Turbo assembler

PROMETHEUS

Určeno pro počítače:
Sinclair ZX Spectrum, Didaktik Gama,
Didaktik M, Didaktik Kompakt

© 1990 PROXIMA Ústí nad Labem

I. Trocha sebechvály aneb odkud to přišlo

Při psaní programů v assembleru na Spectru člověk brzy narazí na problém velikosti paměti - do paměti se musí vejít assembler, zdrojový text, přeložený kód a pokud možno nějaký prostředek pro ladění. Nabízejí se dvě řešení, buď si koupit něco většího než je Spectrum - toto řešení má malý zádrhel, chce to peníze - nebo získat programy, které s tím malým množstvím pamětí, které je k dispozici, zacházejí úsporně. Nápady, jako je kompilace přímo z pásku, jsou silné kostrbaté a pro lid nevlastnící ani disketovou jednotku ani microdrive prakticky nepoužitelné.

Rozhlédne-li se uživatel kolem sebe na nabídku assemblerů a ladících prostředků zjistí následující fakta:

GENS - nejstarší a nejrozšířenější assembler, zdrojový text je uložen jako textový soubor, poměr mezi délkou zdrojového textu a překladu je asi 8:1, snad nejhorší editor, který si Ize vymyslet, assembler je dlouhý přibližně 8,5 KB, překládá poměrně rychle, silně obtěžující je neustálá otázka "Table size:", kde implicitní hodnota obvykle nestačí a po hlášení "No table space" je nutno kompilaci opakovat s větší tabulkou, také dotaz "Option:" při každé kompilaci práci zbytečně neurychluje, program je relokovatelný, což. jeho hodnotu ,velmi zvyšuje, k programu existuje monitor MONS - bohužel práce s ním není pohodlní pro uživatele nepoužívající hexa soustavu, ani služby, které monitor poskytuje už nejsou na úrovni doby, monitor je dlouhý 5 KB a je relokovatelný.

Rozšířené verze GENSu:

- GENS3E (Lamač) - vylepšená verze programu GENS, obsahuje Full Screen Editor s automatickou tabelací a 42 znaky na řádek, program je 12 KB dlouhý a zachoval si relokovatelnost.

- Adámkovo rozšíření - podobné jako GENS3E má Full Screen Editor s automatickou tabelací, na řádku je 64 znaků (horší čitelnost), program už není relokovatelný.

- MASM (MACROS, ASSEMBLER 80) - u tohoto programu by se dalo polemizovat, jestli je nebo není od programu GENS odvozen, má s ním však velké množství shodných prvků (bohužel i špatných), délka programu je 11 KB, není relokovatelný, má Full Screen Editor s automatickou tabelací, zná „neznámé instrukce“, má křížové reference, celkově lze říci, že to není žádná revoluce, zdrojový text je uložen stejně neefektivně jako u programu GENS.

OCP assembler - program je nerelokovalelný, zdrojový text zabírá více místa než stejný u programu GENS, před kompilací je nutno zadávat delší posloupnost parametrů, má Full Screen Editor, chyby na rozdíl od programu GENS (i modifikací) nevypisuje čísly (tzn. mít po ruce tabulku) ale přímo slovy (práce je pohodlnější).

Laser Genius Assembler - program je relokovatelný, má délku kolem 20 KB (s lopatou na komára), pro svou délku je těžko použitelný bez disketové jednotky nebo microdrive, existuje k němu Laser Genius Monitor, oba programy se však do paměti najednou téměř nevejdou.

Memory Resident System - první assembler, který se snaží uložit efektivně zdrojový text, poměr je okolo 4;1 (tedy při stejné délce pojme dvakrát tak dlouhý zdrojový text), program je dlouhý 11 KB a není relokovatelný (existují 3 verze), má Full Screen Editor a kontroluje instrukce už při vkládání (odstraní se mnoho

chyb), ve snaze zkrátit zdrojový text byla použita některá ne příliš šťastná omezení - maximálně 255 návěští, ve výrazu se smí použít maximálně dva operandy a jeden operátor, nelze použít binární zápis čísel (zvláště první omezení doslova zabíjí dobrý nápad), rozporný je také monitor, dokáže při trasování hlídat assembler a zdrojový text před náhodným přepsáním, bohužel všechny výpisy jsou v šestnáctkové soustavě, v monitoru se lze odkazovat na existující návěští v tabulce symbolů, při psaní MRS byl zřejmě požadavek na krátkost přeceněn (program je dlouhý 12 KB) ale na úkor komfortu obsluhy i možností.

PIKASM - program od T.R.C, je 11 KB dlouhý, není relokovalelný, má Full Screen Editor, zdrojový text ukládá v komprimované formě (poměr 5-6:1), na rozdíl od předchozích editorů má příjemnou vlastnost - po kompilaci zůstává kurzor na tom místě, kde byl před kompilací (při vstupu do editoru neskáče ani na konec ani na začátek), assembler také umí při kompilaci před každou instrukci řídící běh programu (call, jp, ret) vložit volání testu klávesy BREAK - užitečné při ladění bez monitoru (lze zastavit program, který se zacyklil (tzv. kousnul)).

MON, MON2 - monitory, dlouhé 7,5 a 9 1KB, oba jsou relokovalelné, umí krokovat instrukce - tučná i automaticky (s vypisováním po každé instrukci i bez něj- velmi rychlé), vypisovat paměť, měnit obsah paměti, plnit a přesouvat bloky, lze také nadefinoval oblast obrazovky, kam budou mířit výpisy (souvislý úsek řádků na obrazovce).

VAST (PIKOMON, MILIMON, MIKROMON) - monitor dlouhý 4,b KB, umístěný v obrazovkové paměti, původně byt psán pro hledání nekonečných životů, ale lze jej použít i při ladění programů - obzvláště v závěrečné fázi, kdy je v paměti málo místa. Existuje další vývojová verze tohoto programu nazvaná DEVAST ACE, která je vylepšena o další služby a kterou nemá na svědomí T.R.C ale moje maličkost. Na rozdíl od VASTu, existuje DEVAST ACE také v relokovatelné verzi. Pro zajímavost - celý program je dlouhý 4,6 KB a jeho zdrojový text (kompletní) má délku kolem 16 KB. Obě verze programu DEVAST ACE jsou součástí kompletu USER I.

Uvedený přehled ukazuje autorovi známý stav a jeho názory na dostupné programy. Z uvedených programů jsou často používané kombinace:

GENS + MONS

GENS + MON (MON2)

OCP + MONS

Memory Resident System

Laser Genius

Při posouzení dostupných programů vzniká myšlenka vyrobit program, který by spojoval výhody všech předchozích programů a neobsahoval jejich nevýhody.

Nejdůležitější vlastnosti, které by takový program měl mít jsou dvě: co nejefektivnější uložení zdrojového textu a spojení assembleru a monitoru do jednoho celku, tak, aby monitor mohl používat některé podprogramy z assembleru a používat existující tabulku symbolů při výpisech. Proto vznikl tento program.

Assembler a monitor - ladící systém PROMETHEUS vznikal za pomoci programů GENS3E, GENS 3.1, MON, MON2 a VAST. Program GENS byl používán do té doby, než se podařilo odladit první provozuschopnou verzi assembleru PROMETHEUS, další vylepšování se provádělo vždy na starší verzi - assembler byl

překládán sám sebou, monitor vznikl úplně od začátku na assembleru PROMETHEUS. Pro srovnání . zdrojový text v GENS byl dlouhý kolem 40 KB a byt tedy nutně rozdělen na dva, překládal se celkem 40 sekund, po přetažení zdrojového textu do PROMETHEA se zdrojový text zkrátil na polovinu (20 KB), odpadla potřeba místa na tabulku symboly - u obou částí asi po 3 KB, text se tedy vešel do paměti najednou (asi 5 KB kódu assembleru jsou tabulky a pracovní místo, které nebyly ve zdrojovém textu). Celý zdrojový text byl nyní přeložen za 3 sekundy (tedy 2 KB/s).

Nejdůležitější vlastnosti ladícího systému PROMETHEUS:

(Stručný seznam vhodný pro rychlou orientaci případného zájemce)

- délka: 11 KB assembler, 5 KB monitor (celkem 16 KB, monitor lze odpojit a neužívat)
- celý program je relokovatelný, a není závislý na systému Spectra
- poměr mezi zdrojovým textem a přeloženým kódem je 3-4:1 (GENS 8:1)
- tabulka symbolů je organickou částí zdrojového textu, vytváří se okamžitě při psaní textu, vždy je ihned přístupná - tzn. že pro kompilaci už není třeba ani byte navíc
- rychlost kompilace je asi 2-3 KB/s - l0x víc než GENS
- editor je řádkově orientovaný, má automatickou tabelaci - skrolování a stránkování je velmi rychlé, umožňuje mazání a kopírování bloku, vyhledávání a nahrazování řetězců, každý řádek je při odeslání syntakticky kontrolován
- obvyklé kazetové operace - SAVE, VERIFY, LOAD, možnost vyhrát i přihrát (na zcela libovolné místo) kus zdrojového textu
- příkaz GENS umožňuje nahrát zdrojový text ve ve formátu assembleru GENS (MASM)
- monitor chránící zdrojový text a ladící systém
- několik trasovacích režimů
- možnost definovat okna se zákazem zápisu, čtení nebo běhu
- možnost zcela ovlivnit složení, tvar a polohu čelního panelu, počet a též způsob výpisu každé položky
- při krokování umí monitor zjistit časovou náročnost krokovaného programu
- počítá cykly procesoru (tzv Téčka) - jako jediný

Tyto a další vlastnosti jsou detailně popsány v dalším textu.

II. Instalace ladícího systému:

Na kazetě je program uložen jako blok CODE dlouhý asi 18500 bytů - obsahuje 5000 bytů monitoru, 11000 bytů assembleru a 2500 bytů instalačního programu a tabulky relokačních adres.

Program lze nahrát na libovolnou adresu v rozmezí adres 24000 až 47000 a od této adresy spustit. Pro nahrání a spuštění programu použijte tyto příkazy. Místo adr vložte adresu, kde budete chtít mít ladící systém uložen. Pokud se s PROMETHEEM teprve seznamujete, nahrajte jej třeba na adresu 24000.

```
CLEAR adr-1: LOAD ""CODE adr: RANDOMIZE USR adr
```

Po spuštění se objeví instalační obrazovka obsahující následující informace:

Instalation address:xxxxx_ - kde xxxxx je adresa kam byl program nahrán - tuto adresu je možno přepsat pomocí kláves 0 až 9 a DELETE (CAPS SHIFT+0) na libovolnou adresu v rozsahu adres 23296 až 49500 (23296 až 54500 bez monitoru). Na tuto adresu bude PROMETHEUS umístěn a od ní ho bude možno z BASICU kdykoliv opět spustit. Umístíte-li program od adresy 23296, nelze se již vrátit do BASICU - program přepíše systémové proměnné. Budete-li používat instalaci na adresu pod nastavený RAMTOP, nelze se již vrátit do BASICU bezpečně. Umístíte-li ladící systém tak, že přepíše systémové proměnné (tzn. na adresu nižší než 23755), nesmí Vaše programy používat služby BASICU (tisk znaků atd.) a povolovat přerušení v módu IM 1. Obvykle je nejlépe použít pro instalaci adresu 25000.

Monitor:Yes - instalovat je možno buď celý ladící systém nebo jen assembler, změnu volby docílíte stisknutím klávesy M. Instalovat jen samotný monitor není možné protože používá mnoho podprogramů assembleru.

Dále lze nastavit barvy, které bude assembler používat. Je to barva papíru - klávesa P, barva inkoustu - klávesa I a jas těchto dvou barev - klávesa B. Při editaci je jeden řádek vyznačen jinou barvou - nastavení je obdobné, je nutné stisknout současné navíc CAPS SHIFT.

PROMETHEUS umí psát dvěma druhy písma - obvyklým a dvojnásobně tlustým (může být lépe čitelné) - přepínání mezi oběma typy písma klávesou D.

Při editaci vypisuje assembler (i disassembler) instrukce malými písmeny a návěští velkými písmeny, je-li však někdo zvyklý používat jen velká (jako GENS) nebo malá písmena (MRS), muže toto nastavení změnit klávesou C - výsledek je hned vidět na instalační obrazovce.

Poslední vlastnost ladícího systému, kterou je možno ovlivnit při instalaci je klávesnicové echo - mění se klávesou X jedním směrem a klávesami CAPS SHIFT + X druhým směrem - současné nastavení je slyšet při stisku libovolné klávesy.

Po nastavení požadovaných parametrů stiskněte klávesu ENTER, program se přesune na zvolenou adresu, přepíší se absolutní adresy (relokace) a program se spustí.

Tuto instalaci můžete provádět při každém použití ladícího systému (obvykle stačí jen stisknout ENTER) nebo si požadovanou konfiguraci vyhrát na pracovní kazetu - nejlépe následovně:

Připište krátký program v Basicu:

```
1 RANDOMIZE USR adr:STOP
```

```
9999 CLEAR adr-1:LOAD ""CODE:RUN
```

Kam místo adr a adr-1 dosadíte čísla (adresa kde je program) a na kazetu uložíte pomocí:

```
SAVE "PROMETHEUS" LINE 9999:
SAVE "prometheus" CODE adr,16000 (11000 bez monitoru)
```

kde opět místo adr bude adresa. Tuto konfiguraci nahrajete pomocí zcela obyčejného LOAD "" a z Basicu budete spouštět jednoduše příkazem RUN. První řádek z tohoto programu je dobré vložit vždy - PROMETHEUS se bude spouštět příkazem RUN a nehrozí chyba, která by mohla vzniknout při psaní RANDOMIZE USR adr, o škodlivosti funkce USR s chybnou adresou snad nemusím nikoho poučovat.

III. Assembler

PROMETEUS:

1. Editor:

Assembler obsahuje pohodlný řádkový editor, který se hodně podobá editoru BASICu. V editoru se zadávají také všechny příkazy - odpadá tedy nutnost neustálého přecházení z editoru do systému a naopak spojená se ztrátou pozice v editoru jako je tomu u MRSu a rozšířených verzí GENSu. Editor umožňuje blokové operace - mazání a přesouvání - a vyhledávání a nahrazování řetězce v textu.

Přístupový (přístupný) řádek - barevně odlišený řádek ve výpisu.

Editační řádek - úplně spodní řádek, je na něm kurzor.

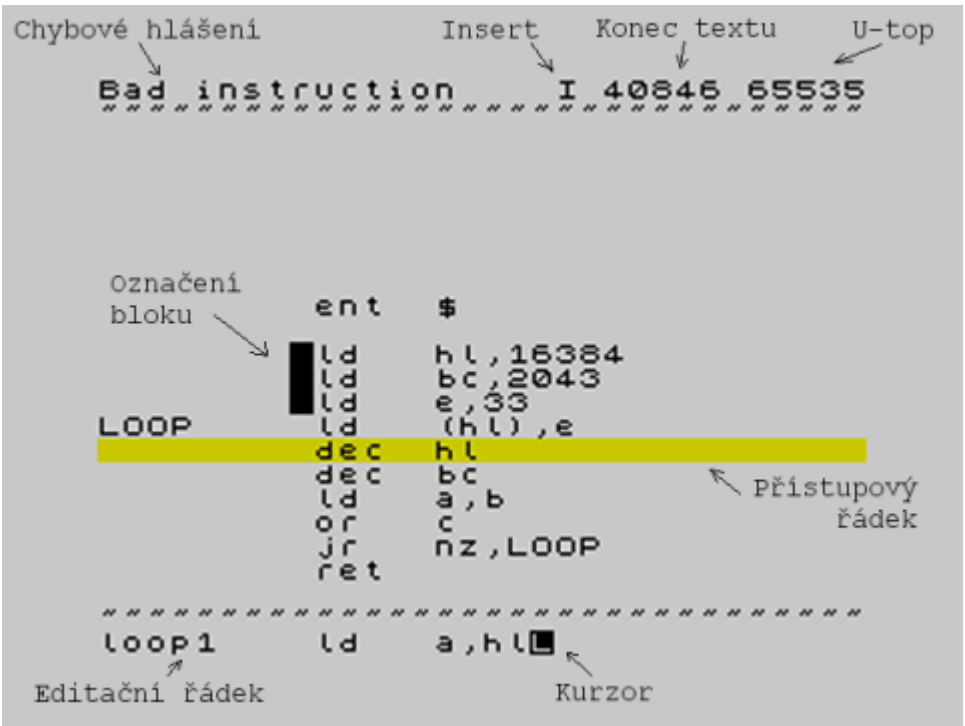
Stavový řádek obsahuje toto (zleva doprava):

• sdělení nebo chybové hlášení

• informaci o nastaveném režimu (Insert/Overwrite)

• adresu posledního bytu zdrojového textu

• adresu posledního bytu oblasti pro zdrojový text (obdoba RAMTOPu)



Následující obrázek vám ukazuje typickou obrazovku při práci s ladícím systémem:

Vkládání příkazů se provádí stisknutím klávesy SYMBOL SHIFT a odpovídajícího písmene při prázdném editačním řádku, tedy kurzor je úplně vlevo a za ním není žádný znak (ani mezera). Nejsou-li tyto podmínky splněny, napíše se znak na daně klávese se SYMBOL SHIFTem, jsou-li splněny, vypíše se celý příkaz (jako klíčové slovo) nebo se ihned provede (některé příkazy - skok na konec zdrojového textu atd.).

