokální návěští, která nemohou být použita z druhé strany takto definovaných hranic.

Lokální návěští jsou od normálních odlišena tím, že začínají znakem ':'. Jinak mají všechny vlastnosti shodné s normálními návěštími. Např.:

```
Start  ld   b,10
       call Incr
       ret
```

```
Incr    LOC
:0      inc  c
        djnz :0
        ret
        ENDL
```

V tomto příkladu je vidět, že byla vytvořena rutina pro inkrementci registru C. Její tělo bylo označeno jako lokální blok. Aby byl k této rutině přístup z vně bloku, muselo být před pseudoinstrukci LOC umístěno globální návěští Incr. Po překladu bude mít Incr stejnou hodnotu jako :0, protože je pseudoinstrukce LOC nevýkonná, ale lokální návěští je z vně bloku nedostupné.

Díky lokálním návěštím umožňuje Masm vytvářet knihovní rutiny, které mohou být později použity. Výhoda spočívá v tom, že uživatel nemusí hlídat jména návěští, případná shoda je řešena otázkou 'jsi ve stejném lokálním bloku nebo ne?'. Odpověď je jednoznačná.

Protože by jeden lokální blok asi charakter assembleru nezměnil, můžete jich použít více. Konkrétně 256 lokálních bloků. Toto číslo bylo stanoveno po mnoha testech a ukázalo se jako postačující. Při tomto počtu lokálních bloků není nutné modifikovat ani zvětšovat položky v tabulce symbolů, práce s lokálními bloky nevyžaduje tedy žádnou zvlášť přidělenou paměť.

Rovněž může dojít k situaci, že si uživatel v lokálním bloku vytvoří další lokální blok. Nebo si jej do lokálního bloku vloží pomocí inkluse, aniž by se zabýval, jak byl ten či onen knihovní soubor napsán. Ani tyto starosti vás nebudou trápit při práci v Masm. On totiž dovoluje tzv. vnořené lokální bloky. Např.:

```
Výstup LOC                  ; začátek výstupní rutiny
       cp   #80             ; změna proti knihovní versi
       jr   ge,:end
       LOC                  ; sem byl vložen knih. soubor
       .
       .
:end   ENDL                 ; konec knih. souboru
:end   ENDL                 ; konec výstupní rutiny
```

V obou blocích je definováno lokální návěští :end. Při překladu ale chyba hlášena nebude, protože je každé definováno v jiném lokálním bloku. Snad už není nutné připomínat, že z jednoho lokálního bloku nelze volat lokální návěští z bloku jiného. Toto tvrzení platí směrem nahoru i dolů.

Hlavní program je chápán jako lokální blok úrovně 0 a v něm můžete rovněž používat lokální návěští. Na začátku a konci textu jsou pomyslné pseudoinstrukce DEFL a ENDL.

Je nutné ještě připomenout, že počet úrovní vnoření je také omezen. Může jich být 'pouze' 20. I to se při testech ukázalo jako postačující.

Velký význam mají lokální bloky při práci s

makroinstrukcemi. Představme si definici makroinstrukce:

```
Copy    mac
lab     ld   a,(hl)      ; vezmi bajt z (hl)
        ld   (de),a      ; polož jej na (de)
        inc  hl          ; zvětši ukazatele
        inc  de
        djnz lab         ; čítání cyklu
        endm
```

Takto je makroinstrukce nadefinována a můžeme ji i použít. Co se však stane, použijeme-li ji vícekrát. Návěští lab bude definováno podruhé a bude hlášena chyba. Nekolidující řešení s použitím lokálních bloků je zde:

```
Copy    mac
        LOC
:lab    ld   a,(hl)      ; vezmi bajt z (hl)
        ld   (de),a      ; polož jej na (de)
        inc  hl          ; zvětši ukazatele
        inc  de
        djnz :lab        ; čítání cyklu
        ENDL
        endm
```

Tuto makroinstrukci můžete použít i vícekrát aniž by byla hlášena chyba.

4.6 Povely assembleru

Povely assembleru, stejně jako pseudoinstrukce neovlivňují mikroprocesor Z-80 při činnosti, neboť nejsou překládány do strojového kódu. Až na příkaz *F jsou určeny pro definování a změny formátu výstupu na obrazovku a tiskárnu.

Povelem assembleru rozumíme řádku zdrojového textu, která začíná znakem '*'. Následující písmeno, které určuje povel, může být jak velké, tak malé. Masm to nerozlišuje. K dispozici jsou následující povely:

```
*C          řízení tisku přeloženého kódu
*D          určování číselné soustavy pro tisk adres
*E          formátování
*F jméno    inkluse
*Hřetězec   definice hlavičky
*M          řízení tozkladu makroinstrukcí
*S          zastavení výpisu
*L          řízení výpisu
*Wřetězec   zastavení výpisu
```

Povel *C

Tento povel určuje, zda bude při překladu tisknut také přeložený kód (hexadecimálně). Povel *C+ zapíná tisk, *C- jej vypíná. Přednastavena je volba *C+.

