U V O D D O S T R O J N I H O K O D U S P E C T R A

(Slaďikar) - ZO SVAZARM KAROLINKA 1985

Pripadnym ctenarum, kterym se toto dilko dostane nahodou do rukou, bych hned uvodem rad sdelil, ze je napsano jen pro naproste zacatecniky v tomto oboru a nemuze se zabyvat vsim, co Spectrum dokaze. Ostatne, ani autori to sami nevi. Jedinym duvodem k jeho napsani bylo poskytnout zacinajicim Spectristum impuls k pochopeni strojoveho kodu a Assembleru Z-80 specialne pro Spectrum 48 k.

Autori totiz sami na sobe poznali, jak nesnadne je z dostupnych ceskych publikaci porozumet strojovemu kodu, nehlede k tomu, ze pro mikroprocesor Z-80 temer zadna literatura neexistuje a ke Spectru uz vubec nic. Vsechny odborne publikace jsou navic psany zvlastni "assemblerovstinou", jejiz autori zrejme predpokladaji, ze to budou cist jen odbornici, kteri tomu rozumi. (Proc by to ovsem cetli, kdyz tomu rozumi, ze?). Take priklady, ktere mnohde sem tam uvadeji, jsou sice spravne, ale pro nas zpocatku vubec nic platne, protoze, kdyz je zavedame do Spectra, nestane se obvykle vubec nic !

Jeden z krasnych prikladu, kdy si nekdo musel dat hodne zalezet na tom, aby nepovolanym osobam co nejvice znemoznil pochopeni strojoveho kodu, je i v manualu Spectra v kap. 25. Autori manualu tam uvadeji kratky strojovy program LD BC,99 RET. A hned pod tim jeho preklad 1,99,0,201. Opravdu perfektne zvoleny priklad specialne vybrany ke zmateni myseli. Clovek nezasvecene veci nikdy nepochopi, proc ma vlozit prave tato cisla, hlavne tu nulu, a to jeste za cislem 99. A uz vubec nemuze ani objevit, ze by se mel vlastne velice podivit, proc po PRINT USR adresa pocitac vypise na obrazovku vlevo nahore cislo 99. Proc by mel vubec neco napsat ? Jeste se k tomu vratime.

I autorum chvili trvalo, nez pochopili, ze nejlepsi metodou, jak studovat assembler Z-80, je vytvaret takove programy, ktere dokazi udelat "neco", co je videt, nebo slyset. Nu a zde vam chceme predlozit podrobny postup a popis (aspon doufame) nekolika kratkych programu, jejichz prostudovanim, rozebranim a postupnym rozsirovanim se snad vytvori ta potrebna "jiskra" - impuls, jez nas uvede do onoho noveho sveta obrovskych moznosti - strojoveho kodu.

Nebudeme se zde zabyvat priklady uziti strojoveho kodu pro matematicke operace, k tomu je obvykle nejvhodnejsi BASIC, ale ukazeme si nektere priklady, ktere jasne predvedou, v cem je Basic pomaly, o kolik pomalejsi, poznate sami.

Jiste jste si dobre vsimli, jakym zpusobem se na obrazovku nahrava z magnetofonu obrazek. Cokoliv, co je na ni zobrazeno, je ulozeno v "obrazovkove" pameti, ktera pocina (viz manual Spectra) bunkou (adresou) 16.384 a zabira dalsich 6.912 osmibitovych pametovych mist (bunek, nebo adres) smerem nahoru. Vsechny bunky s adresou nizsi nez 16.384, t.j. 0 az 16.383, zabira pamet ROM a tedy jejich obsah je dan a nelze zmenit.

Ale vratme se k nahravani obrazku. Nejprve se zobrazi prvni linka odshora, (spravneji nulta linka v nultem radku), potom nulta linka v dalsim radku, atd., az se zobrazi vsech osm prvnich hornich radku. Potom se stejnym zpusobem zacnou zobrazovat dalsi linky dalsich osmi radku a nakonec posledni radky. Na zaver pak se obrazek vybarvi, protoze atributy tisku, t.j. barvy INK, PAPER atd. jsou ulozeny v pameti az v poslednich 768 bunkach z onech 6.912.

Cili prvnich hornich vodorovnych osm bodu obrazovky (uplne vlevo nahore na obrazovce) je ulozeno v prvni bunce obrazovkove

pameti, ktera ma cislo 16.384, ve forme osmibitoveho cisla, kde jednicky se zobrazi jako body inkoustu, t.j. kresba, nuly se nezobrazi. Prosim, zkuste si to overit pomoci Basicoveho prikazu. Zadejte prosim (lze to ucinit i v TASWORDU) POKE 16384, 255, nebo nejake cislo v binarni forme (napr. BIN 00110011) a po ENTER se vlevo nahore na obrazovce vytiskne prvnich osm bodu v pripade, ze jste zadali 255, nebo-li jinak receno, ze kazdou jednicku v binarnim cisle jeden bod.

Zde jen malou pripominku, ze do Basicu se z tohoto textu dostanete takto : stisknutim SYMBOL SHIFT a klavesy A se dostanete do menu, tam stisknutim klavesy B a nasledne ENTER do Basicu, kde muzete experimentovat a zpet do menu se dostanete pomoci prikazu GO TO 25.

Nuze a tim jsme vlastne vytvorili prvni, snad vubec nejjednodussi strojni program. Timto zpusobem muzeme tedy na obrazovce "malovat" a na libovolna mista tisknout body, nebo cele radky bodu. Zkuste to jeste i na nektere jine adrese.

Dalsi bunka - cislo 16.385 obsahuje dalsich osm bitu, t.j. 8 bodu horni linky na obrazovce a tak to pokracuje jako u nahravani z magnetofonu az k cislu 22.527. Od adresy s cislem 22.528 po 23.295 jsou ulozeny stributy tisku, t.j. informace o barve inkoustu a papiru a pod. Ty plati vzdy jiz pro cely znak 8x8 bodu. Ted si zkuste, co to udela, kdyz i do techto bunek pameti zadate nejaka cisla. Napr. POKE 22528,20 a podobne.

Tim jsme si vysvetlili, jakym zpusobem pracuje u Spectra obrazovkova pamet a muzeme se pustit do dalsi casti.

Priklad : potrebujeme prenest obsah obrazovky (obrazek s pod.) nekam jinam, na jine misto pameti, aby se nam neztratil, a odkud bychom jej mohli kdykoliv vyvolat zpet na obrazovku. Nic snazsiho - jde to i v Basicu a zde si ucinime srovnani, jak rychle to dokaze Basic a jak rychle nas strojni program. Vlozte napr. tento Basicovy program :

```
  5 REM Preneseni obrazu do pameti pocinaje adresou  c. 50000
 10 CLEAR 49999 : STOP
 20 FOR n = 0 TO 6911
 30 POKE 50000 + n, PEEK (16384 + n)
 40 NEXT n
```

Program je myslim jasny. Radek 10 nastavuje RAMTOP na adresu 49.999. RAMTOP je prekazka, ktera nedovoli, aby ji program v Basicu prekrocil a nemohl nam tak vniknout do pameti nad ni a pripadne ji zmenit. Radek 20 je zacatek citace-smycky, ktera probehne 6.912 krat a ma za ukol na radku 30 pomoci prikazu POKE vlozit do pameti 50000 + n vzdy pri kazdem pruchodu smyckou obsah pametove bunky s cislem adresy 16384 + n . To provede pomoci prikazu PEEK (16384 + n). Cislo musi byt v zavorce, jinak by vzal jen obsah bunky c. 16384 a pricetl k nemu velikost c. n. Radek 40 je konec smycky. Provedte nejdrive GO TO 10.