Zápis instrukce - na řádek lze zapsat maximálně 31 znaků, prvních 9 pozic je vyhrazeno pro návěští (maximálně 8 znaková), dalších 5 pozic je vyhrazeno pro mnemoniku a zbývající místa jsou vyhrazena pro operandy. Poznámky je možno psát pouze na zvláštní řádek a to hned od začátku za středník - nelze psát komentáře za instrukce (ony by se tam stejně nevešly). Pokud se někomu zdá 31 znaků málo, nezbývá mu než si buď zvyknout nebo používat jiný assembler. K tomuto omezení jsem dospěl na základě zkušeností svých i svých přátel, téměř nikdo, koho znám nepíše komentáře do zdrojového textu (maximálně na papír s programem), je to práce navíc a zdrojový text to neúměrně prodlužuje - a zpomaluje to překlad i nahrávání (to zvláště). Omezení má také své světlé stránky, zdrojový text je dobře čitelný a přehledný.

Mezerník automaticky tabeluje, instrukce je možno psát libovolně, malými nebo velkými písmeny, program rozliší návěští a zkontroluje vkládaný řádek, najde-li chybu, ohlásí ji a řádek nezařadí do výpisu.

Editační klávesy:

CAPS SHIFT + 1 (EDIT) - přenese přístupový řádek do editačního řádku.

CAPS SHIFT + 2 (CAPS LOCK) - zapíná/vypíná CAPS LOCK, ten pracuje trochu jinak než obvykle, původně píše Spectrum při zapnutém CAPS LOCKu pouze velká písmena, nyní funguje spíše jako jakýsi invertor CAPS SHIFTu, když je zapnut CAPS LOCK a není stisknut CAPS SHIFT píše Spectrum velká písmena, a je-li stisknut CAPS SHIFT, píše Spectrum malá písmena.

CAPS SHIFT + 3 (TRUE VIDEO) - přechod ve výpisu na následující stránku ve výpisu, na konci textu skočí na poslední řádek.

CAPS SHIFT + 4 (INV. V1DE0) - přechod ve výpisu na předcházející stránku ve výpisu, na začátku textu skočí na první řádek.

CAPS SHIFT + 5 (šipka doleva) - posun kurzoru doleva o jeden znak v editačním řádku, pokud je kurzor na levém okraji editačního řádku, nic se neprovede.

CAPS SHIFT + 6 (šipka dolů) - posun přístupového řádku na předchozí řádek ve výpisu, na konci textu neprovede nic.

CAPS SHIF'T + 7 (šipka nahoru) - posun přístupového řádku na následující řádek ve výpisu, na začátku textu neprovede nic.

CAPS SHIFT + 8 (šipka doprava) - posun kurzoru vpravo o jeden znak v editačním řádku, nepracuje na konci řádku.

CAPS SHIF'T + 9 (GRAPH) - vymaž přístupový řádek a posuň se na předchozí řádek.

CAPS SHIFT + 0 (DELETE) - vymaž znak vlevo před kurzorem a posun kurzor doleva.

SYMBOL SHIF'T + W - přepínání režimů INSERT a OVERWRITE (písmeno O nebo I ve stavovém řádku), při stisku EDIT se přepne režim OYERWRITE a po odeslání se vrátí zpět režim INSERT, chcete-li zkopírovat jeden řádek, vyeditujte ho, přepněte na 1NSERT a vložte na požadované místo, chcete-li úplně přepsat řádek, napište text, přepněte na OVERWRITE a odešlete.

SYMBOL SHIFT + Q - vymazání editačního řádku, ne při prázdném řádku.

CAPS SHIF'T + SYMBOL SHIFT (EXTEND MODE) - nastaví okraj bloku - příslušnost řádku k bloku je signalizována černým čtvercem na deváté pozici, mezi návěštím a instrukcí - označení je vidět pouze ve výpisu, nikoliv při hledání nebo tisku, není rozlišen začátek a konec, program si pamatuje vždy dva okraje a při zadání

nového odstraní ten, co byl zadán dříve a přidá nový okraj, chcete-li nastavit blok, nastavte jeden okraj do přístupového řádku, stiskněte CAPS SHIFT + SYMBOL SHIFT, nastavte druhý okraj bloku do přístupového řádku a stiskněte CAPS SHIFT + SYMBOL SHIFT.

ENTER - odeslání instrukce nebo příkazu, odeslání instrukce má za následek její syntaktickou kontrolu a zařazení do výpisu - podle režimu (I/O) se buď zařadí za instrukci v přístupném řádku nebo ji přepíše.

SYMBOL SHIFT + E - jdi na konec zdrojového textu, jde o příkaz, který se ihned provede, je ho tedy nutno zadat při prázdném editačním řádku.

SYMBOL SHIFT + K - jdi na začátek zdrojového textu, jde o příkaz, který se ihned provede, je ho tedy nutno zadat při prázdném editačním řádku.

SYMBOL SHIFT + H - přepínání výpisu čísel mezi desítkovou a šestnáctkovou soustavou, ovlivní výpisy ve stavovém řádku a v tabulce symbolů.

Příkazy pro blokové operace:

COPY (SYMBOL SHIF'T + C) - přenese nastavený blok před přístupový řádek - nelze kopírovat blok do sebe.

DELETE (SYMBOL SHIFT + D) - vymaže nastavený blok zdrojového textu.

FIND (SYMBOL SHIF'T + F) - vyhledávání (viz dále), nerozlišují se malá a velká písmena, není-li řetězec nalezen, zůstane výpis na místě.

REPLACE (SYMBOL SHIFT + Z) - zaměňování řetězců.

Vyhledávání řetězců:

FIND s:xxxxx - vyhledání řetězce "xxxxx" od začátku zdrojového textu.

FIND s - vyhledání naposledy zadaného řetězce od začátku zdrojového textu.

FIND b:xxxxx - vyhledání řetězce "xxxxx" v bloku od jeho začátku.

FIND b - vyhledání naposledy zadaného řetězce v bloku od jeho začátku.

FIND :xxxxx - vyhledání řetězce "xxxxx" od řádku, který je právě přístupný buď do konce zdrojového textu nebo do konce bloku - toto v případě, že posledně byl prohledáván blok.

FIND - vyhledání naposledy zadaného řetězce od řádku, který je právě přístupný bud do konce zdrojového textu nebo do konce bloku, opět to závisí na posledním způsobu prohledávání.

FIND b:_ - tímto příkazem se lze dostat okamžitě na začátek bloku (za dvojtečku nutno zapsat jednu mezeru!).

Příklady:

- chci vyhledat všechny výskyty návěští "KAREL", zadám "FIND s:karel" (nerozlišují se malá a velká písmena), po nalezení prvního výskytu zad.~n pokaždé jen "FIND" pro nalezení dalšího, jakmile se po zadání "FIND" jen obnoví výpis na stejném místě, jedná se o poslední výskyt řetězce "KAREL".

- chci se opět dostat na první výskyt řetězce "KAREL", zadám jen "FIND s" a dostanu se na první výskyt.

- chci vyhledat výskyt návěští "KAREL" pouze v polí návěští (toto je nejrychlejší způsob> jak se dostat ve výpisu na požadované místo), zadám "FIND s:KAREL " - za návěští vložím mezeru!

- chci vyhledat první instrukci ld a,(hl) od začátku zdrojového textu, zadám příkaz "FIND s:ld a,(hl)", mezi mnemoniku a operandy musím vložit tolik mezer, aby jejich počet plus počet písmen mnemoniky dal součet 5.

REPLACE :xxxxx - nahradí všechny výskyty řetězce "yyyyy" vyhledaného příkazem "FIND :yyyyy" v daném řádku řetězcem "xxxxx" a vyhledá další výskyt řetezce "yyyyy".

REPLACE - nahradí všechny výskyty řetězce, který byl zadán příkazem FIND, řetězcem, který byl zadán příkazem REPLACE.

Příklad:

- chci najít a vyměnit některé výskyty řetězce "KAREL" řetězcem "JOSEF", zadám příkaz "FIND s:karel", po nalezení výskytu řetězce "KAREL" udělám:

- bud zadám "REPLACE :JOSEF" (podruhé a dole už jen "REPLACE") když budu chtít výskyt nahradit.

- nebo zadám "FIND" když budu chtít tento výskyt ponechat beze změny, což opakuji tak dlouho dokud neprojdu všechny výskyty řetězce "KAREL".

Mazání zdrojového tetu a nastavení prostoru pro něj:

CLEAR (SYMBOL SHIFT + X) - příkaz vymaže zdrojový text a vyčistí tabulku symbolů (ponechá v ní jen uzamčené symboly - viz kapitolu 4. Tabulka symbolů), protože jedná o "nebezpečný" příkaz a protože si autor sám omylem několikrát vymazal zdrojový text, je nutné ho proto nyní psát ve tvaru "CLEAR y", šance, že člověk omylem vloží příkaz a písmeno "y" je malá, pokud je vložen příkaz CLEAR bez písmene "y" (nebo "Y"), nic se neprovede.

U-TOP (SYMBOL SHIFT + U) - uživatelská zarážka, toto je poslední adresa, kam smí být uložen zdrojový text a kam smí být provedena kompilace (kap. 3. Překlad). Za příkazem U-TOP musíte uvést číslo nebo matematický výraz (bude vyhodnocen zleva doprava, může obsahovat čísla i existující definovaná návěští (jinak chyba), tento příkaz lze použít i jako kalkulačku, výsledek je trvale vidět vpravo nahoře, k nastavení zarážky na adresu 65535 můžete použít i příkaz "U-TOP -1", pomocí uživatelské zarážky získáte chráněnou oblast paměti, kam můžete uložit třeba tiskové podprogramy a jiné, pro assembler i monitor je oblast od zarážky až do konce paměti zakázaná a nepřístupná.

Chyby při editování:

Bad mnemonic - chyba v zápisu mnemoniky. Neexistující mnemonika nebo také návěští zapsané v poli mnemoniky.

Bad operand - v instrukci použit chybný operand.

Big number - použité číslo je větší než 65535.

Syntax horror - syntaktická chyba, některá položka je delší než může být, dvě operace ve výrazu vedle sebe, ...

Bad string - špatně zapsaný řetězec, uvozovky se píše do řetězce dvojitě, řetězec nesmí být prázdný.

Bad instruction - instrukce zapsána formálně správné, použitá instrukce však nemá možnost pracovat s touto kombinací operandů.

Memory full - paměť přidělená pro zdrojový text je vyčerpána, můžete se pokusit zkrátit zdrojový text (viz kapitolu 6. Formát zdrojového textu).

XXXXX unknown - chyba při použití návěští, které v tabulce symbolů není nebo tam je ale nemá v současné době definovánu žádnou hodnotu - příkaz U-TOP, ve kterém se objevilo návěští XXXXX.

Source ERROR - tato chyba je hlášena, když program zjistí, že došlo k poškození zdrojového textu - to se může stát jen vnějším zásahem - program, jenž byl spuštěn příkazem RUN přepíše kus zdrojového textu. většinou není možné jiné řešení, než opět nahrát do paměti PROMETHEUS (když je chyba ve zdrojovém textu, je skoro jisté, že byl poškozen i program) a znovu zdrojový text, a proto: Před každým použitím příkazu RUN po zásahu do zdrojového textu je lépe uložit si zdrojový text na kazetu. Po vložení zdrojového textu do paměti (ručně z klávesnice) je to nutné ještě více.

2. Magnetofonové operace:

Assembler PROMETHEUS umožňuje (kdyby ne, bylo by to na pováženou) ukládat, kontrolovat a číst zdrojové texty (í jejich části) z magnetofonu.

SAVE (SYMBOL SHIFT + S) – ukládání zdrojového textu:

Příkaz určený pro uložení zdrojového textu na kazetu.

SAVE :karel - uloží celý zdrojový text (zdrojový text a tabulku symbolů) na kazetu, blok bude uložen jako BASICovské CODE pod jménem „karel“.

SAVE - uloží celý zdrojový text na kazetu, jako jméno bude vzato naposledy použité jméno - toto se hodí jestliže při verifikaci došlo k chybě a je nutno opakovat opět ukládání.

SAVE b:karel - uloží na kazetu část zdrojového textu (nastavený blok) a celou tabulku symbolů - vybírání návěští, která jsou ve zvoleném bloku použita, by bylo časově a paměťově náročné - proto chcete-li na kazetu uložit opravdu jen část zdrojového textu, musíte vymazat všechno ostatní a potom vyčistit tabulku symbolů (viz kapitola 4, tabulka symbolů).

SAVE b - uloží zvolenou část zdrojového textu se jménem naposledy použitým v příkazu SAVE nebo LOAD.

LOAD (SYMBOL SHIFT + L) – čtení zdrojového textu:

Příkaz sloužící k přihrání zdrojového textu z kazety. Chcete-li nový zdrojový text pouze nahrát, musíte ten starý nejprve vymazat příkazem "CLEAR y" - SS + X

LOAD : - nahraje první nalezený zdrojový text z kazety do počítače.

LOAD :karel - nahraje do počítače zdrojový text se jménem "karel".

LOAD - nahraje do počítače zdrojový text se jménem, které bylo naposledy použito v příkazech LOAD nebo SAVE.

Použijí-li se místo jména samé mezery, bude to bráno jako kdyby nebylo zadáno žádné jméno a načte se první nalezený blok CODE.

Nahrávání probíhá následovně:

1. prohledává se kazeta dokud se nenajde blok CODE *se správným jménem.*

2. po nalezení takového bloku se zjistí, zda se blok vejde do volné paměti, pokud ne, je hlášena chyba Memory full.

3. blok se nahraje na konec paměti určené pro zdrojový text (adresa posledního bytu je napsána vpravo ve stavovém řádku).

4. zjisti-li se. že se nejedná o zdrojový text tohoto assembleru, příkaz je přerušen.

5. začne st vkládání nahraného zdrojového textu do zdrojového textu, který je v paměti od minula, při této operaci se každý nový řádek převede na textový tvar a vloží do původního textu - vkládá se stejně jako při vkládání z klávesnice - za přístupný řádek, bude-li při vkládání nalezena chyba, objeví se chybové hlášení a je možno chybu opravit, po opravě program pokračuje ve vkládání, při přihrání bloku se do paměti musí vejít současně dvě tabulky symbolů, stará (ta se zvětšuje o nová návěští) a nová, ze které se berou návěští při převodu řádků na textový tvar - tyto podrobnosti Vás nemusí zajímat, pokud se překládá za zdrojový text (nejčastější situace), pak problémy s místem nastanou jen těžko.

6. stav vkládání lze zobrazit stiskem libovolné klávesy.

7, stiskne-li se při vkládání klávesa SPACE, vkládání se ukončí a v editačním řádku bude právě vkládaná instrukce. Odešlete-li řádek zpět, bude v textu zdvojen, proto stisknete-li tuto klávesu: nechtěně, vymažte editační řádek a stiskněte ENTER

8. pokud dojde při vkládání k chybě a v editačním řádku je chybná instrukce, lze vkládání ukončit vložením nějakého příkazu nebo povelu (třeba přepnutím mezí desítkovou a šestnáctkovou soustavou).

9. vkládání bude chvíli trvat - po dobu vkládání je na obrazovce nápis Wait please".

VERIFY (SYMBOL SHIFT + V) – ověření nahrávky:

Příkaz sloužící ke kontrole na pásce uloženého textu, je ho nutné provést ihned po příkazu SAVE, nepotřebuje žádné parametry, použije jméno z příkazu SAVE a testuje blok pouze s tímto jménem.

GENS (SYMBOL SHIFT + G) – konverze z GENSu:

Příkaz slouží k přihrání zdrojového textu ve formátu assembleru GENS (MACROS), syntaxe je úplně stejná jako u příkazu LOAD, zdrojový text nahraje a vkládá stejně jako u příkazu LOAD, před vkládáním zdrojového textu z GENSu je výhodné text nejprve upravit - zkrátit řádky tak, aby odpovídaly formátu PROMETHEA, přemístit možné komentáře na zvláštní řádky atd. - je s tím méně práce než opravovat řádky až při vkládání.

1e-li konec prostoru pro zdrojový text nastaven na adresu 65535 (#FFFF), nebude rozeznán konec souboru a vkládání se zastaví na chybě - dojde k pokusu vzít jako řádek zdrojového textu několik prvních bytů z ROM, přerušte vkládání vložením nějakého povelu - viz bod 8. Chcete-li se tomu vyhnout, nastavte konec paměti pro zdrojový text třeba na adresu 65500 (příkaz U-TOP).