Povel *D

Povel *D určuje, v jaké soustavě bude zobrazován čítač adres v překladovém protokolu. *D+ zapíná tisk adres dekadicky, *D- hexadecimálně. Přednastavena je volba *D-.

Povel *E

Povel *E způsobí tisk tří prázdných řádek na obrazovce nebo tiskárně. To je užitečné k oddělování modulů.

Povel *F

Zcela jistě nejvýkonnější povel assembleru. Je obecně známo, že poměr délky zdrojového textu a přeloženého kódu je dost velký. Většinou se pro toto vyjádření používá hodnota 8. Je proto jasné, že psát celé zdrojové programy do paměti není možné. Program, u kterého se očekává výsledná délka řekněme 16 kB, by měl zdrojový text (stále používáme konstantu 8) délky 128 kB. Jedinou možností je připravit zdrojové programy na vnější médium a pak je při překladu kousek po kousku vkládat a překládat. Jsou zde dva postupy jak tento překlad provádět. Hlavním faktorem pro volbu je druh vašeho média.

Pokud používáte pouze magnetofon, bude postup následující:

- zdrojový text, který budete chtít při překladu přihrávat musíte nahrát z programu Masm pomocí příkazu Incl:
- do paměti si nahrajete zdrojový text, do kterého se bude provádět inkluse;
- do místa, kam se má provádět inkluse, vložíte příkaz *F ve tvaru

```
*F jméno
```

  před jménem musí být vložena mezera; je-li jméno prázdné, bude nahráván a překládán první nalezený soubor na pásce;
- začnete překládat a na požádání assembleru zapnete magnetofon a pustíte mu soubor určený pro inklusi;
- Masm je dvouprůchodový assembler, proto vás o nahrávku požádá dvakrát.

Používáte-li microdrive, pak je postup tento:

- zdrojový text, který budete chtít při překadu přihrávat nahrajete pomocí příkazu Put na microdrive;
- do paměti si nahrajete zdrojový text, do kterého se bude provádět inkluse;
- do místa, kam se má provádět inkluse, vložíte příkaz *F ve tvaru:

```
*F jméno
```

  před jménem musí být vložena mezera a jméno musí začínat předponou z čísla microdrivu a znaku ':';

- začnete překládat; assembler si sám přihraje zdrojový text z microdrivu;

Jako vyrovnávací paměť pro překlad slouží tzv. Include buffer, jehož velikost můžete měnit pomocí příkazu Mem. Po nahrání Masm do paměti je velikost nastavena na hodnotu 256 bajtů. To je velikost vhodná pro překlad z microdrivu, nikoliv z pásky. Pokud budete pro překlad používat pouze magnetofon, bude výhodné si buffer zvětšit (např. na 1kB).

Budete-li velikost 'Include' bufferu měnit, pamatujte na jednu důležitou zásadu: velikost bufferu při nahrávání příkazem Incl musí totožná s velikostí při překladu. Je proto vhodné zvolit si pro inklusi z pásky jednu velikost bufferu a tu používat. Stojí za to zabývat se chvíli touto problematikou a zvolit optimální velikost.

Povel *H

Povel *Hřetězec způsobí, že se řetězec bere jako záhlaví a bude se tisknout vždy po provedení příkazu *E. Při použití *H se automaticky generuje i povel *E.

Povel *M

Povel *M určuje,zda se v překladovém protokolu budou tisknout rovněž rozvoje makroinstrukcí. Volba *M+ způsobí tisk rozvoje makroinstrukcí, *M- zobrazuje pouze jejich volání. Přednastavena je volba *M-.

Povel *S

Tento povel zastaví překlad a čeká na stisknutí tlačítka. To je vhodné pro čtení řádek ve střední části programu, které jsou jinak při překladu špatně přístupné.

Povel *L

Povel *L ovládá tisk překladového protokolu. Ten bude tisknut při volbě *L+. V případě *L- zobrazován nebude. Přednastavena je volba *L+.

Povel *W

Povel *Wřetězec způsobí zastavení překladu a zobrazení řetězce. Poté čeká program na stisknutí tlačítka. Řetězec je vždy tisknut na obrazovku. Výstup na tiskárnu zůstává nezměněn.
Tento příkaz lze využít například při překladu z více microdrivových kazet:

```
        .
        .
*W Vloz cartridge 1
*F 1:jmeno_z_1._cartridge
        .
        .
*W Vloz cartridge 2
*F 1:jmeno_z_2._cartridge
        .
        .
```

Tato posloupnost povelů způsobí, že bude provedena inklu-
se ze dvou souborů, z nichž každý musel být nahrán na jinou
cartridge. Před inklusí 1. souboru budete požádáni o jeho
vložení. Po inklusi bude překládán program z hlavní paměti,
pak budete požádáni o vložení 2.souboru. To se bude opakovat
i při druhém průběhu.

