Ivan Jedlička

# MEMORY RESIDENT SYSTEM

**Programový systém pro vývoj a ladění programů ve strojovém kódu – assembleru mikroprocesoru Z80 (verze pro mikropočítače ZX Spectrum, Delta a Didaktik gama)**

CODE 535041 713044
SKADD6800 26624 144

Ivan Jedlička

# MEMORY RESIDENT SYSTEM

## Programový systém pro vývoj a ladění programů ve strojovém kódu – assembleru mikroprocesoru Z80 (verze pro mikropočítače ZX Spectrum, Delta a Didaktik gama)

program start

Vydala MLADÁ FRONTA
ve spolupráci se ZENITCENTREM,
centrem mládeže, vědy a techniky SSM

# O B S A H :






ISBN 80-204-0055-9

# O ČEM JE TENHLE PROGRAM ?

Na položenou otázku si programoví skauti budou umět odpovědět sami. Berete-li však do rukou kazetu s programem MRS jako takřka naprostí začátečníci assemblerového programování, vstupujete do nepoznaného labyrintu protkaného sítí temných chodeb. Úvodní slova vám do nich chtějí vnést aspoň trochu mihotavého světla (prozářit je nakonec budete muset sami monumentálními výboji svého intelektu).

Mezi spectristy dosud běhá několik zahraničních i porůznu upravovaných editorů a překladačů assembleru a monitorů paměti s rozličnými možnostmi ladění a analýzy tvořených či zkoumaných programů. Assembleroví metuzalémové nostalgicky vzpomenou dua GENS/MONS s prošedivělými skráněmi. Krátkou sérii variací na dané téma završil nablýskaný LASER GENIUS...a tím to pro světem zapomenuté ZX Spectrum skončilo. Žádný z dosavadních programů však neposkytoval "v jednom kuse" tolik možností jako právě MRS. Proto byste se měli v nové výzbroji aspoň rámcově zorientovat, než půjdete na zteč.

Začněte editorem EDI - do něj budete zapisovat assemblerové instrukce svého vznikajícího programu. Pozorně si pročtěte, jaké povinnosti vám EDI předepisuje i jaké možnosti nabízí. Protože MRS žádá zakončit program slovíčkem END, nezapomeňte na ně. Jinak by se s vámi modul ASM odmítl bavit. Právě on překládá zdrojový text do strojového kódu. Kam tento kód umístit, vám řeknou stránky věnované pseudoinstrukci ORG a direktivě *C.

Představte si, že právě přeložený kousek programu je jednou knížkou. Proč? Protože MRS má tu skvělou vlastnost, že si tuhle knížku umí uložit do své knihovny (modulu LIB). V ní je založena ne proto, aby na ni padal prach; naopak - můžete se na ni kdykoli obrátit a využít její obsah. Ovšem to chce vědět, jak s ní komunikovat, když je v ní prakticky už jen strojový kód. Budete-li chtít z budoucích knížek (částí programu) volat třeba kapitolu (adresu) VSTUP, označíte ji kouzelným slůvkem ENT (vstupní bod) v té části programu, kde VSTUP v editoru definujete. Knihovník si po překladu poznamená absolutní adresu symbolizovanou názvem VSTUP, a kdykoli se na ni nějaký jiný kousek programu bude chtít obrátit (instrukcí CALL VSTUP, LD HL,VSTUP apod.), knihovník už bude umět zařídit potřebné. Budete-li z právě psaného kousku assembleru chtít volat nějakou z adres uložených jako vstupní body, musíte na to knihovníka předem upozornit dalším magickým slovíčkem EXT (externí bod). Když je před překladem umístíte do textu a připíšete k němu názvy vstupních bodů v knihovně už

uložených, knihovník - za nezbytné podpory modulu LNK - pak knížky propojí ("slinkuje"). Všem výrazům označeným EXT přidělí (podle předtím uložených hodnot ENT) správné absolutní adresy a tak celý program vnitřně prováže. A když jste se nikde nespletli, budou si všechny knížky spolu povídat podle vašich představ. Z toho rovněž vyplývá, že byste neměli spouštět program obsahující pseudoinstrukce EXT, dokud knihovnu neprovážete modulem LNK - na to pozor!

Do knihovny pochopitelně ukládáme jen takové kousky programu, které nám už bezvadně fungují. Knihovník napevno uloží jen takovou knížku, které dáme nějaký titul (každé jiný). Jinak ji po překladu uloží bez názvu a při každém dalším překladu ji přepíše.

Co však, když chcete něco v nějaké knížce upravit a nikde nemáte záznam jejího zdrojového textu? Je přece popsána jen strojovým kódem! Nevadí. Z editoru si modulem DIS vyvoláte zpětný překlad pro převedení jejího obsahu ve tvaru zdrojového textu přímo do editoru. S výsledným assemblerovým výpisem, který se však objeví bez původních návěští, můžete naložit, jak libo - přidat či ubrat instrukce a upravený text zase založit do knihovny. Když jej založíte pod stejným názvem, zruší se registrace původní knížky (ačkoli její obsah i rozsah zůstane nezměněn) - knihovník vám s ní už nedovolí komunikovat. Vaše upravená knížka se připojí na konec knihovny (a byla-li tam nějaká bez názvu, přepíše ji).

Modulem DIS můžete do editoru posadit zdrojový text jakékoli části paměti - třeba i romky. Při další práci s textem je samozřejmě dobré vědět (či poznat), co jsou data a co instrukce, aby nedošlo k nějakému omylu.

Něco málo můžete upravit i přímou změnou obsahu adres paměti pomocí ladicího modulu DBG - ale nezapomeňte, že tyto změny se samozřejmě neobjeví ve zdrojovém textu (zvláště máte-li jeho měněnou část už někde na pásku). Modul DBG vám mimo jiné umožní prohlížet obsah paměti a modifikovat ji; ale to je jen malá část jeho možností, které máte k dispozici pro ladění svých programů. Jako každý správný debugger ("spray na programové štěnice") má i DBG možnost zpětného překladu, abyste se mohli podívat na assemblerový tvar uloženého strojového kódu. Oproti produktu modulu DIS je tento překlad jen orientační, nikam se neukládá. Prohlížení obsahu paměti a všech registrů procesoru vám pomůže sledovat, co se v nich děje za chodu programu. Často jím budete potřebovat krokovat manuálně, instrukci za instrukcí, ale jde to i automaticky - ve sledovacím režimu a jeho variacích s body přerušení, stanovenými okny paměti a registru PC, hlídáním vámi zakázaných instrukcí atd., atd...

Když jste při ladění odhalili chybičku (jednu ze sta, jak už tomu tak bývá), můžete se z modulu DBG zase vrátit zpět - třeba do editoru - a pokračovat ve tvorbě programu ... i dalších chyb. Proto se snažte zvládnout všechny kombinace programového ladění, které modul DBG nabízí.

MRS dovoluje použít nejvíce 255 návěští (symbolických adres). To je dáno autorovou snahou docílit komprimací a jinými úspornými prostředky maximálního efektu na malé ploše (do volné paměti se vejde přes 4 000 řádek zdrojového textu). Proto při práci s větším programem budete muset trochu víc přemýšlet o jeho struktuře a postupném ukládání jeho autonomních částí do knihovny, abyste s limitovaným počtem návěští vystačili. Po uložení každé části do knihovny a překladu nového zdrojového textu zmizí všechna předchozí návěští (přepíšou se novými). Vlivem komprimace trvají některé manipulace editoru s dlouhým textem podstatně déle než s krátkým. I proto je lepší častěji knihovat (a následně spojovat modulem LNK). Ale nelekejte se, na tak malém počítači (a pohříchu nejen na malém) nebývá nic absolutního. Až se s programem MRS sžijete, zjistíte, že vám u jiných obdobných bude zase chybět to, co MRS umí.

Když spojená část programu nejeví známky slabosti, můžete si knihovnu zapsat na pásek či jiné médium, abyste o ni nějakou fatální chybou nepřišli. Vždy je lepší dát do bezpečí, co zrovna funguje, než o to v nejbližším okamžiku nenávratně přijít. Uschovanou knihovnu si můžete do počítače zase načíst (nová odstraní starou). Na vnější paměť samozřejmě můžete průběžně zapisovat i samotný zdrojový text či jakoukoli část paměti.

O tom všem a přemnohém jiném se dozvíte z dalších stránek manuálu. Při svých prvních kontaktech s programem se nebojte chybovat, i když vám program občas zmizí z obrazovky (stane se i profesionálovi). Nedílnou součástí osvojení programů, jako je MRS, je výuka vlastními chybami. Budete-li se studiu věnovat opravdu pilně, do dvou, tří týdnů už vám to půjde bez velkého listování manuálem. I když toho není zrovna málo, snažte se ovládnout vše, co MRS nabízí. Cokoli vynecháte, nutně sníží efektivitu vašeho programování - a to by byla škoda.

Malá rada - usilujte o to, aby vám vstřebávané detaily jako střípky zapadaly do vynořující se celkové představy o koncepci programu. Když pochopíte, CO všechno MRS umí a JAK to dělá, už to nezapomenete. Už budete vždy vědět, že můžete udělat to či ono, i když si zrovna nevzpomenete, co máte zmáčknout. Tlačítko, které vám vypadne z paměti, si připomenete pohledem do manuálu. Bez vědomí koncepce, celostního pohledu, se s čímkoli zachází hůř, neboť smysl uniká. Přeji dobrý assemblerový start vám i paní MRS (neb Mrs. je "english spoken" dáma).

A nemohu opomenout ani poděkování nakladatelství Mladá fronta. I když to nemá v popisu práce, stává se prvním vydavatelem počítačového programu, který si může kdokoli koupit v naší maloobchodní síti. Byť si mírně zpožděný, přesto historický moment, v jakém se sluší smeknout.

Ladislav Zajíček

# 0. ÚVOD

Systém **MRS** je určen k vývoji programů ve strojovém kódu/assembleru mikroprocesoru Z80.

**Obsahuje následující moduly:**

- Řídicí modul **MRS** zabezpečuje komunikaci uživatele se systémem a umožňuje mu přístup k ostatním částem programu.
- Editor **EDI** je určen k zápisu a opravě zdrojových programů v assembleru Z80.
- Sestavovací program **ASM** (Assembler) překládá zdrojový text, zapsaný v editoru EDI, do binárních modulů a ukládá je do knihovny.
- Spojovací program **LNK** (Linker) vytváří z binárních modulů vykonatelný program.
- Program pro obsluhu knihovny binárních modulů **LIB** (Library) poskytuje některé funkce nezbytné při práci s knihovnou.
- Ladicí program **DBG** (Debugger) slouží k ladění programů.
- Zpětný překladač **DIS** (Disassembler) vytváří z binárního kódu zdrojový text, který lze zpracovat ostatními moduly.

Systém je vytvořen v jazyku assembler Z80 pro aplikaci na domácím počítači ZX Spectrum a s ním plně kompatibilních (ZX Spectrum+, Delta, Didaktik Gama).

# 1. SPUŠTĚNÍ SYSTÉMU

Systém MRS se dodává na magnetofonové kazetě ve třech verzích. Každá z nich obsahuje tři části.

V první části je basikový program, jehož hlavní úlohou je načíst další dvě části strojového kódu do počítače. První část obsahuje i ukázku toho, jak by měl vypadat uživatelský program pro práci s periferiemi (viz část 3.5.).

Ve druhé části je vlastní systém MRS. Jeho strojový kód je uložen na adresách D100H-FF00H. Prostor paměti FF00H-FF40H je využit pro zásobník (po spuštění systému reg.SP obsahuje FF40H). Startovací adresa (studený start) systému MRS je v basikové části uložena na řádce 30. Na řádce 40 je adresa teplého startu - jejím voláním se uživatelská paměť neinicializuje (v ní uložený zdrojový text nezmizí).

Po načtení systému z pásku se automaticky provede příkaz na řádce 30. Kdykoli během programování se můžeme vrátit do interpretu Basiku a podle potřeby volit příkaz GO TO 30 pro inicializaci systému nebo GO TO 40 pro vstup do ASM se zachováním zdrojového textu.

Třetí část obsahuje základní modul binární knihovny (její funkce viz část 6.2.). Podle umístění modulu v paměti se rozlišují tři verze systému MRS. Modul SYSMOD (základní modul knihovny) je v každé verzi uložen od jiné adresy:

1. verze - od 6800H
2. verze - od 8800H
3. verze - od A800H

Hexadekadická adresa uložení se objevuje i v názvu zaváděcího programu, který načítá vlastní systém MRS a modul SYSMOD: MRS6800, MRS8800, MRSA800.

Prostor paměti od konce modulu SYSMOD nahoru je využit pro ukládání binárních programů uživatele. Zdrojový text assembleru se ukládá od adresy D100H (začátek systému MRS) směrem dolů.

Od adresy FE00H leží pracovní paměť systému MRS. Na tuto paměť se v dalším textu odvolávám názvem BUFFER.

Každá verze tvoří uzavřený celek, proto nelze použít např. vlastní systém MRS6800 a SYSMOD 8800 apod. Příslušná verze se jako jeden celek načte příkazem LOAD "" nebo LOAD "MRSXXXX", kde XXXX symbolizuje číslo verze.

# 2. ŘÍDICÍ MODUL

Řídicí modul se ohlásí zprávou MRS na poslední řádce obrazovky. Tato řádka je vyhrazena komunikaci uživatele se systémem - v dalším textu jí budeme říkat dialogová řádka. Uživatel do ní zapisuje příkazy z klávesnice. K opravě zapsaného textu slouží tlačítka:

**CAPS SHIFT/5** - posun kurzoru na znak vlevo
**CAPS SHIFT/8** - posun kurzoru na znak vpravo
**CAPS SHIFT/9** - vložení mezery na pozici kurzoru
**CAPS SHIFT/0** - výmaz znaku na pozici kurzoru
**ENTER** - vložení textu zapsané řádky do programu

Všechny znaky, které systém jakkoli zpracovává, musejí být zapsány malými písmeny. Systém rozeznává i velká písmena, která můžeme použít např. na poznámky nebo jako textové konstanty (viz modul EDI).

Jestliže při zápisu textu překročíme délku dialogové řádky nebo pokusíme-li se vložit nedefinovaný příkaz, ozve se varovný zvukový signál.

Dialogová řádka má kapacitu 64 znaků. Po vložení více než 32 znaků se její text posouvá vlevo a mizí z obrazovky.

Modul MRS rozeznává tyto příkazy:

**EDI** - aktivace editoru
**ASM** - aktivace assembleru
**LNK** - aktivace spojovacího programu
**LIB** - aktivace programu obsluhy knihovny
**DBG** - aktivace ladicího programu
**MON** - aktivace interpretu Basiku
**ALD** - postupná aktivace modulů ASM, LIB, DBG (viz modul EDI)
**RUN** - spuštění uživatelského programu od adresy definované pseudoinstrukcí ENT modulu LIB (viz modul LIB)

Jiné příkazy jsou vyhodnoceny jako chybné za doprovodu zvukové signalizace. Kurzor se automaticky umístí na první znak příkazu pro opravu chyby. Tentýž proces proběhne, jsou-li za správně zapsaným příkazem uvedeny nesprávné parametry (viz popis jednotlivých příkazů).

Zpětný překladač DIS je zvláštním programem pro vytváření zdrojových textů. Aktivovat jej lze pouze z modulu EDI.

# 3. EDITOR

## 3.1. Příkazový režim

Editor umožňuje práci ve dvou režimech - příkazovém a obrazovkovém. Po vyvolání je v režimu příkazovém, který se ohlásí zprávou EDI v dialogové řádce.

**Příkazy editoru:**

**INI** - inicializace pracovní oblasti editoru a příprava pro vložení nového textu. Obrazovka se vymaže, kurzor se nastaví na první pozici a editor přejde do obrazovkového režimu.

**LN- LN+ LN=** - všechny tři příkazy umístí kurzor na určenou řádku textu: LN-N o N řádek směrem k začátku textu; LN+N o N řádek směrem k jeho konci; LN=N umístí kurzor přímo na řádku N. Příkaz LN-0 umístí kurzor na první řádku; LN+0 na poslední řádku; LN=0 na posledně opravovanou řádku. Příkaz LN= může obsahovat i řetězec znaků; např. LN=VSTUP. Kurzor se umístí na první řádku, která v poli návěští obsahuje zadaný text. Když taková řádka neexistuje, hledání se zastaví a uživatel může zadaný řetězec opravit. Po vykonání příkazu se na obrazovku vypíší řádky od nalezené až do konce obrazovky, resp. textu, kurzor se umístí na 1. pozici a editor přejde do obrazovkového režimu (viz dále). LN je anglicky LiNe (řádka).

**DLB** - za příkazem následují dvě dekadická čísla ve tvaru N-M, např. DLB 10-100. Příkaz vymaže řádky s pořadovými čísly v intervalu N-M včetně; poté editor očekává další příkaz. DLB je anglicky **DeLete Block** (vymaž blok).

**CPB** - za příkazem následují dvě dekadická čísla ve tvaru N-M. Příkaz zkopíruje řádky s pořadovými čísly N-M před řádku, na níž byl kurzor při přechodu do dialogového režimu; např. CPB 23-45. Když je tato řádka uvnitř intervalu N-M, považuje se to za chybu a příkaz se nevykoná (viz též obrazovkový režim). CPB je anglicky **CoPy Block** (zkopíruj blok).