Pokud dojde při kazetových operacích ke stisku klávesy BREAK (SPACE), v editačním řádku zůstane naposledy zadaný povel. Pokud dojde v příkazu LOAD, GENS nebo VERIFY k chybě, vypíše se chybové hlášení "Tape error" a opět je v editačním řádku posledně zadaný příkaz. .

Pokud budete chtít do assembleru vložit zdrojový text z jiného assembleru, musíte ho převést na formát assembleru GENS, každý řádek začíná dvěma byty číslo řádku (pro PROMETHEUS není podstatné), potom text řádku (stačí vždy jedna mezera místo několika) a na konci řádku je kód 13, takto upravený zdrojový text vyhrajte z paměti jako CODE a můžete ho příkazem GENS dostat do PROMETHEA. Další použitelný způsob (možná jednodušší) je popsán v kapitole 5. Ostatní (viz GENSOR).

3. Překlad:

ASSEMBLY (SYMBOL SHIFT + A) - příkaz provede kompilaci zdrojového textu. Za příkazem ASSEMBLY je možno uvést parametr "b" (nebo "B") - v tomto případě bude kompilován pouze nastavený blok zdrojového textu.

Při kompilaci neexistuje kompilační výpis - tato možnost byla pro jednoduchost a zvýšení rychlosti odstraněna, upřímně řečeno autorovi (a nejen jemu) se zdála zbytečná. Pokud se chcete dozvědět, na jaké adrese je která instrukce (to byla jediná námitka proti odstranění kompilačního výpisu), napište tam nějaké návěští (pokud už tam není a po skončení kompilace se podívejte do tabulky symbolů na jeho hodnotu (viz kapitola 4. tabulka symbolů), lze také samozřejmě použít disassembler monitoru. Potřebujete-li z nějakých důvodů (?) protokol o překladu, přeneste zdrojový text do nějakého jiného assembleru (GENS pomocí programu GENSOR - viz kapitolu 5. ostatní - GENSOR) a s jeho pomocí ho získáte.

Kompilace je prováděna rychlostí přibližně 2 KB strojového kódu za sekundu tedy ani dlouhý zdrojový text není překládán déle než 3-4 sekundy, u krátkých zdrojových textu je kompilace takřka okamžitá.

Kompilace je dvouprůchodová, při prvním průchodu se pouze zjišťují hodnoty návěští v poli návěští (nikoliv u příkazu EQU), při druhém průchodu se již generuje strojový kód, nyní musí být známa všechna návěští. Ukládání strojového kódu do paměti je kontrolováno tak, aby nebyl přepsán ladící systém, zdrojový text a oblast paměti nad uživatelskou zarážkou.

Možná chybová hlášení:

Bad PUT (ORG) - při druhém průchodu kompilace došlo k pokusu o přepsání ladícího systému, zdrojového textu nebo oblasti nad uživatelskou zarážkou generovaným strojovým kódem.

XXXXX unknown - použité návěští "XXXXX" nemá dosud přiřazenu žádnou hodnotu.

Already defined - pokus o druhé definování hodnoty návěští.

Big number - pokus o zapsání velkého čísla do 8 bitů, u příkazů relativních skoků to znamená, že vzdálenost je příliš velká, u instrukcí typu (ix+d) smí být d v rozsahu -128 až 127, příkazy typu ld a,N smí být N v rozsahu -256 až 255 (podrobněji dále).

Dojde-li k chybě při kompilaci, překlad se zastaví a obnoví se výpis tak, že řádek, na kterém došlo k chybě je nyní přístupný (jedná-li se o chybu "Already defined" tak ono podruhé definované návěští je samozřejmě v polí návěští, nikoliv v poli operandů). Pokud kompilace skončí úspěšně, vypíše se hlášení "Assembly complete".

RUN (SYMBOL SHIFT + R) - příkaz slouží ke spuštění programu od místa zadaného příkazem ENT (Entry point – viz. dále), před spuštěním programu se vždy provede kompilace - příkaz zahrnuje příkaz ASSEMBLY (překlad je stejně velmi rychlý a není třeba zadávat dva příkazy), pokud nebude nalezen právě jeden příkaz ENT, vypíše se chybové hlášení "ENT ?". Také tento příkaz může obsahovat parametr "b" - před spuštěním se přeloží pouze zvolený blok zdrojového textu. Před spuštěním se vymaže obrazovka, nastaví se barvy, které používá ladící systém, a po návratu se čeká na stisk libovolné klávesy a potom se opět vyčistí obrazovka a obnoví výpisy. Pokud jste někdy pracovali s programem GENS, víte, že tato část příkazu RUN je značné zpříjemnění práce - nemusíte koukat, jak vám výsledky proběhnuvšího programu poodjedou z obrazovky pryč nebo jak na vás svítí špatně nastavené atributy. Strojový kód je spouštěn se zakázaným přerušením, registr IY není PROMETHEEM používán a jeho hodnota se nemění, na zásobníku je asi 100 bytů místa.

Instrukce:

PROMETHEUS překládá všechny základní instrukce (známé) a také většinu tzv. "neznámých" instrukcí - jediné "neznámé" instrukce, které assembler nezná jsou všechny rotace na adrese (ix+d) s následným přenosem do zvoleného registru, pokud budete chtít tyto instrukce použít, musíte je zapsat pomocí pseudoinstrukce DEFB (instrukce nebyly zařazeny protože se v praxi nepoužívají a zvětšily by délku assembleru). Následuje seznam těch "neznámých" instrukcí, které assembler používá:

Instrukce, které používají poloviny indexregistrů - IX,IY. Dolní polovina registru IX je značena LX, horní polovina bude označena HX, obdobně pro registr IY je LY a HY.

inc	inc hx	inc ix	inc hy	inc ly
dec	dec hx	dec lx	dec hy	dec ly
ld	ld hx,N	ld ix,N	ld hy,N	ld ly,N
	ld b,hx	ld b,lx	ld b,hy	ld b,ly
	ld c,hx	ld c,lx	ld c,hy	ld c,ly
	ld d,hx	ld d,lx	ld d,hy	ld d,ly
	ld e,hx	ld e,lx	ld e,hy	ld e,ly
	ld hx,hx	ld hx,lx	ld hy,hy	ld hy,ly
	ld lx,hx	ld lx,lx	ld ly,hy	ld ly,ly
	ld a,hx	ld a,lx	ld a,hy	ld a,ly

add	add a,hx	add a,lx	add a,hy	add a,ly
adc	adc a,hx	adc a,lx	adc a,hy	adc a,ly
sub	sub hx	sub lx	sub hy	sub ly
sbc	sbc a,hx	sbc a,lx	sbc a,hy	sbc a,ly
and	and hx	and lx	and hy	and ly
xor	xor hx	xor lx	xor hy	xor ly
or	or hx	or x	or hy	or ly
cp	cp hx	cp lx	cp hy	cp ly

Rotace SLIA (shift left inverted arithmetic -investovaný aritmetický posun doleva) - posune doleva zvolený registr, vystupující bit je uložen do CARRY, vstupující bit je jednička.

slia	slia b	slia c	slia d	slia e
	slia h	slia l	slia (hl)	slia a

U instrukcí relativních skoků a instrukce DJNZ se jako u ostatních assemblerů píše absolutní adresa. Při kompilaci se vypočte relativní adresa a uloží se, pokud je relativní adresa mimo rozsah -128 až 127, ohlásí se chyba "Big number".

Assembler používá některé pseudoinstrukce - slouží bud k zápisu čísel a textů do paměti nebo k ovládání ukazatele kompilace, umístění strojového kódu a k nastavování startu pro příkaz RUN a hodnot návěští - následuje seznam:

ORG výraz - nastaví čítač adres a ukazatel uložení do paměti na hodnotu výrazu. Ukazatel uložení do paměti určuje, kam se bude ukládat strojový kód, čítač adres určuje kde bude přeložený program pracovat - hodnota tohoto čítače se používá při dosazování hodnot návěští při prvním průchodu. Ukazatel uložení do paměti i čítač, instrukcí se vždy zvětšují o délku pravě přeložené instrukce. Pokud tento příkaz nepoužijete, bude se program překládat a ukládat ihned za zdrojový text.

PUT výraz - nastaví ukazatel uložení do paměti na hodnotu výrazu. Tento příkaz se používá v případě, že je potřeba uložit strojový kód jinam než bude spouštěn. Chcete třeba přeložit program tak, aby pracoval v obrazovkové paměti tedy, od adresy 16384, protože překládat přímo do obrazovky není možné, vložte za příkaz ORG 16384 ještě příkaz PUT 60000 - program se bude překládat tak, aby pracoval od adresy 16384 ale ukládat se bude od adresy 60000. Nelze přehodit pořadí, ve kterém jsou obě pseudoinstrukce uvedeny, protože příkaz ORG nastavuje také ukazatel uložení do paměti! Příkaz PUT byl zaveden jako alternativa k možnosti ukládat přeložený program za tabulku symbolů bez ohledu na pseudoinstrukci ORG, kterou poskytuje assembler GENS (MASM), toto řešení je však podstatně přizpůsobivější, programátor se nemusí starat, kde tabulka symbolů končí a může si sám nastavit, kam se bude překlad ukládat. Chcete-li vyrobit program, který se po spuštění sám rozmístí po paměti, můžete to vyrobit třeba takto:

```
                org 60000
START           ld hl,START1        ;adresa, kde je 1. blok uložen
                ld de,34000         ;adresa, kde má 1. blok pracovat
                ld bc,BLOK1LEN      ;délka 1. bloku
                ldir                ;přesuň 1. blok
                ld de,35000         ;adresa, kde má 2. blok pracovat
                ld bc,BLOK2LEN      ;délka 2. bloku
                ldir                ;přesuň 2. bloku
                jp 30000            ;skok na začátek 1. bloku
END1
                org 30000           ;první blok bude na adrese 30000
                put END1            ;jeho kód se ukládá stále dále
START1          ???                 ;zde je text prvního bloku
                ???
BLOK1LEN        equ $-START1        ;BLOK1LEN bude obsahovat délku
END2
                org 35000           ;druhý blok bude na adrese 35000
                put END2            ;kód se ukládá stále dále
START2          ???                 ;zde je text druhého bloku
                ???
BLOK2LEN        $-START             ;BLOK2LEN bude obsahovat délku
```

Návěští EQU výraz - přiřadí použitému návěští hodnotu výrazu. V okamžiku, kdy dochází při druhém průchodu ke zpracování této instrukce už musí být známé hodnoty všech návěští, která jsou použita ve výrazu, jinak dojde k chybě.

```
LABEL1    equ LABEL2+1    ;takhle to bude fungovat, protože
LABEL2                    ;návěští LABEL2 bylo definováno
                          ;už při prvním průchodu překladu
LABEL1    equ LABEL2+1    ;takhle to bohužel fungovat nebude
LABEL2    equ 32000       ;protože návěští LABEL2 ještě nemá
                          ;definovanou hodnotu
```

ENT výraz - tento příkaz slouží k nastavení adresy, od které bude spuštěn strojový kód v případě, že bude použit příkaz RUN.

```
          ent START       ;program bude spuštěn od místa, na
                          ;němí se nachází návěští START
                          ;v poli návěští
          ent $           ;program bude spuštěn od místa, na
                          ;kterém se vyskytuje tento příkaz
```

DEFB výraz,výraz,..,výraz - uloží od adresy, na kterou ukazuje ukazatel uložení do paměti, výsledky výrazů jako osmibitové hodnoty - musí ležet v rozmezí -256 až 255, jinak je hlášena chyba "Big number". Potom jsou zvýšeny hodnoty čítače adres a ukazatele uložení do paměti o počet bytů, které byly příkazem DEFB naplněny (počet výrazů).

DEFW výraz,výraz,..,výraz - podobné jako příkaz DEFB, ukládají se šestnáctibitové hodnoty (napřed nižší byte a potom vyšší byte). Ukazatel uložení do paměti a čítač adres se zvětší o počet bytů, které byly naplněny (2*počet výrazu).

DEFM řetězec - uložení řetězce do paměti, tento příkaz byl u assembleru rozšířen:

```
          defm "karel"    ;uloží do paměti postupně ASCII
                          ;kódy jednotlivých písmen
          defm 'karel'    ;provede téměř totéž, poslední
                          ;znak bude mít ASCII hodnotu větší
                          ;o 128 - investovaný znak, pokud
                          ;netušíte, k čemu je to dobré.
                          ;podívejte se na příklady programů
```

DEFS výraz - vyhodnotí se výraz a o jeho hodnotu se zvětší ukazatel uložení do paměti a čítač adres. Tato pseudoinstrukce se používá k vynechání místa o požadované délce.

```
BUFFER    defs 100      ;vynechání 100 bytů
BUFFER2   defs 1000     ;vynechání 1000 bytů
```

Výrazy:

Assembler umožňuje vložit na místo, kde je v instrukci číselná hodnota, matematický výraz. Matematický výraz se skládá z jednoho nebo několika návěští či konstant oddělených od sebe operátory. Všechna čísla jsou převedena na rozsah 0-65535 (záporná čísla jsou přičtena k číslu 65536 a tím jsou převedena na rozsah 0-65535), v tomto tvaru jsou provedeny operace a výsledek je použit podle potřeb instrukce. Výpočet je prováděn zleva doprava, není zachovávána priorita operátorů. Pokud není možno výraz zapsat tak, aby ho bylo možno provádět zleva doprava, musíte ho rozepsat na dva řádky a mezivýsledek přiřadit nějakému návěští a to použít na dalším řádku, autor se s touto eventualitou ještě nesetkal - nejčastější výrazy používají pouze jeden operátor.

Konstanty:

Jsou povoleny tyto konstanty:

desítkové:	1234	47	-32
šestnáctkové:	#ABCD	-#30	#a3
dvojkové:	%10111	%0111	-%011010
znakové:	"a"	"aB"	"""" (uvozovka se musí vložit dvojitě)

Je možnost odkázat se na aktuální hodnotu čítače adres:

$ - na toto místo bude dosazena hodnota čítače adres

```
START       ;zde je část textu, jejíž
LENGHT      ;délku potřebujeme zjistit
????        ;a vložit do návěští LENGHT
equ $-START ;toto je na konci oné části
```

Návěští:

Návěští je posloupnost čísel, písmen a znaku "_", která začíná písmenem. Znaků lze použít maximálně 8 a pro identifikaci jsou důležité všechny.

Aritmetické operátory:

Sčítání	+
odčítání	–
násobení	*
dělení	/
zbytek po dělení	?

Chcete-1i získat dolní byte návěští TAB, vložte TAB?256, horní byte získáte pomocí výrazu TAB/25b.

Důležité upozornění: při dělení nulou není hlášena chyba a výsledek je 65535!

4. Tabulka symbolů:

Možnosti, které poskytuje PROMETHEUS při práci s tabulkou symbolů, jsou jedním ze znaků, které ho výrazně odlišují od ostatních assemblerů.

Tabulka symbolů existuje neustále a je vždy možné prohlédnout si abecedně seřazený výpis návěští a jejich hodnot, pokud jim byly nějaké přiřazeny. Neustálá existence tabulky symbolů se může zdát nevýhodná - je to však vlastnost, která umožňuje výrazné zkrácení zdrojového textu a také zrychlení kompilace.

Příkazy pro práci s tabulkou:

TABLE (SYMBOL SHIFT + T) - tento příkaz vypíše tabulku symbolů na obrazovku, zobrazí se jedna stránka (40 návěští) a očekává se stisk nějaké klávesy, pokud to bude klávesa SPACE, vypisování tabulky se ukončí, pokud to klávesa SPACE nebude, vypíše se další stránka. Návěští jsou vypsána ve dvou sloupcích, před návěštím je vypsána bud mezera nebo hvězdička (její význam je popsán dále), za návěštím je vypsána hodnota, která byla návěští přiřazena při posledním překladu, pokud mu žádná hodnota přiřazena nebyla, je vypsáno pět teček. Hodnota návěští je vypsána v nastavené číselné soustavě (desítkové či šestnáctkové viz kapitola 1. Editor). Možnost přečíst si tabulku symbolů lze použít, zajímají-li Vás adresy některých částí programu nebo když si nejste jisti (u dlouhých textů se to občas stane), která návěští jste už použili a která ne.

TABLE p - provádí výpis tabulky symbolů na obrazovku a také na tiskárnu, pro podrobnosti se podívejte na příkaz PRINT. Na každý řádek se vytiskne jedno návěští a jeho hodnota, jedna stránka obrazovky se tedy vypíše na 40 řádků papíru.