4.7 Překlad

Překlad zdrojového textu do strojového kódu provádí příkaz
Ass (Assembly). Předpokladem pro správnou funkci je bezchyb-
ně napsaný zdrojový text v paměti počítače, resp. na vněj-
ším médiu a dostatek volné paměti pro jeho činnost.
Po volbě příkazu Ass budete požádáni o vložení velikosti
tabulky symbolů (Table size:), jejíž délka je statická. Masm
vám nabídne poslední volbu. Pokud potvrdíte prázdný řetěz,
určí si velikost tabulky Masm sám. Ta je úměrná délce zdro-
jového textu. Je-li celý zdrojový text v paměti, většinou
takto určená velikost postačí. V případě, že je část zdrojo-
vého textu na vnějším médiu, je slušné assembleru pomoci a
délku bufferu mu vložit. Dojde-li v průběhu překladu k napl-
nění tabulky symbolů, bude hlášena chyba 'No Table space'.
Když jste vložili velikost tabulky symbolů jste dotázáni
na volby činností assembleru při překladu (Options :). Vklá-
dá se číslo 0..127, prázdný řetěz se chápe jako 0.

K dispozici jsou následující volby, jejich součet vyjadřu-
je vaše požadavky:

1 Při této volbě bude na konci překladu vygenerován seznam
všech použitých návěští. Návěští jsou abecedně srovnána
a doplněna hodnotami. Počet sloupců, ve kterých se budou
návěští tisknout, se volí na adrese 25010.

2 Při této volbě probíhá překlad naprosto normálně, vyge-
nerovaný kód není ale ukládán do paměti. V překladovém
protokolu je vygenerovaný kód zobrazen normálně.

4 Tato volba způsobí, že se nebude provádět výpis překla-
dového protokolu.

8 Tato volba přepne výstupní směr z obrazovky na tiskárnu,
kde bude zobrazen celý protokol.

16 Velice důležitá volba. Strojový kód se bude překládat naprosto normálně, ukládat se ale bude od konce tabulky symbolů výše. Po přehrání na své původní místo bude pochopitelně spustitelný. Tímto způsobem můžeme psát programy, které poběží např. v oblasti, kde leží nyní Masm.

32 Tato volba vypne kontrolu umístění strojového kódu do paměti. Kompilace se tím jistě zrychlí, číhá tu ale nebezpečí, že si poškodíte omylem nějakou důležitou část paměti a dojde ke zhroucení. Buďte velice opatrní při zapnutí této volby.

64 Velice výkonná volba, která jako první nebyla obsažena v programu Gens. S její pomocí můžete generovat tzv. EQU--soubor, který nese informaci o použitých návěštích. Podrobný popis je v následující kapitole. Při této volbě budete dotázáni na další parametry, jejichž význam je vysvětlen tamtéž.

Budete-li chtít například provést překlad, aniž by Masm kontroloval umístění strojového kódu a tiskl překladový protokol na tiskárnu, bude volba 32 + 8 = 40.

Informace o tom, jak probíhá překlad, naleznete v kapitole 6.6.

Překladový protokol má následující formát:

```
8000 3E0A     ....2 Start  Ld   a,10          ; komentář
^    ^        ^     ^      ^    ^             ^
1    6        15    21     28   33            46
```

Jednotlivé pozice zobrazují:

1 Hodnota čítače adres (zde zobrazena hexadecimálně, povelem *D+ lze přepnout na zobrazení dekadické). V případě pseudoinstrukcí EQU a DEFL je na této pozici hodnota výrazu za instrukcí, tedy nová hodnota návěští.
6 Na této pozici jsou v hexadecimálním tvaru umístěny kódy, které jsou ukládány do paměti. Jsou zobrazeny maximálně 4 bajty (to platí i v případě pseudoinstrukcí definujících obsah paměťových buněk).
15 Od této pozice je vyhrazeno 5 znaků pro zobrazení čísla řádku. Řádky jsou číslovány od 1 s krokem také 1.
21 Od této pozice je zobrazováno návěští, které je na tomto řádku definováno.
28 Na této pozici začíná jméno výkonné instrukce, pseudoinstrukce nebo makroinstrukce (u makroinstrukce je povoleno jméno delší než 4 znaky, při jeho použití je zbytek řádky posunut).
33 Od této pozice mohou být uvedeny parametry všech tří druhů instrukcí.
44 Za normálních okolností je tato pozice určena pro komentář. Pokud není předcházející text tak dlouhý, aby zasáhl až sem.

Vznikne-li během překladu chyba, bude překlad zastaven. Pokud se jedná o opravitelnou chybu, skočí Masm do editoru, kursor umístí na řádku s chybou a bude oznámeno číslo chyby. Tím se oprava zdrojového programu značně urychlí. Kódy chyb

jsou uveřejněny na závěr této kapitoly. Při ostatních (fatálních) chybách zobrazí chybové hlášení, čeká na stisk libovolného tlačítka a pak přechází do hlavního menu.