Ted si muzete na obrazovku cokoliv namalovat, nebo napsat, nebo i nahrat nejaky obraz a po spusteni programu pomoci prikazu GO TO 20 plus ENTER, bude obrazek, ktery je prave na obrazovce, postupne prenesen take do pameti pocinaje cislem adresy 50.000 az po adresu 56.911 vcetne. Chvili to potrva, pak ohlasi O.K.

Po skonceni prenosu tohoto bloku dat vymazeme obrazovku napr. pomoci ENTER a nas program v Basicu zmenime nyni tak, aby nam tentokrate prenesl informace (obrazek) - neboli blok dat, zpet, t.j. z pameti s adresou 50.000, kam jsme jej ulozili pred chvili zpatky znovu do obrazovkove pameti 16.384 a tim tedy i znovu na obrazovku.

Program upravime takto (nebo radeji napiseme novy na dalsi

radky) :

```
100 REM Preneseni obrazu z pameti na obrazovku
110 FOR n = 0 TO 6911
120 POKE 16384 + n, PEEK (30000 + n)
130 NEXT n
```

Nyni si pripravte stopky a po zadani GO TO 100 plus ENTER je spustte a stopnete si tak, jak dlouho to bude tomuto Basicovemu programu trvat, nez cely obrazek prenese. Muzete pri tom v klidu pozorovat, jak je obrazek postupne prenasen, jak se zobrazuji postupne jeho jednotlive linky a radky a jeho zaverecne "vybarveni", obdobne jako pri nahravani z magnetofonu. Je take asi stejne tak "rychle" prenasen, t.j. tak pomalu, ze se tento zpusob neda prakticky pri nicem pouzit. Potrebujeme takovy program, aby dokazal prenaset tyto obrazky, cili jakekoliv bloky dat, mnoham rychleji, temer ihned. Lze to ucinit ? Ano a sice velmi snadno a rychle pomoci nasledujiciho strojniho programu, ktery je prakticky presnou obdobou jiz uvedenych programu Basicovych.

Pokud jste Spectrum mezitim nevypli, mame v pameti od adresy c. 58.000 az po adresu 56.911 vcetne stale ulozen vas obrazek a dale dva Basicove programy na jeho premistovani, ktere sice jiz nebudeme potrebovat, ale nepreskazeji nam a muzete je klidne ponechat pro pripadnou pozdejsi demonstraci. Nas nasledujici strojovy program si musime nekam ulozit. Nejlepe opet za RAMTOP, tedy nahoru nad obrazek, napr. od adresy c. 57.000 vyse.

Nyni vlozte v Basicu tento program - vysvetlime si jej pozdeji :

```
200 REM Program pro vlozeni strojoveho kodu pro preneseni
      obrazu z pameti na obrazovku
210 FOR n = 0 TO 11
220 READ a
230 POKE 57000 + n, a
240 NEXT n
250 DATA 33,80,195,17,0,64,1,0,27,237,176,201
```

Po zadani GO TO 200 plus ENTER se nam do pameti pocinaje adresou c. 57.000 ulozi 12 osmibitovych cisel strojoveho programu, ktery je schopen prenaset obraz (blok dat o delce 6.912 bytes), z pameti s pocatecni adresou 30.000, do obrazove pameti s pocatecni adresou 16.384 - tedy na obrazovku. Tento strojni program uvedeme do chodu (odstartujeme) obvyklym RANDOMIZE USR 57000 plus ENTER. Po stisknuti ENTER se nam obraz, ulozeny na adrese 30.000 az 36.911 vcetne, prenese na obrazovku. Jak rychle to bude provedano, posoudite sami.

Nyni neco k vysvetleni vyse uvedeneho programu na radku 200 az 250. Radek 210 je pocatek citace-smycky, ktera ma za ukol 12-krat ulozit do dvanacti pametovych bunek o adresach 57.000+n dvanact osmibitovych cisel (dat) strojoveho kodu. Osmibitova cisla jsou v dekadickem vyjadreni cisla od nuly do 255 vcetne.

Jak jsme k temto cislum (at uz jsou vyjadrena dekadicky, binarne, ci hexadecimalne) vlastne prisli ? Tato cisla jsou strojovym prekladem programu napsaneho v jazyku symbolickych adres (JSA) - assembleru Z-80, ktery je stejny pro vsechny mikropocitace s mikroprocesorem Z-80, jen cisla "obrazovky" a podobne specialita plati ovsem jen pro Spectrum 48k. Jiny mikropocitac muze k tomu vyuzivat treba jine adresy.

Program v assembleru, presneji receno v jazyku symbolickych adres (JSA) pro Z-80 vypada takto :

```
                                                    Preklad
1.    LD HL, 50000      ; Nastaveni pocatecni adresy  (33,80,195)
2.    LD DE, 16384      ; Nastaveni cilove adresy     (17,0,64)
3.    LD BC, 6912       ; Nastaveni pocitadla         (1,0,27)
4.    LDIR              ; Provedeni prenosu           (237,176)
5.    RET               ; Navrat do Basicu            (201)
```

Cisla vlevo na zacatku radku nepatri k assembleru, uvedli jsme je jen jako ocislovani radku pro vysvetleni tohoto programu. Nektere prekladaci programy pouzivaji take cislovani radku, pro funkci programu vsak nemaji rovnez zadny vyznam, jen ulehcuji jeho zapsani. Proto se jimi nebudeme dale zabyvat a vsimneme si dalsich jednotlivych sloupcu. V pravem sloupci jsou uvedeny komentare (poznamky) k jednotlivym radkum. Musi byt vzdy oddeleny strednikem. Jsou to nevykonne poznamky, neco jako v Basicu REM. Vlastni zapis programu je tedy jen v prostrednim sloupci. Presna pravidla o forme zapisu jsou uvedena napr. i v publikaci ASSEMBLER Z-80, kterou vydalo JZD Slusovice a i jinde. Uvedeme asspn toto : program zapsany v Assembleru Z-80 se sklada ze sledu instrukci a kazda instrukce (na jednom radku) je tvorena po sobe jdoucimi ctyrmi poli (sloupci), oddelenymi od sebe navzajem urcitymi oddelovaci, napr. takto :

```
   1.sloupec   2.sloupec   3.sloupec   4.sloupec
   Navesti     Op.kod      Operandy    Poznamky

Priklad :
   NAV1        LD          HL,nn       ; presun nn do HL
```

Do techto ctyr poli (sloupcu) je nutno psat vzdy v predepsanem poradi, i kdyz jsou nektera pole vynechana. Oddelovace jsou bud carka, mezera, nebo tabulator. Strednik oddeluje nevykonne poznamky. Ty mohou zacit kdekoliv - i zcela vlevo.