**SAV** - uložení zdrojového textu na magnetofonový pásek. Tato verze systému využívá rutiny romky ZX Spectra, ale poskytuje k nim vlastní připojení. Za příkazem SAV následuje libovolný název textu, který musí být od příkazu oddělen jednou mezerou. Délka názvu je max. 10 znaků; např. SAV ASSEMBLER. Když název chybí, považuje se to za chybu (ozve se výstražný zvukový signál). SAV je zkráceně **SAVe** (ulož). Editor sám určí paměťový rozsah textu pro jeho zápis na pásek. Zdrojový text je uložen se standardní hlavičkou, která má v posledních dvou bajtech zapsané číslo 1. Podle toho pak editor při čtení záznamu pozná, že jde o zdrojový text (jinou hlavičku odmítne). Před odesláním příkazu SAV (tlačítkem ENTER) je nutné spustit nahrávací magnetofon. Po zápisu textu nebo po jeho přerušení funkcí BREAK (CAPS SHIFT/SPACE) se opět přihlásí editor.

**MER** - načtení zdrojového textu z magnetofonu. Je-li v editoru již nějaký text uložen, nepřepíše se, ale v případě úspěšného načtení je do editoru vložen před řádku, na níž byl kurzor v momentu přechodu z režimu obrazovkového do příkazového. Zde jsou opět využity rutiny romky ZX Spectra. Za příkazem MER následuje jedna mezera a název požadovaného textu (max. 10 znaků), např. MER ASSEMBLER. Když název nezadáme, načte se první text, který editor na pásku najde. Editor rozeznává jen vlastní textové soubory (jiné, byť stejnojmenné, nenačte). Během hledání textu se zadaným názvem vypisuje editor do dialogové řádky názvy nalezených souborů, dokud nenajde hledaný - pak vedle něj vypíše hlášení LOADING. Vyhledávání a načítání lze přerušit funkcí BREAK. Poté se řízení vrátí do editoru. MER je zkráceně **MERge** (sluč). Když během načítání dojde k chybě, objeví se hlášení IO ERROR (**IO** značí **Input/Output**, **error** je chyba). Po stisku tlačítka ENTER se pak řízení vrátí editoru. Když editor zjistí, že se text do paměti už nevejde, oznámí to hlášením MEM FULL; MEM značí **MEMory**, (paměť), spolu s **full** (plný) to znamená paměť plná. Návrat do editoru zajistí stisk tlačítka ENTER. Proces vkládání textu po jeho načtení z pásku může trvat až 20 sekund. Během nich je v dialogové řádce hlášení WAIT (čekej). Po vložení textu se opět přihlásí editor zprávou EDI.

**LOA** - příkaz má podobnou funkci jako MER s tím rozdílem, že z pásku načtený text přepíše v paměti případně již uložený. Jinak o LOA platí

v zásadě vše, co o MER. Po načtení textu se řízení ihned vrací editoru. LOA je zkráceně **LOAd** (zaveď).

SUS - obdoba příkazu SAV. Slouží k zapsání zdrojového textu na jinou vnější paměť, jejíž obsluhu si realizuje uživatel sám (viz část 3.5.). SUS je zkráceně **Save USer's** (user's je česky uživatelův).

MUS LUS - obdoby příkazů MER a LOA; ostatní jako u SUS. MUS je zkratka **Merge USer's**, LUS - **Load USer's**.

ALD - speciální příkaz pro postupné volání jednotlivých modulů ASM, LNK, DBG (název ALD je složen z prvních písmen jmen všech tří modulů). Pokud překlad nebo spojování skončí s chybami, volání ostatních modulů se neuskuteční a řízení se vrátí editoru (blíže viz popis jednotlivých modulů).

DIS - aktivace zpětného překladače (disassembleru). Za příkazem následuje jedna mezera, pak počáteční a koncová adresa paměti, jejíž obsah chceme disassemblovat. Forma zápisu: DIS XXXX-YYYY. Adresy se zadávají hexadekadicky, bez znaku #. Instrukce na adrese YYYY se už nepřekládá. Text se ukládá na řádek, na němž byl kurzor. Výrazy v adresovém poli jsou zobrazeny hexadekadicky. Když hodnota na některé adrese neobsahuje kód platné instrukce, je přeložena pomocí pseudoinstrukce DB (viz popis překladače). Když chceme získat i hodnoty čítače adres, můžeme získaný zdrojový text normálně přeložit a na základě výpisu překladu doplnit výrazy v poli adresy o návěští.
Příkaz DIS lze využít, když chceme pokračovat v práci na programu, který jsme dosud vytvářeli v jiném systému (či ve strojovém kódu), nebo když chceme do odladěného programu vložit některé moduly systému MRS, aby se program na něm stal nezávislý (viz popis modulů LIB a LNK). S jeho pomocí můžeme upravit i obsah modulů uložených v knihovně (viz modul LIB).
Když se při zpětném překladu přeplní paměť, vypíše se hlášení MEM FULL a řízení se po stisku tlačítka ENTER vrátí systému MRS.

INS - speciální obdoba příkazu DIS. Slouží k vložení zdrojového textu, který nebyl vytvořen systémem MRS. Za příkazem následují dvě hexadekadické konstanty, podobně jako za DIS. První konstanta

udává počáteční adresu uživatelova podprogramu, druhou je inicializován reg. HL před prvním voláním podprogramu. Tento registr je k dispozici uživatelovu modulu a při dalších voláních má hodnotu, jakou měl při výstupu z podprogramu. Úlohou uživatelského podprogramu je při každém volání vložit do paměti (počínaje adresou FEC0H) další řádek zdrojového textu a vynulovat indikátor CY. Když už není k dispozici další řádek zdrojového textu, CY se převrátí na log. 1 a řízení se vrátí editoru.
Každá řádka textu je syntakticky zkontrolována a v případě správné syntaxe je do textu vložena ve zhuštěném tvaru před řádek, na němž byl kurzor (viz příkaz DIS). Chybné řádky textu se zobrazí v první řádce obrazovky a v pravém dolním rohu se objeví jejich pořadové číslo. Řádky je pak možné opravit tak, jak je uvedeno v části 3.3. Opravený tvar se do textu vloží stiskem tlačítka ENTER. V případě chyby zazní zvukový signál a cyklus se opakuje. Příkaz INS můžeme přerušit funkcí BREAK (CAPS SHIFT/SPACE) - v dialogové řádce se objeví hlášení BREAK a po stisku tlačítka ENTER se řízení vrátí systému MRS.

**N64** - příkaz k výpisu 64 znaků na řádce. Týká se práce editoru v obrazovkovém režimu (viz dále) a výpisu překladu z modulu ASM. V tomto módu se kurzor zobrazuje v podobě inverzního znaku.

**N32** - příkaz k výpisu 32 znaků na řádce.

**MON RUN ASM LNK LIB DBG** - tyto příkazy zavolají příslušný modul systému a odevzdají mu řízení.

## 3.2. Obrazovkový režim

Po příkazech LN+, LN-, LN= nebo INI přejde editor do obrazovkového režimu. Na obrazovku se vypíše text od řádky nastavené příkazem LN (po INI je obrazovka prázdná), kurzor je v levém horním rohu obrazovky a editor očekává vložení dalšího příkazu, resp. textu.

Pro pochopení obrazovkového režimu je nutné si uvědomit, že editor sám už upravuje zdrojový text pro překladač. Proto se během zápisu textu kontroluje správnost syntaxe a v dalším textu se budou prolínat informace o editoru s informacemi o assembleru.

Obecný tvar řádky zdrojového textu je rozdělen do čtyř polí:

INSTRUKCE

```
LABEL     TYP     ADR              POZN
-----------------------------------------------------------
VSTUP     LD      A,5              5 do reg.A
          ENT     VSTUP
;POZNAMKA NA VOLNE RADCE
*A
```

**LABEL** - pole návěští. Začíná na nulté pozici zleva a má délku 7 znaků. Buď je celé vyplněné mezerami, nebo obsahuje návěští (řetězec max. 6 písmen a číslic). Návěští musí začínat písmenem, nikoli číslicí! Zbytek pole musí obsahovat mezery.

**TYP** - pole typu instrukce. Začíná na 7. pozici zleva a má délku 5 znaků. Pole je buď vyplněno mezerami, nebo obsahuje platnou instrukci Z80, resp. pseudoinstrukci assembleru. Zbytek pole musí obsahovat mezery.

**ADR** - pole adresy. Začíná na 12. pozici zleva a má délku 20 znaků. Když je pole TYP vyplněno mezerami, musí je obsahovat i pole ADR. V opačném případě je v něm platná adresa pro instrukci nebo pseudo-instrukci ze sousedícího pole TYP. Pole ADR musí být zakončeno mezerou, ev. zbytek pole musí obsahovat mezery.

**POZN**- pole poznámky. Začíná na 32. pozici zleva a má délku 32 znaků. Může obsahovat libovolné znaky, protože v něm umístěný zápis se nepřekládá. Při psaní v módu 32 znaků na řádce se text při překročení délky 32 znaků posouvá doleva mimo obrazovku, což nás upozorňuje, že jsme přešli do tohoto pole. Pole poznámky nemůže být na řádce, která neobsahuje instrukci.

Kromě uvedené řádky, sestávající ze čtyř polí, pracuje systém ještě se dvěma - řádkou poznámky a řádkou pro řízení překladu. Tyto dvě řádky nejsou členěny na pole, proto se v nich stisk kombinace tlačítek CAPS SHIFT/1 a CAPS SHIFT/4 považuje za chybu (viz část 3.3.).

Řádka poznámky začíná znakem ; (středníkem) na nulté pozici zleva. Může obsahovat jakékoli znaky, protože se nepřekládá. Řízení překladu se ovládá řádkou, která začíná znakem * (hvězdička). Obsah řádky podléhá syntaktické kontrole, její zbytek musí obsahovat mezery. Platné příkazy překladu jsou uvedeny v části 4.3.

Při zápisu instrukce musíme dát pozor, abychom u ní příliš dlouhými výrazy nevjeli do pole poznámky. Vše, co by tak přesáhlo pole ADR, by pak při překladu nebylo zpracováno.

Pole adresy může kromě prvků přesně předepsaných instrukcí obsahovat i prvky, které se vyhodnocují během překladu. Jsou to tzv. výrazy obsahující symboly a matematické operátory. Výraz v systému MRS může mít jeden z těchto tvarů:

| | |
|---|---|
| symbol | |
| symbol + symbol | |
| symbol - symbol | |
| symbol * symbol | |
| symbol / symbol | |
| symbol ! symbol | (operace OR) |
| symbol & symbol | (operace AND) |
| symbol @ symbol | (operace XOR) |
| - symbol | |
| >.symbol | (vyšší bajt dvoubajtové konstanty) |
| <.symbol | (nižší bajt dvoubajtové konstanty |

**Symbol může mít jeden z těchto tvarů:**

- návěští
- dekadická konstanta (řetězec číslic)
- hexadekadická konstanta (řetězec číslic a písmen a,b,c,d,e,f; tento řetězec musí začínat znakem #)
- znaková konstanta (libovolný zobrazitelný znak mezi dvěma apostrofy)
- znak $ (hodnota čítače instrukcí)

Kromě instrukce Z80 může být v poli TYP i některá z následujících pseudoinstrukcí:

**ORG** - v adresové části musí být výraz
**DS** - v adresové části musí být výraz
**DB** - v adresové části musí být jeden nebo více výrazů oddělených čárkami; výrazem může být i řetězec zobrazitelných znaků mezi dvěma apostrofy
**DW** - v adresové části musí být jeden nebo více výrazů oddělených čárkami
**EQU** - v poli návěští musí být návěští a v poli adresy musí být výraz
**ENT** - v poli adresy musí být jedno nebo více návěští oddělených čárkami
**EXT** - v poli adresy musí být jedno nebo více návěští oddělených čárkami
**END** - pole adresy musí být prázdné

**Příklady:**

```
        ORG  #aaaa
        DS   #05
        DS   pocet
        DB   #ff,#a0
        DB   sloup,"ahoj"
        DW   adresa,kurzor,obraz
pocet   EQU  #33*#22+adr0
        ENT  vystup
        EXT  vstup,vystup,nahoru
        END
```

Kromě END a ORG mohou být ostatním pseudoinstrukcím předřazena návěští (u EQU být musí).

## 3.3. Zápis řádky

Řádka se píše běžným způsobem z klávesnice, přičemž lze využít tato speciální tlačítka:

**CAPS SHIFT/8** - posun kurzoru na následující znak,
**CAPS SHIFT/5** - posun kurzoru na předchozí znak,

**CAPS SHIFT/0** - výmaz znaku na pozici kurzoru; zbytek řádky za kurzorem vpravo se posune o 1 pozici doleva,

**CAPS SHIFT/9** - vložení mezery na pozici kurzoru; zbytek řádky za kurzorem vpravo se posune o 1 znak doprava,

**CAPS SHIFT/4** - přesun kurzoru na začátek dalšího pole vpravo,

**CAPS SHIFT/1** - přesun kurzoru na začátek pole, resp. na začátek pole předcházejícího (doleva),

**SYMBOL SHIFT/SPACE** - přesun kurzoru na poslední znak pole ADR,

**SPACE** - vyplnění zbytku pole mezerami a skok kurzoru na začátek dalšího pole vpravo; tato funkce je potlačena v poli poznámky, v řádce poznámky, nebo když je mezera součástí řetězce či znakové konstanty - v těchto případech se mezera uloží do textu normálně.

Další řídící znaky způsobí ukončení zápisu řádky. Když je v některém poli zjištěna syntaktická chyba, ozve se výstražný signál a kurzor se nastaví na začátek chybného pole, abychom mohli provést opravu.

Kvůli úplnosti byly některým znakům, které systém MRS nepoužívá, přiřazeny nestandardní kódy. Tak můžete psát i znaky { | } ~ [ \ ] současným stiskem tlačítka SYMBOL SHIFT a A, S, D, F, Q, W, E.

Syntakticky správně zapsaná řádka se poměrně komplikovaně komprimuje (kvůli uložení komprimátu do paměti) a opět se expanduje na výpis zdrojového textu srozumitelný člověku . Potom se porovnává prvotně vložený text s touto řádkou; když se jejich obsah liší, systém vypíše expandovanou řádku místo původní a zazní výstražný zvukový signál. Uživatel pak řádku může opravit. Tento komplikovaný systém je uživateli prakticky skryt. Pokud jsou totiž řádky stejné (a tak tomu je vždy, když je řádka správně zapsána), systém normálně pokračuje ve své práci.

**Pomocí uvedeného procesu jsou zjišťovány tyto chyby:**

- přetečení (příliš velká konstanta se komprimací změní),
- neformalizovaný zápis konstanty (hexadekadická konstanta musí být zapsána sudým počtem bajtů, před dekadickou nesmějí být nuly),
- formálně správné, ale neexistující instrukce (např. ADD HL,IX - se přetvoří na ADD IX,IX a následuje hlášení o chybě).

Moduly EDI, ASM a DBG správně vyhodnocují i nestandardní instrukce. Např. SLL (operační kódy CB30H-CB37H) a instrukce obdobné instrukcím operujícím s registry H a L (dosazením prefixu DDH, resp. FDH ) se pak

vztahují k operacím s polovinami indexových registrů IX, resp. IY. Tyto instrukce se zapisují tak, že k symbolu reg. H se připíše X, resp. Y (XH, resp. YH). Obdobně u reg. L - XL, resp. YL (X a Y je tu svého druhu "mnemonickým prefixem"). Tak např. YH znamená vyšší bajt reg. IY, XL nižší bajt reg. IX. (H je anglicky **High**, L je **Low**). Tak instrukce LD XL,1 naplní číslem 1 nižší polovinu reg. IX. Ale ne každý prefix k reg. H a L je možný - systém však chybné obměny odhalí a upozorní na ně.

*(Pozn. prekl.: Tyto nestandardní instrukce pracují bezchybně pouze s mikroprocesory Z80, vyrobenými firmami Zilog a Mostek. U ostatních mikroprocesorů může jejich použití přinášet nepředvídatelné výsledky).*

U systému MRS jsou změny ve standardním zápisu těchto instrukcí:

EX AF, AF' - píše se bez apostrofu: EX AF, AF
IM 0, IM 1, IM 2 - píší se bez mezery: IM0, IM1, IM2

V případě správně zapsané řádky je další akce systému závislá na volbě jednoho z těchto řídících znaků:

**CAPS SHIFT/7** - kurzor se umístí na začátek předešlé řádky. Když je na 1. řádce obrazovky, její obsah se posune o 1 řádku směrem dolů. Když je na 1. řádce celého textu, zazní jen zvukový signál.

**CAPS SHIFT/6** - kurzor se umístí na začátek následující řádky. Když je na poslední řádce obrazovky, její obsah se posune o 1 řádku směrem nahoru. Když je za poslední řádkou textu, zazní jen signál.