Na konci překladu zobrazí Masm, jak velkou část tabulky symbolů použil. Pokud byl definován vstupní bod pro spuštění příkazem Run a překlad proběhl bez chyb, bude zobrazeno hlášení 'Executes vstupní_adresa'. Na tuto vstupní adresu skáče příkaz Run.

Jsou-li v programu použity nedefinované symboly, je jejich seznam zobrazen (v pořadí použití) na konci překladu ve formátu 'Undefined Chybějící_návěští'.

4.8 Křížové reference

Při psaní extrémně dlouhých programů se můžete dostat do situace, že vám nepomůže ani inkluse. Velká tabulka symbolů a výsledný kód se už do paměti nevejdou. Přijde tedy okamžik, ve kterém budete muset zdrojový text rozdělit a překládat po částech. Přeložené části budou potom sestaveny do cílového programu. Při takovém rozdělení vzniká velice nepříjemná situace. Pokud překládáme část programu, kde jsou použita návěští z části jiné bude hlášena chyba, že tato návěští nejsou definována. Masm má prostředek, který vás v tomto okamžiku zachrání. Tímto postředkem jsou křížové reference.

Křížovou referencí se rozumí 'poskytnutí assembleru informací o návěštích v ostatních blocích, které nejsou právě překládány'.

Prvním úkolem je, získat ze všech bloků, které nebudou při překladu přítomny v paměti. Musíme vytvořit soubory, které informují o hodnotách návěští v těchto blocích. Můžeme je označovat např. EQU soubory. S jejich pomocí budeme nakonec překládat zdrojový text, který se odvolává na symboly v těchto blocích.

EQU soubor generuje Masm na požádání (volba 64 vpřekladu). Pokud si ji zvolíte máte možnost vkládat požadavky pro generaci EQU souborů.

Jsou k dispozici následující volby, jejich součet vyjadřuje vaše požadavky:

1 Do výsledného souboru budou zařazena vybraná návěští. Tato návěští jsou označena znakem '_' na svém začátku. Tímto způsobem si můžete označit pouze vstupní body do hlavních rutin a jejich návěští použít pro křížovou referenci.

2 Do výsledného souboru zařadí tato volba definovaná návěští. To jsou návěští, u kterých je na konci prvního průchodu známa jejich hodnota.

4 Do výsledného souboru zařadí tato volba nedefinovaná návěští. To jsou návěští, u kterých není na konci prvního průchodu známa jejich hodnota.

8 Tato volba způsobí, že bude vygenerovaný soubor uložen do textu místo do paměťového bufferu, který je za tímto účelem v paměti vytvořen. O tomto přeložení se můžete přesvědčit ihned po překladu tím, že přejdete do editoru a skočíte na konec textu.

16 Pro vás, kteří máte k dispozici microdrive, nabízí Masm další možnou volbu. Tou je nahrání souboru na microdrive ve formátu, který se používá pro záznam zdrojových textů. Při této volbě budete dotázáni na jméno souboru, pod kterým bude zaznamenán na microdrivu. Jméno musí být napsáno podle pravidel, která byla uvedena např. v Put.

32 A aby byla nabídka kompletní, nabízí Masm (možná zbytečnou) ještě jednu volbu a tou je zařazení také lokálních návěští do souboru. Ty nejsou mimo lokálních blok prakticky použitelné, ale může nastat situace, kdy se to hodí.

Budete-li chtít například vytvořit na microdrivu soubor se všemi nedefinovanými návěštími, zvolíte volbu 16 + 4 = 20. Nebo budete potřebovat všechna lokální návěští, která byla definována: 32 + 2 = 34.

Pokud neurčíte volbu 8 ani 16, bude soubor uložen do paměťového bufferu, odkud může být nahrán na pásku nebo microdrive příkazem Xrf. Jeho popis je uveden v kapitole 3.2.

Proběhne-li generace EQU souboru bez chyby, bude zobrazeno hlášení 'Xref file finished'.

A nyní je to už jednoduché. Budete-li překládat část zdrojového textu, ve kterém jsou použita návěští z jiné části (ze které máme informace o definicích návěští) stačí si pouze vygenerovaný soubor přihrát do překládaného textu nebo použít povel assembleru *F a tím provést inklusi souboru. Assembler tak bude mít přístupné informace o všech návěštích a překlad projde bez chybových hlášení.

Zkusíme si krátký příklad. Máme následující program:

```
        org   #8000
Start   ld    a,10      ; 1. část
        call  Print
        ld    a,"a"
        call  Test
        ret

Print   rst   #10       ; 2. část
        ret
Test    cp    2
        ret
```

Tento program nám poslouží pro demonstraci. Budeme muset z nějakého důvodu program rozdělit, takže v případě překladu můžeme v paměti pouze první skupinu instrukcí. Vytvoříme si tedy soubor, který ponese informace o návěštích druhé části.