NAVESTI je sestnactibitova konstanta, cesky receno, je to nami zvoleny "nazev" cisla urcite adresy pametove bunky, do ktere se ulozi prvnich osm bitu nasledujici instrukce. (t.j. instrukce, ktera nasleduje bezprostredne za navestim). Techto osm bitu je cislo kodu, pod kterym je tato instrukce (napr. LD HL,nn) uvedena v seznamu instrukci a jejich CODE. (ld hl,NN = CODE 33 dekadicky, nebo 21 Hexadecimalne atd.)

OPERACNI KODY A OPERANDY - v seznamu instrukci, o nemz jiz byla zminka, je uvedeno, jake operacni kody a s jakymi operandy muzeme pouzit. Jejich blizsi vysvetleni je uvedeno rovnez v prislusnych publikacich, zabyvajicich se mikroprocesorem Z-80. Nas strojni program nektere z nich take pouziva a my se nyni k nemu vratime, abychom si jej blize vysvetlili.

Cely princip prenosu obrazu (bloku dat) a tedy princip tohoto programu spociva ve specialni instrukci mikroprocesoru Z-80, instrukci pro blokovy prenos dat, nazvanou LDIR (v manualu Spectra uvedeno malymi pismeny ldir). Ma CODE ED B0 hexadecimalne, neboli 237, 176 dekadicky). Je to instrukce pro blokovy prenos dat s inkrementaci (pricitanim jednicky) a opakovanim, cili dalo by se rici se zabudovanym citacem, ktery si samozrejme muzeme nastavit - v nasem pripade prave na onech prenasenych 6.912 obsahu bunek (bytes). Schematicky znazorneno :

```
LDIR znamena:    (DE) <-- (HL);  DE <-- DE+1;  HL <-- HL+1;
                 BC <-- BC-1;  opakovani az do stavu  BC = 0.
```

Na tomto schematu je onen citac prave ten registrovy par BC, ktery se po nastaveni na potrebnou hodnotu sam odpocitava az k

nule a mezitim provadi i ty dalsi potrebne cinnosti. Jeste se musime zminit take o jedne veci. Registr HL (nebo DE, ci BC) se sklada ze dvou osmibitovych registru (bunek) H a L, proto do nej ukladame 16-ti bitova cisla (cislo NN sestnactibitove jsou vlastne dve cisla osmibitova). Vetsinou je to prave adresa, napr. 16.384 a pod. A zde je ten duvod, proc jsme pomoci POKE nemohli ulozit rovnou 16-bitove cislo, ale museli jsme jej rozlozit na dve 8-bitova cisla a ta ulozit do dvou adres pameti jdoucich po sobe. A navic se musi tato cisla vlozit v obracenem poradi - t.j. nejdrive cast cisla mene vyznamna (jednotky) a potom cast vyznamna (tisice). Pri pouziti prekladace to za nas cvsem prekladac udela sam, proto v assembleru muzeme psat i sestnactibitova cisla v normalni dekadicke forme, binarne, nebo i hexadecimalne, dle toho, jak je pouzity prekladac uzpusoben. Nejvhodnajsi zapis je v hexadecimalnich cislech, protoze tak jsou 16-ti bitova cisla lehce rozlozitelna na dve 8-mi bitova a naopak. Pri rucnim prekladu pomoci tabulky instrukci si musime poradit sami.

Ale vratme se k instrukci LDIR a k jejimu schemat.znazorneni : (DE) <-- (HL) nam ukazuje, ze z pametove bunky, ktera ma cislo HL (v nasem pripade 50.000), prenese jeji obsah do pametove bunky cislo DE (16.384). V zavorce jsou prava proto, ze prenasejí obsah bunky a ne cislo adresy.
DE <-- DE+1 jiz v zavorce neni a proto znaci, ze (16384 <-- 16384+1 = 16385) registr DE se nam kazdym pruchodem citace zvysi o jednicku a totez je u HL. Myslim, ze jiz chapete, jak uvedeny program pracuje. Ted si jej rozebereme podrobneji.

Aby instrukce LDIR mohla pracovat, musime ji nastavit vychozi parametry.

Na prvnim radku pomoci instrukce LD HL,50000 (uloz do registru HL cislo 50000) jsme nastavili cislo pocatecni adresy, od ktere chceme uskutecnit prenos dat. Kam je chceme prenest, nastavime na druhem radku pomoci LD DE,16384. Na tretim radku nastavime citac na hodnotu = pocet dat, ktera chceme prenest. V nasem pripade je to 6912. Nu a na ctvrtem radku je jiz samotna instrukce LDIR. Na dalsi radek je treba vlozit instrukci, ktera vrati mikroprocesor (nebo posle) napriklad zpet do Basicu, nebo na nejakou jinou adresu. Kdybychom tam napsali napr. takovou instrukci, ktera by jej poslala zpet na prvni radek, bude program probihat do nekonecna atale dokola a my bychom jej nemohli zastavit. My jsme pouzili instrukci RET. Navratova adresa do Basicu je ulozena normalnim zpusobem v zasobniku a jeji vyvolani zpusobi prave instrukce RET. (Je to neco jako RETURN v Basicu). Viz manual Spectra.

Nu a preklad takto zapsaneho programu do strojoveho kodu, pokud nechceme pouzit prekladac, t.j. nektery z programu assemblaru, je jiz jednoduchy. Zacneme postupne od radku prvniho, t.j. LD HL,50000. V tabulce instrukci (CODE) najdeme, ze LD HL,nn = 33 dekadicky
50000 = musime rozlozit na dve 8-mi bitova cisla, napr. takto :
50000-256*INT(50000/256) da prvni cislo = 80
INT (50000/256) da druhe cislo = 195

A jiz zde mame prva 3 cisla, 33,80 a 195.

Dale postupujeme stejne. V tabulce CODE najdeme LD DE,nn (17), rozlozime 16384 na dve cisla 0 a 64, atd., az skoncime instrukci LDIR a RET.

Jiste jste si vsimli, ze pri rozlozeni 16-ti bitoveho cisla na dve 8-mi bitova, musime tato cisla vlozit do dvou po sobe jdoucich adres vzdy obracene, nez bychom ocekavali. Tak to musi byt - viz manual Spectra, kapitola 25, odkud je i zpusob

rozkladu 16-ti bitoveho cisla. Pomoci nej muzeme i prikazem POKE vkladat do adres 16-ti bitova cisla takto :

```
POKE n, v - 256 * INT (v / 256)
POKE n + 1, INT (v / 256)
```

a naopak k ziskani teto hodnoty pomoci PEEK :

```
PEEK n + 256 * PEEK (n + 1)
```

Co jeste dodat? Ted to chce vzit manual Spectra a podivat se na kapitolu 25 a hlavne 26 a vyjasni se nam i zahada tam uvedeneho programu LD BC, 99 RET.