**CAPS SHIFT/3** - posun textu o 1 celou obrazovku směrem nahoru; kurzor se umístí na začátek 1. obrazové řádky.

**CAPS SHIFT/2** - posun textu o 1 celou obrazovku směrem dolů; kurzor se umístí na 1. obrazovou řádku.

**SYMBOL SHIFT/ENTER** - řádka, na níž je umístěn kurzor, se stane 1. řádkou obrazovky.

**CAPS SHIFT/SYMBOL SHIFT** - výmaz řádky, na níž leží kurzor; text pod vymazanou řádkou se posune o 1 řádku vzhůru. Kurzor se umístí na nejblíže posunutou řádku. Pokud byl kurzor za poslední řádkou

textu, ozve se zvukový signál a příkaz se neprovede. Vymazávaná řádka není syntakticky kontrolována.

**CAPS SHIFT/ENTER** - za řádku, na níž leží kurzor, se vloží její kopie a kurzor se posune na ni.

**ENTER** - pod řádku s kurzorem se vloží prázdná řádka. Tak se vytvoří místo pro vložení další řádky. Kurzor se objeví na jejím začátku. Níže položené řádky se posunou o 1 řádku dolů; když je kurzor na poslední řádce obrazovky, všechny řádky se posunou o 1 řádku nahoru. Když je kurzor za poslední řádkou textu, ozve se jen zvukový signál.

**CAPS SHIFT/SPACE** - editor vypíše hlášení EDI do dialogové řádky a přejde do příkazového režimu.

Odeslání jiného řídícího znaku, než jsou zde uvedené, se považuje za chybu (ozve se výstražný signál).

## 3.4. Chybová hlášení

Na většinu chybně provedených akcí reaguje editor výstražným zvukovým signálem, případně i nastavením kurzoru na chybné pole. Jsou však dva chybové stavy, při nichž se hlášení o chybě objeví v dialogové řádce (do systému se vrátíme stiskem tlačítka ENTER). Hlášení

**MEM FULL**

znamená, že paměť vyhrazená zápisu textu nedostačuje. Pak je nutno text zkrátit nebo vymazat moduly z binární knihovny (viz část o obsluze knihovny). Prakticky však k tomu dochází zřídka, protože díky komprimaci se do volné paměti vejde až 5 000 řádek zdrojového textu.

Když dojde k přeplnění paměti, vyvoláme editor, příkazem LN=0 přejdeme do obrazovkového režimu, tlačítky CAPS SHIFT/SYMBOL SHIFT vymažeme řádku, při jejímž vložení došlo k přeplnění, a tlačítky SYMBOL SHIFT/ENTER obnovíme výpis obrazovky.

Jinou příčinou výskytu tohoto hlášení je zaplnění tabulky návěští. Tabulka je uložena za textem a dynamicky se rozšiřuje. Její maximální kapacita je 255 návěští. Ta návěští, která nejsou během překladu použita, se po

ukončení překladu z tabulky automaticky vymažou. Místo v tabulce tedy získáme tak, že vymažeme libovolné návěští všude, kde se v textu vyskytuje, a text znova přeložíme.

## 3.5. Práce s periferiemi

Editor a program pro obsluhu knihovny mají přímo implikovány funkce pro práci s magnetofonovým záznamem. Protože pro ZX Spectrum bylo vyrobeno mnoho odlišných typů vnějších paměťových zařízení, nelze pro jejich obsluhu vytvořit nějaký univerzální program. Proto byl do systému zabudován modul, který volá interpret Basiku. S jeho pomocí můžete vytvořit svůj vlastní obslužný podprogram. Pro ten účel jsou v systému příkazy LUS (obdoba LOA), SUS (obdoba SAV) a MUS (obdoba MER). Provedením těchto příkazů editor nebo program pro obsluhu knihovny uloží potřebné informace na pevně určené místo v paměti a řízení odevzdá interpretu Basiku. Obslužný podprogram pak vykoná potřebné operace a vrátí řízení systému MRS.

**Informace jsou v paměti uloženy od adres:**

FE00H - adresa začátku paměti (2 bajty)
FE02H - adresa konce paměti (2 bajty)
FE04H - návratová adresa (2 bajty)
FE06H - kód příkazu (1 bajt)
FE07H - kód výsledku operace (1 bajt)
FE08H - od této adresy je uložen název, zadaný uživatelem; název souboru je ukončen binární nulou

**Kódy jednotlivých příkazů:**

0 - zápis binární knihovny na vnější paměť
1 - zápis zdrojového textu na vnější paměť
2 - načtení binární knihovny do počítače
3 - načtení zdrojového textu do počítače
4 - načtení a vložení zdrojového textu do textu stávajícího (příkaz MUS)

Když je kód příkazu 0 nebo 1, adresa začátku a konce paměti určuje oblast, která se bude zapisovat na vnější paměť.

Když je kód příkazu 2 nebo 3, soubor se načte od adresy, z níž byl původně přenesen na vnější paměť. Adresy začátku a konce paměti označují její volný rozsah.

Při příkazovém kódu 4 se soubor načítá do paměti tak, aby poslední bajt načítaného zdrojového textu byl uložen na koncové adrese paměti. Jen v tomto případě systém testuje kód výsledku operace na adrese FE07H. Když operace skončila úspěšně, obslužný podprogram uloží na tuto adresu hodnotu 0 a editor zařadí načtený text podobně jako u příkazu MER. Když obslužný podprogram uloží na uvedenou adresu hodnotu 1, editor případně načtený program ignoruje.

V každém případě obslužný podprogram odevzdá řízení návratové adrese uložené na adresách FE04H a FE05H buď instrukcí CALL, nebo z Basiku příkazem USR.

V basikové části dodaného programu jsou pro vaši informaci zařazeny příkazy pro obsluhu magnetofonu. Aktivují se systémovými příkazy SUS, LUS. Mohou vám být rovněž malým vodítkem pro tvorbu obdobných podprogramů pro jiné periférie.

# 4. PŘEKLADAČ ASSEMBLERU

## 4.1. Spuštění překladu

Překladač assembleru se aktivuje příkazem ASM, za nímž může - jako jeho další parametr - následovat název překládaného modulu (zdrojového textu). Když překladač najde stejnojmenný modul v binární knihovně, uloží překlad na jeho místo. Pokud modul zadaného názvu v knihovně neexistuje, překladač přeloží zdrojový text jako nový modul a připojí jej na konec knihovny.

Jestliže v příkazu ASM název modulu chybí, překladač vytvoří modul s prázdným názvem na konci knihovny. Takový modul je v knihovně "dočasně" - každý další překlad jej vymaže.

Jinými slovy - uživatel by měl nejdříve přeložit a odladit modul bez názvu, a teprve když je vše v pořádku, zařadit jej příkazem ASM i s názvem do binární knihovny.

Překladač sleduje hranice paměti, která je modulu přidělená, a to buď podle velikosti místa, jež původní modul zabíral, nebo podle prostoru od konce knihovny do konce volné paměti.

Překladač překládá text připravený editorem do příslušné paměti a na obrazovce vytváří výpis překladu (pokud o něj uživatel požádá - viz direktivy v části 4.3.). Řádka textu se zobrazí na dvou až třech řádkách obrazovky takto:

| | | | | |
|---|---|---|---|---|
| 1. - | 5. | znak | - | pořadové číslo řádky |
| | 7. | znak | - | mezera nebo kód chyby |
| 9. - | 12. | znak | - | hexadekadický výpis hodnoty čítače adres |
| 14. - | 21. | znak | - | hexadekadický výpis strojového kódu |

Druhá (a v případě poznámky i třetí) řádka obrazovky obsahuje assemblerový výpis. Když byl příkazem N64 nastaven mód výpisu 64 znaků na řádce, bude výsledný výpis v jedné (resp. dvou) řádkách.

Výpis můžeme kdykoli přerušit tlačítky CAPS SHIFT/S a poté opět spustit stiskem CAPS SHIFT/Q.

Po skončení překladu se vypíše informace o počtu chyb

**ERRORS: N** kde N je počet zjištěných chyb.

Pokud MRS během překladu objeví chyby, modul není do knihovny zařazen. Protože však překlad do knihovny probíhá vždy (je-li modul bez názvu, je umístěn na konec knihovny nebo do stejnojmenného modulu), pak při chybném překladu bude modul se stejným názvem zničen. Funkce ostatních modulů však zůstanou zachovány.

Jakmile se celý text přeloží (na poslední řádce textu musí být pseudoinstrukce END), systém prohlédne tabulku symbolů a v překladu nepoužité symboly z ní vymaže.

Jestliže si vyžádáme direktivou *A výpis všech řádek (viz část 4.3.), na závěr překladu se vypíše tabulka použitých návěští. Jejich názvy jsou uspořádány abecedně, za každým z nich je vypsána jeho hexadekadická hodnota. U návěští označených pseudoinstrukcí EXT jsou jejich případně neznámé hodnoty označeny znaky XXXX.

## 4.2. Vyhodnocení výrazů

Výraz v poli adresy se vyhodnocuje běžným způsobem - dekadická a hexadekadická konstanta má hodnotu danou svým zápisem, znaková konstanta má hodnotu danou hodnotou ASCII znaku, který ji reprezentuje, a hodnota návěští je dána okamžitou hodnotou čítače adres v momentě, kdy bylo návěští identifikováno.

**Speciálními unárními výrazy jsou:**

- symbol, který definuje hodnotu vyšších osmi bitů šestnáctibitové hodnoty
- symbol, který definuje hodnotu nižších osmi bitů šestnáctibitové hodnoty

Např. když chceme zjistit, zda je obsah registru L shodný s obsahem 8 nižších bitů návěští ADRESA, můžeme to provést takto:

```
LD    A,L
CP    <ADRESA
```

Jinou možnost, jak návěští přiřadit hodnotu, představuje pseudoinstrukce EQU (viz níže).

Symbol $ během překladu nabývá hodnotu čítače instrukcí.

## 4.3. Řízení překladu pseudoinstrukcemi a direktivami

Pseudoinstrukce řídí překlad zdrojového textu a direktivy řídí vlastní činnost překladače. Pseudoinstrukce byly zmíněny již v části věnované editoru (3.). Zde si probereme jejich funkce ve vztahu k assembleru.

**ORG** - určuje obsah čítače adres. Výraz v poli adresy se vyhodnotí a jeho hodnota je přiřazena čítači adres. Prvky výrazu musejí být známy již při prvním průběhu překladu. Tzn. že součástí výrazu nesmí být návěští, které se nevyskytuje před touto pseudoinstrukcí.
Forma zápisu např.: ORG #8000 nebo ORG START apod. ORG je zkratka anglického slova **ORiGin** (počátek uložení strojového kódu do paměti - viz i direktiva *C).

**DS** - určuje obsah čítače adres tak, že v momentě, kdy překladač narazí na pseudoinstrukci , zvýší svůj obsah o hodnotu výrazu uvedenou za DS v poli adres. Výsledkem je rezervování paměti v rozsahu daném výrazem. Např. DS #20 rezervuje 32 adres, DS KOUSEK rezervuje počet adres daný výrazem KOUSEK. Podobně jako v případě ORG musí být hodnota výrazu známa již v prvním průběhu překladu. DS je totéž, co v některých překladačích DEFS, což je anglická zkratka slov **Define Space** (definuj prostor).

**DB** - ukládá do paměti hodnoty určené výrazy v poli adres. Tyto hodnoty musejí být v číselném rozsahu jednoho bajtu (0..255). Když je v adresové části řetězec znaků v apostrofech, uloží se do paměti číselné hodnoty ASCII kódů příslušných znaků řetězce. Příklady: DB #FF nebo DB "AHOJ". DB je totéž, co v jiných assemblerech DEFB. Jde o zkratku anglických slov **Define Byte** (definuj bajt).

**DW** - ukládá do paměti hodnoty určené výrazy v poli adres. Každá zabírá 2 bajty. Např. DW #4000 nebo DW ADRESA. DW je ekvivalentem DEFW jiných překladačů. Je to zkratka anglických slov **Define Word** (definuj slovo).

**EQU** - přiřadí návěští hodnotu danou výrazem v poli adres. I tato hodnota musí být známa již v prvním průběhu překladu.

Příklady:

DESET EQU #0A nebo ADR100 EQU KONEC-START. EQU je zkratka anglického **EQUate** (rovnat se).

**ENT** - určuje návěští v poli adres jako bod vstupu (angl. **ENtry poinT**). Význam této pseudoinstrukce je popsán v části věnované spojovacímu programu.

**EXT** - určuje návěští v poli adres jako vnější (externí) symbol (angl. **EXternal poinT**). S významem této pseudoinstrukce se seznámíte v části o spojovacím programu (5.).

**END** - ukončení překladu části zdrojového textu (text se přeloží od začátku až po tuto pseudoinstrukci).

Pseudoinstrukce ENT a EXT můžeme použít jen tehdy, když je v paměti přítomna knihovna binárních modulů (alespň základní modul SYSMOD). V opačném případě je výsledek jejich použití nepředpokládatelný.

Pro řízení práce překladače se do textu vkládají speciální řádky začínající znakem * (hvězdička). Těsně za ní musí následovat znak symbolizující některou z těchto direktiv:

***A** - při překladu se na výstupu (obrazovce, tiskárně) vypíší všechny řádky. Pokud chceme získat jen výpis tabulky symbolů (viz část 4.1.), direktivu *A vložíme až před řádku s pseudoinstrukcí END.

***E** - na výstupu se vypíší jen chybné řádky. Tato hodnota je předdefinována na začátku překladu.

***L** - nasměrování výpisu na tiskárnu (viz část 4.5.).

***P** - má smysl jen ve spojení s direktivou *L. *P vyšle na tiskárnu řídící kód pro přechod na tisk nové stránky (kód 0CH, angl. **Form Feed**). Dekadická konstanta za *P určuje počet řádek; po jejich vytištění se kód vyšle.

***T** - výpis textu na terminál (obrazovku). Tato hodnota je nastavena na začátku překladu.

Platnost každé z těchto direktiv začíná řádkou umístěnou pod touto direktivou a platí do výskytu jiné direktivy (včetně ní).

Poslední direktiva:

*CNNNN- určuje, od které adresy (NNNN hexadekadicky) se bude ukládat binární kód překladu textu, uvozeného touto direktivou. *C úzce souvisí s pseudoinstrukcí ORG (viz další odstavec). Když NNNN=0, binární kód se nevytváří - tak můžeme získat výpis programu bez narušení obsahu paměti. Platnost direktivy *C ukončuje pseudoinstrukce ORG nebo nový výskyt *C.

## 4.4. Uložení modulu do knihovny (pseudoinstrukce ORG)

Přeložený modul bez názvu nebo s novým názvem se překládá na konec knihovny. Kromě vlastního binárního kódu se do knihovny uloží i informace o názvu a délce modulu (uložení proběhne na začátku překladu) a informace o vstupních (ENT) a vnějších (EXT) návěštích. To znamená, že systém uloží binární kód na místo, jaké mu sám určí. Do tohoto procesu normálně nezasahujeme. Můžeme tak ale učinit pseudoinstrukcí ORG. S její pomocí určíme, od které adresy se bude kód do paměti ukládat. Direktiva *C určuje, kam se v paměti bude kód ukládat. V obou případech je nutné:

- aby byl překlad bez názvu (tak se zabrání uložení kódu do binární knihovny),
- aby se v programu nevyskytovaly pseudoinstrukce ENT a EXT.

Pokud tato pravidla nedodržíme, může dojít k narušení binární knihovny!

Použití ORG a *C si ozřejmíme na příkladu. Předpokládejme, že chceme vytvořit program, který má být uložen v paměti od adresy F000H. Když program začne příkazem ORG #F000, pokus o překlad skončí chybovým hlášením MEM FULL, protože na adrese F000H je kód systému MRS a překladač jej nedovolí přepsat.

Problém se vyřeší, když za ORG #F000 vložíme řádek *C#8000. Program se pak překládá tak, jako by se měl ukládat od adresy F000H (tomu odpovídají všechny absolutní hodnoty návěští), ale ukládá se od adresy 8000H. Po skončení překladu přesuneme binární kód krátkým programem

na adresu F000H nebo jej zapíšeme na pásek od adresy 8000H a zpět do počítače načteme od adresy F000H (systém MRS se tak samozřejmě přepíše).

Když potřebujeme zjistit rozsah programu v paměti, stačí označit poslední řádku (pseudoinstrukci END) návěštím a pomocí modulu DBG (viz dále) zjistit hodnotu návěští a tím i hodnotu první volné adresy za programem.

## 4.5. Použití tiskárny

Jak už bylo řečeno, direktiva *L způsobí, že se výpis začne vytvářet na připojené tiskárně. Systém je dodáván bez obslužného programu, protože způsoby připojení tiskárny k ZX Spectru jsou velmi rozmanité. V knihovně SYSMOD je však zařazen vstupní bod PNTCHR - na této adrese je instrukce skoku JP (normálně adresuje instrukci RET).

Když si napíšeme vlastní tiskovou rutinu (tzv. **driver**, čte se drajvr, česky ovladač) pro tisk jednoho znaku (s ev. inicializací tiskárny), zařadíme ji do knihovny a odpovídajícím způsobem zmodifikujeme adresovou část XXXX skokové instrukce JP XXXX na adrese PNTCHR, získáme možnost tisknout výstupy ze systému na své tiskárně.

Obslužný podprogram tisku, který si vytvoříme, bude prostřednictvím reg. A přebírat kód určený k odeslání na tiskárnu. Systém MRS používá řídící kódy 0DH (CR), 0AH (LF) a 0CH (FF) - **CR** je **Carriage Return** (návrat vozíku), **LF Line Feed** (posun o řádku), **FF Form Feed** (posuv formuláře).