Do paměti nahrajeme druhou část, před začátek programu vložíme pseudoinstrukci Org s požadovanou adresou a přeložíme s volbou 18 (definovaná návěští na microdrive). Pro volbu Org=#800b vznikne soubor (nazveme jej 'jméno') obsahující informace:

```
Print   =     #800b
Test    =     #800d
```

Nyní si do paměti nahrajeme první část programu, a do něj vložíme povel *F. Např. takto:

```
        org  #8000
Start   ld   a,10
        call Print
        call Test
        ret

*F 1:jmeno
```

Tento zdrojový program už lehce přeložíte bez vzniku chyb.

Další využití křížových referencí může být v případě, kdy se váš program odvolává na strojový kód uložený v paměti. Je mnohem přehlednější, když voláte jednotlivé rutiny jménem a ne adresou. Pokud by došlo k přeadresování vstupních bodů tohoto strojového kódu, museli byste také všechny adresy změnit. To samé platí v případě, že jsou ve zdrojovém programu tyto návěští nadefinovány pomocí pseudoinstrukcí EQU.
Mnohem výhodnější je toto řešení. Před generací strojového kódu, na který se budeme odvolávat, označíme všechny vstupní body vybranými návěštími a provedeme překlad s generováním souboru všech vybraných návěští (třeba na microdrive). Ten si potom můžeme přihrát při překladu povelem *F. Je jasné, že pokud budeme při opravě strojového kódu provádět překlad stále tímto způsobem, nemusíme v našem zdrojovém textu provádět žádné změny. Tyto změny jsou zachyceny v EQU souboru, který je předává našemu zdrojovému programu.

4.9 Seznam chyb

Jak bylo uvedeno v kapitole 4.7, dělíme v programu Masm chyby na opravitelné a neopravitelné (fatální). Mezi chyby neopravitelné patří všechny chyby typu : přeplnění některého z bufferů nebo chyby na vstupních a výstupních zařízeních. Ostatní jsou chyby opravitelné.

Opravitelné chyby

1 Chyba v kontextu v tomto řádku.
2 Neznámá mnemonika.
3 Špatně uspořádaný příkaz.
4 Vícenásobně definovaný symbol.
5 Řádek obsahuje nedovolený znak.
6 Některý z operandů je nedovolený.
7 Použitý symbol je reservované slovo.
8 Neshoda registrů.
9 Mnoho registrů v příkazu.
10 Výraz, který bude použit jako 8-bitový,je větší než 255.
11 Instrukce 'JP (IX+n)' a 'JP (IY+n)' nejsou dovoleny.
12 Chyba v uspořádání assemblerovského povelu.
13 Nedovolená reference. EQU nebo DEFL pro neznámý výraz.
14 Dělení nulou.
15 Přeplnění při násobení.
16 Definice makroinstrukce v těle jiné makroinstrukce.
17 Definice makroinstrukce nez jména.

18 Volání makroinstrukce v těle jiné makroinstrukce.
19 Špatně definovaný ORG.
20 Mnoho úrovní lokálních bloků.
21 Mnoho lokálních bloků.
22 Použití pseudoinstrukce ENDL bez definování LOC.
23 Lokální makroinstrukce nejsou povoleny.
24 Vnořené inkluse nejsou povoleny.

Neopravitelné chyby

No Table space

Toto chybové hlášení znamená, že byla tabulka symbolů naplněna. Prostor, který jsme jí vyčlenili už nestačí. Je proto nutné provést nový překlad a při něm tabulku symbolů zvětšit.

No space for Xref

Není prostor pro generování EQU souborů. EQU soubor je generován v místě bývalé tabulky symbolů. Délka jednotlivých položek je ale o něco delší než položka v tabulce symbolů. Může se proto stát, že velikost tabulky symbolů postačí pro návěští, při generování EQU souboru již paměť nezbývá. Pokud provádíte generaci s nastavenou volbou 16, pak k této chybě dojít nemůže.

No macro space

Stejně jako tabulka symbolů, je i makrobuffer pevné délky. Dojde-li při překladu k jeho naplnění, je nutné použít příkaz Mem z hlavního menu a velikost změnit.

Bad Memory

Tato chyba bude hlášena editorem, dostane-li se zdrojový text nebezpečně blízko ke konci paměti. V tomto okamžiku si zdrojový text nejpve nahrajte, pak můžete přemýšlet jak tuto situaci řešit.

User Break

Při překladu došlo k přerušení. K tomuto přerušení může dojít rovněž v několika jiných příkazech.

Tape error

Podobná chyba jako 'Mdrive error'. Při komunikaci s magnetofonem došlo k chybě. Je nutné zopakovat překlad.

Bad Name

Bylo použito špatné jméno. Při nahrávání z microdrive musí být jméno specifikováno. Nelze použít prázdné jméno jako u magnetofonu.