TABLE c - vyčištění tabulky symbolů, z tabulky symbolů se odstraní ta návěští, která se nevyskytují ve zdrojovém textu a nejsou uzamčena. Při vymazání řádku nebo bloku zdrojového textu se z tabulky symbolů neodstraní ta návěští, která se vyskytovala pouze ve vymazané části zdrojového textu - tato akce může být časově náročnější a nevyplatí se ji provádět neustále protože by mohla zdržovat, také několik bytů za to nestojí. Zbytečná návěští jsou vidět když si po provedení kompilace vypíšete tabulku symbolů - nemají definovanou hodnotu - pokud jejich počet vzroste, tak použijte příkaz pro vyčištění tabulky symbolů. Pokud nemáte problémy s volnou pamětí (pak je každý byte dobrý) můžete na tento příkaz zapomenout, při každém přečtení zdrojového

textu z kazety (tuto akci je nutno provádět obvykle několikrát za den - při ladění programů ve strojovém kódu není nouze o stavy, kdy se zkoušený program odmítá vrátit zpět, případně se rovnou po spuštění vymaže) se ve zdrojovém textu zachovají jen použitá návěští.

Oddělená kompilace – kompilace po částech:

TABLE l - uzamkne všechna návěští v tabulce symbolů, tato návěští se při kompilací chovají tak, jako by byla stále definována, příkaz TABLE c tato návěští neodstraní. Popsání smyslu tohoto příkazu následuje.

TABLE u - odemkne všechna zamčená návěští v tabulce symbolů, opak příkazu TABLE 1 (písmeno L, nikoliv číslice 1).

Dva předchozí příkazy jsou určeny pro umožnění oddělené kompilace důvodem může být velký rozsah zdrojového textu. Zdrojový text je nutno rozdělit na jakýsi základ (pomocné podprogramy - tiskové rutiny, práce s obrazovkou, data,..) a na hlavní část (toto rozdělení může být i několikavrstevné); důležité je, aby se základ (nižší úroveň) neodkazovala na návěští v hlavní části (vyšší úroveň). Nyní se odladí základ, zamkne tabulka symbolů a vymaže zdrojový text. Po vymazání zdrojového textu zůstanou v paměti všechna návěští, která obsahoval a je možno se na ně odvolávat v dalším zdrojovém textu.

Tuto možnost Vám snad lépe osvětlí následující příklad:

V příkladu je program rozdělen na tři vrstvy, v první vrstvě jsou uloženy texty, druhá část obsahuje program, který vytiskne text, jehož adresa mu bude zadána a poslední (nejvyšší) vrstva bude volat tisk textu.

```
                org 65000
TEXT1           defm "Zkusebni " ;toto je první část programu,
                defm 'text c.l'  ;není v ní žádný odkaz na
TEXT            defm "Pokusny "  ;návěští ve vyšší vrstvě
                defm 'text c.2'
```

Tuto část programu zkompilujte (ASSEMBLY, zamkněte tabulku symbolů (TABLE l) a vymažte zdrojový text (CLEAR y).

```
                org 64000
PRINT           ld a,(hl)        ;toto je druhá část programu,
                and 127          ;odkaz na nižší úroveň v ní
                rst 16           ;sice není, mohl by však být,
                ld a,(hl)        ;odkaz do vyšší úrovně se
                inc hl           ;zde vyskytovat nesmí
                and 128
                jr z,PRINT
                ret
```

Nyní opět zkompilujte zdrojový text (ASSEMBLY, uzamkněte opět tabulku symbolů (TABLE l), nyní jsou v ní již tři návěští, a opět vymažte zdrojový text (CLEAR y).

```
                org 63000
MAIN            ld a,2           ;poslední a nejvyšší úroveň
```

```
call #1601      ;programu, může se odkazovat
ld hI,TEXT1     ;i na návěští definovaná
call PRINT      ;v obou nižších úrovních
ret
```

Po vložení poslední části zdrojového textu jsou v tabulce symbolů návěští TEXT1, TEXT2, PRNIT a MAIN. První tři návěští jsou uzamčena.

Je zřejmé, že v uvedeném případě nemá rozdělení tohoto krátkého zdrojového textu na kusy smysl, ilustruje však způsob, jakým lze zpracovávat i dlouhé zdrojové texty.

Tento způsob práce byl použit při ladění monitoru, z kompilace zdrojového textu byla zachována tabulka symbolů a přeložený kód, do paměti byl vkládán zdrojový text monitoru a laděn - tímto způsobem bylo možno odladit rozumným způsobem program dlouhý 11 KB, odpovídající zdrojový text by byl dlouhý celkem asi 40 KB a nevešel by se za žádných okolností spolu s assemblerem do paměti, o potřebě místa pro přeložený kód ani nemluvě.

Při kompilaci jednotlivých vrstev je vhodné nižší vrstvy chránit tím, že se uživatelská zarážka nastaví pod ně, nehrozí nebezpečí, že již hotové části budou přepsány zdrojovým textem nebo generovaným strojovým kódem.

Při každém zahájení práce je nutno opět zkompilovat všechny vrstvy, aby se do tabulky symbolů dostala potřebná návěští, a vyplatí se tedy spíše vyhrát na kazetu celou paměť, tedy ladící systém i se zdrojovým textem a již přeloženými vrstvami programu.

Může se to zdát, že se nahrává zbyteční velký rozsah paměti, ale z časových důvodů je to lepší, než při zapnutí počítače provádět některé akce vždy znovu, navíc tento postup umožňuje mít z paměti i další programy a data a nemuset je pokaždé nahrávat z deseti různých kazet po dlouhém hledání a převíjení kazet z jednoho konce na druhý, takto popsaný způsob jsem si nevycucal z palce ale sám jsem ho používal při práci na monitoru k tomuto ladícímu systému - assembler už byl plné funkční v této době. Můžete použít následující modifikaci tohoto způsobu:

1. odlaďte si základní vrstvu.

2. pro ladění další vrstvy si vyrobte jakousi instalaci, jenž bude obsahovat krátký zaváděcí BASIC program a dva bloky CODE - první bude tvořit ladící systém spolu s tabulkou symbolů vzniklou odladěním základní vrstvy, druhý pak přeložený kód a případně nějaká data nebo cokoliv jiného - rozdělení je proto, že mezi ladícím systémem a přeloženým kódem je obvyklá velká mezera a nemá smysl nahrávat několik KB nul, to ovšem neplatí když překládáte před ladící systém a ne za něj.

Zaváděcí DASIC může vypadat třeba takto:

```
1 RANDOMIZE USR 24e3:STOP
9999 CLEAR 23999:LOAD ""CODE:LOAD ""CODE:RUN
```

Odpovídající bloky dostanete na kazetu pomocí příkazů:

```
SAVE "Instalace" LINE 9999:
SAVE "Code 1" CODE 24e3,konec-23999:
SAVE "Code 2" CODE začátek,délka
```

kde, "konec" je konec zdrojového textu (první číslo ve stavové řádce, "začátek" a "délka" jsou čísla určující polohu a rozsah přeložené nižší vrstvy a případně dat. V tomto příkladu je assembler umístěn na adrese 24000. Pro uložení instalace na kazetu lze také výhodně použít kompresní programy z kompletu USER II, který připravujeme.

3. v instalaci si můžete také nastavit monitor podle svých požadavků a nemusíte jej pak pokaždé znovu nastavovat.

4. po odladění další vrstvy si vyrobíte opět další instalaci - a takto pořád dokola, dokud není program celý hotov.

Můžete samozřejmě také základní vrstvu přeložit a z dalších vrstev se odkazovat už přímo na adresy, to má nejméně dvě nevýhody - při používání monitoru se nebudete moci odkazovat na návěští v již hotové vrstvě (nebudou v tabulce) a v případě, že se rozhodnete pro změnu některé části v nižší vrstvě, budete muset přepsat každý odkaz na adresy ve vyšší vrstvě protože se tyto adresy posunou, použijete-li výše popsaný postup, stačí vyrobit si novou instalaci s novými hodnotami v tabulce symbolů a vyšší vrstvu nemusíte měnit (pokud ovšem nezměníte nějak více základní vrstvu - jiná návěští nebo jiný smysl).

Nutnost použití odděleně kompilace se však díky značnému zkrácení zdrojového textu (proti GENS je poloviční odsouvá až na hranici kolem 6-8 KB generovaného strojového kódu, kratší strojový kód lze ladit najednou - jaké to má výhody snad není potřeba moc zdůrazňovat.

5 Ostatní příkazy a možnosti:

Doposud nebyly uvedeny některé příkazy, které lze v assembleru používat, zde je jejich celkový přehled:

PRINT (SYMBOL SHIFT + P) - tento příkaz slouží k vytištění zdrojového textu na tiskárnu nebo plotter. Program bere jednotlivé řádky a posílá je znak za znakem do kanálu systému Spectra #3 - kanál pro tiskárnu. Každý řádek je ukončen kódem 13 (ENTER) a nejsou v něm použity žádné tabelátory - mezery jsou opravdu tištěny všechny. Pokud v tomto příkazu použijete parametr "b", bude vytištěn pouze nastavený blok. Chcete-li vytisknout tabulku symbolů, použijte příkaz "TABLE p". Provádění lze zastavit stiskem klávesy SPACE. Jakým způsobem si připojíte svou tiskárnu na kanál #3 je vaše věc - stačí, když použijete program, který "umí“ provádět LPRINT.

MONITOR (SYMBOL SHIFT + M) - příkaz slouží k přechodu do monitoru (viz dále). Použijete-li parametr "a", provede se před spuštěním monitoru kompilace zdrojového textu - je to výhodnější než zadávat příkazy ASSEMBLY a MONITOR postupně. Pokud jste si při instalaci monitor odpojili, spuštění se neprovede.

BASIC (SYMBOL SHIFT + B) - vrátí řízení do systému Spectra, do BASICU. Pokud se Vám však podařilo přepsat některé systémové proměnné Spectra (program pracující s obrazovkou Vám "utekl" a prošel přes tuto oblast) použijte pro návrat do BASIC raději příkaz NEW. Pro nové spuštění PROMETHEA použijte adresu, na které je instalován - při návratu je assembler ve stejném stavu, v němž jste ho opustili (pokud jste ho ovšem z BASIC nějak "nevylepšili"). Při návratu se nastaví potřebné hodnoty do registrů IY a SP, zapne přerušeni v módu IM 1 a provede se skok do hlavní prováděcí smyčky (bude se interpretovat další příkaz), při tomto návratu není potřeba nastavovat registr HL' !

NEW (SYMBOL SHIFT + N) - příkaz provede stejnou akci jako v BASIC příkaz NEW - vymaže celou paměť až po nastavený RAMTOP. Do assembleru byl zařazen protože při trasování nebo při běhu se muže podařit přepsat systémové proměnné BASICU a hrozí, že se při návratu pomocí příkazu BASIC systém zhroutí. Pokud nevíte, zda se nepřepsala také systémová proměnná RAMTOP, podívejte se na její hodnotu (je to hodnota na adresách 23730 a 23731) a případně ji opravte předtím, než použijete příkaz NEW (toto lze provést

bud v monitoru nebo tak, že vyrobíte krátký program, který to provede a ten spustíte příkazem RUN (z assembleru):

```
org 23296
ent $
ld hl,23999
ld (23730),hl
ret
```

Program přeložíte a spustíte příkazem "RUN b" - předtím si ho nastavte jako blok - potom můžete bez obav provést příkaz NEW. Pokud používáte rutiny z ROM Spectra, které přepisuji systémové proměnné, je někdy lepší používat pro návrat do systému vždy příkaz NEW - po skončení vašeho programu totiž nemusí být v tom stavu, v jakém by být měly.

QUIT (SYMBOL SHIFT + Q) - pokud bude tento příkaz použit tak zaručeně jako poslední, k použití jiných už nebudete mít příležitost - provádí RESET (RANDOMIZE USR 0).Tímto příkazem pouze elegantně ukončíte práci a zcela vyčistíte počítač. Protože tento příkaz je mimořádně nebezpečný a jeho nechtěné použití by mohlo vyvolat u uživatele silnou depresi (už se stalo), je nutno za něj přidat parametr "y", pouze takto se provede.

Gensor:

Způsob, jak dostat zdrojový text z programu GENS do tohoto assembleru byl již zmíněn, může nastat také opačná potřeba, k tomuto účelu byl vytvořen krátký program GENSOR.

GENSOR používá ke své práci systém Spectra - kanál #3, avšak místo do tiskárny ukládá znaky do paměti ve stejném tvaru, jaký používá program GENS (MASM). Pro Vaši informaci uvedu, jak vypadá jeden řádek v assembleru GENS:

1. dva byty obsahují číslo řádku

2. následuje text proložený maky CHR$(9), které znamenají posun na další tabelační pozici a nahrazují znak mezery. Tyto znaky v textu být nemusejí, stačí, když je na jejich místě alespoň jedna mezera - takto například pracuje MONS při zpětném překladu ze strojového kódu do zdrojového textu.

3. každý řádek je ukončen znakem: CHR$(13) - ENTER.

Program GENSOR se skládá z krátké části v BASICU a přibližně z 200 Bytů strojového kódu. Ke své činnosti potřebuje, aby byla oblast paměti od adresy 55000 až ke konci prázdná - sem se ukládá zdrojový text ve formátu assembleru GENS (MASM).

Nahrajte do paměti PROMETHEUS (můžete navíc odpojit monitor aby bylo více místa) a instalujte ho třeba na adresu 25000. Nahrajte do assembleru zdrojový text - pokud by přesahoval adresu 55000, budete ho muset rozdělit. Jestliže je adrese konce zdrojového textu menší než 55000 můžete nahrát program GENSOR příkazem LOAD "GENSOR". Po nahrání přepíšete v příkazu RANDOMIZE USR adr, proměnnou "adr" skutečným startem assembleru (nyní 25e3).

Nyní můžete zadat příkaz GO TO 0 (ne RUN!), provede se inicializace a program GENSOR se připojí na tiskový kanál #3, potom se provede spuštění assembleru.

Nastavte začátek bloku na začátek zdrojového textu a konec dejte o maximálně 30 stránek dál, nyní zadejte příkaz, "PRINT b" - zadaný blok půjde místo na tiskárnu do programu GENSOR, který ho ukládá za sebe - omezení na 30 stránek je dáno velikostí paměti, do které se může zdrojový text ukládat (kdyby byly jednotlivé

řádky zdrojového textu delší než je obvyklé - data nebo texty vyplňující celý řádek - vložte raději konec bloku dříve než po 30 stránkách). Zadejte příkaz BASIC a nahrajte právě vytvořenou část zdrojového textu na kazetu, po návratu do assembleru nastavte další část zdrojového textu a opakujte dokud celý zdrojový text není nahrán na kazetě.

Tento způsob přenesení zdrojového textu do programu GENS můžete použít i u jiných assemblerů, máte-li zdrojový text ve formátu pro GENS, můžete ho snadno přenést do PROMETHEA. Tento způsob však nelze použít u těch assemblerů, které jsou umístěny nad adresou 55000 – např. MRS (zde by bylo nutno modifikovat program GENSOR tak, aby používal pro svou práci jinou oblast paměti.

6. Formát zdrojového textu:

Tato kapitola je určena všem, kteří by se rádi dozvěděli jak vypadá zdrojový text, jakým způsobem je možno ho ještě zkrátit (v mezích možností samozřejmě) a proč vlastně dosahuje assembler v některých ohledech takové výkony (rychlost a délka či spíše úspornost uložení zdrojového textu) proti jiným assemblerům.

Na počátku byly tyto myšlenky hlavní myšlenky:

- k tomu, aby se dala instrukce zcela jednoznačně rozlišit, stačí jejich operační kód, informace o prefixech a tvaru případného číselného operandu, tedy informace, které lze zapsat do dvou bytů, toto řešení výrazně urychlí překlad, bude stačit jen brát operační kódy, přidávat prefixy, nebo vyhodnocovat výrazy a dosazovat jejich hodnoty do místa pro číselné operandy - nebude potřeba zjišťovat operační kódy a hlídat syntaxi příkazů při kompilaci - tyto akce se provádějí už při editování (v této době čeká počítač na člověka a má dost času aby tyto akce provedl) - vedlejším efektem pak je, že už při psaní zdrojového textu dojde k vychytání spousty chyb a překlepů.

- návěští ve zdrojovém textu mohou být uložena jen jako odkazy do tabulky symbolů, tyto odkazy zaberou jen dva byty, u návěští dlouhých alespoň tři znaky vzniká úspora - tabulka symbolů se bude vytvářet již při psaní zdrojového textu, nebude tedy potřeba vytvářet ji až při překladu, také každý odkaz na návěští bude nasměrován při editaci a nebude potřeba při překladu tabulku prohledávat.

Nyní šlo o to, jak to provést s nejmenším možným nárokem na čas - třeba jedna stránka výpisu vlastně znamená jakési disassemblování 20 instrukcí - o rychlosti, s jakou se to provádí se můžete přesvědčit sami (provádí se po každém odeslání instrukce).