Hlavním cílem vstupního bodu PNTCHR je umožnit tisk výstupu z překladače. Kromě toho nám umožňuje dostat se k nekomprimovanému zdrojovému textu programu. Když např. po připojení disketové jednotky chceme zařadit zdrojové texty pod systém CP/M, napíšeme příslušný podprogram, zmodifikujeme adresu PNTCHR a překládaný program uvedeme direktivami *A a *L. Uživatelský podprogram shromáždí v paměti znaky jedné řádky (až po dvojici znaků CR, LF), odstraní výpis čítače adres, čísla řádky a binárního kódu a vlastní zdrojový text uloží na disketu. V tomto smyslu je vstupní bod PNTCHR opačný k příkazu editoru INS.

## 4.6. Chybová hlášení

Můžeme je rozdělit do dvou skupin. V první řadě to jsou závažné chyby, které okamžitě ukončí překlad - v dialogové řádce se objeví kód chyby a po stisku tlačítka ENTER se řízení vrátí systému MRS. Jde o tyto chyby:

**NO END** - překladač nenašel pseudoinstrukci END, která musí ukončovat překládaný text.

**MEM FULL** - toto hlášení upozorňuje, že se pokoušíme uložit binární kód mimo paměť, která je mu vyhrazena. Buď je vytvářený kód příliš dlouhý, nebo byla nesprávně použita pseudoinstrukce ORG či direktiva *C. Tato chyba se zjistí až při skutečném pokusu o zápis do paměti. Jak už bylo uvedeno výše, vhodným použitím ORG a *C lze program přeložit pro práci na libovolném místě paměti.

**BREAK** - hlášení potvrzuje, že byla stisknuta tlačítka CAPS SHIFT/SPACE, jimiž můžeme přerušit překlad během jeho druhého průběhu (tehdy probíhá výpis zdrojového textu). BREAK nejčastěji využijeme v případě, kdy jsme zapomněli odstranit direktivu *A a nechceme čekat, až výpis dlouhého textu skončí.

Do druhé skupiny patří chyby ve zdrojovém textu. Překladač je odhalí během překladu. Tyto chyby se hlásí kódem, který je vypisován před polem návěští. Chybné řádky se vždy zobrazí na výstupu - tuto funkci systému nelze vypnout.

**Chybové kódy:**

**U** - v adresové části instrukce se vyskytlo návěští, které nebylo definováno (nevyskytlo se v poli návěští nebo jeho hodnota nebyla definována pseudoinstrukcí EXT). Návěští v adresové části pseudoinstrukce EQU musejí být definována před tím, než se vyskytnou v adresové části instrukcí.

**M** - v poli návěští se vyskytuje návěští, jehož hodnota již byla určena (buď už se vyskytlo v poli návěští, nebo už bylo definováno pseudoinstrukcí EXT jako vnější návěští). Stejným kódem bude označena řádka, v níž je jako vstupní bod (ENT) definováno návěští, které se už jako

vstupní bod v binární knihovně vyskytuje. Kódem M se označí všechny řádky, v nichž se taková návěští vyskytují.

D - v adresové části je návěští, které bylo definováno vícekrát (jeho výskyt v poli návěští byl označen kódem M). Toto hlášení má v podstatě jen informativní charakter, ulehčuje opravu chyby vícenásobně definovaného návěští.

R - osmibitový výraz v adresovém poli instrukce překročil rozsah jednoho bajtu (výsledek jde mimo interval 0..255).

E - návěští určené jako vnější (EXT) je součástí výrazu nebo je v poli instrukce, která připouští jen osmibitovou hodnotu.

Z - při operaci dělení došlo k pokusu o dělení nulou.

Výraz, při jehož zpracování byla zjištěna chyba, má hodnotu 0. Pokud se ovšem v adresové části instrukce vyskytuje návěští určené jako vnější, vždy se do binárního kódu ukládají na aktuální adresu informace, které překladač na závěr použije pro přípravu tabulky nezbytné pro práci spojovacího programu. Proto se ve výpisu objeví i hodnoty, které pro uživatele nemají smysl.

## 4.7. Použití relativních skoků

Když se v adresové části instrukce JR objeví výraz obsahující návěští, výsledná hodnota se automaticky upraví tak, aby obsahovala správné posunutí (angl. **displacement**) vůči instrukci. Když ve výrazu návěští není, výsledná hodnota výrazu se přímo zapíše do adresové části instrukce. Např. JR LABEL+4 má ve své adresové části hodnotu LABEL+4-$-2, kde $ je hodnota čítače instrukce v čase jejího překladu. Po překladu bude mít instrukce JR #10 ve své adresové části hodnotu 10H.

# 5. SPOJOVACÍ PROGRAM

Spojovací program se aktivuje příkazem LNK bez parametru. Úloha tohoto programu je velmi jednoduchá, protože většinu jeho práce vykoná už překladač. Spojovací program jen prohledá seznam vnějších návěští v celé binární knihovně, ke každému z nich najde odpovídající vstupní návěští a jeho hodnotu uloží na všechna místa binárního kódu, kde se toto návěští vyskytuje v adresové části. Vnější návěští, které mezi vstupními návěštími není nalezeno, se vypíše na obrazovku pod hlavičkou

**UNDEFINED EXTERNALS:**

tj. v překladu nedefinované vnější body. Nakonec se vypíše ještě informace o počtu N takovýchto bodů:

**ERRORS: N**

Pokud byl spojovací program spuštěn příkazem LNK, řízení se vrátí modulu MRS. Když byl aktivován kombinovaným příkazem ALD, řízení se v případě výskytu chyby vrátí editoru, jinak ladicímu programu.

Jinými slovy - programová návěští, která mají být volána z jiných samostatně překládaných modulů, musíme v modulu, v němž se vyskytují, označit pseudoinstrukcí EXT. Návěští definovaná pomocí EXT musejí reprezentovat šestnáctibitovou hodnotu a nesmějí být součástí výrazu. Spojovací program správně doplní adresové části, v nichž jsou návěští označena jako EXT. Tak se propojí více modulů do jednoho celku.

Zásadně musíme dbát na to, abychom nespustili program s vnějšími návěštími, aniž byl předem spojen spojovacím programem, nebo takový, který obsahuje nedefinované vnější body. Adresová část příslušné instrukce by po překladu obsahovala nedefinovanou hodnotu, vedoucí nejspíše k programovému kolapsu.

# 6. PROGRAM PRO OBSLUHU BINÁRNÍ KNIHOVNY

## 6.1. Ovládání programu pro obsluhu binární knihovny

Program se spouští příkazem LIB bez parametru. Po svém spuštění se přihlásí výpisem

**LIB**

v dialogové řádce a očekává jeden z těchto příkazů:

**LST** - vypíše seznam modulů uložených v knihovně a jejich vstupní návěští v hexadekadickém vyjádření. Stejně jako výpis překladače i tento můžeme zastavovat a spouštět tlačítky CAPS SHIFT/Q nebo přerušit pomocí CAPS SHIFT/SPACE - tehdy systém vypíše hlášení BREAK a po stisku tlačítka ENTER se vrátí do řídicího modulu.

**DEL** - vymaže poslední modul knihovny. Moduly z vnitřní části knihovny nelze vymazat tak, aby se provedla její komprimace (zkrácení jejího rozsahu). Když třeba budeme chtít použít vstupní návěští se stejným názvem, který už obsahuje jiný vnitřní modul, provedeme výmaz modulu bez ušetření paměti tak, že pod stejným názvem přeložíme prázdný modul (bude obsahovat jen pseudoinstrukci END). Poslední modul bez názvu se automaticky vymaže na začátku jakéhokoli překladu. Základní modul knihovny SYSMOD vymazat nelze.

**SAV** - podobně jako při ukládání zdrojového textu zapíšeme tímto příkazem binární knihovnu na magnetofon. Před odesláním příkazu musí být magnetofon již spuštěn. Po skončení zápisu se opět přihlásí modul LIB. V posledních dvou bajtech hlavičky záznamu jsou zapsány nuly. Tak knihovna rozeznává své záznamy od jiných. Je-li v knihovně před zápisem binární knihovny přítomen modul bez názvu, vymaže se.

**LOA** - slouží k načtení binární knihovny z magnetofonu. Načtení se provede podobně jako v případě zdrojového textu. V případě, že by hrozilo přepsání zdrojového textu uloženého v paměti, je čtení záznamu přerušeno a objeví se chybové hlášení MEM FULL; po stisku

tlačítka ENTER se vrátíme do modulu MRS. Nová knihovna vždy nahradí starou.

**SUS, LUS** - viz část 3.5.

**ENT** - za tímto příkazem následuje návěští, které je v knihovně označeno jako vstupní bod. Od tohoto návěští se pak program spouští příkazem RUN. Pokud ENT není definován, příkaz RUN okamžitě vrátí řízení systému MRS.

**RUN** - příkaz spustí uživatelský program od adresy uvedené u příkazu ENT. Poslední instrukce RET v uživatelském programu vrátí řízení systému MRS.

**MON EDI ASM LNK ALD DBG** - ukončují práci programu LIB a aktivují příkazem zvolený modul.

## 6.2. Základní modul knihovny

Tento modul se jmenuje SYSMOD; v knihovně musí být vždy přítomen. Obsahuje různé programy systému MRS, označené jako vnější návěští. Uživatel tak nemusí vědět, na jakých absolutních adresách jsou moduly uloženy, ale určuje je názvy pomocí pseudoinstrukce EXT. Modul SYSMOD obsahuje tato vstupní návěští:

**MEMORY** - obsahuje první volnou adresu za knihovnou. Tato hodnota se inicializuje až při vstupu do ladicího programu.

**MEMTOP** - obsahuje první obsazenou adresu. MEMORY a MEMTOP umožňují práci s dynamickou pamětí. Např. když zadáme:

```
EXT  MEMORY,MEMTOP
LD   DE,(MEMORY)
LD   HL,(MEMTOP)
```

budeme mít v registrech DE a HL hranice volné paměti. Běžící program má k dispozici paměť v rozsahu od (DE) do (HL)-1.

**DCBN** - konverze řetězce dekadických číslic na šestnáctibitovou binární hodnotu.

| | | |
|---|---|---|
| vstup | HL | adresa řetězce |
| výstup | HL | adresa konce řetězce (řetězec končí libovolným znakem kromě číslic 0..9) |
| | DE | konvertovaná binární hodnota |
| | A | ukončovací znak |
| mění se | AF,DE,HL | |

**HXBN** - konverze řetězce hexadekadických číslic na šestnáctibitovou binární hodnotu.

| | | |
|---|---|---|
| vstup | HL | adresa řetězce |
| výstup | HL | adresa konce řetězce (řetězec končí libovolným znakem kromě 0..9, A..F) |
| | DE | konvertovaná binární hodnota |
| mění se | AF,DE,HL | |

**BNHX** - konverze osmibitové hodnoty na dvě hexadekadické číslice.

| | | |
|---|---|---|
| vstup | A | konvertované číslo |
| | HL | adresa uložení výsledku |
| výstup | HL | adresa za uloženým párem znaků |
| mění se | AF,HL | |

**BNDC**- konverze šestnáctibitové hodnoty na řetězec dekadických číslic.

| | | |
|---|---|---|
| vstup | BC | konvertované číslo |
| | HL | adresa uložení výsledku |
| výstup | HL | adresa za řetězcem dekadických číslic |
| mění se | AF,DE,HL | |

Číslo je uloženo jako řetězec ASCII znaků, předřazené nuly jsou vypuštěny.

**INKEY** - čtení jednoho znaku z klávesnice s automatickým opakováním. Modul čeká, dokud není stisknuto nějaké tlačítko. Po opětovném

volání akci automaticky opakuje - podržení tlačítka má tedy stejný účinek jako jeho opětovný stisk, který je provázen zvukovým signálem.

vstup -
výstup A ASCII kód stisknutého tlačítka
mění se AF

**SCAN** - zjištění, zda a jaké tlačítko je stisknuto

vstup -
výstup A A=0, když není stisknuto nic nebo jde o "nelegální" kombinaci tlačítek; jinak reg. A obsahuje ASCII kód znaku
mění se AF

**CLEAR** - výmaz obrazovky

vstup -
výstup -
mění se AF,BC,DE,HL

**OUTCHR** - vypsání znaku na pozici "pomyslného" kurzoru a jeho posun doprava (když je kurzor na konci řádky, přejde na novou, když je na konci obrazovky, ta se posune o 1 řádku nahoru). Kurzor se na obrazovce neobjevuje (to je zabezpečeno modulem SETCRS), jen se modifikuje proměnná CURSOR. OUTCHR správně ošetří tyto řídící znaky:

19H - posun kurzoru vlevo
18H - posun kurzoru vpravo
0DH - ukončení řádky, přechod na novou (CR)

vstup A kód znaku
výstup -
mění se AF

**CURSOR** - adresa pozice kurzoru na obrazovce (pro modul OUTCHR). První z obou bajtů obsahuje číslo sloupce (0..31), druhý číslo řádky (0..23).

**SETCRS** - kurzor v podobě blikajícího čtverečku se na svém původním místě vymaže a objeví se na pozici určené proměnnou CURSOR.

| | | |
|---|---|---|
| vstup | HL | nová pozice kurzoru (řádka v H, sloupec v L) |
| výstup | - | |
| mění se | AF | |

**MRS** - operace s dialogovou řádkou. Modul vypíše požadovaný text v dialogové řádce, načte vstupní text a podle požadavku provede analýzu tříznakových příkazů. Při volání tohoto modulu máme k dispozici všechny funkce uvedené u popisu modulu MRS. Tzn., že pro editaci textu lze využít tlačítka CAPS SHIFT/5,8,9,0 a pro odeslání textu tlačítko ENTER.

| | | |
|---|---|---|
| vstup | A | pozice (0..31) v dialogové řádce, od níž se má vypsat požadovaný text |
| | HL | adresa vypisovaného textu |
| | B | délka vypisovaného textu |
| | C | při C=0 se nic nečte, jinak se čte text z klávesnice a ukládá se do vyrovnávací paměti (**buffer**) od adresy FE00H |
| | DE | adresa tabulky tříznakových příkazů; když (DE)=0, načtený text se neanalyzuje |

V tabulce následuje po sobě vždy tříznaková zkratka příkazu a adresa obslužného programu. Tabulka je zakončena binární nulou. Když je analýza požadována, modul nevrátí řízení, dokud uživatel nenapíše řádku začínající dovoleným příkazem.

| | | |
|---|---|---|
| výstup | B | počet načtených znaků bez 0DH (ENTER) |
| | DE | adresa načteného řetězce; pokud byla požadována analýza tříznakových příkazů, v DE je adresa prvního znaku za příkazem |
| | HL | obsahuje adresu příslušného podprogramu jen v případě požadované analýzy |
| mění se | AF,BC,DE,HL | |

**MRS2** - umožňuje opravy textů načtených modulem MRS. Musí mu předcházet volání modulu MRS se čtením řetězce. Modul MRS2 se musí volat instrukcí JP MRS2!

| | |
|---|---|
| vstup | použije informace získané od modulu MRS |
| výstup | stejný jako u modulu MRS |
| mění se | AF,BC,DE,HL |

**SCHENT** - vyhledává vstupní bod v knihovně. Dá se využít např. tehdy, když jeden program produkuje data, která má zpracovat jiný modul knihovny. Pomocí SCHENT může daný program najít tabulku v jiném modulu, do něhož má uložit výstupní data.

| | | |
|---|---|---|
| vstup | HL | adresa řetězce obsahujícího požadované vstupní návěští; řetězec musí končit nulou |
| výstup | Z | při Z=1 je chybná syntaxe nebo v knihovně není zadaný název |
| | HL | při Z=0 obsahuje adresu názvu; v knihovně na této adrese je 6 znaků názvu a 2 bajty adresy, kterou tento název označuje (např. adresa tabulky, jejíž název je označen jako ENT) |
| mění se | AF,BC,DE,HL | |

**PNTCHR** - pro tisk jednoho znaku na tiskárně (viz část 4.5.).

| | | |
|---|---|---|
| vstup | A | kód tištěného znaku |
| výstup | - | |
| mění se | AF | |

**INVPIX** - invertuje znak na obrazovce.

| | | | |
|---|---|---|---|
| vstup | H | řádka | (0..23) |
| | L | sloupec | (0..31) |
| výstup | - | | |
| mění se | AF | | |

**II** - seznam nestandardních instrukcí (viz část 7.4.).

Využití modulu SYSMOD může zpočátku působit těžkosti, protože se s ním v obdobných systémech nesetkáte. Je to však velmi pohodlný a silný prostředek, jak dokumentuje následující příklad:

Chceme vymazat obrazovku, cyklicky načítat znak a zobrazit jej na obrazovce, dokud není načten znak '!'.

```
        EXT     CURSOR,OUTCHR,INKEY,CLEAR
START   CALL    CLEAR
        LD      HL,0
        LD      (CURSOR),HL
LOOP    CALL    INKEY
        PUSH    AF
        CALL    OUTCHR
        POP     AF
        CP      '!'
        JR      NZ,LOOP
        RET
        END
```

Po překladu (příkazem ASM) a spojení (příkazem LNK) je program připraven k práci. Vidíme, že se nemusíme starat o adresy daných modulů a program nemusíme modifikovat, i když se systém MRS změní. Odpovídajícím způsobem se totiž změní i modul SYSMOD.