5. Příklad

Součástí Masm je také krátký příklad psaní programu ve strojovém kódu. Jedná se o program 'Had', který jste si přeložili již v úvodní části manuálu. Tento program byl vytvořen v 'Oxford Computer Publishing software' a byl zdarma ! dodáván jako součást první verse OCP-assembleru. I k programu Masm je dodáván zdarma, jen tak na ukázku.

6. Trochu víc o Masm

Tato kapitola je určena pro vás, kteří potřebujete od Masm o něco víc, než je schopen ve standartní konfiguraci nabídnout, a musíte v jeho těle provést určité změny. Snad vám tato kapitolka trochu pomůže.

6.1 Formát textu

Zdrojový text je uložen v paměti následovně :

```
dw    ln              ; číslo řádku, 2 bajty
dm    "Label"         ; navěští, normálně do šesti bajtů
db    tab             ; tabelátor, chr 9
dm    "Ld"            ; instrukce
db    tab             ; tabelátor, chr 9
dm    "a,10"          ; parametry
db    tab             ; tabelátor, chr 9
dm    "; text"        ; komentář
db    eoln            ; konec řádky, chr 13
```

Toto je formát plné řádky. Všechny její části nejsou povinné. Možné formáty jsou popsány v kapitole 2.2. Důležité je zjištění, že jsou jednotlivé části odděleny tabelátory.
Pokud řádka obsahuje např. pouze instrukci ret, bude formát následující:

```
dw    ln              ; číslo řádku, 2 bajty
db    tab             ; tabelátor, chr 9
dm    "Ret"           ; instrukce
db    eoln            ; konec řádky, chr 13
```

Bude-li v řádce pouze komentář, bude formát:

```
dw    ln              ; číslo řádku, 2 bajty
db    tab             ; tabelátor, chr 9
db    tab             ; tabelátor, chr 9
db    tab             ; tabelátor, chr 9
dm    "; text"        ; komentář
db    eoln            ; konec řádky, chr 13
```

Pozice každé části řádky je jednoznačně určena tabelátory.

6.2 Tisk

Při výstupu na tiskárnu používá Masm standartní metodu:

1. Otevře kanál č. 3.
2. Výstup provádí přes #10, tedy instrukcí Rst #10.

Pokud budete interfejs obsluhovat vlastním programem, musíte změnit kanálovou informaci o umístění výstupní rutiny. Nejjednodušší metoda, která rovněž respektuje přítomnost interfejsu 1, je tato:

```
Init    ld   hl,(chans)   ; #5c4f
        ld   de,#000f     ; ofset pro kanál tiskárny
        add  hl,de
        ld   de,NewOut    ; start nové výstupní rutiny
        ld   (hl),e       ; ulož ho do kanál. informací
        inc  hl
        ld   (hl),d
        ret

NewOut  ; zde leží nová výstupní rutina
        ; znak, který se má vyslat přichází v Acc
        .
        .
        ret
```

Masm nevyužívá žádnou z mimořádných schopností tiskáren, které jsou v současnosti vyráběny. Aby byly výpisy v pořádku požaduje pouze tisk znaku a přechod na další řádku (cr,#0d). Veškeré tabelace jsou tvořeny pomocí mezer a kódy výstupních znaků jsou tedy 13 a 32..126.

6.3 Beta disk

Hned na začátku této kapitoly je nutno prohlásit, že neuvádí metodu, jak spustit Masm s interfejsem Beta. Spíše se snaží vyložit, proč s Masm Beta nefunguje.

Interfejs Beta vyráběla firma Technology Research. Poslední známá verse byla Beta 128 v. 5.03. Poté firma zkrachovala a je otázkou, zda se dnes může někde koupit. Beta je připojitelná k mechanice pružného disku velikosti 3", 3+1/2" a 5+1/4". Její systém je naprosto něčím odlišným od systému umístěného v interfejsu 1. Na úrovni strojového kódu se ke vnitřním rutinám přistupuje podobně jako ke službám OS operačního systému CP/M (jeden vstupní bod, číslo služby ve vybraném registru).
Beta nepoužívá k aktivaci adresu #0008 jako microdrive, její aktivační body leží v nevyužitém prostoru ROM.

Aby byla práce stylová, rozhodl jsem se využít pro Masm sekvenční soubory, které jsou v manuálu inzerovány. Tedy něco podobného jako má microdrive. Na vlastní zodpovědnost mohu prohlásit, že v rutinách pro obsluhu kanálu je tolik chyb, že jsou nepoužitelné. Musíme pouze držet palce, aby do nich nikdo nezabloudil. Rovněž otevírání souboru, jak nám radí manuál, nelze použít. Jen náhodou se počítač nezhroutí.

Druhou možností je použít pro záznam zdrojových textů soubory typu Code. Jejich délka je však omezena pouze na 64kB, nelze číst sekvenční soubory vzniklé při disassemblování dlouhého kódu. Nelze je ani překládat.