Každý řádek zdrojového textu má tvar:

první byte je operační kód instrukce, jsou zde ale také pseudoinstrukce a další "neinstrukce" jako je prázdný řádek a komentář, ty mají také přiřazeny své kódy, aby se odlišily od obyčejných instrukcí, mají v druhém bytu řádku (informační byte) takovou kombinaci prefixů, kterou nemá žádná instrukce, je kombinace prefixů #DD a #FD, a v posledních třech bitech je uloženo číslo 7 (mimo prázdný řádek, ten tu má uloženu 0), hodnota informačního bytu závisí také na tom, jestli je nebo není před instrukcí či neinstrukcí návěští nebo ne - pro signalizaci této skutečnosti je určen 3 bit informačního bytu, když je návěští použito, je zde zapsána 1, odpovídající hodnota je zapsána v závorce, následuje přehled kódů "neinstrukcí":

"Neinstrukce"	Operační kód	Informační byte	
prázdný řádek	0	48	(56)
komentář	1	55	
ent	2	55	(63)
equ	3		(63)

org	4	5S	(63)
put	5	55	(63)
defb	6	55	(63)
defm	7	55	(63)
defs	8	55	(63)
defw	9	55	(63)

- druhý byte je informační byte, obsahuje tyto údaje:

7 bit - #CB prefix (203), 0 ne, l ano

6 bit - #ED prefix (237)

5 bit - #DD prefix (221)

4 bit - #FD prefix (251)

3 bit - v poli operandů je návěští 2-0 bit - typ číselného výrazu podle

tohoto se dosazuje výsledek výrazu, význam hodnoty je následující:

0 - Instrukce nemá číselný operand

1 - jeden byte, rozsah -256 až 255

2 - dva byty, mezi -65536 až 65535

3 - jeden byte, rozsah -128 až 128

4 - jeden byte, typu (ix+d)

5 - typ (ix+d),n

6 - rst p (p se vloží do op. kódu)

7 - neinstrukce

Uvedené rozsahy je možno používat v textu

- další byty jsou použity jen v případě, že je 3 bit informačního bytu nastaven na 1 nebo když poslední tři bity informačního bytu neobsahují každý nulu, tvar je následující:

1. bit 3 je 1, bity 0 až 2 jsou všechny 0

potom v dalších dvou bytech je zapsáno pořadí daného návěští v tabulce a na konci je číslo 192+2 (délka).

2. bit 3 je 0, bity 0 až 2 nejsou jenom 0

v dalších bytech je zapsán celý výraz, návěští jsou nahrazena svým pořadím v tabulce symbolů, ostatní znaky jsou zapsány zcela obvyklým způsobem, na konci je číslo 192+délka výrazu.

například výraz 2*LABEL+#23

je zapsán takto:

"2","*",128+H,L,"+","#","2","3",192+8

kde H a L jsou horní a dolní byte pořadí návěští LABEL v tabulce symbolů.

výraz (2*LABEL+#23) bude uložen také takto, jestli se jedná nebo nejedná o adresu se zjistí z operačního kódu.

3. bit 3 je 1, bity 0 až 2 nejsou jenom 0

uložení bude stejné jako v předchozím případě, před výrazem budou dva byty obsahovat pořadí návěští a délka bude o dva byty větší.

Tyto informace Vám při vlastním programování nebudou naprosto k ničemu, nicméně plynou z nich způsoby, jak zkrátit zdrojový text a přitom nezměnit jeho smysl.

Možnosti zkrocení zdrojového textu:

Dostane-li se programátor do situace, že potřebuje několik desítek bytů paměti a tyto nejsou k dispozici, pokusí se zkrátit zdrojový text tak, aby byl při stejném obsahu kratší. Několik způsobů, které zaručeně vedou k výsledku:

1. Vyčistěte tabulku symbolů od zbytečných návěští příkazem "TABLE c".

2. Odstraňte komentáře a prázdné řádky.

3. Pokud se ve zdrojovém textu vyskytují příkazy DEFB nebo DEFW za sebou, snažte se dostat co nejvíce výrazů do jednoho příkazu (bude méně operačních kódů a informačních bytů) – např:

```
            defb 255
přepisem    defb 0      na defb 255,0,32 ušetříte 4 byty
            defb 32
```

4. Zvykněte si, pokud používáte texty s posledním invertovaným znakem, používat rozšíření příkazu DEFM - sekvenci:

```
            defm "Invertovany tex"      ;v GENSu to jinak
            defb "t"+128                ;nelze udělat
```

lze zapsat také takto: defm 'Invertovany text' , je to přehlednější a kratší (zvyk bývá železná košile).

5. U konstant použijte desítkovou soustavu, stejná čísla lze zapsat nejkratším způsobem, je to sice méně přehledné, ale dá se s takto ušetřit mnoho - hlavně u binárního zápisu, počet znaků potřebný na zápis stejného čísla je různý, třeba kód mezery je možno zapsat:

```
32          v desítkové soustavě
#20         v šestnáctkové soustavě
" "         jako znak
% 100000    ve dvojkové soustavě
```

zápis v desítkové soustavě je viditelně ze všech nejkratší. Také je vhodné místo 2+3 psát rovnou 5.

6. Zkraťte návěští, o kolik znaků zkrátíte návěští, o tolik se zkrátí tabulka symbolů a tedy i celý zdrojový text, vyplatí se ovšem zkracovat větší množství návěští najednou, používáte-li třeba návěští LOOP s číslem, pak je vhodné nahradit stejnou část kratší sekvencí - třeba LP - zde si však dejte dobrý pozor abyste prohozením nezpůsobili, že dříve různá návěští budou nyní stejná, po prohození návěští samozřejmě musíte vyčistit tabulku symbolů.

7. Vyskytuje-li se ve zdrojovém textu některá konstanta velmi často a je-li dlouhá alespoň 3 znaky, je vhodné její hodnotu přiřadit některému návěští a nahradit ji všude tímto návěštím - tento postup není dobrý jen pro zkrácení ale i pro zpřehlednění a budete-li chtít konstantu změnit, stačí ji změnit na jednom místě a ne všude! Je-li tato konstanta použita v příkazu DEFB (DEFW) a použijete-li jednopísmenné návěští, může se Vám vejít na jeden řádek vícekrát než původní konstanta.

Zajímavé je, že (moje zkušenosti) zdrojový text u dlouhého programu je obvykle tak dlouhý, jak jen může být. V závěrečné fázi psaní je nutno používat některé z uvedených způsobů, jak získat ještě alespoň 100 bytů navíc. Tato skutečnost se projevila už při psaní assembleru PROMETHEUS - assembler lze překládat najednou, ale nezbude téměř ani byte volný - tehdy jsem to považoval za náhodu. Později se mi podobná situace zopakovala při psaní ORFEA, BAD DREAMu a DESKTOPU (2 kusy zdrojového textu). Ani kolega při tvorbě SCREEN TOPu nebyl ušetřen stejného zjištění. Po konfrontaci našich empirických poznatků jsme dospěli k názoru, že jde zřejmé o dosud neznámý přírodní zákon, který by se dal volně vyjádřit slovy: Co se může naplnit, to se také naplní. To je hezké, má to jednu chybu, že k zaplnění dojde obvykle poněkud předčasně - zde však jde o obyčejný zákon schválnosti, který nikdy nezklame.

Uvedené způsoby nejsou doporučením jak psát zdrojové texty, vedou totiž až na některé výjimky k horší přehlednosti. Je to způsob, jak se vyhnout nutností použít oddělenou kompilaci. Pokud se Vám podaří zjistit, že se zdrojový text nevejde do paměti už v polovině prací, nezbude Vám než ho rozdělit, začnou-li problémy s pamětí až na konci, stojí za pokus zkrátit zdrojový text - obvykle se to podaří - stále je to lepší než se zabývat dělením zdrojového textu a používat oddělenou kompilaci.

Tabulka symbolů:

Tabulka je uložena za zdrojovým textem a má následující tvar:

Čítač	dva byty	počet návěští v tabulce
	dva byty	odkazy do vlastní tabulky, počet
	.....	odkazů je roven počtu návěští,
tabulka	.....	v horním bytu odkazu je v bitech
vektory	.....	č. 6 a 7 uložena informace o
	.....	definování nebo uzamčení daného
	dva byty	návěští
	dva byty, jméno	dva byty obsahují hodnotu daného
vlastní	.....	návěští, poslední znak jména je
tabulka	.....	invertován, identifikátory jsou
	.....…	abecedně seřazeny, jejich počet
	dva byty, jméno	je roven počtu návěští

Podrobnosti editoru:

Na tomto místě se dozvíte o způsobu, kterým probíhá převádění řádku z textového tvaru do tvaru, ve kterém je uložen. Nejdříve se oddělí návěští v poli návěští (pokud tam nějaké je), potom se rozdělí instrukce na mnemoniku a dva operandy (operand může být i prázdný), zjistí se o jaký typ mnemoniky a operandů se jedná a prohledá se tabulka, jestli taková instrukce existuje. Při prohledávání operandů se nejprve hledají registry a když se nenajdou, snaží se assembler vyhodnotit operand jako výraz, tento způsob umožňuje, že je možno použít jako návěští zcela libovolnou kombinaci znaků, nejsou zde žádná rezervovaná slova pouze u některých slov (jména registrů a dvojregistrů) je nutno použít takový zápis, aby byt vyhodnocen jako výraz:

Např. chcete použít jako návěští řetězec "SP" - tento řetězec by se vyhodnotil jako registr SP, napíšete-li však SP+0, bude vše v pořádku.

```
ld hl,sp   ;takto zadaný řádek bude vyhodnocen jako chyba
ld h1,sp+0      ;při tomto zadání už bude
                ;přijat a slovo "sp" bude ;vyhodnoceno jako návěští
ld hl,+sp       ;pokud to zadáte takhle, dojde také k přijetí, zmizí však
                ;znaménko "+" a při vyeditování a zpětném vložení bude opět
                ;hlášena chyba (totéž při LOAD)
```

Poslední poznámka se týká používání závorek pro označení adresy - pokud je v závorce číselná hodnota (výraz), stačí zadat jen tu pravou, levá závorka bude doplněna. Pokud však totéž uděláte a v závorce má být registr (např. (hl)), bude slovo "hl" vyhodnoceno jako návěští, nikoliv jako registr - zde je nutno závorku uzavřít. Použití závorek při uzávorkování výrazu není povoleno!

```
ld a,(hl)       ;toto se vezme jako ld a,(hl)
ld a,(hl        ;toto bude vyhodnoceno jako ld a,(HL), tedy HL, je návěští !
ld a,(label     ;toto stačí k tomu aby to bylo bráno jako ld a,(LABEL)
```

III. Monitor PROMETHEUS:

Monitor PROMETHEUS byl vytvořen tak, aby vhodně doplnil možnosti, které poskytuje assembler PROMETHEUS. Nabízí mnoho funkcí, které usnadňují ladění programů. Při návrhu ovládám byla zachována shoda s programem VAST od T.R.C prakticky ve všech shodných funkcích - uživatelé tohoto programu jistě rádi tuto skutečnost ocení. (platí i pro DEVAST ACE).

Po vstupu do monitoru se vymaže obrazovka a vypíše čelní panel, pokud nebyl čelní panel předefinován, bude vypadat takto:

Na obrázku vidíte v horní části výpisové okno (jsou v něm celkem tři výpisy - disassembling, znakový a číselný výpis obsahu paměti), pod tím jsou vypsány dva řádky instrukcí od aktuální adresy a ve spodní čisti pak hodnoty vybraných registrů, několik adres na zásobníku a informační/editační řádek .

Po vypsání čelního panelu je očekáván stisk nějaké klávesy případně více kláves, po stisknutí klávesy se zjistí, odpovídá-li stisknutá klávesa nějaké funkci, pokud ano, přejde řízení do této funkce, pokud ne, opět se čeká na stisk klávesy.

1. Přístup k paměti:

Návrat do systému assembleru - klávesa Q:

Po stisku klávesy Q se provede návrat do assembleru, po návratu bude assembler ve stejném stavu, v jakém byl opuštěn, toto neplatí v případě, že byla prováděna zpětná kompilace - v assembleru bude změněný zdrojový text.

Nastavení aktuální adresy - klávesa M:

Po stisknutí klávesy M se ve spodním řádku objeví otázka Memory a kurzor, nyní je možno vložit matematický výraz (pravidla jsou stejná jako v assembleru, ten bude vyhodnocen zleva doprava, v případě, že dojde při vyhodnocování výrazu k chybě, zazní zvukový signál a do editačního řádku se na několik sekund vypíše chybové hlášení, potom se opět objeví editační řádek s textem, který v něm byl při odeslání a je umožněna oprava chyby. Pokud chcete přerušit vkládání, stiskněte klávesu EDIT (CAPS SHIFT + 1), řízení se vrátí do hlavní smyčky. Hodnota výrazu bude přiřazena aktuální adrese.

Posun na další instrukci - klávesy CAPS SHIFT + 6:

Při použití této volby se zjistí délka instrukce na aktuální adrese a aktuální adresa se zvětší o tuto délku. Po provedení se obnoví výpis čelního panelu.

Posun o jeden byte - klávesa ENTER:

Aktuální adresa bude zvětšena o jedničku, po provedení se obnoví výpis čelního panelu. Pokud je aktuální adresa nastavena na hodnotu 65535, bude mít po stisku klávesy ENTER hodnotu 0 - počítá se modulo 65536.

Posun o jeden byte zpět - klávesy CAPS SHIFT + 7:

Odečtení jedničky od aktuální adresy, po provedení se obnoví výpis čelního panelu. Ukazuje-li aktuální adresa na 0, pak po provedení této operace bude ukazovat na 65535.

Vnoření o jednu úroveň - klávesy CAPS SHIFT + 8:

Při prohlížení programu ve strojovém kódu se často objeví nutnost podívat se do nějakého podprogramu a neztratit přitom současné místo - v prohlíženém podprogramu mlže dojít ke stejné potřebě. Proto existuje funkce vnořování - po zvolení této funkce se objeví dotaz "Memory", současná hodnota aktivní adresy se uloží na zvláštní zásobník (kapacita 10 adres) a aktuální adresa je nastavena vloženou hodnotu. Pokud je zásobník plný, příkaz se neprovede. Obnoví se výpis čelního panelu.

Vynoření o jednu úroveň - klávesy CAPS SHIFT + 5:

Vynoření je opak funkce vnoření. Funkce odebere ze zásobníku hodnotu a nastaví na ni aktuální adresu, pokud na zásobníku není žádná adresa, nic se neprovede. Obnoví se výpis čelního panelu.

Smazání obrazovky - klávesy CAPS SHIFT + 0:

Tato funkce smaže obrazovku, nastaví barvy, které používá assembler (i monitor) a obnoví se výpis čelního panelu.

Smazání výpisového okna - klávesy CAPS SHTFT + 2:

Smazání části obrazovky, která je určena pro výpisy - tato část obrazovky je volitelný počet řádků - viz 12. Editor čelního panelu. Maže se pouze pixelová část obrazovky, attributy zůstávají původní (pokud chcete smazat i attributy, použijte předchozí funkci).

2. Podprogramy a breakpoint:

Volání podprogramu - klávesu SYMBOL SHIFT + H:

Po stisku klávesy se objeví dotaz "Call" na adresu, na které má být volán podprogram. Po vložení adresy (zadávejte velmi pozorně, pokud se zmýlíte, může to mít neodstranitelné následky) se nastaví všechny registry na hodnoty uvedené v čelním panelu, podle stavu indikátoru se provede zavolání podprogramu bud s povoleným nebo zakázaným přerušením. Mód přerušení bude takový, jaký byl nastaven naposledy - ladící systém pracuje pod zakázaným přerušením a nemění nastavený mód, pouze po návratu do systému BASIC je nastaveno IM 1 (příkazy BASIC, NEW). Po návratu z podprogramu se uloží všechny registry, zjistí se jestli při návratu bylo nebo nebylo povoleno přerušení a obnoví se výpis čelního panelu. Při volání se správně ukládá také hodnota registru R. Při použití tohoto příkazu si ověřte, zda je registr SP nastaven na správné místo - nesmí být samozřejmě nastaven do oblasti paměti ROM (1-16385), takto se na zásobník neuloží návratová adresa (ROM nelze přepsat) a program se nevrátí na správné místo!

Nastavení startu pro BREAKPQINT - klávesa W:

Tato funkce vezme hodnotu aktuální adresy a uschová ji pro potřebu běhu s BREAKPOINTem - viz následující funkce.

Běh s pomocí BREAKPOINTu - klávesu SYMBOL S. + U:

Funkce vyzvedne tři byty na aktuální adrese, uschová je a na jejich místo vloží skok do monitoru. Potom se nastaví všechny registry na hodnoty uvedené v čelním panelu, povolí nebo zakáže se přerušení a provede skok na adresu zadanou stiskem klávesy W. Pokud program proběhne místem s BREAKPOINTem, vrátí se zpět do monitoru, budou uloženy všechny registry, zjištěn stav přerušení a na své místo se opět uloží odebrané tři byty. Z uvedeného je zřejmé, že tato funkce není použitelná v paměti ROM. Opět je správně zpracován registr R. U této instrukce není potřebné hlídat hodnotu SP, důležité je pouze to, aby procesor při provádění programu na BREAKPOINT narazil.