Pokud budeme chtít později odstranit závislost programu na systému MRS, příkazem DIS modulu EDI uložíme příslušné části kódu programu MRS, jak bylo popsáno výše.

# 7. LADICÍ PROGRAM

## 7.1. Úvod

Ladicí program se aktivuje příkazem DBG bez parametru. Návrhu modulu DBG byla věnována velká pozornost, protože ladění programu je nejčastější činností programátora. Po spuštění vypíše modul DBG do spodních dvou řádek obrazovky informaci o sledovaných stavech (stavovou informaci):

**X PCPC INSTRUKCE SZAPC**
**AA BBCC DDEE HHLL XHXL YHYL SPSP**

**X** - jednoznakový status poskytující informaci o stavu ladicího programu

**PCPC** - hodnota reg. PC

**INSTRUKCE** - assemblerový výpis instrukce uložené na adrese uvedené u reg. PC. Hodnota operandu je vypsána hexadekadicky. Instrukce relativního skoku mají v adresové části cílovou adresu, nikoli odchylku vůči obsahu reg. PC.

**SZAPC** - stavové indikátory (**Sign, Zero, Auxilliary Carry, Parity/Overflow, Carry**). Přítomnost příslušného znaku na obrazovce znamená, že indikátor je ve stavu log.1. V opačném případě se na jeho místě objeví znak '-'.
*(Pozn. překl.: Místo Auxilliary Carry se běžněji používá název Half Carry, symbolizovaný písmenem H.)*

**AA** - hodnota reg. A
**BBCC** - hodnota reg. BC
**DDEE** - hodnota reg. DE
**HHLL** - hodnota reg. HL
**XHXL** - hodnota reg. IX
**YHYL** - hodnota reg. IY
**SPSP** - hodnota reg. SP

Obsahy všech registrů jsou zobrazovány hexadekadicky.

I když se stavová informace promítá do spodních dvou řádek obrazovky, je možné zajistit i výpis dialogové řádky (viz dále).

## 7.2. Zadávání hodnot pro ladicí program

Když chceme zadat nějakou hodnotu (nastavit velikost nějakého registru, definovat adresu paměti apod.), máme k dispozici tyto možnosti:

- dekadickou konstantu,
- hexadekadickou konstantu,
- návěští,
- libovolnou kombinaci uvedených tří prvků spojených operacemi + nebo - (ve tvaru a + b či a-b). Když návěští označuje vstupní bod v knihovně, musí být uvedeno jako druhý člen výrazu.

Na syntaxi zápisu jsou kladeny stejné požadavky jako při užívání symbolů v modulu EDI. Syntakticky nesprávný zápis je odmítnut zvukovým signálem; poté jej můžeme opravit.

Když jako součást výrazu uvedeme návěští, ladicí program nejdříve prohlédne tabulku návěští posledně překládaného zdrojového textu, a když tam zadané návěští nenajde, prohledá tabulku vstupních bodů v binární knihovně. Když ani tam návěští není, považuje příkaz za chybný a nabídne nám jej k opravě. Zadaná veličina získá hodnotu, která je označena návěštím. Např. když se v přeloženém programu vyskytuje návěští START, nastavíme registr PC na toto návěští (tak do něj vložíme adresu symbolizovanou návěštím START) příkazem RP START (RP znamená Registr PC, RB je BC apod.).

Když chceme na obrazovce vidět zpětný překlad knihovního modulu CLEAR, zadáme A CLEAR (samozřejmě, že v tomto případě by se v posledně přeloženém zdrojovém textu nemělo návěští CLEAR vyskytovat ještě jednou).

Uvedená vlastnost je velmi užitečná, zvláště když nemáme k dispozici vytištěný protokol překladu. Není nutné znát absolutní adresy laděného programu ani se nemusíme starat o to, že po každé opravě se program v paměti může přesunout. Po každém překladu se návěštím přiřadí aktuální hodnota a uživatel příslušná návěští označuje jen jejich symbolickým názvem.

Při zadávání příkazů modulu DBG se využívají stejné prostředky jako při práci s dialogovou řádkou v ostatních modulech.

Když odešleme prázdnou řádku, příslušná veličina se nezmění. Např. když po RP stiskneme ENTER, obsah reg. SP se nezmění.

## 7.3. Příkazy ladicího programu

Tyto příkazy se zadávají stiskem jednoho tlačítka. Po napsání příslušného znaku se příkaz vykoná ihned, není nutné jej potvrzovat stiskem tlačítka ENTER. Pak už ovšem nelze opravit chybně zadaný příkaz. Po napsání příkazu očekává modul DBG zpravidla nějakou hodnotu, kterou zadáváme podle zásad uvedených v části 7.2.

Dále je uveden popis jednotlivých příkazů; způsob jejich použití bude vysvětlen v části 7.4.

R - příkaz pro nastavení hodnot registrů. Systém po jeho zadání očekává název registru, tj. jeden ze znaků P, A, B, D, H, X, Y, S. Když napíšeme jiný znak, ozve se výstražný signál a systém čeká na nový příkaz (musíme jej zadat celý znova). Po správně zadaném příkazu se v dialogové řádce objeví hlášení:

RN:

kde N je název registru, jehož hodnotu zadáváme. Musíme vždy zadat hodnotu šestnáctibitovou (např. pro celý pár BC, nikoli jen pro reg. B nebo C). Jestliže hodnotu nezadáme a stiskneme ENTER, obsah žádného registru se nezmění.
Když změníme obsah reg.PC, automaticky se změní i obsah reg.SP na hodnotu, kterou měl při spuštění modulu DBG.

I - příkaz pro nastavení bodu přerušení. Ladicí program očekává adresu zadanou podle části 7.2. a na tuto adresu vloží bod přerušení. Pak do dialogové řádky vypíše hlášení:

N:

a čeká na zadání dekadického čísla, které uvádí, kolikrát musí program tímto bodem projít, aniž se přeruší chod laděného programu.

Pokud hodnotu nezadáme, běh programu se přeruší ihned, jakmile na bod přerušení narazí. Když např. zadáme N:1000, program projde bodem 1 000krát bez zastavení, ale při 1001. setkání s ním se zastaví.
Při každém průchodu bodem přerušení proběhne test, zda není stisknuta tlačítková kombinace CAPS SHIFT/SPACE. Pokud tomu tak je, chod programu se rovněž přeruší.
Po přerušení programu je řízení odevzdáno modulu DBG, který do spodních dvou řádek vypíše stavovou informaci a bude čekat na další příkaz. Zároveň se objeví jednoznakový status B (BREAK), který signalizuje, že laděný program se zastavil vlivem bodu přerušení. Obnovit chod programu můžeme buď zrušením bodu přerušení (příkaz O), nebo provedením jedné instrukce (příkaz S). Ve druhém případě bod přerušení zůstává na svém místě s hodnotou N:0. Proto se na tomto místě běh programu opět zastaví. Hodnotu N můžeme změnit novým zadáním příkazu I.
Zadat lze jen jeden bod přerušení. Zadáním dalšího automaticky zrušíme předchozí.
Bod přerušení je realizován instrukcí RST 10H, jíž se přepíše instrukce na adrese bodu přerušení. Protože DBG využívá systémovou proměnnou CURCHL ZX Spectra na adrese 5C51H, měl by se uživatel její změně vyhnout. Systém v bodu přerušení píše do paměti, nelze tedy takový bod zadat v paměti ROM.
Využití bodu přerušení je zřejmé - sporné místo v programu můžeme označit třeba návěštím BREAK, program opět přeložit a před jeho spuštěním nastavit příkazem I BREAK bod přerušení do tohoto místa. V něm se program zastaví a my se můžeme podívat na obsahy registrů, paměti, ev. provést změny jejich obsahu atd.

O - příkaz pro zrušení bodu přerušení

G - příkaz ke spuštění laděného programu. Registry se inicializují na hodnoty, které vidíme ve výpisu stavové informace, a program odstartuje. Pokud program úspěšně proběhne až po svou poslední instrukci návratu, řízení se vrátí modulu DBG. Reg. PC se po ukončení běhu programu inicializuje na hodnotu, jakou měl v momentu jeho spuštění příkazem G, takže program můžeme tímto příkazem zase spustit.

**CAPS SHIFT/G**- příkaz má stejnou funkci jako příkaz G, ale s tím rozdílem, že chod programu se přeruší při prvním setkání s bodem přerušení bez ohledu na to, jaký počet přerušení N byl zadán příkazem I.

Po zadání příkazů G a CAPS SHIFT/G je maskovatelné přerušení zablokováno. Pokud je potřebujeme uvolnit, musíme do laděné části programu vložit instrukci EI. Po návratu do modulu DBG se přerušení opět zablokuje. Při uvolněném přerušení je nutné, aby reg. IY obsahoval hodnotu 5C3AH (na kterou je modulem DBG inicializován).

**S** - příkaz pro provedení jedné instrukce (jednoho kroku) na adrese uvedené u reg. PC. Vždy poté se řízení vrací modulu DBG. Jednoznakový status bude obsahovat písmeno S (angl. **Step** - krok), indikující krokování programu. Když během krokování narazíme na bod přerušení, objeví se ve statusu písmeno B.

Každá instrukce je před svým provedením analyzována, zda neporušuje některou podmínku, kterou stanovil uživatel. (viz část 7.4.). Pokud instrukce nevyhovuje stanoveným omezením, neprovede se, zazní výstražný signál a na místě statusu se objeví znak symbolizující příčinu odmítnutí instrukce. Analýza instrukce s opakováním (LDIR apod.) může trvat až 20 sekund. Analyzování však můžeme vypnout (viz další příkaz).

Při krokování se laděný program nemodifikuje, proto můžeme krokovat i v paměti ROM.

**CAPS SHIFT/S** - příkaz se stejnou funkcí jako má příkaz S s tím rozdílem, že neprobíhá analýza podmínek stanovených pro provádění instrukcí.

**C** - rovněž příkaz podobný příkazu S. Jeho zvláštností je, že za právě prováděnou instrukci vloží vnitřní bod přerušení. To je výhodné hlavně tehdy, když při krokování narazíme na instrukci CALL. Nemusíme pak ztrácet čas krokováním podprogramu, který tato instrukce volá; ten se provede celý normálním způsobem - bez krokování. Po návratu z něj se běh programu zastaví v bodu vnitřního přerušení, tj. na instrukci hned za CALL. Pak můžeme v krokování zase pokračovat. Není tedy nutné, abychom krokovali podprogramy romky a ty, které už máme v našem programu odladěné. Musíme si

ovšem být jisti, že se program za instrukci CALL opravdu vrátí. Rovněž by bylo zbytečné vkládat vnitřní bod přerušení do definovaného bajtu, který se za CALL může vyskytnout apod.
Vnitřní bod přerušení nesouvisí s bodem přerušení, který vytvoří příkaz I - kromě případu, kdy je takový bod přerušení nastaven v programové části prováděné normálním způsobem. Tehdy se běh programu v definovaném bodě přeruší, ale už se nezastaví po návratu do bodu vnitřního přerušení.
Stejně jako u příkazu S je každá instrukce předem analyzována.

**CAPS SHIFT/C** - příkaz má stejnou funkci jako příkaz C, ale neprovádí se analýza instrukcí.

**CAPS SHIFT/J** - příkaz pro přeskočení jedné instrukce. Instrukce adresovaná registrem PC se neprovede a obsah PC se zvýší tak, aby adresoval instrukci následující. Přeskočená instrukce se neanalyzuje. Tento příkaz můžeme použít např. tehdy, když se budeme chtít vyhnout provedení nějaké subrutiny (přeskočíme instrukci CALL) apod.

T - příkaz pro provádění programu ve sledovacím režimu (angl.**Tracing** - trasování). Modul DBG analyzuje každou instrukci před jejím provedením a každá přípustná instrukce je ihned provedena. Při sledování se instrukce ve stavové informaci nevypisují, protože bychom je nestačili číst. Ostatní informace jsou však průběžně vypisovány. Pokud některá z instrukcí poruší podmínky, které jsme stanovili, provádění programu se přeruší a na místě statusu se objeví znak charakterizující příčinu přerušení programu. Také v tomto režimu se program zastaví, jestliže narazí na bod přerušení. Sledovací režim je vlastně zautomatizované krokování, kterého můžeme jinak dosáhnout též opakovaným zadáváním příkazu S.

N - příkaz pro provádění programu ve sledovacím režimu, v jehož průběhu se nevypisuje žádná stavová informace. Díky tomu je toto sledování podstatně rychlejší než při použití příkazu T. Jinak platí pro příkaz N stejné podmínky jako pro příkaz T.

W - příkaz pro nastavení paměťových oken a oken pro reg. PC. Po zadání příkazu je očekáván buď znak M, jímž se definuje paměťové okno, nebo znak P pro definování okna pro reg. PC. Když napíšeme jiný

znak, program se vrátí do výpisu stavové informace. Jinak se do spodních dvou řádek vypíší čtyři páry hexadekadických čísel - každá dvojice vymezuje rozsah jednoho okna. Pokud chceme některé z čísel změnit, zadáme pořadovou číslici v rozsahu 1..8, označující, které číslo chceme změnit. Při zápisu jiného znaku se program vrátí do výpisu stavové informace. Jinak očekává vložení nové hodnoty zvoleného čísla. Hodnotu zadáme podle zásad uvedených v části 7.2. Poté se program vrátí do výpisu stavové informace.
Použití oken je vysvětleno v části 7.4.

P - tímto příkazem určíme paměťový úsek, jehož obsah se zobrazí vždy, když se obsah některé adresy v tomto úseku změní. Po zadání příkazu se v dialogové řádce objeví zpráva:

**P:**

a systém čeká na vložení hodnoty podle části 7.2. Hodnota P určuje první adresu zobrazované části paměti. Poté se objeví zpráva:

**L:**

K ní zapíšeme dekadické číslo určující počet obrazovkových řádek, které se rozhodneme rezervovat pro výpis obsahu úseku paměti. Tím určíme délku úseku - na jedné řádce se vypisují obsahy 4 adres. Když po L: stiskneme ENTER, funkce zobrazování úseku se zruší.
Paměťový úsek se pochopitelně nezobrazuje po aktivaci příkazu G, ale jen při příkazech S, C, T a N.

M - příkaz pro modifikaci obsahu paměti. Po jeho volbě se objeví zpráva:

**M:**

Poté zadáme adresu podle pravidel uvedených v části 7.2. V dialogové řádce se vypíše obsah 4 adres, počínaje adresou zadanou. Výpis je hexadekadický i znakový.

**Obsah adres můžeme modifikovat pomocí těchto tlačítek:**

**0-9,A-F** - vloženou hexadekadickou číslicí se přepíše hodnota, na kterou ukazuje kurzor. Ten se pak posune na další číslici. Mezery mezi nimi automaticky přeskakuje. V případě potřeby lze vypsanou řádku posunout doleva i doprava a tak zobrazit obsah další adresy. V každém případě se číslo adresy na začátku řádky vztahuje k prvnímu zobrazovaného bajtu.

**CAPS SHIFT/5** - posun kurzoru na předcházející znak vlevo. Mezery se automaticky přeskakují. Když je kurzor na prvním znaku výpisu, text řádky se posune doprava a na místě kurzoru se objeví hodnota bajtu z nižší adresy.

**CAPS SHIFT/8** - posun kurzoru na další znak vpravo. Mezery se automaticky přeskakují. Když je kurzor na posledním znaku výpisu, text řádky se posune doleva a na místě kurzoru se objeví hodnota bajtu z vyšší adresy.

**CAPS SHIFT/1** - zobrazení obsahu nižších 4 adres.

**CAPS SHIFT/4** - zobrazení obsahu vyšších 4 adres.

**CAPS SHIFT/SYMBOL SHIFT** - bajty, které byly modifikovány na obrazovce, zůstanou v původním stavu. Do paměti se zapíše čtveřice bajtů teprve tehdy, když posouváním textu v řádce zmizí některý řádek z obrazovky.

**ENTER** - modifikovaná čtveřice bajtů se z obrazovky uloží do paměti a modul DBG se vrátí do výpisu stavové informace.

**SPACE** - je akceptován pouze tehdy, je-li kurzor na první číslici obsahu adresy. Po stisku tlačítka SPACE můžeme zapsat přímo do paměti jeden libovolný znak, jako bychom psali normální text. Po vložení znaku se kurzor posune na první číslici obsahu další adresy, kde tento postup - stisknutí tlačítka SPACE a vložení znaku - můžeme opakovat.

Pokud stisknete jiné než povolené tlačítko, odpoví vám systém varovným zvukovým signálem.

U příkazu M můžeme využít toho, že odesláním prázdného příkazu se původní hodnota příkazu nemění (jednou vloženou hodnotu příkazu program zachovává až do její změny). Např. jestliže jsme jednou zadali příkaz M:4000, pak když příště zvolíme příkaz M a budeme opět chtít zobrazit obsah adres od 4000H, nemusíme již tuto adresu vepisovat, ale stačí po M stisknout ENTER a výpis paměti se zobrazí zase od adresy 4000H.