Buď se tedy rozhodnete pro život 'rozlamovače dlouhých souborů' nebo zbývá klasické řešení. Napsat celou obsluhu nahrávání a čtení pomocí Beta disku znovu. Takový kód, který je určen pro Masm, má délku přibližně 3 kB. Tak dlouhý kód nebylo možné vložit do Masm.

Pokud se chcete věnovat konkurenci firmy Technology Research , kterou je Miles Gordon Technology (dnes konkurují neexistující firmě), dostanete se určitě k jejich výrobku zvanému Disciple. Ten také není bez chyb, ale výrobce se alespoň pokusil o stejný formát komunikace se ZX-Spectrem jako je u Interfejsu 1.

Situace u interfejsu Beta je vám tedy známa a firma ji asi už nezmění, u Disciplu je naděje, že jeho autoři snad jednou své rutiny doladí.

6.4 Put a Get

Při komunikaci s magnetofonem používá Masm dvě standartní rutiny umístěné v ROM. Pro nahrání na magnetofon #04c2 a z magnetofonu #0556. Všechny soubory jsou typu Code.

U microdrivu je tomu trochu jinak. Zdrojové texty včetně souborů pro křížovou referenci jsou vytvářeny jako sekvenční soubory. Přeložený strojový program je typu Code. Pro komunikaci jsou používány výhradně Hook-kódy. Algoritmy záznamu a čtení jsou následující:

```
Put:  otevři soubor s tímto jménem:                (hook 22)
      když existuje, zeptej se na smazání:
         jestliže je odpověď ano soubor smaž       (hook 24)
         jinak z příkazu vyskoč:
      nahraj nový soubor na microdrive:            (hook 26)
      uzavři tento soubor.                         (hook 23)

Get:  otevři soubor s tímto jménem:                (hook 22)
      když neexistuje, vyskoč na chybu:
      nahraj nový soubor do paměti:                (hook 25)
      uzavři tento soubor.                         (hook 23)
```

Jsou tedy používány soubory typu 'temporary', které při vzniku jakékoliv chyby zanikají, nenastává tedy problém 'nezavřených kanálů', který by byl v tomto případě dost těžko řešitelný.

6.5 Paměťová mapa

Tato kapitola vám přiblíží, v kterých paměťových prostorech se můžete v počítači pohybovat.

Masm je nahráván od adresy 25000. Jeho důležité adresy na konci jsou odvozovány od začátku zdrojového textu.

V následující tabulce bude :

```
TxtSt     - začátek zdrojového textu,
TxtEnd    - konec zdrojového textu,
InclBuff  - délka bufferu pro inklusi,
MacBuff   - délka makrobufferu,
TableSize - velikost tabulky symbolů).
```

Informace o začátku textu je vám přístupná příkazem Text.

Adresa	Význam
25000	Začátek Masm, zde skok je na skutečný start
25003	Skok na rutinu, která zapíná 64-znakový výstup. Ten budete potom moci použít ve svých programech, které budou pod Masm spuštěny (kanál 'S').
25006	Adresa začátku zdrojového textu.
25008	Adresa konce zdrojového textu + 1.
25010	Počet sloupců pro tisk použitých symbolů.
TxtSt-MacBuffer-InclBuff	Začátek bufferu pro inklusi.
TxtSt-MacBuffer	Začátek makrobufferu.
TxtSt	Začátek textu.
TxtEnd	Konec textu + 1.
TxtEnd+2	Začátek tabulky symbolů.
TxtEnd++2+TableSize	Konec tabulky symbolů+1; místo pro uložení strojového kódu při volbě 16, nebo když nebyl určen ORG.

6.6 Překlad Masm

Tato kapitola stručně popisuje, jakým způsobem se provádí v programu Masm překlad.

Masm je dvouprůchodový assembler. Pro překlad je tato verse nejpoužívanější. Jednoprůchodové překladače mají buď omezené možnosti, nebo vyžadují mnohem víc paměti pro záznam informací. Překladače tříprůchodové se používají pro speciální účely, poskytují ale vyšší komfort při psaní zdrojových programů.

Při prvním průchodu musí assembler sestavit tabulku symbolů. Každé návěští, na které narazí je zapsáno do tabulky symbolů a je doplněna jeho hodnota. Při zápisu je kontrolováno, zda se jeho jméno neshoduje s některým s reservovaných slov. Definice makroinstrukcí jsou zapisovány do makrobufferu. Při relativních skocích vzad je kontrolováno přetečení. Během celého prvního průchodu se generují instrukce, nejsou však ukládány do paměti. Je pouze potřeba znát jejich délku pro výpočet hodnoty čítače adres.

V druhém průchodu by měli být všechny potřebné informace k dispozici. Znovu se prochází celý zdrojový text a je generován strojový kód. U všech relativních skoků je kontrolováno přetečení. Rovněž je prováděna kontrola uložení strojového kódu (pokud není vypnuta).

Při jakékoliv chybě bude překlad zastaven. Jedná-li se o opravitelnou chybu, bude řízení předáno editoru. V opačném případě se skáče do hlavního menu s chybovým hlášením.