Totiz - z programu v jazyce Basic muzeme odskocit do strojoveho kodu, ktery jsme si za jistym ucelem vyvinuli, a po jeho vykonani se instrukci RET (posledni instrukce ve strojovem kodu) vracime do Basicu, nebo lepe receno za to misto, odkud byl proveden skok. (Neco podobneho jako GOSUB a RETURN v Basicu). Spectrum pritom umoznuje vynest ze strojoveho kodu obsah registroveho paru BC.

Podrobneji : vracime-li se ze strojoveho kodu instrukci RET do Basicu, pak instrukce USR, ktera skok do strojoveho kodu vyvolala, priradi obsah registroveho paru BC necemu. To neco je napr. RANDOMIZE, PRINT, LET A, apod.

Strojovy kod muzeme volat mnoha zpusoby, napr. :

1. RANDOMIZE USR 50000

nebo

2. LET A = USR 50000

nebo

3. PRINT USR 50000

Program pritom vzdy skoci na adresu 50000 a provede tam patricne instrukce. Pri navratu instrukci RET se v pripade 1. priradi do zdrojoveho cisla pro RND obsah BC. V pripade c. 2 nam pri navratu funkce USR da do promenne A (neni totozna s registrem A) obsah BC, tudiz obsah, ktery byl nacten z portu 31. Port 31 je i napr. pouzit u interfejsu s MHB 8255A ing. Soldana.

V pripade c. 3 se proste obsah registroveho paru BC vytiskne na obrazovku.

Proto i ve vyse uvedenem programu LD BC, 99 RET bude obsah registroveho paru BC vytistan na obrazovku. Jestlize napiseme program LD HL, 99 RET, vytiskne se nam na obrazovku opet obsah registru BC, nebude v nem vsak ulozeno to nase cislo 99, ale pravdepodobne jina hodnota - a ta se vytiskne. Cislo 99 je ulozeno v jinem registru (HL) a proto se nemuze vytisknout. Navic by melo dojit ke zhrouceni Basicu, protoze jsme zmenili HL, jestlize jsme pri tom nepouzili instrukce PUSH a POP.

Musime mit na pameti, ze Basic take pracuje se vsemi registry a tudiz, skaceme-li do strojoveho kodu, mely by nejdrive nasledovat instrukce typu PUSH vsech registrovych paru, ktere budeme v nasem strojovem programu pouzivat a na konci stroj. programu opet obnoveni registrovych paru na puvodni hodnotu pomoci instrukci POP. Netyka se to vsak registroveho paru BC.

Tak napriklad :

Instrukce PUSH HL ulozi obsah HL do zasobniku (jako do uschovny) a instrukci POP HL tuto hodnotu ze zasobniku (z uschovny) vybereme a znovu ulozime zpet do HL.

Pokud se toto ulozeni hodnot jednotlivych registru a jejich opetne obnoveni na puvodni hodnotu neprovede, muze se stat, ze zmenime Basicu jiste registry a on se z toho potom "zhrouti", protoze nevi, co ma delat.

Takze : chceme-li prenest ze strojaku nejake hodnoty (dvoubytove), nejsnaze to provedeme prave pomoci registru BC tim, ze do nej pred instrukci RET ulozime pozadovanou hodnotu, (to jest 16-ti bitove cislo), pomoci napr. LD BC,nn, nebo i jinak a potom provedeme RET, tedy navrat do Basicu.

Na zaver teto casti jeste program pro ulozeni obrazovky do pameti :

```
                                        Preklad
LD HL, 16384    ;nast. pocatecni adresy (33,0,64)
LD DE, 50000    ;nast. cilove adresy    (17,80,195)
LD BC, 6912     ;nast. citace           (1,0,27)
LDIR            ;provedeni prikazu      (237,176)
RET             ;navrat do Basicu       (201)
```

C A S T D R U H A

V prvni casti jsme se seznamili s usporadanim obrazovkove pameti a zkusili malou ukazku, jak "vyrobit" strojni program rucne. Protoze to je ponekud zdlouhave, budeme nadale pouzivat "prekladac", ktery nam program zapsany v assembleru ssm prelozi do strojniho kodu a pripadne i nahraje na pasek.

Je mozne pouzit ruzne druhy prekladacu, napr. GENS 3, my si podrobne vysvetlime priklad v Edit/Assembleru. Je sikovny, jednoduchy na obsluhu, prehledny - ma vice pismen na radek, a dovoluje snadne opravy textu, Mnohem snadneji, nez napr. GENS 3. Ten ma ovsem zase jine vyhody. Pozdeji se treba seznamime i s nim.

Jaky priklad zvolit? Uvedeme si jeden, ktery jsme zkouseli pro graficku tiskarnu, jez dokazala tisknout graficke znaky definovane grafiky, ale nemeli jsme pro ni program COPY pro tisk obrazovky - pro tisk obrazku. Presneji receno, tiskla obrazky jen po jednotlivych linkach, takze musela natisknout postupne vsech 24*8=192 linek, coz znacne dlouho trvalo. Chteli jsme, aby tiskla najednou cely znak 8*8 bodu, to jest vsech osm linek najednou, coz by rychlost tisku 8 krat zvysilo.

Program v Basicu pro tuto upravu byl celkem jednoduchy : bylo jen treba realizovat potrebne citace tak, aby prenesly vzdy postupne jednotlive znaky (8*8 bodu) z obrazovky do nekterehe ze znaku definovane grafiky (UDG). A to pocinaje levym hornim ctverecekem obrazovky (AT 0,0) a konce pravym dolnim (AT 23,32).

Prenos jsme provedli do UDG "A", to jest do prvniho znaku definovane grafiky, jehoz pocatecni adresa (skutecna) je 65368 a zabira dalsich 7 adres (celkem 8) pametovych mist za sebou, smerem nahoru (65369,65370,....).

Vypadal takto :

```
100 FOR p=0 TO 2          (3 bloky radku)
110 FOR r=0 TO 7          (8 radku v kazdem bloku)
120 FOR s=0 TO 31         (pocet znaku na radek)
130 FOR i=0 TO 7          (pocet linek v jednom znaku 8*8 bodu)
140 POKE USR "A"+i, PEEK (16384+i*256+s+r*32+2048*p)
150 NEXT i
160 LPRINT TAB s; "A"     ("A" = UDG grafika)
170 NEXT s
180 NEXT r
190 NEXT p
```

Fungovalo to, ale dost pomalu. Muzete si to vyzkouset i bez tiskarny napr. tak, ze na radek 160 date :

```
160 PRINT AT r+11+p*8, s; "A"    ("A" = UDG)
```

nebo podobne, ale o to nejde. Sermo to jen jako cvicny priklad, ktery mame za ukol "vyrobit" ve strojnim kodu.

Protoze nasobeni v assembleru bychom obtizne realizovali tak, jak je uvedeno, nejprve si tento Basicovy program trochu upravime :

```
 90 LET x = 0
100 FOR p = 0 TO 4096 STEP 2048
110 FOR r = 0 TO 224 STEP 32
120 FOR s = 0 TO 31
130 FOR l = 0 TO 1792 STEP 256
140 POKE USR "A" + x, PEEK (16384+l+s+r+p)
150 LET x = x + 1
160 NEXT l
170 LPRINT  a tak dale
180 NEXT s : NEXT r : NEXT p
```

Tim jsme se zbavili nasobeni a ted lze tento program prelozit temer "doslovne" do assembleru a strojoveho kodu.