Následující příklad vložte jako zdrojový text do assembleru a přeložte ho na adresu 50000. Místo instrukce call SUBROUT vložte instrukci ret. Co bude program dělat jste asi poznali sami - pokud ne, pak uvidíte při jeho zkoušení.

Příklad:

```
49999   ....
50000   ld hl,16384     ;na tomto místě stiskněte W
50003   ld bc,6144      ;počet průchodů
LOOP    ld a,(hl)
50005   cpi
50006   ld (hl),a
50007   inc hl
50008   dec bc
```

```
50009     ld a,b
50010     or c
50011     jr nz,LOOP
50013     call SUBROUT     ;na tomto místě stiskněte klávesy
50016     ....             ;SYMBOL SHIFT + U
```

Označení "na tomto místě" znamená, že v okamžiku stisknutí klávesy W bude aktuální adresa rovna 50000 a v okamžiku stisku kláves SYMBOL SHIFT + U bude aktuální adresa rovna 50013.

Varovný příklad:

Při vybírání místa pro vložení BREAKPOINTU kontrolujte důsledně, aby BREAKPOINT (3 byty) nepřepsal jinou část programu, kterou bude běh požadovat:

```
50000     ld hl,43210      ;na této instrukci stiskněte W
50003     call PROCEDUR    ;někdo uprostřed programuje volání
50006     ret              ;na této instrukci má být konec

PROCEDUR  ld a,(hl)        ;kdyby byl použit BREAKPOINT
50008     inc hl           ;na adrese 50006, přepíší se
50009     ret              ;také adresy 50007 a 50008 což nelze
                           ;potřebovat a je tedy nutno zvolit jiný
                           ;způsob projdení této části programu
```

Tento případ není příliš častý, je však nutno jej vždy zvážit. Může se totiž stát zdrojem naprosto nepochopitelných věcí - někdy se program zhroutí, mnohem horší mohou být nepochopitelné rozdíly mezi tím, co by program měl dělat a mezi tím co dělá.

Ještě bych se zmínil o tom, proč není BREAKPOINT udělán stejně jako u M.R.S., jeho autoři použili jiný způsob (zdánlivě lepší), který nemá výše uvedený nedostatek. Použili totiž instrukci RST 16, která je dlouhá jeden byte, a vlastnosti ROM na Spectru. Toto řešení má však některé vady - nezachovává hodnoty záložních registrů, potřebuje k činnosti přítomnost systémových proměnných a neumožňuje v takto testovaných programech použití tisku znaků právě pomocí RST 16.

Použití BREAKPOINTu si důkladně rozmyslete, není zrovna nejbezpečnější a může Vám práci nepříjemně prodloužit. Pokud to není nezbytné, používejte místo BREAKPOINTu raději rychlé trasování.

3. Magnetofonové operace:

Uložení bloku na kazetu - klávesa S:

Assembler umožňuje ukládat na kazetu jak bezhlavičkové bloky, tak bloky CODE (se standardní hlavičkou jako v BASICU). Postup je následující:

- stiskněte klávesu S

- na dotaz "First" vložte adresu prvního bytu bloku, který má být nahrán na kazetu.

- na dotaz "Last" vložte poslední adresu zvoleného bloku - na dotaz "Leader" můžete odpovědět dvěma způsoby:

vložením čísla - v tomto případě bude na kazetu uložen bezhlavičkový blok, jako leader (značkový byte) bude použito toto zadané číslo, nahrávání se provádí hned po odeslání, nečeká se na další stisk klávesy.

vložením dvojtečky a jména (použije se prvních 10 znaku, ostatní budou odděleny stejně jako v BASICU) - v tomto případě bude vytvořena hlavička, ve které bude uložen začátek bloku (First), jeho délka (Last-First+l) a zadané jméno, po stisku klávesy ENTER se počká na stisk nějaké klávesy a pak se začne ukládání bloku na kazetu (magnetofon musíte spustit sami).

Uložení bloku na kazetu - klávesy SYMBOL SHIFT + S:

Tento příkaz pracuje stejně jako předcházející příkaz, pouze místo dotazu na "Last" se ptá na "Lenght", tedy délku ukládaného bloku - zadání parametrů bloku je takto úplně stejné jako v BASICU. Chcete-li například uložit přeložený program na kazetu, provedete to nejlépe takto:

- na začátek programu vložte návěští A0START - písmeno A a číslice 0 jsou tam proto, aby při výpisu tabulky symbolů nebylo třeba toto návěstí hledat - bude hned na začátku.

- za konec zdrojového testu programu vložte sekvenci

```
A0LENGHT        equ $-A0START
```

(písmeno A a číslo 0 jsou tu ze stejných důvodů jako u návěští A0START)

- proveďte kompilaci příkazem "MONITOR a" a v monitoru zvolte nyní popisovanou funkci, na dotaz "First" vložíte A0START a na dotaz "Lenght" A0LENGHT

- návěští A0LENGHT Vám může poskytovat informaci jak dlouhý je přeložený strojový kód, která není nikdy na škodu

Nahrání bloku z kazety do počítače – klávesa J:

Funkce se zeptá na adresu, kam se má blok nahrávat - dotaz "First", na poslední adresu nahrávaného bloku - dotaz "Last", a na značkový byte nahrávaného bloku - dotaz "Leader". Potom se nahraje první blok, který bude nalezen, v případě, že nebude splňovat parametry, které byly zadány, nebo bude zjištěna chyba v paritě bude ohlášeno chybové hlášení "READ/WRITE Error". Tato chyba bude hlášena také v případě, že blok bude zasahovat ladící systém nebo zdrojový text - další provádění se přeruší ještě před zadáním značkového bytu.

Nahrání bloku z kazety - klávesy SYMBOL SHIFT + J:

Funkce pracuje jako funkce předešlá, liší se jen tím, že se místo dotazu "Last" používá dotaz "Lenght".

Přečtení hlavičky nebo značkového bytu – klávesa Y:

Provádí se čtení hlavičky z kazety - vypíše se typ bloku (0 je BASIC, 1 je číselné pole, 2 je znakové pole a 3 je CODE), potom jméno bloku, jeho počáteční adresu, jeho délku a další informaci, jejíž význam závisí na typu bloku. Po vypsání informací se očekává stisk klávesy, pokud se jedná o klávesu J, bude nahrán blok podle údajů zjištěných z hlavičky. Jinak se vrací řízení do hlavní smyčky.

4. Paměťové výpisy:

Výpis disassembleru - klávesy SYMBOL SHIFT + 4:

Po stisku klávesy se do výpisového okna vypisují disassemblované instrukce. Výpis lze přerušit stiskem klávesy EDIT (CAPS SHIFT + 1). Počáteční adresa výpisu je právě nastavená aktuální adresa.

Výpis disassembleru od zadané adresy - klávesa V:

Příkaz se dotáže na adresu, od níž má být disassemblování prováděno, otázkou "First". Další akce jsou stejné jako v předchozím příkazu.

Způsob výpisu adres - klávesy SYMBOL SHIFT + C:

Při vypisování hodnot dvoubytových číselných operandů může disassembler pracovat ve třech režimech, přepínána těchto režimů je cyklické:

1. vypíše se běžným způsobem číselná hodnota

2. pokud se hodnota, která má být vypsána, rovná hodnotě některého návěští v tabulce symbolů, vypíše se místo čísla toto návěští, stejně tak se návěští vypisuje místo adresy před instrukcí.

3. vypisuje návěští stejné jako v režimu č. 2, navíc pokud se hodnota zmenšena o jednu rovná hodnotě některého návěští, bude vypsáno: "návěští+l".

Po instalaci ladícího systému je nastaven vypisovaní režim č. 3

Povolení/zákaz výpisu adres - klávesa C:

Monitor umožňuje zakázat výpis adres před instrukcemi, tento zákaz se netýká návěští, která mohou být vypisována před instrukcemi při zvolení režimů 2 a 3.

Změna číselné s soustavy - klávesy SYMBOL SHIFT + 3:

Změna číselné soustavy se týká všech čísel vypisovaných ve všech typech paměťových výpisů, netýká se ovšem výpisu registrů v čelním panelu - toto lze ovlivnit v editoru čelního panelu.

Výpis disassembleru na tiskárnu - klávesa D:

Program umožňuje posílat výpis disassembleru na tiskárnu, komunikace se provádí stejně jako v assembleru přes kanál č.#3 operačního systému Spectra. Při zvolení této funkce se program zeptá na první "First" a na poslední "Last" adresu úseku, který má být vypsán na tiskárnu - instrukce začínající na poslední "Last" adrese už vypsána nebude. Vypisování lze přerušit stiskem kláves CAPS SHIFT + ENTER po každé instrukci, každý řádek je pro kontrolu také vypisován do výpisového okna.

Zpětný překlad do zdrojového textu - klávesy SYMBOL SHIFT + D

Funkce se podobá předchozí, výpis strojového kódu směřuje místo na tiskárnu do zdrojového textu v assembleru - vkládání se provádí za přístupový řádek (zcela jako při např. příkazu LOAD). Opět dotaz na "First" a "Last", instrukce počínající na adrese "Last" se také do zdrojového textu nedostane. Pokud dojde k chybě, ohlásí se chybové hlášení a uživateli je dána možnost nalezenou chybu opravit, bude-li hlášena chyba "Memory full", nezbude než stiskem EDIT (CAPS SHIFT + 1) celou akci přerušit. Chcete-li převod předčasně ukončit, stiskněte současně klávesy CAPS SHIFT + ENTER. Při zpětném překladu se opět každý řádek vypisuje do výpisového okna. Více Vám napoví příklad - viz přílohy.

Znakový výpis paměti – klávesa O:

Protože při prohlížení obsahu paměti nepotřebujeme vždy vidět jen strojový kód, umožňuje monitor vypisování čísel v paměti jako znaků. Každý řádek výpisu obsahuje adresu a dvacet pět znaků od této adresy. Znaky s kódy v rozsahu 32 až 127 se vypisují obvyklým způsobem, kódy v rozsahu 0 až 31 jsou nahrazeny tečkou. Pokud je kód znaku větší než 127, odečte se číslo 128 a znak se vypíše stejně jako předtím, pouze bude prohozena barva papíru a barva inkoustu - inverzní výpis. Jako první adresa se vezme aktuální adresa. Výpis se provádí tak dlouho, dokud je stisknuta libovolná adresa. Přerušení výpisu a návrat do hlavní smyčky zajistí stisk klávesy EDIT.

Znakový výpis od zadané adresy - klávesy SYMBOL SHIFT + O:

Naprosto stejný výpis jako v předchozím případě, výpis se provádí od adresy zadané na dotaz "First".

Výpis čísel – klávesa L:

Poslední výpis umožněný monitorem je číselný výpis od aktuální adresy. Na řádku bude vypsána adresa, pět jednobytových čísel a pět znaků odpovídajících těmto číslům. Všechno ostatní je shodné jako u znakového výpisu.

Výpis čísel od zadané adresy - klávesu SYMBOL SHIFT + L:

Výpis se bude provádět od adresy zadané na dotaz "First". Jinak se tato funkce neliší od předchozí.

5. Vyhledávání:

Zadání posloupnosti a vyhledání prvního výskytu – klávesa G;

Příkaz umožňuje vyhledávat 5 bytů dlouhou posloupnost, některé byty mohou být pro porovnávání nevýznamné. Po zvolení se objeví dotaz "1. byte" až "5. byte", můžete vložit bud požadované číslo (výraz) nebo dvojtečku - takto se označují nevýznamné byty v hledané posloupnosti. Po odeslání pátého čísla se provede

hledání, pokud bude posloupnost nalezena, nastaví se na adresu jejího výskytu aktuální adresa, potom se obnoví výpis čelního panelu. Hledání se provádí od adresy, která je větší o 1 než aktuální adresa až do konce paměti.

Příklady:

Hledáte výskyt posloupnosti znaků "ABcd", zadejte tedy postupně:

1. byte "A"
2. byte "B"
3. byte "c"
4. byte "d"
5. byte :

Hledáte instrukci ld hl,33333, zadejte:

1. byte #21
2. byte 33333?256
3. byte 33333/256
4. byte :
5. byte :

Kód instrukce ld hl,N buď znáte, najdete ho v tabulce nebo do prázdného místa v paměti tuto instrukci vložíte (viz editace paměti) a podíváte se na její operační kód.

Vyhledání dalšího výskytu zadané posloupnosti – klávesa N:

Aktuální adresa se zvětší o jedničku a od této adresy se hledá další výskyt posloupnosti zadané předchozím příkazem, pokud bude posloupnost nalezena, aktuální adresa se nastaví na její začátek.

6. Plnění a přenos bloku:

Přenos bloku - klávesa I:

Program se zeptá na první "First" adresu bloku, na poslední "Last" adresu bloku a na místo, kam má být blok přenesen "To". Po zadání se provede kontrola, jestli se místo kam má být blok přenesen nepřekrývá s ladícím systémem nebo zdrojovým textem, a provede se přesun. Dojde-li při testu ke zjištění kolize těchto dvou částí paměti, provádění se přeruší a vypíše se chybové hlášení "READ/WRITE Error". Místa odkud a místa kam se bude přenášet se mohou překrývat, blok bude vždy přesunut dobře.

Přenos bloku - klávesy SYMBOL SHIFT + I:

Příkaz se od předchozího liší pouze způsobem zadání parametrů bloku, místo otázky "Last" je nyní otázka "Lenght" délku přenášeného bloku.

Plnění bloku – klávesa P:

Program se zeptá na začátek "First" a na konec "Last" bloku, provede se kontrola, zda blok nezasahuje do paměti obsazené ladícím systémem a zdrojovým textem - pokud ano, ohlásí se chyba "READ/VRITE Error", pokud ne, zeptá se program ještě na to, čím "With" chcete plnit zvolený blok, potom se blok vyplní.

Plnění bloku – klávesy SYMBOL SHIFT + P:

Obdoba předchozí funkce - blok se tentokrát zadává parametry začátek "First" a délka "Lenght".

7. Editace paměti:

Výraznou vlastností monitoru PROMETHEUS je, že umožňuje zapisovat v monitoru do paměti přímo instrukce, není tedy nutno používat tabulky operačních kódů. Tato funkce využívá podprogramy pro kompilaci z assembleru.

Jednorázová editace paměti – klávesa S:

Po stisku klávesy SPACE se objeví v editačním řádku kurzor úplně vlevo, nyní je možno zadat instrukci strojového kódu a to zcela stejným způsobem jako při vkládání zdrojového textu, tedy instrukce začíná od desátého znaku na řádku - u tohoto příkazu funguje automatická tabelace, stačí stisknout ještě jednou mezerník. Po vložení instrukce se provedou oba průchody kompilace a instrukce bude uložena na aktuální adresu. Při ukládání instrukce se kontroluje, aby instrukce nebyla uložena do oblasti ladicího systému se zdrojovým textem ani za uživatelskou zarážku. Pokud chcete instrukci za uživatelskou zarážku přece jen vložit, musíte se vrátit do assembleru, přepsat její hodnotu, vrátit se do monitoru a vložit instrukci. Chcete-li do paměti přece jen zapsat číslo nebo text, použijte odpovídající způsob jako ve zdrojovém textu - pseudoinstrukce DEFB, DEFW, DEFM se stejným účinkem. Tento příkaz sice slouží k editování paměti, můžete ho použít i pro nadefinování hodnoty nějakého (i nového) návěští - použijte pseudoinstrukci EQU, více se dozvíte v příloze. Pokud bude ve vložené instrukci nalezena chyba, ohlásí se do editačního řádku na několik sekund chybové hlášení, bude-li to pokus o zápis do zakázané oblastí, vypíše se chybové hlášení "Bad PUT (ORG)". Ve fázi vkládání instrukce do editačního řádku lze funkce přerušit stiskem EDIT.

Opakovaná editace paměti - klávesa E:

Chcete-li do paměti vložit více instrukcí, není jednorázová editorce příliš pohodlná - je nutno neustále přeskakovat za právě vloženou instrukci - proto použijte tento příkaz. Po zvolení funkce se v editačním řádku také objeví kurzor, stejně se funkce chová i dále, po zapsání instrukce do paměti, se ukazatel posune za vloženou instrukci, obnoví se výpis čelního panelu (důležité jsou dva disassemblované řádky) a je možno vložit další instrukci. Při vkládání můžete používat před instrukcí také návěští, bude mu přiřazena hodnota ukazatele pro ukládání instrukce. Při psaní se lze odvolávat na všechna již existující definovaná návěští - tedy napřed návěští použít v poli návěští a pak se na něj lze odvolat v poli operandů, jinak bude hlášena chyba, odvolávat se samozřejmě lze i na návěští definovaná v průběhu poslední kompilace nebo definovaná dříve pomocí jednorázové editace pseudoinstrukcí EQU - viz příloha.