**D** - příkaz ke zobrazení paměti. Po jeho volbě se zobrazí zpráva:

**D:**

Nyní zapíšeme adresu podle zásad uvedených v části 7.2. Poté se zobrazí obsah 64 adres od adresy zadané.

**Další volitelné funkce:**

| | | |
|---|---|---|
| **CAPS SHIFT/6** | - | zobrazení obsahu vyšších 64 adres |
| **CAPS SHIFT/7** | - | zobrazení obsahu nižších 64 adres |
| **A** | - | vložení a následné hledání vloženého řetězce znaků |
| **H** | - | vložení a následné hledání řetězce hexadekadických konstant |
| **ENTER** | - | návrat k výpisu stavové informace |

Stisk jiného tlačítka se ohlásí jako chyba.

Když zadáme příkaz A (resp.H), v dialogové řádce textu se vypíše zpráva:

**a:** (resp. **h:**)

Dále vložíme řetězec znaků, resp. hexadekadických čísel. Při opravě zápisu můžeme použít tlačítka CAPS SHIFT/0 (DELETE) pro výmaz znaku. Tlačítkem ENTER zadávání ukončíme.

DBG začne hledat zadaný řetězec v paměti. Jakmile ho najde, zobrazí obsah 64 adres od adresy obsahující první znak řetězce. Poté čeká na další příkaz. Pokud není řetězec nalezen, ozve se signál. DBG prohledává paměť od adresy zadání řetězce směrem k vyšším adresám; když dojde na adresu FFFFH, pokračuje od adresy 0.

Jestliže po příkazu A (resp. H) stiskneme jen ENTER, DBG vyhledá posledně zadaný řetězec.

A - příkaz pro výpis zpětného překladu. V dialogové řádce se objeví zpráva:

A:

K ní zadáme adresu podle části 7.2. Počínaje touto adresou se zobrazí zpětný překlad (assemblerový výpis) 21 řádek textu. Při dalším stisku tlačítka A se vypíše další řádka přeloženého textu. ENTER nás vrátí k výpisu stavové informace. Adresová část instrukce relativního skoku (JR) obsahuje absolutní adresu, nikoli odchylku.

X - příkaz pro výměnu sad registrů. Provedou se instrukce X AF,AF' a EXX.

L - příkaz pro posun (kopii) obsahu dialogové řádky do vyšších řádek obrazovky. Po jeho zadání se objeví zpráva:

N:

Poté zadáme číslo, jímž určíme, kolik dvouřádek obrazovky má být pro posouvání poskytnuto. Potom se před každým výpisem nové stavové informace na spodních dvou řádkách posune předchozí výpis o dvě obrazové řádky nahoru. Tzn., že předchozí stavová informace nezmizí, ale jen se posune výš a uvolní místo informaci aktuální. Po příkazu L v režimu T se zobrazuje i assemblerový výpis instrukce. Pomocí příkazů T a L tak můžeme na obrazovce sledovat i výpis instrukcí prováděných před přerušením programu.
Když zadáme N:0, stavová informace se zobrazuje jen na spodních dvou řádkách, ale ve sledovacím režimu se vypisuje i assemblerový tvar instrukce. Jestliže po zprávě N: stiskneme pouze ENTER, funkce L se zruší.

Q - návrat z/do řídicího modulu MRS

## 7.4. Ladění programů

Při hledání chyb v programu nad ním především nesmíme ztratit kontrolu. Obecně o ni můžeme přijít třemi způsoby:

1) přepsáním paměti,
2) provedením některé instrukce s nepříjemnými následky (např.HALT),
3) zacyklením programu.

Dobrý ladicí program tedy musí poskytovat prostředky na ochranu části paměti před přepsáním, zastavit program před provedením nějaké instrukce, která by nás zbavila kontroly, a umožnit přerušení běhu programu stiskem nějakého tlačítka. DBG toho dosahuje následujícími způsoby:

Ad 1) Ve sledovacím nebo krokovacím režimu (příkazy T, N, S, C) nelze přepsat paměť na adrese, která je uvnitř zadaného paměťového okna (viz příkaz W). Když máme nějaké paměťové okno nastaveno na určitý interval, každý pokus o zápis do tohoto prostoru paměti ve sledovacím nebo krokovacím režimu přeruší chod programu. Poté se vypíše stavová informace se statusem M, který oznamuje, že došlo k pokusu modifikovat chráněnou oblast paměti.

Někdy je však třeba něco zapsat i do chráněné paměti. DBG to umožňuje pomocí příkazů CAPS SHIFT/S (CAPS SHIFT/O), které provedou jednu instrukci bez kontroly.

Tak např. zjistíme, že do proměnné pod názvem TEMP se zapisuje nesprávná hodnota, ale nevíme, kdy a odkud. Příkazy W.M3TEMP a W:M4TEMP nastavíme druhé paměťové okno tak, aby chránilo proměnnou TEMP před přepsáním (pořadová čísla 3 a 4 výpisu hodnot oken se vztahují k 1. a 2. adrese druhého okna ve výpisu). Poté program spustíme ve sledovacím režimu příkazem N. Při každém pokusu o zápis do adresy TEMP se běh programu zastaví a my můžeme provést analýzu problému. Když je zápis v pořádku, povolíme provedení zastavené instrukce příkazem CAPS SHIFT/S a program opět spustíme příkazem N.

Když program přeložíme a spustíme příkazem ALD, je první paměťové okno automaticky inicializováno tak, aby chránilo zdrojový text našeho programu a systém MRS.

Podobnou funkci jako paměťová okna mají okna pro reg. PC. Definujeme je příkazem WP. Jejich použití není tak kritické jako je tomu v případě oken paměťových, protože DBG umí sledovat a krokovat i programy v paměti ROM - můžeme tedy okna pro reg. PC určit i v ní.

Okna reg. PC jsou velmi užitečná při definici dalších bodů či oblastí přerušení chodu programu. Takováto přerušení jsou ovšem možná jen ve sledovacím režimu.

První okno pro reg. PC je inicializováno v intervalu 0-FFFFH; reg. PC tedy může nabývat jakékoli hodnoty.

**Ad 2)** V základním modulu knihovny SYSMOD je 16 bajtů rezervováno pro definici zakázaných instrukcí (takových, jež nemají být provedeny), které určí sám uživatel. Když program spustíme jedním z příkazů S, C, T, N a DBG narazí v průběhu programu na zakázanou instrukci, program se přeruší a vypíše stavovou informaci se statusem I - tak jsme informováni o výskytu zakázané instrukce.

Chceme-li přesto zakázanou instrukci provést, stiskneme CAPS SHIFT/S (CAPS SHIFT/O).

Zmíněných 16 adres v modulu SYSMOD je pod vstupním bodem II. Definici zakázaných instrukcí provedeme po příkazu M:II. Pak zapíšeme do tabulky jednobajtové operační kódy pro každou zakázanou instrukci (instrukce s prefixem EDH se zadávají dvěma bajty: prefix a operační kód). Tabulka musí být ukončena nulou (proto nemůžeme definovat jako zakázanou instrukci NOP, která má kód 00H, ).

Jako zakázané tedy můžeme definovat např. všechny instrukce operující s reg. IY (operační kód FDH) nebo jednu konkrétní instrukci, jako je třeba EXX (operační kód D9H) či instrukci LDIR (dva bajty operačního kódu za sebou - EDH, B0H).

Podobným způsobem jako zakázané instrukce se vyhodnocují kombinace bajtů, které nejsou obsaženy v instrukčním souboru Z80 (např. sled bajtů DDH, FDH). Rovněž v tomto případě se provádění programu zastaví, ale při provedení jedné instrukce příkazem CAPS SHIFT/S se první bajt nedovolené kombinace přeskočí.

V základním modulu SYSMOD je jako zakázaná definována instrukce HALT (operační kód 76H), protože modul DBG běží se zablokovaným přerušením (viz výše).

**Ad 3)** Před provedením každé instrukce ve sledovacím režimu (příkazy T a N) DBG zjišťuje, zda je stisknuta kombinace tlačítek CAPS

SHIFT/SPACE (BREAK). Když ano, běh programu se zastaví a na obrazovce se vypíše stavová informace. Dále můžeme pokračovat krokováním programu nebo jeho spuštěním příkazy N, T, G.

Definováním bodu přerušení s velkým počtem průchodů (příkaz I) můžeme dosáhnout toho, že program spuštěný příkazem G budeme moci přerušit tlačítky CAPS SHIFT/SPACE (BREAK) - ovšem pouze v okamžiku, kdy bude program při stisku tlačítek procházet bodem přerušení.

# 8. PŘÍKLAD NA ZÁVĚR

V této kapitole si ukážeme použití strojového kódu a systému MRS na uceleném příkladě. Při jeho výběru jsem se snažil přihlédnout k několika protichůdným kritériím. Program by měl přinést praktický užitek, měl by dokumentovat výhody strojového kódu oproti Basiku, měl by být dostatečně jednoduchý, ale i přiměřeně složitý.

Po delší úvaze jsem nakonec vybral program na třídění textových řetězců podle abecedy. Tento problém se vyskytuje poměrně často a třídění v jazyce Basic trvá nesnesitelně dlouho. Existují různé algoritmy, ale nejvyššího zrychlení dosáhneme, pokud modul napíšeme ve strojovém kódu. V textu je několik úmyslných chyb, abych mohl ukázat postup jejich odhalování a oprav.

Vývoj každého programu probíhá ve třech etapách:
- návrh algoritmu,
- jeho zápis ve zvoleném programovacím jazyce,
- napsání a odladění programu na příslušném počítači.

Naším cílem je vytvořit modul, který bude možné volat z jazyka Basic a který bude třídit údaje v tomto jazyce vytvořené. Předpokládáme, že jsme definovali pole řetězců příkazem DIM A$(N,M), čímž je definováno pole N textových řetězců, z nichž každý má délku M znaků. Podle informací z příručky pro ZX Spectrum (kapitola 24), se proměnné v paměti Spectra ukládají od adresy, jejíž hodnota je uložena na adrese VARS = 23627. Bez újmy na obecnosti můžeme předpokládat, že proměnná A$(N,M) je uložena jako první; stačí, když ji jako první definujeme v našem programu. Z tvaru uložení této proměnné, jak je popsán v kapitole 24, vyplývá, že na adrese (VARS)+4 je ve dvou bajtech uložena hodnota N = počet řetězců a na adrese (VARS)+6 je ve dvou bajtech uložena hodnota M = délka řetězce. Připomínám, že zápisem (VARS) označuji dvoubajtovou hodnotu uloženou na adrese VARS.

Pro třídění řetězců použijeme velmi jednoduchý algoritmus. Postupně procházíme množinu řetězců a porovnáváme vždy dva sousední. Přitom využijeme skutečnost, že ASCII kód znaku je přímo úměrný svému pořadí v abecedě; porovnáváme vlastně numerické hodnoty znaků na stejných pozicích v porovnávané dvojici řetězců. První dvojice odlišných znaků určí zároveň i pořadí řetězců v abecední posloupnosti.

Jakmile při porovnávání dvojice řetězců zjistíme inverzi (narušení abecední posloupnosti), tyto dva řetězce navzájem vyměníme, poznamenáme si tuto skutečnost do nějaké proměnné a pokračujeme v porovnávání následující dvojice řetězců. Když projdeme celou množinu řetězců, zkontrolujeme, zda jsme v daném průchodu provedli nějakou výměnu. Pokud ano, zopakujeme celý cyklus znovu, v opačném případě jsou řetězce setříděny.

Tento velmi jednoduchý postup doplníme jedním trikem, který náš program trochu zkomplikuje, ale třídění především delších řetězců se významně zrychlí. Pro dané řetězce si vytvoříme seznam adres, na kterých jsou umístěny. Procházet budeme potom tento seznam adres a namísto výměny dvou řetězců (což mohou být i stovky bajtů), budeme vyměňovat jen dvoubajtové adresy. V průběhu třídění se tedy vyměňují jen tyto adresy a na konci potom obsahují informaci o setřídení řetězců. I-tá adresa v seznamu adres je vlastně adresa i-tého řetězce v abecedním třídění. Na základě těchto informací vyměňujeme řetězce v poli A$(N,M). Pokud nechceme pro setříděné řetězce rezervovat nové místo v paměti, musíme si trochu namáhat mozek a vymyslet algoritmus, jak bychom je mohli vyměňovat v původním poli s využitím jen jednoho pomocného řetězce. Tato část algoritmu je z celého programu nejsložitější; doporučuji čtenáři, aby se problém pokusil pochopit a vyřešit samostatně a v četbě dalšího textu aby pokračoval až potom.

Body 1 - 4 se opakují pro i = 1 až N-1, kde N je počet řetězců.

1) Vezmu i-tou hodnotu ze seznamu adres; tato hodnota tedy udává adresu řetězce, který je na i-té pozici v abecedním třídění.
2) Řetězec na této adrese vyměním s i-tým řetězcem v poli A$(N,M).
3) Ve zbylých adresách ze seznamu adres, tedy na pozicích i + 1 až N, najdu adresu i-tého řetězce z pole A$(N,M).
4) Místo této hodnoty zapíši i-tou hodnotu ze seznamu adres. Tímto způsobem seznam adres zachytí stav, který vznikl v poli A$(N,M) výměnou dvojice řetězců.

Nesmíme zapomenout: vždy musí platit, že hodnota na i-té pozici v seznamu adres musí udávat adresu i-tého řetězce v abecedním třídění.

Napišme nyní uvedený algoritmus v assembleru procesoru Z-80. Hlavní program se skládá ze tří částí a jedné samostatné procedury pro porovnávání řetězců. Jako pracovní paměťové buňky potřebujeme proměnné FLAG, LENGTH, COUNT a ADDSTR, které definujeme pomocí příkazů

překladače DS. Proměnné v jazyce Basic jsou uloženy od adresy, jejíž hodnota je na adrese VARS. Seznam adres budeme vytvářet na začátku volné paměti, adresa volné paměti je na adrese RAMTOP; jako pracovní oblast použijeme paměť využívanou pro práci s tiskárnou (je na adrese 23296 a má délku 256 bajtů; na tento rozměr tedy omezíme délku tříděných řetězců). Dále budeme psát vlastní text programu velkými písmeny; pokud jej čtenář bude chtít přepsat do Spectra, musí použít písmena malá , protože systém MRS rozeznává pouze malá písmena.

Úvod našeho programu tedy může vypadat takto:

```
;
;SSORT      USPORADANI RETEZCU V POLI A$(N,M) PODLE ABECEDY
;
;VSTUP:     POLE A$(N,M), KTERE MUSI BYT DEFINOVANO JAKO PRVNI
;           PROMENNA V PROGRAMU V JAZYCE BASIC, DELKA RETEZCE
;           (HODNOTA M) MAXIMALNE 256
;
;VYSTUP:    RETEZCE V POLI A$(N,M) JSOU SERADENY PODLE ABECEDY
;
TMPBUF     EQU    23296     ADRESA PRACOVNI PAMETI
VARS       EQU    3627      ADRESA ADRESY PROMENNYCH
RAMTOP     EQU    23730     ADRESA ADRESY VOLNE PAMETI
FLAG       DS     1         INDIKATOR VYMENY RETEZCU
LENGTH     DS     2         DELKA RETEZCE
COUNT      DS     2         POCET RETEZCU
ADDSTR     DS     2         ADRESA ULOZENI RETEZCU
```

Napišme nejprve podprogram pro porovnání řetězců s názvem CMPS. Vstupem do podprogramu jsou adresy dvou porovnávaných řetězců ve dvojicích registrů BC a DE, délka řetězce je na adrese LENGTH a výstup podprogramu bude v příznaku Cy.

```
;CMPS       PODPROGRAM PRO POROVNANI DVOJICE RETEZCU
;
;VSTUP:     <BC>,<DE> - ADRESY RETEZCU
;VYSTUP:    Cy = 1 RETEZEC NA ADRESE <DE> JE V ABECEDNI
           POSLOUPNOSTI ZA RETEZCEM NA ADRESE <BC>
;MENI SE:   <AF>
```

```
;
CMPS    PUSH    HL              ULOZENI REGISTRU
        PUSH    DE
        PUSH    BC
        LD      HL,(LENGTH)
        EX      DE,HL
;
;<BC>,<HL> - ADRESY RETEZCU, <DE> - DELKA RETEZCE
;
D1      LD      A,(BC)
        CP      (HL)            POROVNEJ DVOJICE ZNAKU
        JR      NZ,D1           ZNAKY SE NEROVNAJI
        INC     HL              <HL> A <BC> UKAZUJI
        INC     BC              NA DALSI DVOJICI ZNAKU
        DEC     DE              SNIZ A OTESTUJ POCITADLO
        LD      A,D             POROVNANYCH ZNAKU
        OR      E               <DE> = 0 ?
        JR      NZ,D1           NE, POKRACUJ DALSI DVOJICI
D2      POP     BC              OBNOV REGISTRY
        POP     DE
        POP     HL
;
;CY = 1  POKUD HODNOTA ZNAKU NA ADRESE <HL> JE VETSI NEZ
        HODNOTA ZNAKU NA ADRESE <BC>.
;CY = 0  POKUD JSOU RETEZCE STEJNE NEBO HODNOTA ZNAKU NA
        ADRESE <HL> JE MENSI NEZ HODNOTA ZNAKU NA ADRESE
        <BC>

        RET
```