Do Spectra si nahrajeme Edit/Assembler. Po natazeni se nam na obrazovce na modrem podkladu objevi vlevo nahore nuly (coz je cislo prvniho radku) a vpravo vedle nich, oddelen mezerou, blikajici kurzor (ukazovatko) tato vodorovna carecka _ . Pokud to bude jinak, bude program asi chybne nahrany a je treba jej nahrat znovu.

Protoze zatim nemame napsany zadny text, lze ukazovatkem pomoci sipek pohybovat jen vodorovne. Pozdeji, az bude napsan text, vzdy, kdyz bude kurzor blikat, muzeme s nim pohybovat i nahoru, nebo dolu - az na konec textu.

A muzeme zacit psat text programu. Nejprve stiskneme SPACE (mezeru), protoze na prvnim radku zadne navesti nebude. Potom napiseme ORG, dale opet mezeru a cislo 55000. Tim je prvni radek hotov. Psat budeme vse jen malymi pismeny. Tomuto prekladaci je to jednak jedno, je to i pro nas jednodussi - a navic, nedoporucujeme pouzit funkci CAPS LOCK - u nektere verze ji pak jiz nelze zrusit a jsou z toho potize.

Jak se dostat na druhy radek? Lze to jedine takto : stiskneme soucasne oba shifty (tedy CAPS SHIFT a SYMBOL SHIFT), treba i dvakrat po sobe, az se nam na miste blikajiciho ukazovatka (vodorovne carecky) ukaze maly neblikajici ctverecek, ktery znaci, ze jsme se dostali do rezimu prikazu. Potom stiskneme klavesu I a objevi se nam novy radek - opet se samymi nulami na zacatku a ukazovatko prestane blikat. Muzeme napsat novy radek a na dalsi se dostaneme jiz jednoduse - stiskneme ENTER. Nove vkladane radky budou mit na zacatku vzdy jen nuly, jejich precislovani provedeme kdykoliv, nebo nejlepe az na konec pred prekladem programu do strojoveho kodu. Po napsani druheho a dalsich radku uz tedy jen vzdy stiskneme ENTER a automaticky nam naskoci novy radek, (vzdy pri neblikajicim kurzoru). Soucasne se pritom take jiz napsany text automaticky rozdeli do jednotlivych sloupcu :

```
cislo radku     navesti     instrukce     operandy     poznamky

00000                       ORG           55000        ;prvni start
```

cili - nemusime vkladat tolik mezer, kolik je mezi jednotlivymi sloupci, ale musime vzdy vlozit aspon jednu, aby tomu prekladac rozumel. Nesmime take vkladat mezery tam, kam nepatri. Prekladac by je povazoval za konec jednoho sloupce a zacatek dalsiho.

Poznamky muzeme psat kamkoliv, musime jen pred ne umistit strednik (;). Poznamky samozrejme nemusime psat vubec. Pouzivaji se jen ke zprehledneni textu programu v assembleru.

Take sloupec "navesti" je nepovinny, ale nekdy navesti potrebujeme jako symbolicky nazev adresy, kterou pak pouzivame v programu - a nemuzeme ji nahradit cislem, protoze dopredu nevime, jake bude cislo mit. To lze zjistit az po prekladu. Prekladac temto navestim priradi pri prekladu spravna cisla

adres sam - a ta pak pouzije i ve strojnim kodu.

Ale vratme se k nasemu prvnimu radku. ORG je tzv. pseudoinstrukce - a ta se do strojniho programu neprekleda.

Z tohoto radku bude pouzito jen cislo 55000, protoze instrukce ORG 55000 informuje prekladac o tom, ze program ma zacinat na adrese 55000. Tato adresa bude tedy prirazena prvni strojni instrukci, ktera bude nasledovat. Take na pasek bude pozdeji prelozeny program nahran (ulozen) tak, ze po zpetnem natazeni do Spectra bude v jeho pameti opet ulozen pocinaje adresou 55000 a vyse.

Tim jsme si stanovili pocatecni adresu naseho budouciho strojniho kodu, kterou vsak muzeme kdykoliv zmenit pouhym prepsanim cisla instrukce ORG na prvnim radku, a po novem prelozeni prekladacem ulozit na pasek s novou adresou.

Kdyz se nam tedy podarilo dostat se na druhy radek, pisme dle jiz uvedenych pravidel dalsi text. Nejprve napiseme na dva radky instrukci PUSH.

A sice opet bez navesti :

```
PUSH  AF
PUSH  HL
PUSH  DE
```

Instrukce PUSH nam "uschova" do zasobniku (uschovny) obsahy techto registru, at je jiz v nich momentalne uschovano cokoliv, treba i nuly. Pri navratu (pred navratem) do Basicu tyto hodnoty opet vyvolame zpet a znovu ulozime na jejich byvala mista v jednotlivych registrech AF, HL a DE, aby byl zarucen spravny navrat na dalsi misto - instrukci v basicovem programu.

Je mozne, ze to v nasem pripade u vsech registru udelat nemusime, ale pro jistotu to radeji ucinme. (Pozdeji muzete experimentovat). Pri zpetnem vyvolavani pomoci instrukce POP (pred navratem do Basicu) musime jen dat pozor, abychom tak ucinili ve spravnem poradi, protoze obsahy registru AF,HL a DE jsou v zasobniku (uschovne) ulozeny "jedna na druhe" (za sebou), jako listy papiru v zasuvce, pricemz ta prvne ulozena je vespod a posledne ulozena je nahore. Proto budeme muset jako prvni vybrat DE, potom HL a nakonec AF.

Na dalsim radku nam zacina nas strojni program . Zacneme konstrukci tech ctyr (vlastne peti) citacu, ktere potrebujeme. Zajiste se daji "udelat" i jinak. My si pro ne urcime 5 dvojitych pametovych mist (po dvou byte), abychom do nich mohli ukladat jakekoliv 16-ti bitove cislo - pomoci instrukce LD.

Kazdy citac bude tedy "vlastnit" dve po sobe jdouci adresy. (Do kazde z nich lze ulozit 8-bitove cislo - tedy 0-255). Budeme do nich (presneji receno do prvni adresy) ukladat cisla vzdy ve forme 16-ti bitu, aby byla spravne ulozena, jak jsme se jiz dozvedeli v prvni casti.

Na radku 50 je zacatek (prvni adresa) prvniho citace - presneji receno jeho adresa, na niz bude vzdy ulozena jeho momentalni hodnota. Musime napsat navesti, protoze je budeme potrebovat k tomu, abychom mohli obsah citace menit, nebo zjistovat, jakou ma prave hodnotu. Pro zacatek programu ulozime do vsech citacu same nuly. (Stejne jako v Basicu). To nejsnaze provedeme tim, ze ke kazde (prvni i druhe) adrese kazdeho citace napiseme instrukci NOP. Tato instrukce nic neprovadi a proto bude temto adresam prirazen obsah nula.