8. Krokování a trasování:

Krokování a trasování jsou nejdůležitější funkce, které monitor poskytuje. Při trasování a krokování instrukcí se provádí kontroly jestli instrukce nepoužívá zakázané oblasti pro zápis, čtení a běh - takových oblastí je možno pro každou činnost definovat až 5, navíc oblast v níž se nachází ladící systém a zdrojový text se kontroluje na všechny činnosti. Dále se kontroluje, aby se nepoužila instrukce HALT při zakázaném přerušení.

Provádění kontrol je možno vypnout a rychlost trasování se přibližně zdvojnásobí - kontrola na instrukci HALT se vypnout nedá (ta nezdržuje a bylo by to zbytečné), vypínají se kontroly na zakázané oblasti paměti včetně oblasti s ladícím systémem a zdrojovým textem.

U instrukcí CALL a RST lze volit tři režimy - obvyklou simulaci, u vybraných adres (možno až 10 adres) se tyto instrukce provedou přímo (nebude se simulovat volání podprogramu ale celý podprogram se zavolá najednou) anebo se budou všechny instrukce CALL a RST provádět najednou.

Poslední dva režimy zvyšují rychlost trasování, je tu však riziko ztráty kontroly nad programem - režim rychlého volání vybraných podprogramů lze výhodně použít na dokonale odladěných podprogramech pro zvýšení rychlosti trasování. V částech programu, které jsou přímo volány samozřejmé neexistují žádné kontroly. V editačním řádku jsou vypsány indikace stavu kontroly (Control ON/OFF) a způsobu, jakým se budou provádět instrukce CALL a RST (Call NON/DEF/ALL) (žádné/definované/všechny instrukce se budou přímo volat).

Nevypnete-li kontrolu a budete-li krokovat či trasovat své programy se zakázaným přerušením, nemůže se stát, že by monitor ztratil řízení programu. budete-li používat povolené přerušení v módu IM 1 nesmíte měnit hodnotu registru IY (je používán systémem jako ukazatel do systémových proměnných), dávejte si také pozor, aby se registr SP (ukazatel na zásobník) nenastavil do oblasti paměti ROM. Budete-li povolovat přerušení v módu IM 2, musí být nastaveno správně přerušení není trasováno ani nijak kontrolováno - to může být také příčinou některých problémů: je povoleno přerušení a zásobník se zatoulal do oblasti ROM, program se může zhroutit v okamžiku, kdy při trasování nějaké instrukce dojde k přerušení. Pokud Váš program přerušení nepožaduje, pracujte raději se zakázaným přerušením.

Při krokování a trasování počítá monitor také časovou náročnost programu počítadlo T-cyklů (T:00000). Počítání cyklů se hodí při práci s na programech, které musí trvat přesně zadanou dobu - programy LOAD a SAVE, hudební rutiny, barevné efekty na obrazovce a v borderu. Na začátku nastavte počítadlo na nulu a protrasujte tu část, jejíž časovou délku chcete zjistit. Při trasování se nesmí žádná instrukce CALL nebo RST provádět přímo.

Krokování – klávesy SYMBOL SHIFT +Z:

Po stisknutí kláves se provede simulace jedné instrukce na aktuální adrese, před provedením se obnoví hodnoty všech registrů (včetně R) a případně se povolí přerušení, po provedení simulace se hodnoty všech registrů opět uloží, zaznamená se stav přerušení a obnoví se výpis čelního panelu. Pokud je povolena kontrola, provedou se potřebné testy před vlastní simulací, když je zjištěna nějaká kolize, vypíše se chybové hlášení a čeká se na stisk klávesy. Simulace instrukcí CALL a RST se provádí podle zvoleného režimu. Aktuální adresa se nastaví podle toho, jak simulovaná instrukce změní obsah čítače instrukcí.

Pomalé trasování – klávesa T:

Funkce provede simulaci instrukce na aktuální adrese, obnoví výpis čelního panelu, změní aktuální adresu, provede test stisknuté klávesy a pokud není stisknuta klávesa BREAK (CAPS SHIFT + SPACE) opakuje celý cyklus stále znovu. Stisknete-li při trasování současnic klávesy CAPS SHIF'T + ENTER, vynechá se po dobu, kdy budou klávesy stisknuty, obnovování čelního panelu, tato možnost je užitečná pro zrychlení trasování na kratší dobu - třeba pro urychlení některého cyklu. Simulace probíhá stejně jako při krokování. .

Rychlé trasování – klávesa T:

Při zvolení této možnosti se program zeptá na adresu instrukce na níž by se měl zastavit - "Last", a provádí cyklicky tyto akce: simuluje instrukci na aktuální adrese, změní aktuální adresu, pokud jsou stisknuty klávesy CAPS SHIF'T + ENTER obnoví čelní panel, testuje stisk klávesy BREAK (CAPS SHIFT + SPACE) a testuje, jestli se aktuální adresa nerovná zadané poslední adrese, pokud tomu tak není a není stisknut BREAK, skočí opět

na začátek cyklu. V opačném případě se obnoví výpis čelního panelu a rychlé trasování se zastaví. Pokud nevíte, na které adrese by se měl program zastavit (nezáleží na tom - zastavíte ho BREAKem) vložte třeba 0.

9. Práce s registry:

Změna stavu indikátoru přerušení - SYMBOL SHIFT + M:

Přehodí hodnotu indikátoru stavu přerušení z EI na DI nebo naopak. Indikátor určuje v jakém stavu přerušení se bude simulovat každá instrukce, volat podprogram nebo provádět běh s pomocí BREAKPOINTu, po skončení každé takové akce se indikátor nastaví podle skutečnosti po provedení instrukce. Při trasování, krokování, volání podprogramů a běhu s pomocí BREAKPOINTu si dejte pozor na to, aby registr SP neukazoval do oblasti paměti ROM při povoleném přerušení.

Přehození základních a alternativních registrů – klávesy SYMBOL SHIFT + B:

Provede výměnu základní a alternativní sady registru, ve strojovém kódu je to ekvivalentní instrukcím EXX a EX AF, AF' . Při trasování nebo krokování jsou základní vždy ty registry, které jsou viditelné na čelním panelu. Chcete-li se tedy při krokování podívat na hodnoty alternativních registrů, musíte před dalším krokováním registry vrátit.

Nastavení obsahu registrů - klávesy SYMBOL SHIFT + N:

Po zvolení této možnosti se v editačním řádku objeví nápis "ld ", vložte jméno registru, jehož hodnotu chcete změnit, čárku nebo mezeru a napište číslo (výraz), které chcete do tohoto registru zapsat, registry jsou tyto:

Jednobytové registry:	A,B,C,D,E,H,L,HX,LX,HY,LY,I,R
Dvoubytové registry:	AF,BC,DE,HL,LY,IY,SP
Počítadlo T-cyklů:	T
Paměťové ukazatele:	X,Y (viz editor čelního panelu)
Stavový registr :	F

U stavového registru (registr F) je možno měnit jednotlivé bity následujícím způsobem:

ld f,c - změní hodnotu CARRY flagu z NC na C a naopak

ld f,s - změní hodnotu SIGN flagu z P na M a naopak

ld f,z - změní hodnotu ZERRO flagu z NZ na Z a naopak

ld f,p - změní hodnotu PARITY flagu z PO na PE a zpět

10. Oblasti a parametry:

Nastavení DEFB oblastí – klávesa 1:

Monitor dovoluje nastavovat oblasti, které budou při výpisu disassemblerem vypsány jako jednobytová data, takových oblasti si můžete nadefinovat celkem 5. Každá oblast je zadána prvním a posledním bytem (včetně). Po stisku klávesy 1 se do výpisového okna vypíší současně nastavené oblasti, každý řádek začíná číslem oblasti (0 až 4), za ním je vypsána adresa prvního bytu oblasti a pokud je stejná jako hodnota nějakého návěští, je vypsáno také toto návěští, nakonec je vypsána adresa posledního bytu oblasti a případně návěští. Nyní můžete stisknou bud klávesu s číslem některé oblasti - tato oblast se vymaže a znovu se vypíše seznam nastavených oblastí (windows-okna), nebo klávesu I, v tomto případě se program zeptá na první "First" a na poslední "Last" adresu oblasti, prověří, jestli platí, že First<=Last, a přidá novou oblast za již definované oblasti, pokud je již všech pět oblastí definováno, klávesa I nebude reagovat. Stisknete-li jinou klávesu než 0,1,2,3,4,I program se vrátí do hlavní smyčky. Za oblast s jednobytovými hodnotami je považován také prostor paměti obsazený ladícím systémem a zdrojovým textem.

Nastavení DEFW oblastí – klávesa 2:

Další typ oblastí, které lze používat jsou oblasti s dvoubytovými hodnotami (tabulky adres), jejich nastavení je úplně stejné jako u oblastí DEFB.

Nastavení oblastí, z nichž se nesmí číst - klávesa 3:

Při ladění programu je výhodné mít jistotu, že program nečte data z míst v paměti, která k tomu nejsou určena, proto byla do programu dána možnost tato místa instrukcím zakázat. Každá instrukce, která nějakým způsobem čte data z paměti, např. přímo z adresy, z adresy v registru, pomocí indexregistrů nebo přes SP registr (tedy i RET, POP a EX (SP),HL), bude nejprve prověřena, zda tak nečiní z oblasti, kde to má zakázáno, pokud ano, její simulace se neprovede a bude ohlášeno chybové hlášení. Za oblast se zákazem čtení je navíc považován paměťový prostor obsazený ladícím systémem a zdrojovým textem. Vlastní definice těchto "No read" oblastí je stejná jako u DEFB oblasti.

Nastavení oblastí se zákazem zápisu - klávesa 4:

Důležitější, než zjišťovat, zda program nečte data z míst v paměti, z nichž by neměl, je mít možnost hlídat program, aby nezapisoval do jiných než povolených úseků paměti. U každé instrukce, která provádí zápis do paměti (včetně instrukcí CALL, RST, PUSH, EX (SP),HL) se provede kontrola jestli se nepřepíší zakázané oblasti, pokud ano, simulace se neprovede a bude hlášena chyba "READ/WRITE Error". Nastavení je obdobné jako u DEFB oblastí. Oblast s ladícím systémem a zdrojovým textem je také považována za oblast se zákazem zápisu - anglicky No write.

Nastavení oblastí se zákazem běhu - klávesa 5:

Posledním typem oblastí, kde je možnost něco zakázat, jsou oblasti se zákazem běhu. Jakmile by se po provedení nějaké instrukce ocitl program v oblasti se zakázaným během, instrukce se nevykoná a ohlásí se chyba "RUN Error". Zadávání je stejné jako u všech předchozích. Použití snad ani není potřeba nějak vysvětlovat, například když se program nevrací tam, kam by měl, zakažte vše, kromě vlastní oblasti s programem, jakmile se nějaká instrukce při trasování pokusí o skok mimo povolenou oblast, bude hlášena chyba - tyto problémy vyvolává nejčastěji chybná práce se zásobníkem návratových adres pomocí instrukcí PUSH a POP - obvykle něco chybí nebo přebývá.

Definování adres pro přímé volání - klávesa 6:

Výhodný způsob, jak urychlit trasování programů, je provádět vybrané příkazy CALL a RST přímo (nesimulovat každou instrukci ve volaných podprogramech volat celý podprogram najednou. K tomuto účelu si

můžete zvolit celkem deset adres. Pokud nastavíte režim volání definovaných podprogramů (Call DEF) bude se každý příkaz CALL nebo RST, jehož adresní část je rovna některé adrese ze seznamu definovaného tímto příkazem, nahrazovat zavoláním podprogramu na této adrese. Při ladění programu se tato možnost výhodně používá na již odladěné a dobře fungující podprogramy (nehrozí tu již nebezpečí ztráty kontroly, podprogram se vrací zpět tam, odkud byl zavolán) - tisk znaku, test klávesnice pokud není používáno přerušení - a také na podprogramy, jejichž práci není možno zpomalit - nahrávání a zvukové podprogramy. Ukázky použití jsou v příloze. Editace je podobná jako u definování oblastí, vymazávání je možno navíc provádět klávesami 0 až 9.

Změna režimu provádění CALL a RST - klávesa X:

Jak bylo dříve uvedeno, mohou se indukce CALL a RST provádět třemi různými způsoby, které se liší hlavně rychlost (a bezpečností - bohužel nepřímo úměrně. Tyto režimy jsou indikovány v editačním řádku (NON/DEF/ALL) a jejich význam už byl popsán dříve.

Zapnutí/vypnutí kontroly instrukcí - SYMBOL SHIFT + X:

Tímto příkazem lze najednou vypnout provádění všech kontrol, nejméně dvojnásobně to zvýší rychlost provádění trasování. Tuto možnost používejte uváženě.

11. Editor čelního panelu:

Vstup do editoru – klávesy SYMBOL SHIFT + W:

Po přepnutí do editoru čelního panelu se vymaže obrazovka, obnoví výpis čelního panelu a rozsvítí se jedna z položek v čelním panelu. Nyní je možno pro každou položku definovat její umístění, způsob výpisu a další možnosti - viz dále.

Ovládání editoru čelního panelu:

CAPS SHIFT + 1 (EDIT) - opuštění editoru čelního panelu

4 - přechod na následující položku v seznamu (viz dále)

3 - přechod na předchozí položku v seznamu

5 - posun položky o jednu pozici doleva.

6 - posun položky o jednu pozici dolů.

7 - posun položky o jednu pozici nahoru.

8 - posun položky o jednu pozici doprava.

A až Z - nastavení velikostí položky (0 až 25), pro registry znamená, jestli se budou nebo nebudou vypisovat, pro paměťové výpisy se takto mění počet adres, které se vypisují.

SYMBOL SHIFT + D - zapnutí nebo vypnutí výpisu také v desítkové soustavě.

SYMBOL SHIFT + H - zapnutí nebo vypnutí výpisu také v šestnáctkové soustavě.

SYMBOL SHIFT + B - zapnutí nebo vypnutí výpisu také ve dvojkové soustavě.

SYMBOL SHIFT + C - zapnuti nebo vypnutí výpisu také ve znakové formě

SYMBOL SHIFT + T - změna typu výpisu u paměťových výpisů - vypisují se buď jednobyty nebo dvoubyty.

SYMBOL SHIFT + S - změna směru výpisu u paměťových výpisů - obsahy jednotlivých adres jsou vypisovány buď vedle sebe nebo pod sebe.

Seznam položek, které lze používat:

Editační řádek - lze ovlivnit pouze, který řádek bude používán pro tyto účely.

Výpisové okno - lze ovlivnit počet řádků, kletě budou použity, lze také změnit polohu okna.

Výpisové okno disassembleru - při obnovování čelního panelu se vypisuje několik řádků instrukcí strojového kódu od aktuální adresy, počet řádků a polohu okna lze nastavit.

Stav přerušení - výpis EI nebo DI.

Jednobytové registry - A,B,C,D,E,H,L,I,I2,HX,LX,HY,LY a F - lze volit soustavy, ve kterých budou jejich hodnoty vypisovány - registry mohou být vypisovány ve všech najednou, u registru F je možno volit (SYMBOL SHIFT + B) mezi binárním výpisem a výpisem splněných podmínek.

Dvojbytové registry - AF,BC,DE,HL,SP,IX,IY - podobá se jednobytovým registnům.

Výpis počítadla T-cyklů - lze nastavit stejně jako dvoubytový registr.

Adresové výpisy od zadané adresy (ukazatele X,Y) u těchto výpisů lze volit počet bytů, které budou vypisovány, lze měnit směr, kterým se jednotlivé hodnoty vypisují, lze měnit také jestli se budou vypisovat jednobytové nebo dvoubytové hodnoty.

Adresové výpisy od adresy v dvojregistru - (BC),(DE),(HL),(SP),(IX),(IY) - nastavit lze stejné vlastnosti jako u předcházející položky.

Nejjistější způsob, jak si osvojit způsob použití editoru čelního panelu, je ho zkusit použít - nemůžete nic zkazil (program můžete nahrát přece klidně znovu), případně se podívejte na příklad v příloze.

V. Příklady práce se systémem

Ve všech příkladech budu používat znak podtržítko tam, kde by myla být mezera - uvidíte-li tedy "_", znamená to, že musíte při vkládání vložit mezeru z klávesnice - stisknou klávesu SPACE. Ostatní mezery si systém připíše sám.

1. Zpětný překlad:

Nainstalujte PROMETHEUS na adresu 24000 - včetně monitoru. Po instalaci přejděte do monitoru a provádějte následující operace:

• stiskněte SPACE a vložte "ldbytes_equ_#556" - do tabulky symbolů se přidá návěští LDBYTES a bude mít hodnotu #556.