7. Závěr

Program Masm byl koncipován tak, aby poskytl programátorovi maximální komfort s ohledem na malou paměť mikropočítače ZX-Spectrum. Kromě assembleru je jeho součástí celostránkový editor, který jistě přispěje k tomu, že bude tvorba programů příjemnou záležitostí.

Hodně zdaru při tvorbě nových programů.

Příloha A : Instrukce

Nyní následuje seznam seznam výkonných instrukcí mikroprocesoru:

ADC	ADD	AND	BIT	CALL	CCF	CP	CPD	CPDR
CPI	CPIR	CPL	DAA	DEC	DI	DJNZ	EI	EX
EXX	HALT	IM	IN	IND	INDR	INI	INIR	JP
JR	LD	LDD	LDDR	LDI	LDIR	NEG	NOP	OR
OTDR	OTIR	OUT	OUTD	OUTI	POP	PUSH	RES	RET
RETI	RETN	RL	RLA	RLC	RLCA	RLD	RR	RRA
RRC	RRCA	RRD	RST	SBC	SCF	SET	SLA	SRA
SRL	SUB	XOR						

Nyní následuje seznam pseudoinstrukcí Masm:

DB	DEFB	DEFL	DEFM	DEFS	DEFW	DL	DM	DS
DW	ELSE	END	ENDL	ENDM	ENT	EQU	IF	LOC
MAC	ORG	=						

Nyní následuje seznam předdefinovaných makroinstrukcí:

BRK	ERR	MAIN	RCAL

Příloha B : Povely assembleru

Následuje seznam všech možných povelů assembleru:

*C	řízení tisku přeloženého kódu
*D	určování číselné soustavy pro tisk adres
*E	formátování
*F jméno	inkluse
*Hřetězec	definice hlavičky
*M	řízení tozkladu makroinstrukcí
*S	zastavení výpisu
*L	řízení výpisu
*Wřetězec	zastavení výpisu

Příloha C : Volby assembleru

Následuje seznam možných voleb pro překlad:

1 Na konci překladu generován seznam použitých návěští.
2 Vygenerovaný kód není ale ukládán do paměti.
4 Nebude se provádět výpis překladového protokolu.
8 Výpis na tiskárnu.
16 Strojový kód se bude ukládat za tabulku symbolů.
32 Vypnutí umístění strojového kódu do paměti.
64 Generace EQU souboru.

Nyní následuje seznam voleb pro generaci EQU souborů (na tyto volby jste dotázáni pouze po vložení volby 64 pro generaci EQU souboru):

1 Do souboru budou zařazena také vybraná návěští.
2 Do souboru zařadí tato volba také definovaná návěští.
4 Do souboru zařadí tato volba také nedefinovaná návěští.
8 Vygenerovaný soubor bude uložen do textu.
16 Vygenerovaný soubor bude uložen na microdrive.
32 Do souboru zařadí tato volba také lokální návěští.

Příloha D : Seznam chyb assembleru

Nyní následuje seznam opravitelných chyb:

1 Chyba v kontextu v tomto řádku.
2 Neznámá mnemonika.
3 Špatné uspořádaný příkaz.
4 Vícenásobně definovaný symbol.
5 Řádek obsahuje nedovolený znak.
6 Některý z operandů je nedovolený.
7 Použitý symbol je reservované slovo.
8 Neshoda registrů.
9 Mnoho registrů v příkazu.
10 Výraz, který bude použit jako 8-bitový, je větši než 255.
11 Instrukce 'JP (IX+n)' a 'JP (IY+n)' nejsou dovoleny.
12 Chyba v uspořádání assemblerovského povelu.
13 Nedovolená reference. EQU nebo DEFL pro neznámý výraz.
14 Dělení nulou.
15 Přeplnění při násobení.
16 Definice makroinstrukce v těle jiné makroinstrukce.
17 Definice makroinstrukce nez jména.
18 Volání makroinstrukce v těle jiné makroinstrukce.
19 Špatně definovaný ORG.
20 Mnoho úrovní lokálních bloků.
21 Mnoho lokálních bloků.
22 Použití pseudoinstrukce ENDL bez definování LOC.
23 Lokální makroinstrukce nejsou povoleny.
24 Vnořené inkluse nejsou povoleny.

Následuje seznam neopravitelných chyb:

No Table space
No space for Xref
No macro space
Bad Memory
User Break
Tape error
Bad Name

Příloha E : Reservovaná slova

Nyní následuje seznam reservovaných slov. Tyto symboly nesmí být použity jako návěští, mohou však tvořit jeho část. Assembler ani zde nerozlišuje velká a malá písmena.

$	A	AF	AF'	B	BC	C	D	DE	E
EQ	GE	H	HL	I	IX	IY	L	LT	M
NC	NE	NZ	P	PE	PO	R	SP	XH	XL
YH	YL	Z							