Pisme tedy :

```
00050 acit1 nop     ;ulozi nulu
00060       nop
00070 acit2 nop
```

atd. az po radek 140.

Jako navesti jsme zvolili slova ACIT1, ACIT2, atd., coz jsou zkratky : Adresa CITace 1, Adresa CITace2 a pod., az adresa citace 44. Navesti ami mit i vice pismen, prekladac vsak

"uznava" a rozeznava jen 6 znaku. Musi zacinat pismenem a nesmi to byt tzv. rezervovana slova, jako napr. NOP, PUSH, LD, HL a pod., viz jejich seznam v navodu k Edit/Assembleru. Zadne z techto rezervovanych slov neni delsi nez ctyri pismena a neobsahuje cisla ani jine znaky. Ostatne prekladac by nas na to pri prekladu stejne asi upozornil ohlasenim chyby.

Po napsani radku 50-140 jsme si tak vlastne urcili v budoucim programu 5 pametovych mist (16-ti bitovych) pro nasich 5 citacu, ktere potrebujeme. Ten paty citac (44) je nahrada za LET x = v basicovem programu vyse uvedenem.

Protoze ulozeni nul do adres citacu pomoci NOP je jen prozatimni, dokud do nich neulozime neco jineho (pak tam jiz nebude NOP = 0, ale nejake jine cislo), musime si vytvorit take "nulovani" citacu, (a budeme je i potrebovat) jako ma Basic napr. FOR 1 = 0 TO 7.

Proto na dalsich radcich zdanlive zbytecne znovu do citacu budeme ukladat nuly. Jen citac 1 nechame az na konec programu. Citace budeme totiz muset po jejich skonceni citani vzdy zase vynulovat, aby mohly opet znovu plnit svou funkci - zacit zase citat od zacatku, to jest od nuly.

Proto na dalsi radek budeme psat :

```
000150 zcit2 ld hl,0     ;uloz do 16-ti bitoveho registru HL nulu
000160       ld (acit2),hl     ;prenes tuto nulu do adresy ACIT2
000170 zcit3 ld hl,0
000180       ld (acit3),hl
```

a tak dale, viz priloha s napsanym celym programem.

Jak vidite, opet jsme pouzili navesti - vzdy na zacatku nulovani kazdeho citace, protoze tyto vstupni body (adresy) budeme take potrebovat, abychom na ne mohli v programu "odskakovat" a "spoustet" tim jednotlive citace (nebo i vice) opet od nuly.

Je to stejne jako ve vyse uvedenem prikladu v Basicu. A protoze tato navesti jsou na radcich, kde zacina nulovani jednotlivych citacu, nazvali jsme je ZCIT2, ZCIT3, atd., (jako Zacatek CITace 2, 3 a pod.). Mohli jsme je ovsem nazvat jakkoliv jinak - dle pravidel.

Instrukce LD HL,0 znamena : do registroveho paru (registru) HL uloz cislo 0.

Instrukce LD (ACIT2),HL znamena : cislo ulozene v registru HL prenes a uloz do adresy citace 2, cili do (ACIT2). ACIT2 musi byt v zavorce, protoze se jedna o obsah ulozeny na teto adrese, nikoliv o cislo adresy. Cislo se ulozi do obou adres, t.j. do adresy ACIT2 i do ACIT2 + 1. Jestlize bude ACIT2 napr. 55011, pak se cislo (tentokrate 0) ulozi do 55011 i do 55012, protoze instrukci LD HL (nebo LD (ACIT2),HL) prenasime vzdy najednou dvoubytovou hodnotu, t.j. 16-ti bitove cislo, i kdyz je to treba nula, jako v tomto pripade.

Kdyz mame takto vytvoreny jak pameti pro citace, tak i jejich nulovani, muzeme pristoupit k prelozeni Basicoveho radku

```
140 POKE USR "A" + x, PEEK (16384+l+s+r+p)
```

x je v nasem pripade citac 44. l = citac 4, s = citac 3, r = citac 2 a p = citac 1. Presneji receno, obsah jejich adres ACIT44, ACIT4, ACIT3, ACIT2 a ACIT1.

Protoze dalsi radek budeme opet pouzivat ke skoku, musime mu zase pridelit navesti, abychom se meli nac odvolavat, respektive abychom mohli mikroprocesor poslat na tuto adresu, i kdyz jeji cislo neznáme. Nazveme je treba POKEA - coz nam bude pripominat, ze zde zacina POKE USR "A" a PEEK, neboli vlastni premisteni dat z pameti obrazovky do UDG "A".

```
Piste :   00230 pokea ld de,(acit1)
          00240       ld hl,(acit2)
          00250       add hl,de
```

a tak dale

LD DE, (ACIT1) znamena : prenes do registru DE obsah, ACIT1 to jest obsah adresy citace 1, neboli cislo, ktere je na teto adrese ulozeno. Prenese se opet cele dvoubytove (16-ti bitove) cislo, i kdyz je to treba 0 nebo i a pod.

Podobne LD HL,(ACIT2) prenese obsah citace 2 do HL. Na dalsim radku jsme pouzili instrukci ADD HL,DE , ktera nam cisla ulozena v HL a DE secte a vysledek ulozi do HL. K tomuto vysledku jeste potrebujeme pricist obsahy citacu 3 a 4 a cislo 16384 (coz je zacatek - prvni adresa - obrazovky), soucet uskladnit v HL a potom to pouzit jako adresu pameti obrazovky, odkud chceme precist (a prenest) jedno byte dat (8bitu), (t.j. jednu vodor. linku 8 bodu dlouhou) odtud do pameti UDG "A", neboli na adresu 65368 + x (dle basicoveho programu), nebo-li na adresu 65368 + obsah citace 44.

Toto provedeme takto :

Jednotlive soucty pomoci ADD HL,DE mame jiz provedeny na radku 290 a tim zaroven i uskladneny v registru HL. Nyni napiseme :

```
          00300       ld a,(hl)
```

coz znamena : do registru A (ktery je velikosti jen jednoho byte, nebo-li 8 bitu) preneseme (HL), coz je obsah ulozeny na adrese, jejiz cislo prave obsahuje registr HL. Musi byt v zavorce, protoze jinak by se preneslo cislo v HL a nikoliv cislo z pametovano mista s adresou, ktera je v HL.

Do registru A, ktery je jen 8-bitovy a tudiz jen pro mala jednobytova cisla, by se stejne dvoubytove cislo z HL ulozit nedalo a take ani takova instrukce (LD A,HL) neexistuje a proto i prekladac by ohlasil chybu.