První část programu připraví pole adres a převezme informace o počtu a délce řetězců. Seznam adres zakončíme dvěma nulovými bajty, vlastně adresou 0, která bude sloužit jako zarážka. Když při prohlížení seznamu adres narazíme na adresu 0, znamená to, že jsme prošli celý seznam.

```
SSORT   LD      HL,(RAMTOP)
        INC     HL
        PUSH    HL              ZDE SE VYTVORI SEZNAM ADRES
        LD      HL,(VARS+4)
```

```
        LD      (COUNT),HL      POCET RETEZCU
        LD      HL,(VARS+6)
        LD      (LENGTH),HL     DELKA RETEZCE
        LD      HL,VARS+8
        LD      (ADDSTR),HL     ADRESA PRVNIHO RETEZCE
;
;ZASOBNIK - ADRESA SEZNAMU ADRES, BC - DELKA RETEZCE
;HL - ADRESA PRVNIHO RETEZCE, DE - POCET RETEZCU
;
A1      EX      DE,HL           HL-POCET, DE-ADRESA RETEZCU
        EX      (SP),HL         HL-ADRESA SEZNAMU, ZASOB. - POCET
        LD      (HL),E          DO SEZNAMU SE ULOZI
        INC     HL              ADRESA DALSIHO RETEZCE
        LD      (HL),D          NEJDRIV NIZSI, POTOM VYSSI BAJT
        INC     HL
        EX      (SP),HL         HL-POCET, ZASOB. - ADRESA SEZNAMU
        EX      DE,HL           DE-POCET, HL-ADRESA RETEZCU
        ADD     HL,BC           HL NA DALSI RETEZEC
        DEC     DE              SNIZ POCITADLO RETEZCU
        LD      A,D             TESTUJ POCITADLO DE
        OR      E               NA 0
        JR      NZ,A1           SEZNAM ADRES NEUPLNY
        POP     HL              HL-ADRESA KONCE SEZNAMU ADRES
        LD      (HL),A          SEZNAM ADRES ZAKONCIME
        INC     HL              DVEMA NULOVYMI BAJTY
        LD      (HL),A          VYUZIJEME, ZE A=0
```

Druhá část programu provede vlastní třídění řetězců; jejich uspořádání se zachytí v poli adres.

```
B1      XOR     A               VYNULUJ A
        LD      (FLAG),A        NASTAV INDIKATOR VYMENY
        LD      HL,(RAMTOP)
        INC     HL              HL-ADRESA SEZNAMU ADRES
        LD      C,(HL)          BC-ADRESA PRVNIHO RETEZCE
        INC     HL
        LD      B,(HL)
        INC     HL
B2      LD      D,B             DE-ADRESA PRVNIHO
```

```
        LD      E,C         POROVNAVANEHO RETEZCE
B3      LD      C,(HL)      BC-ADRESA DRUHEHO
        INC     HL
        LD      B,(HL)      POROVNAVANEHO RETEZCE
        INC     HL
        LD      A,B         KDYZ ADRESA DRUHEHO RETEZCE=0,
        OR      C           TAK JE SEZNAM ADRES VYCERPAN
        JR      Z,B4        POKRACUJ TESTEM PROMENNE FLAG
        CALL    CMPS        JINAK POROVNEJ RETEZCE
;
;KDYZ CY=0, VYMENA ADRES RETEZCU V SEZNAMU ADRES SE NEPROVEDE
;A PROGRAM POKRACUJE NA NAVESTI B2, KDE SE ADRESA DRUHEHO
;RETEZCE (OBSAH REGISTRU BC) PRESUNE NA MISTO ADRESY PRVNIHO
;RETEZCE (DO REGISTRU DE), REGISTR BC SE NAPLNI ADRESOU DALSIHO
;RETEZCE A POKRACUJE SE V CYKLU POROVNAVANI,JINAK SE RETEZCE
;V SEZNAMU VYMENI, REGISTR DE PAK OBSAHUJE ADRESU PRVNIHO
;RETEZCE A STACI POKRACOVAT NA NAVESTI B3, KDE SE REGISTR BC NAPLNI
;ADRESOU DALSIHO RETEZCE.
;
        JR      NC,B2       VYMENA ADRES SE NEPROVEDE
        LD      A,1
        LD      (FLAG),A    POZNAC VYMENU ADRES RETEZCU
        PUSH    HL          ULOZ ADRESU DO SEZNAMU ADRES
        DEC     HL
        LD      (HL),D      ULOZ ADRESY DO SEZNAMU
        DEC     HL          V OPACNEM PORADI, NEJDRIV DE
        LD      (HL),E      POTOM BC
        DEC     HL          PROTOZE SE DO PAMETI UKLADA
        LD      (HL),B      SESHORA DOLU, UKLADA SE
        DEC     HL          NEJDRIV VYSSI A POTOM NIZSI BAJT
        LD      (HL),C
        POP     HL          OBNOV ADRESU DO SEZNAMU ADRES
        JR      B3          A POKRACUJ
B4      LD      A,(FLAG)    KONEC JEDNOHO PRUCHODU
        OR      A           BYLA PROVEDENA VYMENA ?
        JR      NZ,B1       ANO - POKRACUJ NOVYM PRUCHODEM
```

V třetí části programu se řetězce v poli A$(N,M) uspořádají podle informací obsažených v seznamu adres, realizuje se tedy algoritmus popsaný v bodech 1 až 4 (viz výše).

```
        LD      BC,(ADDSTR)   BC UKAZUJE NA RETEZCE V POLI A$
        LD      HL,(RAMTOP)   HL UKAZUJE NA ADRESY RETEZCU
        INC     HL            V SEZNAMU ADRES
C1      LD      E,(HL)        DE-ADRESA RETEZCE, KTERY JE
        INC     HL            NA I-TEM MISTE V POSLOUPNOSTI
        LD      D,(HL)        RETEZCU PODLE ABECEDY
        INC     HL            KDYZ JE NASLEDUJICI ADRESA 0,
        LD      A,(HL)        TAK SE ALGORITMUS PROVEDL
        INC     HL            PRO I=1 AZ N-1
        OR      (HL)          A RETEZCE JSOU USPORADANY
        RET     Z             RIZENI SE VRATI DO BASIKU
        PUSH    HL            ULOZ ADRESU DO SEZNAMU ADRES
;
;NASLEDUJE VYMENA RETEZCU, JEJICHZ ADRESY JSOU V REGISTRECH BC
;A DE (BOD 2 ALGORITMU). NA VYMENU SE POUZIJE PRACOVNI PAMET
;NA ADRESE TMPBUF. NA ZACATKU SE UCHOVA OBSAH REGISTRU HL, BC
;A DE, KTERE JSOU POTREBNE PRI REALIZACI BODU 3 A 4
;VLASTNI VYMENA RETEZCU NEBUDE DAL KOMENTOVANA, DOPORUCUJI
;CTENARI, ABY BEDLIVE SLEDOVAL OBSAHY REGISTRU PRED
;POUZITIM INSTRUKCE LDIR.
;
        PUSH    HL
        PUSH    DE
        PUSH    BC
        PUSH    DE            ZDE ZACINA VYMENA RETEZCU
        LD      BC,(LENGTH)
        PUSH    BC
        LD      HL,TMPBUF
        EX      DE,HL
        LDIR
        POP     BC
        POP     DE
        POP     HL
        PUSH    HL
        PUSH    BC
```

```
                LDIR
                POP     BC
                POP     DE
                PUSH    DE
                LD      HL,TMPBUF
                LDIR                    VYMENA RETAZCU SKONCENA
                EX      DE,HL
                POP     BC
                POP     DE
                EX      (SP),HL
;
;REGISTR BC OBSAHUJE ADRESU RETEZCE V POLI A$, REGISTR DE
;OBSAHUJE ADRESU RETEZCE ZE SEZNAMU ADRES, REGISTR HL
;OBSAHUJE ADRESU NASLEDUJICI ADRESY ZE SEZNAMU ADRES (DE
;JE HODNOTA ZE SEZNAMU A HL JE UKAZATEL DO SEZNAMU).
;V ZASOBNIKU JSOU ULOZENY ADRESY DALSICH RETEZCU V POLI A$
;A ADRESA DALSI ADRESY V SEZNAMU ADRES. TYTO HODNOTY JSOU
;POTREBNE PRO OPAKOVANI BODU 1 AZ 4
;REALIZOVAT BODY 3 A 4 ZNAMENA NAJIT V SEZNAMU ADRES,
;JEHOZ ZACATEK JE V HL, HODNOTU, KTERA JE V BC A PREPSAT
;JI HODNOTOU, KTERA JE V DE
;
C3              LD      A,(HL)
                CP      C               POROVNANI NIZSICH BAJTU
                INC     HL              DRIV NEZ SE PROVEDE PRIPADNY
SKOK
                LD      A,(HL)          NUTNO PRECIST I VYSSI BAJT,
                INC     HL              ABY HL BYL NA DALSI ADRESE
                JR      NZ,C3           NIZSI BAJTY JSOU RUZNE
                CP      B               JINAK POROVNEJ VYSSI BAJTY
                JR      NZ,C3           POKRACUJ V HLEDANI
                DEC     HL              JINAK PREPIS NALEZENOU HODNOTU
                LD      (HL),D          HODNOTOU V REGISTRU DE
                DEC     HL
                LD      (HL),E
                POP     BC              BC-ADRESA NA DALSI RETEZEC V A$
                POP     HL              HL-NA DALSI ADRESU V SEZNAMU
                JR      C1              POKRACUJ BODY 1 AZ 4
                END                     KONEC TEXTU PROGRAMU
```

Program máme na papíře, následuje třetí krok - jeho uložení do počítače. Z kazety nahrajeme program MRS, příkazem EDI vyvoláme editor, příkazem INI přejdeme do obrazovkového režimu a uvedený text pečlivě opíšeme, (komentáře můžeme vynechat).

Většinu chyb, jichž se dopustíme při opisování, systém zachytí; pokud se vyskytnou jiné obtíže, než zde popsané, je nutno odstranit chyby, které vznikly při přepisu.

Po napsání celého textu se vrátíme do příkazového režimu (klávesou BREAK) a program přeložíme příkazem ASM. Volat spojovací program příkazem LNK je zbytečné, protože jsme nepoužili pseudoinstrukce EXT a ENT. Případné chyby, které modul ASM odhalí, opravíme a příkazem SAV SSORT modulu EDI uložíme zdrojový text na kazetu, abychom ho v průběhu ladění náhodou nevymazali.

Ladit začneme od procedury CMPS. Vstupní údaje pro tuto proceduru připravíme někde v paměti příkazem M modulu DBG. Vyvoláme modul DBG a napíšeme text

**m #8000 < enter >**
**j a n a j i r i < enter >**

a tak od adresy #8000 uložíme ASCII kódy dvou řetězců 'jana' a 'jiri'.
Pomocí příkazů

**rb#8000 < enter >**
**rd#8004 < enter >**
**rpcmps < enter >**
**m length < enter > 0400 < enter >**

uložíme do registrů BC a DE adresy řetězců, do registru PC adresu procedury CMPS a do proměnné LENGTH délku porovnávaných řetězců (4). Příkazem N spustíme proceduru CMPS a ladicí modul za chvíli vypíše základní stav začínající znakem E. Podprogram tedy proběhl úspěšně; zkontrolujeme ještě, že se registry BC, DE, HL nezměnily a že Cy flag obsahuje správnou hodnotu. V našem případě musí být Cy = 1, protože řetězec na adrese DE - 'jiri' je za řetězcem na adrese BC - 'jana'. Stejným způsobem ověříme funkci podprogramu v případě, že jsou řetězce přehozeny nebo se porovnávají dva totožné řetězce. Pokud všechny testy proběhnou úspěšně, považujeme podprogram za odladěný.

Abychom mohli odzkoušet první část programu, musíme připravit pole A$ v jazyce Basic. Příkazem MON vrátí systém MRS řízení programu Basic.

Výraz za příkazem CLEAR opravíme na 6*4096+7*256-1, čímž vytvoříme na začátku volné paměti prostor 256 bajtů, který program SSORT používá na seznam adres. Řádek, v kterém se nahrávají binární moduly systému MRS, nahradíme např. řádkem

```
20 DIM A$(5,4):LET A$(1)="IVAN":LET A$(2)="VERA":
   LET A$(3)="JANA":LET A$(4)="PETR":LET A$(5)="ADAM"
```

a řádek 30 preventivně vymažeme, aby se při spuštění příkazem RUN nevymazal zdrojový text (teplý start). Potom systém MRS opět spustíme pomocí příkazu RUN a příkazem DBG vyvoláme ladicí program.
Pomocí příkazů

**rpssort<enter>**
**sssssssss**

provedeme prvních devět instrukcí, které by měly převzít parametry pole A$. Registr BC by měl mít hodnotu 4 (délka řetězce) a registr DE hodnotu 5 (počet řetězců). Pohled na obrazovku nás však přesvědčí, že v první části programu je nějaká chyba; registry BC a DE mají velmi podivné hodnoty.

Když se blíže podíváme např. na instrukci LD BC,(VARS+4), vidíme, že do registru BC dávame hodnotu z adresy (VARS+4), ale požadovaná hodnota je na adrese (VARS)+4, což je rozdíl. Tato velmi typická chyba se zrovna tak týká obsahů registrů DE a HL. První tři řádky od návěští SSORT jsou v pořádku, ale dalších pět řádků vymažeme a nahradíme je řádky

```
LD      HL,(VARS)
INC     HL
INC     HL
INC     HL
INC     HL
LD      E,(HL)
INC     HL
LD      D,(HL)
INC     HL
LD      C,(HL)
INC     HL
```

```
LD      B,(HL)
INC     HL
LD      (LENGTH),BC
LD      (COUNT),DE
```

Příkazem Q ukončíme práci ladicího modulu, příkazem EDI vyvoláme editor. Příkazem **ln=ssort** se nastavíme na začátek programu a opravíme ho. Program opět přeložíme, aby se oprava promítla i do binárního kódu a opět ho vyzkoušíme. Pro změnu to můžeme udělat tak, že příkazem DBG vyvoláme ladicí modul a pomocí příkazů

**rpssort<enter>**
**ib1<enter>**
**n**

nastavíme registr PC na začátek programu, bod přerušení na návěští B1 a program provedeme až po tento bod v sledovacím režimu.
Pomocí příkazů

**mlength<enter>**
**<enter>mcount<enter>**
**<enter>maddstr<enter>**

zkontrolujeme postupně obsah proměnných COUNT, LENGTH a ADDSTR. V prvním případě je obsah prvních dvou bajtů 04 00, ve druhém případě 05 00 a ve třetím případě 26 5F. To znamená LENGTH = 4, COUNT = 5 a zkontrolujeme ještě hodnoty na adrese #5F26.
Klávesou <enter> ukončíme příkaz M a příkazem

**d#5f26<enter><enter>**

zobrazíme obsah paměti od adresy #5F26. I zde je všechno v pořádku, na adrese #5F26 jsou naše řetězce, které jsme definovali v programu Basic. Zbývá ještě ověřit, zda jsou od adresy (RAMTOP)+1 uloženy adresy těchto řetězců, zakončené dvěma nulami.
Napíšeme tedy

**d#6700<enter><enter>**

a od adresy #6700 jsou skutečně uloženy hodnoty 26 5F 2A 5F 2E 5F 32 5F 36 5F 00 00 tak, jak to má být.

Stejným způsobem ověříme činnost programu až k návěští C1, tedy vlastní třídění.
Pomocí příkazů

**ic1 <enter>**
**n**

provedeme program až k návěští C1 v sledovacím režimu. Program se na toto místo úspěšně dostane a nám zbývá ověřit, jestli hodnoty na adrese #6700 obsahují informace potřebné pro uspořádání řetězců podle abecedy. Protože adresa #6700 byla poslední, kterou jsme příkazem D zobrazili, stačí napsat příkaz

**d <enter> <enter>**

Na adrese #6700 jsou hodnoty 36 5F 26 5F 2E 5F 32 5F 2A 5F 00. Pomocí příkazu

**m#5f36 <enter>**

si ověříme, že na adrese #5F36 je text ADAM. Stejným způsobem prozkoumáme i ostatní adresy, čímž si potvrdíme, že adresa na i-té pozici je adresa i-tého řetězce v pořadí podle abecedy. Ladění programu tedy úspěšně pokračuje, zbývá otestovat jeho závěrečnou část. Není třeba nastavovat bod přerušení, protože poslední instrukce RET vrátí řízení modulu DBG. Spustíme program příkazem

**n <enter>**

ale program se ani po 10 sekundách (což je pro tak jednoduchý kód opravdu dost) nepřihlásí. Přerušíme jeho běh klávesou BREAK a začneme krokovat příkazem S, abychom zjistili, kde se zacyklil.

Program neustále provádí instrukce od návěští C3, pomocí nichž hledá hodnotu registru BC v seznamu adres; jako ukazatel slouží registr HL. Registr HL už má hodnotu přes #7000, i když seznam adres je v intervalu #6700-#670B. Z nějakého důvodu se tedy hodnota registru BC nenašla.