• stiskněte M a vložte "ldbytes" - nastavíte aktuální adresu na hodnotu LDBYTES, na prvním disassemblovaném řádku je místo adresy návěští.

• stiskněte SYMBOL SHIFT + 4 - objeví se disassemblerový výpis obsahu paměti, po několika řádcích se objeví instrukce "call 1511", přerušte výpis - klávesa EDIT (CAPS SHIFT + 1).

• stiskněte SPACE a vložte pomocí EDIT "ldedge1_equ_1511", opět se podívejte disassemblerem na totéž místo, instrukce "call 1511" se změnila na "call LDEDGEl", další adresa je hned na příští instrukci "jr nc,1387".

• stiskněte SPACE a vložte "ldbreak_equ_1387", při dalším listování pak uvidíte návěští LDBREAK hned na třech místech současně, listujte dál a uvidíte ještě další výskyt a také několik výskytů návěští LDEDGEl, přerušte výpis (EDIT) a použijte ho znovu k nalezení další adresy - tentokrát to je adresa 1396.

• stiskněte SPACE a vložte "ldwait_equ_1396", obnovte výpis a zjistěte další adresu - teď to je adresa 1507.

• stiskněte SPACE a zadejte "ldedge2_equ-1507", po obnově výpisu najděte instrukci "jr nc,LDBREAK+1", tento skok skáče na instrukci, která leží hned za instrukcí na adrese LDBREAK, když mezi tyto dvě instrukce vložíte další, nemusí program správně pracovat, vyrobíme raději nové návěští.

• nyní stiskněte SPACE a vložte "ldstart_equ_ldbreak+1", číselná hodnota LDBREAK Vás nemusí vůbec zajímat - můžete použít návěští, podíváte-li se na výpis disassembleru, uvidíte další změny tímto způsobem lze nahradit všechny odkazy na adresy návěštími, použitá návěští jsou z výpisu paměti ROM, nebrání Vám však nic v použití třeba návěští LOOPXX pro skoky a CALLXX pro podprogramy - XX je pořadové číslo použitého návěští - podobným způsobem vznikl předváděcí demoprogram.

• protože nahrávací rutina není ještě schopna zpětného překladu, vložte postupně tato návěští:

SPACE "ldleader_equ_1408"

SPACE "ldsync_equ_1423"

SPACE "ldmarker_equ 1480"

SPACE "ldflag_equ_1459"

SPACE "ldverify_equ_1469"

SPACE "ldnext equ_1474"

SPACE "lddec_equ_1476"

SPACE "ld8bits_equ_1482"

SPACE "ldloop_equ_1449"

po provedení se podívejte na výpis disassembleru (můžete i při vkládání), každý odkaz na adresu v hlavní části programu je nahrazen návěštím, nyní se podíváme na oba podprogramy LDEDGE1 a LDEDGE2.

• stiskněte M a vložte "ldedge1", při výpisu narazíte na adresy 1513 a 1517.

SPACE "lddelay_equ_1513"

SPACE "ldsample_equ_1517"

Po vložení proveďte stejným způsobem prozkoumejte ještě podprogram LEDGE2, brzy zjistíte, že má stejné tělo a není tudíž co prohlížet.

• stiskněte M, vložte "ldbytes" a prohlédněte program, potřebná část končí na adrese 1540, nyní provedeme zpětný překlad.

• stiskněte SYMBOL SHIFT + D a na otázku "First" vložte "ldbytes", na otázku "Last" vložte "1540+1".

• po skončení zpětného překladu stiskněte C - zapnete tím vypisování adres před instrukce - při zpětném překladu se vypíná.

• nyní se vraťte do assembleru - stiskněte klávesu Q, pokud jste provedli všechny operace správně, měl by zdrojový text končit adresou 40607.

• stiskněte SYMBOL SHIFT + K a dostanete se na začátek zdrojového textu, najděte instrukci "ld hl,1343", je to sedmá instrukce od začátku.

• nastavte do přístupového řádku instrukci "push hl" - je uložen hned za instrukcí "ld hl,1343", stiskněte dvakrát za sebou klávesy CAPS SHIFT + 9, vymažete dva řádky se zmíněnými instrukcemi, zdrojový text nyní končí na adrese 40598 a jeho začátek vypadá takto:

```
LDBYTES     inc d
            ex af,af´
            dec d
            di
            ld a,15
            out (254),a
            in a,(254)
            …
```

• v tomto okamžiku je nejvyšší čas pro uložení zdrojového textu na kazetu

• vložte "SAVE :loader", spusťte volnou kazetu a stiskněte nějakou klávesu, po provedení vložte "VERIFY", vraťte kazetu a zkontrolujte nahrávku, dojde-li k chybě, zkuste ještě jednou příkaz "VERIFY", bude-li opět bez úspěchu, zadejte opět příkaz "SAVE" (teď už bez jména, vezme se poslední zadané jméno) a celé nahrávání opakujte.

připište před program instrukce:

```
            org 60000
            ent $
START       ld ix,16384
            ld bc,6912
            ld a,255
            scf
            call LDBYTES
            ret
```

• vezměte kazetu s libovolným obrázkem (SCREEN) a nastavte ji za hlavičku obrázku, zadejte příkaz "RUN" a spusťte magnetofon, po nahrání obrázku čeká program na stisk klávesy a pak se vrací do editoru.

• vložte "FIND s:or_ _ _2", změníte-li po nalezení číslo 2 na nějaké číslo z rozsahu 0 až 7, změní se barva pruhů při nahrávání zůstanou však zachovány dvojice barev.

• vložte "FIND s:ldsample_", najděte instrukci "cpi" (můžete také rovnou hledat příkazem "FIND s:cpl", tato instrukce je tu pouze jednou) a zaměňte ji instrukcí "xor %11111111", změníte-li některé jedničky na posledních třech místech za nuly, můžete získat další kombinace barev - mění i složení dvojic barev.

• poslední instrukce, která mění barvy, je instrukce "xor 3" za návěštím LDSYNC.

• vložte příkaz "MONITOR a" - provede se příkaz ASSEMBLY a skok do MONITORU.

• stiskněte M a vložte "ldedge1" - takto nastavíte aktuální adresu na návěští LDEDGE1.

• stiskněte SYMBOL SHIFT + N a vložte "t,0" - vynulování T-stavů.

• stiskněte SYMBOL SHIFT + T a na otázku "Last" vložte "ldsample-1" - rychlé trasování do adresy LDSAMPLE-1.

• po provedení ukazuje počítadlo T-cyklů hodnotu 354, tolik času trvalo provedení tohoto programu:

```
LDEDGE1   ld a,7
LDDELAY   dec a
          jr nz,LDDELAY
```

• vraťte se do assembleru - stiskněte Q

• přepište nahoře uvedené tři řádky takto:

```
LDEDGE1   ld a,7
LDDELAY   dec a
          set 3,a
          out (254),a
          res 3,a
          jr nz,LDDELAY
```

• vložte příkaz "MONITOR a".

• vynulujte počítadlo T-cyklů, nastavte aktuální adresu na LDEDGE1 a spusťte rychlé trasování k adrese LDSAMPLE-1, po provedení ukazuje počítadlo 346 cyklů, tedy o 8 méně, než by mělo, přidejte dvě instrukce "nop" za instrukci "jr nz,LDDELAY", samozřejmě v assembleru, vložte opět příkaz "MONITOR a", zopakujte výpočet T-cyklů - nyní už bude výsledek opravdu 354.

• vraťte se do assembleru, vložte příkaz "RUN" a spusťte kazetu výsledek uvidíte sami. Většina loaderů reaguje na stisk SPACE, pokud budete chtít tuto vlastnost u svého loaderu odstranit, nahraďte instrukci "ret nc" za návěštím LDSAMPLE instrukcí "nop".

2. Trasování I:

V tomto příkladu si vyzkoušíte některé možnosti, které Vám PROMETHEUS poskytuje pro trasování programu. Nainstalujte assembler i s monitorem na adresu 25000.

• vložte příkaz "CLEAR y".

• napište tento krátký program:

```
          org 60000
          ent $
START     xor a
          out (254),a
FILL      ld hl,16384
          ld bc,6912
FILL2     ld a,r
```

```
        ld (hl),a
        inc hl
        dec bc
        ld a,b
        or e
        jr nz,FILL2
        ret
```

• vložte příkaz "RUN" a prohlédněte si výsledek, na barevné televizi to vypadá podstatně lépe, pokud ji nemáte, pak máte smůlu.

• když spustíte program znovu, bude výsledek poněkud odlišný od předchozího - při každém spuštění je výsledek jiný, je to způsobeno registrem R, vložte za instrukci "ld bc,6912" instrukci "ld r,a", po úpravě bude výsledek vždy stejný.

• vložte příkaz "MONITOR a".

• nastavte aktuální adresu na 60000, prohlédněte si přeložený program a zjistěte, kde leží instrukce "ret" - mělo by to být na adrese 60020, vymažte obrazovku (CS+0) a spusťte rychlé trasování (SS+T), na dotaz "Last" vložte adresu instrukce "ret" (60020). Po odeslání vidíte značně zpomaleně všechny akce, které program provádí, nyní můžete:

Nedělat nic, počkat až to samo skončí.

Stisknout CAPS SHIFT + ENTER a sledovat, co se děje s registry.

Stisknout CAPS SHIFT + SPACE a provádění zastavit, případně opět spustit nebo krokovat jednotlivé příkazy.

• chcete-li při trasování sledovat jednotlivé registry, použijte pomalé trasování (T), po spuštění můžete:

Nedělat opět nic – tentokrát program poběží tak dlouho, dokud nenarazí na instrukci, která by vyvolala nějakou chybu.

Stisknout CAPS SHIFT + ENTER, teď se program naopak zase rozběhne rychleji, neuvidíte však obsahy registrů, respektive jejich vypisování se po dobu stisku neprovádí.

Stisknout CAPS SHIFT + SPACE a trasování zastavit.

• nastavte znovu aktuální adresu na 60000, vymažte obrazovku a vypněte provádění kontrol (SS+X).

• zkuste ještě jednou pomalé i rychlé trasování - rozdíl v rychlosti u rychlého trasování bude víc než patrný.

• až se do sytosti přesvědčíte o rozdílu v rychlostech, kontrolu opět zapněte (SS+X), zrychlení trasování vypínáním kontrol provádějte obezřetně - až když jste si jisti, že program nepáchá nějaké nepřístojnosti, několik sekund ušetřených vypnutím kontrol se totiž může snadno změnit na několik desítek minut nového nahrávání systému a zdrojového textu.

• vraťte se do assembleru a před instrukci "ld hl,16384" vložte tyto řádky:

```
LOOP    push af
        call FILL
        pop af
        inc a
        cp 128
        jr c,LOOP
        ret
```

• podívejte se co bude program dělat na příkaz "RUN" nyní.

• přepněte se do monitoru - "MONITOR a".

• přepněte režim provádění instrukcí "CALL" a "RST" na režim ALL, klávesa X.

• nastavte okno zákazu čtení - zakažte oblast adres 0 až 1, klávesa 3, vypíše se seznam definovaných oken - bude prázdný, stiskněte klávesu I a na dotaz "First" vložte 0 a na dotaz "Last" vložte 1, bude-li nyní chtít program při provádění instrukce "ret" číst z těchto dvou adres, bude hlášena chyba.

• nastavte SP registr na 0, aktuální adresu na 60000 a krokujte, případně spusťte pomalé trasování (rychlé se takhle nebude příliš lišit od příkazu RUN - jediný rozdíl je, že program se dá zastavit stiskem kláves CAPS SHIFT + SPACE.

3. Trasování II:

• instalujte PROMETHEUS na adresu 24000.

• nastavte aktuální adresu na #12A2, registr IY na 23610, registr SP na 0 a povolte přerušení.

• spusťte rychlé trasování do adresy 0, uvidíte, jak se pomalu čistí obrazovka, provádí listing BASIC programu (jestli tam nějaký je - vložte třeba 1 RANDOMIZE USR 24e3:STOP) a v editační zóně se objeví kurzor.

• vložte do editační zóny příkazy PLOT 0,0 :DRAW 255,175, klávesy budou reagovat velice vlažně a bude delší dobu trvat než se vypíše řádek po přijetí každé klávesy, až úspěšně vložíte oba příkazy, stiskněte ENTER, po chvilce se objeví vlevo dole bod a začne se kreslit čára.

• předchozí způsob je příliš pomalý, zkusíme ho zrychlit, stiskněte klávesu 6 a vložte jako adresy tato čísla: #F2C (editor), #556 (nahrávání bloku z kazety) a 16 (tisk znaku), nastavte stejně jako předtím zásobník a aktuální adresu, přepněte režim provádění instrukcí "CALL" a "RST" na režim "DEF", spusťte rychlé trasování do adresy 0.

• objeví se editor, na první pohled se může zdát, že se řízení programu vrátilo do systému Spectra - BASICU, vložte však třeba příkaz LIST a po stisku ENTER uvidíte, že ostatní akce jsou zpomaleny, vyzkoušejte si tyto příkazy:

CIRCLE 127,87,87 - po odeslání musíte delší čas počkat, uvidíte, že BASIC kreslí kružnice z čar.

POKE 25000,0 - jakmile dojde k pokusu o zápis nuly na adresu 25000, která leží v oblasti ladícího systému, dojde k ohlášení chyby a provádění se zastaví na instrukci "ld (bc),a", posuňte se za tuto instrukci a spusťte znovu rychlé trasování

LOAD ""SCREEN$ - vemte nějaký obrázek a můžete ho zkusit nahrát

BEEP .05,40 – až se ozve divné cvakání z počítače, tak se jedná o zvuk zadaného příkazu podstatně menší výšky, u Spectra není žádný zvukový procesor a každý zvuk se vyrábí pouhým zapínáním a vypínáním reproduktoru.

4. Zdrojový text:

Na kazetě je vedle ladícího systému PROMETHEUS a programu GENSOR také ještě demonstrační program - zdrojový text. Pokud Vás zajímá, proveďte následující akce:

• nainstalujte assembler na adresu 25000.

• vymažte případný zdrojový text příkazem "CLEAR y" (klíčové slovo CLEAR se vypíše po stisku kláves SYMBOL SHIFT + C).

• vložte nyní příkaz "LOAD :skatecrazy" a spusťte magnetofon, program vyhledá a nahraje požadovaný blok, skončí-li nahrávání úspěšně, vypíše se nápis "Wait please" a program se vkládá řádek po řádku k již existujícímu programu v paměti, chvilku posečkejte, pokud nastane chyba při nahrávání, vypíše se hlášení "Tape error" a v editačním řádku se obnoví posledně vložený příkaz, opakujte nahrávání.

• zdrojový text je úspěšně v paměti, můžete si ho prohlížet klávesami pro skrolování nebo stránkování, pokud jste si program prohlédli, a jste zvědavi co bude dělat, zadejte příkaz "RUN".

• jak jste si už mohli přečíst v návodu, provádí povel RUN nejdřív překlad a vyčištění obrazovky, potom spustí přeložený program od určené adresy, pokud se program po rozběhnutí vrátí zpět do assembleru, čeká na stisk libovolné klávesy, vyčistí obrazovku a poté obnoví výpisy.

Doufám, že se Vám melodie líbila, možná ji znáte, odkud, to ponechám na Vás. Jestli si chcete s rutinkou trochu pohrát, zkuste následující příkazy.

• "FIND s:loop23_" - vyhledá návěští LOOP23 v poli návěští a na obrazovce uvidíte (mimo jiné) tyto dva řádky:

```
LOOP23      djnz LOOP23
            ld a,0
```

• nahradíte-li 0 nějakým číslem v rozsahu 0-7 a spustíte-li program (RUN), uvidíte změnu, další změny podobného charakteru najdete na těchto místech programu:

```
LOOP04      djnz LOOP04
            ld a,0
LOOP09      push de
            ld a,24         ;tady přičítejte 0 až 7
LOOP34      ld a,(de)
            inc de
            and 24          ;přičítejte 0 až 7
LOOP36      djnz LOOP36
            xor 24          ;přičítejte 0 až 7
LOOP38      djnz LOOP38
            xor 24          ;přičítejte 0 až 7
LOOP40      djnz LOOP40
            xor 24          ;přičítejte 0 až 7
```

Další změny můžete provést v programu s návěštím INTERUPT, několik instrukcí za tímto návěštím je tato sekvence:

```
            dec c           ;délka noty
            dec h           ;doznívání pro jeden
            dec l           ;a pro druhý kanál
```

každou instrukci můžete vyloučit (vložte před na na začátek řádku znak ";" - vyrobíte z ní komentář). Budete-li se chtít zabývat průzkumem této hudební rutiny a podaří-li se Vám zjistit, jak se hudba programuje (sám to bohužel nevím), zkuste vyrobit nějaký hudební program (podobný WHAMu). Rutina umí hrát dva kanály s proměnlivou hlasitostí a třetí kanál - bicí nástroje.

Několik slov na závěr.....

Hodně úspěchů při programování?