Co jsme to prave provedli? Do registru A (muzeme si jej predstavit jako promennou A velikosti jen pro 8-bitova cisla) jsme tim ulozili hodnotu z "obrazovky" - t.j. jednu vodorovnou linku z jednoho znaku 8*8 bodu. Nyni ji presuneme na patricne misto, t.j. na adresu (65368 + obsah citace 44) takto :

```
00310       ld de,(acit44)  ;do DE obsah z adresy ACIT44
00320       ld hl,65368     ;do HL cislo 65368
00330       add hl,de       ;soucet DE + HL, vysledek do HL
00340       ld (hl),a       ;stejne jako POKE HL,A v basicu
```

Dejme tomu, ze v citaci ACIT44 bude ulozena nula. Potom radek 330 - ADD HL,DE nam da vysledek v HL = 65368 a radek 340 - LD (HL),A ulozi na adresu 65368 cislo = hodnotu ulozenou v A. Nebo-li to bude totez, jako kdybychom v basicu napsali :

```
POKE 65368, A  =  (POKE HL, A)
```

Zde vidime, proc jsme citac 4 museli udelat "dvojity" - citac 4 a citac 44. Kazdy z nich totiz muze mit jinou hodnotu. Citac 44 by napr. mohl mit rovnou pocatecni hodnotu 65368 a o 1 ji vzdy zvysovat. Potom bychom usetrili i nektere radky programu a radek 340, ktery ma provest POKE HL,A, by mohl byt takto :

```
00340     LD (ACIT44),A
```

Na zaver muzete zkusit tento program minimalizovat a vubec experimentovat jakkoliv. Ale pokracujme.

Prvni prenos z obrazovky mame za sebou a ulozen v 65368. Nyni musime zvysit citac 4 o 256 a citac 44 o 1. Piste :

```
00350       ld de,(acit4)   ;uloz do DE obsah citace 4
00360       ld hl,256       ;uloz do HL cislo 256
00370       add hl,de       ;secti a uloz do HL
00380       ld (acit4),hl   ;uloz do cit.4 novou hodnotu,to jest
                            ;puvodni obsah o 256 vyssi
```

```
00390       ld hl,acit44    ;uloz do HL adresu citace 44
00400       inc (hl)        ;pricti k obsahu adresy v HL jednicku
```

Na radku 390 LD HL,ACIT44 je ACIT44 bez zavorek, to znamena,ze do registru HL se prenese samotne cislo adresy citace 44, nikoliv jeho obsah.
Radek 400 INC (HL) zpusobi pricteni jednicky (inkrementaci) a protoze (HL) je v zavorkach, pricte tuto jednicku k obsahu adresy ulozene v HL, to jest rovnou k obsahu citace 44.

```
00410       ld a,7          ;ulozime do A cislo 7, coz je konecna
                            ;hodnota citace 44, a srovname ji na
                            ;dalsim radku se skutecnym momentalnim
                            ;obsahem citace 44
00420       cp (hl)         ;CP (HL) zpusobi odecteni obsahu adresy
                            ;ulozene v (HL) (proto zavorky), tedy
                            ;obsahu citace 44 od registru A.
00430       jp m,konec      ;je-li vysledek zaporny, skoc na adresu
                            ;KONEC
00440       jp pokea        ;skoc na adresu POKEA
```

Instrukce CP I. je instrukce porovnani. Porovnava obsah registru A s operandem I., v nasem pripade (HL).Ve skutecnosti odecte od registru A operand I. Na vysledku zavisi, jak budou nastaveny stavove indikatory. Toho vyuzijeme na dalsim radku u instrukce JP M,KONEC, ktera provede odskok na adresu KONEC jen tehdy, je-li vysledek CP zaporny, to znamena tehdy, az bude obsah citace 44 = 8. Jinak dalsi radek 440 JP POKEA posle mikroprocesor na adresu POKEA - k prenaseni dalsi casti obrazovky - to jest druhe linky, pak treti atd., az je prenesena vsech 8, tedy cely znak 8x8 bodu.
V pripade, ze je jiz prenesano vsech osm linek (cely prvni znak AT 0,0), skoci program na adresu KONEC a do Basicu, kde vytiskne, nebo jinak pouzije definovanou grafiku v "A". Potom se opet vrati zpet do strojoveho kodu a sice na radek 450 s navestim NAVRAT, coz musi byt ucineno v Basicu prikazem RANDOMIZE USR NAVRAT (na miste slova NAVRAT tentokrate pochopitelne bude skutecne cislo adresy, ktere zjistime po prekladu programu prekladacem).
Citac 4 testovat nemusime, protoze pracuje souhlasne s citacem 44. Piste dale :
000450 navrat push af a tak dale.
Jak jsme jiz uvedli, navesti NAVRAT nam oznacuje adresu, na kterou se mikroprocesor vrati z Basicu, aby pokracoval v nasem strojnim programu. Opet "odlozime" jeho momentalni hodnoty registru AF, HL a DE do uschovy na zasobnik instrukcemi PUSH a muzeme pokracovat.
Nyni, protoze chceme prenest dalsi znak z obrazovky (to jest AT 0,1), zvysime obsah citace 3 o jednicku - stejny postup jako u citace predchazejiciho - t.j. 44.

```
00480   ld hl,acit3     ;ulozi do HL adresu citace 3
00490   inc (hl)        ;obsah teto adresy zvetsi o 1 prictenim 1
00500   ld a,31         ;do A ulozime 31 (pocet znaku na radek)
00510   cp (hl)         ;od A odecte obsah adresy ulozene v HL,
                        ;tedy obsah citace 3.
00520   jp p,zcit4      ;je-li vysledek instrukce CP nezaporny,
                        ;vrati se program na adresu ZCIT4, coz je
```

Zacatek CITace 4, vynuluje ho a take citac 44,...atd...a prenese dalsi znak, skoci do Basicu, pak zpet na NAVRAT a tak dlouho to vse bude opakovat, dokud CP (HL) neda vysledek zaporny, cili

dokud neprenese vsech 32 znaku nulteho radku obrazovky.
Instrukce JP P tentokrat vyuziva jiny priznak indikatoru S, oznaceny P, ktery zpusobi, ze skok na ZCIT4 bude jen pri nezapornem vysledku. (Viz napr. Assembler JZD Slusovice a pod).
Potom zvysime obsah citace 2 takto:

```
00590   ld hl,(acit2)     ;do HL obsah citace 2
00540   ld de,32          ;do DE cislo 32
00550   add hl,de         ;secte HL+DE, vysledek ulozi do HL
00560   ld (acit2),hl     ;vysledek predchoziho souctu, nyni
                          ;ulozeny v HL, prenese do adresy ACIT2,
                          ;cili obsah citace 2 se zvysi o 32.
```

Nyni potrebujeme otestovat stav citace ACIT2 :

```
00570   ld hl,(acit2)     ;muze se vypustit, nebo ne? Zkuste to!
00580   ld de,253         ;do DE dame 253
00590   sbc hl,de         ;od HL odecte DE, vysledek ulozi do HL
00600   jp m,zcit3        ;podle priznaku M provede bud odskok na
                          ;ZCIT3, nebo jde na dalsi radek.
```