Pohled na obrazovku nám prozradí, že registr BC obsahuje hodnotu #5F2E; příkazem

**m#5f2e<enter>**

zjistíme, že se jedná o řetězec 'JANA'. Čím je tenhle řetězec zajímavý? Hodně přemýšlení si ušetříme, když si všimneme, že registr DE, jehož hodnotu máme v seznamu přepsat, obsahuje také hodnotu #5F2E. To vlastně znamená, že řetězec 'JANA' se nikam neposouvá. Tento řetězec byl původně na třetím místě a po setřídění na třetím místě zůstane. Protože však jeho adresu hledáme až od čtvrté pozice, je jasné, že ji nemůžeme najít. Bod 3 algoritmu opravíme na

3) Ve zbylých adresách ze seznamu adres, tedy na pozicích i až N, najdu adresu i-tého řetězce z pole A$(N,M).

Pro program v assembleru to znamená, že před prohledáváním seznamu adresu snížíme ukazatel (hodnotu HL) o 2. Příkazem Q ukončíme práci modulu DBG, příkazem EDI vyvoláme editor a příkazem ln=c3 se posuneme na řádek s návěštím C3. Před tento řádek vložíme dvakrát řádek DEC HL a program opět přeložíme. Část řetězců už byla posunuta; obnovíme tedy pole A$ tak, že příkazem MON vrátíme řízení do Basiku, a program spustíme příkazem RUN. Potom příkazem DBG zavoláme ladicí modul a pomocí příkazů

**rpssort<enter>**
**n**

provedeme celý program ve sledovacím režimu. Program po malé chvíli skončí návratem do ladicího modulu a my už jen pomocí příkazu

**d#5f26<enter><enter>**

ověříme, že řetězce jsou opravdu setříděny. Předpokládejme, že tímto je program odladěn, i když v praxi by to chtělo důkladnější testování. Zbývá už jen jedno: upravit program tak, aby sloužil i bez přítomnosti systému MRS. Jako logické umístění v paměti se nabízí prostor od adres #FE00; upravíme tedy zdrojový text tak, že za první řádek (obsahuje jen ;), vložíme řádek

ORG #FE00. Abychom zjistili délku programu, doplňme ještě poslední řádek programu s textem END libovolným návěštím, např. KONEC.

Když se však program pokusíme přeložit příkazem ASM, překlad skončí výpisem MEM FULL. Na adrese #FE00 je totiž kód systému MRS a překladač ho nedovolí přepsat. Systém MRS naštěstí poskytuje prostředek i proti této komplikaci. Za řádek s textem ORG vložíme instrukci řízení překladu např. ve tvaru *C8E00. Když přeložíme takto upravený program, uloží se vlastní kód od adresy #8E00, ale provádět se dá až po přesunutí na adresu #FE00. Ukážeme si, že tento přesun je možno zajistit pomocí jazyka Basic. Po překladu ještě určíme rozsah a startovací adresu programu. Víme, že program bude začínat na adrese #FE00 = 65024; koncovou a startovací adresu zjistíme pomocí modulu DBG příkazy

**mkonec<enter>**
**<enter>mssort<enter>**

V našem případě KONEC = #FEC2, program má tedy délku #C2 = 194 bajtů, a SSORT = #FE1D = 65053, což je hodnota startovací adresy. Klávesou <enter> ukončíme příkaz M, pomocí příkazů Q a MON vrátíme řízení programu Basic a uložíme program na pásku. Přitom si musíme uvědomit, že vlastní kód je uložen od adresy #8E00 = 36352; uložíme ho tedy příkazem

SAVE "SSORT"CODE 36352,194

Před jeho použitím musíme nastavit RAMTOP alespoň na hodnotu #FE00 - 2 * počet řetězců - 1, protože od (RAMTOP)+1 se ukládá seznam adres. Potom nahrajeme binární kód do paměti příkazem

LOAD "SSORT"CODE 65024

a pokud je proměnná A$(N,M) v našem programu umístěna jako první, tak její řetězce setřídíme podle abecedy příkazem

RANDOMIZE USR 65053

Na závěr ještě jedno upozornění. Pokud pole A$ nahráváme z kazety, tak se původní definice příkazem DIM vymaže a pole už nebude první

proměnná. Znamená to, že v tomto případě musí být příkaz LOAD "JMENO"STRING A$() uveden dříve, než se definuje jiná proměnná.

Schopný čtenář s pomocí příručky k počítači Spectrum snadno upraví program tak, aby dokázal vyhledat určitou proměnnou v seznamu proměnných. Potom může být jméno pole vstupním parametrem pro podprogram, toto jméno se může uložit příkazem POKE na pevné místo v paměti a podprogram SSORT ho odtud přečte.

# PŘÍLOHA

## Zdrojový text některých modulů systému MRS

```
;PODPROGRAMY PRO KONVERZE
;
;
;                                   B N D C
;
BNDC:       LD      A,1
BD0:        LD      D,B
            LD      E,C
            LD      C,A
            EX      DE,HL
            PUSH    BC
BD1:        XOR     A
            LD      B,16
BD2:        ADD     HL,HL
            ADC     A,A
            CP      10
            JR      C,BD3
            SUB     10
            INC     L
BD3:        DEC     B
            JR      NZ,BD2
            OR      #30
            PUSH    AF
            LD      A,L
            OR      H
            JR      NZ,BD1
            EX      DE,HL
            INC     A
            CP      C
            SBC     A,A
            LD      B,A
BD4:        POP     AF
            RET     C
            LD      (HL),A
            ADD     HL,BC
            JR      BD4
;
BNHX2:      LD      A,D
            CALL    BNHX
```

```
                LD      A,E
;
;                                       B N H X
;
BNHX:           PUSH AF
                RRCA
                RRCA
                RRCA
                RRCA
                CALL    BNHX1
                POP     AF
BNHX1:          AND     #0F
                ADD     A,#90
                DAA
                ADC     A,#40
                DAA
                OR      #20
DNUM:           LD      (HL),A
                INC     HL
                RET
;
;                                       H X B N
;
HXBN:           LD      DE,0
                LD      A,(HL)
                CALL    HXBN1
                RET     C
                EX      DE,HL
                ADD     HL,HL
                ADD     HL,HL
                ADD     HL,HL
                ADD     HL,HL
                EX      DE,HL
                OR      E
                LD      E,A
                INC     HL
                JR      HXBN+3
HXBN1:          ADD     A,#C6
                JR      C,HXBN2
```

```
                SUB     #F6
                RET
HXBN2:          AND     #DF
                SUB     7
                RET     C
                ADD     A,#FA
                RET     C
                SUB     #F0
                RET
;
;                                       D C B N
;
DCBN:           LD      DE,0
                LD      A,(HL)
                CP      #3A
                RET     NC
                SUB     #30
                RET     C
                EX      DE,HL
                PUSH    BC
                ADD     HL,HL
                LD      B,H
                LD      C,L
                ADD     HL,HL
                ADD     HL,HL
                ADD     HL,BC
                LD      C,A
                LD      B,0
                ADD     HL,BC
                POP     BC
                EX      DE,HL
                INC     HL
                JR      DCBN+3
;
;K O N S T A N T Y  A  D A T A  P R O  D A L S I  M O D U L Y
;
KPORT           EQU     #FE
SHF1            EQU     1
SHF2            EQU     37
```

```
MAXC:     EQU   64
MAXCS     EQU   31
MAXR      EQU   23
CHLEFT    EQU   #19
CHRGHT    EQU   #18
CRLF      EQU   #0D
FE        EQU   #0C
LF        EQU   #0A
SYSBEP    EQU   #03B5
KREPT:    DB    0
LASTK:    DB    0
KTAB0:    DB    0,'zxcvasdfgqwert12345'
          DB    '09876poiuy',0DH,'lkjh'
          DB    ' ',0,'mnb'
KTAB1:    DB    0,'ZXCVASDFGQWERT'
          DB    #1E,#1F,#1C,#1D,#19
          DB    #13,#12,#18,#17,#1A
          DB    'POIUY',5,'LKJH'
          DB    #11,#07,'MNB'
KTAB2:    DB    0,#3A,#60,#3F,#2F,#7B,#7C,#7D,#7E,0
          DB    #5B,#5C,#5D,#3C,#3E,'!@#$%'
          DB    '_)(',#27,'&";',0,0,0
          DB    3,'= +-^',#14,0,'.,*'
POSCUR:   DW    #5800
CURSOR:
ROW:      DS    1
COLUMN:   DS    1
;
;RUZNE POMOCNE PODPROGRAMY
;
ABSCRS:   LD    A,H
          RRCA
          RRCA
          RRCA
          AND   #E0
          OR    L
          LD    L,A
          LD    A,H
          AND   #18
```

```
        OR      #40
        LD      H,A
        RET
;
SETCHR: LD      HL,(CURSOR)
SETCH0: PUSH    AF
        CALL    ABSCRS
        POP     AF
;
PIXEL:  PUSH    HL
        PUSH    DE
        PUSH    BC
        EX      DE,HL
        LD      L,A
        LD      H,0
        ADD     HL,HL
        ADD     HL,HL
        ADD     HL,HL
        LD      BC,#3C00
        ADD     HL,BC
        LD      B,8
PX1:    LD      A,(HL)
        LD      (DE),A
        INC     D
        INC     HL
        DJNZ    PX1
        POP     BC
        POP     DE
        POP     HL
        RET
;
IIID:   PUSH    HL
        PUSH    BC
        LD      A,8
IID1:   LD      BC,32
        PUSH    DE
        PUSH    HL
        LDIR
        POP     HL
```

```
                POP     DE
                INC     H
                INC     D
                DEC     A
                JR      NZ,IID1
                POP     BC
                POP     HL
                RET

;
IDELLN:         LD      HL,(CURSOR)
                SUB     H
                RET     Z
                LD      B,A
                LD      L,0
                CALL    ABSCRS
II0:            LD      D,H
                LD      A,L
                ADD     A,32
                LD      E,L
                LD      L,A
                JR      NZ,II1
                LD      A,8
                ADD     A,H
                LD      H,A
II1:            CALL    IIID
                DJNZ    II0
                RET

;
IINSLN:         LD      A,MAXR-1
                LD      HL,(CURSOR)
                SUB     H
                RET     Z
                RET     C
                LD      HL,#50A0
                LD      DE,#50C0
                LD      B,A
ID0:            CALL    IIID
                EX      DE,HL
                LD      A,E
```

```
                SUB     32
                LD      L,A
                LD      H,D
                JR      NC,ID1
                LD      A,D
                SUB     8
                LD      H,A
ID1:            DJNZ    ID0
                RET
;
SCROLL:         LD      HL,(CURSOR)
SCROL1:         LD      A,MAXR
                PUSH    HL
                PUSH    DE
                LD      HL,0
                LD      (CURSOR),HL
                CALL    IDELLN
                POP     DE
                POP     HL
                LD      (CURSOR),HL
                RET
;
;                                       S C A N
;
SCAN:           PUSH    BC
                LD      BC,KPORT
                IN      A,(C)
                POP     BC
                CPL
                AND     #1F
                RET     Z
                PUSH    BC
                PUSH    DE
                PUSH    HL
                LD      BC,KPORT+#FE00
                LD      HL,0
                LD      E,1
KS3:            IN      A,(C)
                LD      D,5
```

```
KS4:    RRCA
        JR      C,KS1
        INC     H
        DEC     H
        JR      NZ,KERR
        LD      H,L
        LD      L,E
KS1:    INC     E
        DEC     D
        JR      NZ,KS4
        RLC     B
        JR      C,KS3
        EX      DE,HL
        LD      HL,KTAB0-1
        INC     D
        DEC     D
        JR      Z,KTE
        INC     E
        DEC     E
        JR      Z,ADDD
        LD      HL,KTAB1-1
        LD      A,SHF1
        CP      D
        JR      Z,ADDE
        CP      E
        JR      Z,ADDD
        LD      HL,KTAB2-1
        LD      A,SHF2
        CP      D
        JR      Z,ADDE
        CP      E
        JR      Z,ADDD
KERR:   XOR     A
KRET:   POP     HL
        POP     DE
        POP     BC
        RET
ADDD:   LD      E,D
ADDE:   LD      D,0
```

```
        ADD     HL,DE
        LD      A,(HL)
        JR      KRET
KTE:    INC     E
        DEC     E
        JR      Z,KERR
        JR      ADDE
;
;                               INKEY
;
INKEY:  PUSH    BC
INKEY1: CALL    SCAN
        OR      A
        JR      NZ,IN3
        LD      (LASTK),A
        JR      INKEY1
IN3:    LD      BC,#1000
WAIT1:  DEC     C
        JR      NZ,WAIT1
        DJNZ    WAIT1
        LD      C,A
        CALL    SCAN
        CP      C
        JR      NZ,INKEY1
        LD      A,(LASTK)
        CP      C
        JR      Z,REPEAT
        LD      A,250
        JR      SIN1
REPEAT: LD      A,(KREPT)
        LD      B,A
REP1:   XOR     A
REP2:   DEC     A
        JR      NZ,REP2
        CALL    SCAN
        CP      C
        JR      NZ,INKEY1
        DJNZ    REP1
        LD      A,25
```

```
SIN1:   LD      (KREPT),A
        LD      A,C
        LD      (LASTK),A
        PUSH    HL
        PUSH    DE
        PUSH    AF
        LD      HL,80
        LD      DE,20
        CALL    SYSBEP
        POP     AF
        POP     DE
        POP     HL
        POP     BC
        RET
```

## OUTCHR

```
OUTCHR: CP      LF
        RET     Z
        CP      PE
        RET     Z
        PUSH    HL
        LD      HL,(CURSOR)
        CP      #20
        JP      C,SPEC
        PUSH    HL
        CALL    SETCHR
        POP     HL
        INC     L
        LD      A,MAXCS+1
        SUB     L
        JR      NZ,NEWC
NEWL:   LD      L,A
        LD      A,H
        SUB     MAXR-1
        ADC     A,MAXR-1
        LD      H,A
        CALL    NC,SCROL1
```

```
NEWC:   LD      (CURSOR),HL
        POP     HL
        RET
SPEC:   SUB     CRLF
        JR      Z,NEWL
        LD      DE,NEWC
        PUSH    DE
        SUB     CHLEFT-CRLF
        JR      Z,LVLAVO
        SUB     CHRGHT-CHLEFT
        RET     NZ
        LD      A,MAXC-2
        CP      L
        RET     C
        INC     L
        RET
LVLAVO: DEC     L
        RET     P
        INC     L
        RET
```

## INVPIX

```
INVPIX: PUSH    HL
        PUSH    BC
        CALL    ABSCRS
        LD      B,8
IPX1:   LD      A,(HL)
        CPL
        LD      (HL),A
        INC     H
        DJNZ    IPX1
        POP     BC
        POP     HL
        RET
```

## SETCRS

```
SETCRS: PUSH    HL
```

```
        LD      HL,(POSCUR)
        RES     7,(HL)
        POP     HL
        PUSH    HL
        PUSH    DE
        LD      A,L
        CP      MAXCS+1
        JR      C,SCR1
        LD      A,MAXCS
SCR1:   LD      D,#58
        LD      E,A
        LD      L,H
        LD      H,0
        ADD     HL,HL
        ADD     HL,HL
        ADD     HL,HL
        ADD     HL,HL
        ADD     HL,HL
        ADD     HL,DE
        POP     DE
        LD      (POSCUR),HL
        SET     7,(HL)
        POP     HL
        LD      (CURSOR),HL
        RET
```

CLEAR

```
CLEAR:  LD      HL,#4000
        LD      DE,#4001
        LD      BC,#1800
        LD      (HL),0
        LDIR
        LD      (POSCUR),HL
        LD      (HL),#38
        LD      BC,#300-1
        LDIR
        RET
        END
```

IVAN JEDLIČKA

# MEMORY RESIDENT SYSTEM

Programový systém pro vývoj a ladění programů
ve strojovém kódu - assembleru mikroprocesoru Z80
(verze pro mikropočítače ZX Spectrum, Delta a Didaktik Gama)

Předmluvu napsal Ladislav Zajíček
Přeložili Ladislav Zajíček (kap. 0 až 7) a Petr Karlach (kap. 8)
Odbornou revizi provedl Ladislav Zajíček
Odborně lektoroval Petr Karlach
Obálku a titulní list navrhl Jaroslav Baierle
Sazbu na osobním počítači provedl Ladislav Zajíček
Typograficky upravil na osobním počítači Petr Karlach
Vydala Mladá fronta ve spolupráci se Zenitcentrem SSM
Publikace číslo 5047
Odpovědná redaktorka Božena Fleissigová
Výtvarný redaktor Josef Velčovský
Technický redaktor Miloš Jirsa
Vytiskło maloofsetové středisko MF, Kaliningova 42, Praha 3
3,53 AA, 4,25 VA, 80 stran
Náklad 3000 výtisků. 505/21/81.6
Vydání 1. Praha 1989
03/2 23–022–89
Cena brožovaného výtisku včetně kazety 70 Kčs

Případné reklamace programových kazet uplatňujte
u pobočky ZENITCENTRA, Beroun

03/2 23–022–89
Cena brožovaného výtisku
včetně kazety 70 Kčs

**ISBN 80-204-0055-9**

**Vydala MLADÁ FRONTA**
**ve spolupraci se ZENITCENTREM,**
**centrem mládeže, vědy a techniky SSM**