Instrukce SBC HL,DE nam od registru HL, v nemz mame ulozen obsah citace ACIT2, odecte hodnutu ulozenou v DE (253) s umozni tim porovnani, podobne jako instrukce CP, ktera je ale jen 8-bitova, zde jsme jiz na hranici 8 bitu a proto pouzijeme tuto moznost (SBC), jez nam umozni i 16-ti bitove operace.
Cislo 253 by melo spravne byt 256, ale protoze citac se zvysuje vzdy po 32, klidne muzeme pouzit i cislo 253. K tomu, abychom zjistili, zda vysledek SBC bude zaporny (JP M), ci nikoliv, to vyhovuje. Instrukce SBC totiz odcita spolu s druhym operandem i indikator C (CY) a tak vysledek pri vlozeni 256 by nemusel byt spravny. Kdo chce vyzkouset, prosim, muze. My budeme pokracovat.
Radek 590 JP M,ZCIT3 - v pripade, ze citac 2 jeste nedosahl "plocha" stavu 253, t.j. tehdy, kdyz priznak M je nastaven, protoze vysledek SBC je zaporny, coz nutne znamena, ze stav citace 2 nedosahl 253 a ve skutecnosti tedy ani 256, jde program zpatky na adresu ZCIT3 - tedy na nulovani citace 3 atd...
Teprve po prekroceni obsahu citace 2 pras 253 (256) pujde na dalsi radek (610) na zvetseni citace 1 ;

```
00610   ld hl,(acit1)     ;do HL obsah citace 1
00620   ld de,2048        ;do DE 2048 (hodnota o kterou se ma
                          ;citac 1 zvysit
00630   add hl,de         ;secte HL+DE a vysledek ulozi do HL
00640   ld (acit1),hl     ;na adresu ACIT1 (do citace 1) ulozi
                          ;obsah HL, cili zvysi citac o 2048.
```

A opet testovani - tentokrate citace 1 :

```
00650   ld hl,(acit1)     ;muze se take vypustit?
00660   ld de,6000        ;nastaveni maxima citace 1
00670   sbc hl,de         ;odecteni
00680   jp m,zcit2        ;testovani
```

Tyto ctyri radky neni treba vysvetlovat, je to stejne jako u predchazejiciho citace ACIT2. Jen maximalni hodnota 6000 a krok 2048 citace 1 jsou jine. Opet jsme maximalni hodnotu dali o neco nizsi, nez skutecnych 3k2048, ale to staci. Dalsim krokem by ji prekrocil tak jako tak.
Na dalsich radcich nasleduje nulovani citace 1, pro eventualni dalsi prenos dalsi obrazovky. Nemuzeme v nem zanechat hodnotu

kterou dosahl, to je jasne. Proto :

```
00690  ld hl,0
00700  ld (aciti),hl
```

a nyni jiz zbyva jen ukonceni :

```
00710 konec pop de
00720       pop hl
00730       pop af
00740       ret
```

Pomoci instrukci POP DE, atd., vratime odlozene hodnoty registru DE, HL a AF zpet ze zasobniku tam, kam patri, t.j. do DE, HL a AF a muzeme se vratit do Basicu, coz provede instrukce RET. (Viz manual Spectra).

Kdyz mame program zepsany, je vhodne provest ocislovani (precislovani) radku. Provedeme to takto : stiskneme soucasne oba shifty, abychom se dostali do rezimu prikazu. Ze v nem jsme, nam signalizuje bily ctverecek, ktery se objevi misto kurzoru.

Potom stiskneme SPACE (mezeru) a dostaneme se tak do rozsireneho rezimu prikazu. Dole na obrazovce se objevi COMMAND =>. Napiseme N, nebo N10,10, coz je totez a po stisknuti ENTER se nam radky precislujii od prvniho s cislem 10 po krocich 10.

Chceme-li jiny krok a pod., napiste N neco,neco, atd..,

Nyni si muzeme text programu (neprelozeneho) nahrat na pasku. Nebo i kdykoliv potom, az nam napr. prekladac po prekladu ohlasi, ze jej mame bez chyb.

Nahravka textu na pasek se provede opet v rezimu rozsirenych prikazu (COMMAND) vlozenim Sjmeno. Natazeni nejakeho textu (assembleru psaneho v Edit/Assembleru) provedeme take v COMMAND vlozenim Ljmeno.

PREKLAD programu do strojoveho kodu s naslednym ulozenim na pasku :

Rezim COMMAND a vlozit Ajmeno. Zacne preklad. Ten muzeme kdykoliv pozastavit, napr. proto, abychom zjistili, kde jsou pripadne chyby, nebo cisla adres, treba klavesou ENTER a dalsim stisknutim opet spustit.

Na konci prekladu se objevi zprava o chybach, tabulka navesti a Start the tape..... Pokud je program bez chyb, muzeme jej ulozit na pasku. Dajl se ovsem pouzit i mnohe jine prikazy Edit/Assembleru, ty jsou uvedeny v jeho manualu.

Snad jeste tohle : preklad bez ulozeni na pasku provedeme : v rezimu COMMAND napiseme A/NO, prekladat muzeme kolikrat chceme, prikazem RUN odstartujeme strojni program, ale u tohoto naseho to nezkousejte, vymaze pocitac. Zkouset jej muzete jen bez Edit/Assembleru.

Opravy v textu : stisknout oba shifty, pri ctverecku stisknout I a tim vlozime novy radek tam, kam jsme predtim umistili kurzor. Dalsi novy radek pak uz jen pomoci ENTER.

Pouziti tohoto programu? S malymi upravami se daji napr. jakekoliv jednotlive ctverecky obrazovky uskladnovat v libovolne casti pameti a zobrazovat potom jako definovana grafika a pod.

K podrobnejsimu seznameni se s pouzitymi instrukcemi je potreba pri programovani pouzivat vhodnou literaturu. Napr. jiz zminenou knihu ASSEMBLER Z-80, kterou vydalo JZD Slusovice, kde jsou vysvetlovany i priznaky pro porovnani atd.

K teto casti patri jeste vytisk uvedeneho programu v assembleru.

Nu a nakonec si povime neco o tom, jestli tento program fungoval a o jeho dalsich upravach.

```
00010 start  org  55000     ;stanoveni pocatecni adresy programu
00020        push af        ;ulozeni do zasobniku
00030        push hl
00040        push de
00050 acit1  nop            ;adresa citace 1
00060        nop
00070 acit2  nop            ;adresa citace 2
00080        nop
00090 acit3  nop            ;adresa citace 3
00100        nop
00110 acit4  nop            ;adresa citace 4
00120        nop
00130 acit44 nop            ;adresa citace 44
00140        nop
00150 zcit2  ld hl,0        ;zacatek nulovani citace 2
00160        ld (acit2),hl
00170 zcit3  ld hl,0        ;   "       "      "   3
00180        ld (acit3),hl
00190 zcit4  ld hl,0        ;   "       "      "   4
00200        ld (acit4),hl
00210 zcit44 ld hl,0        ;   "       "      "   44
00220        ld (acit44),hl
00230 pokea  ld de,(acit1)  ;zacatek prenosu dat (PEEK a POKE)
00240        ld hl,(acit2)
00250        add hl,de      ;soucet obsahu citace 1 a 2
00260        ld de,(acit3)
00270        add hl,de      ;pricteni obsahu citace 3
00280        ld de,16384    ;poc.adresa obrazovky
00290        add hl,de      ;pricteni 16384 k HL
00300        ld de,(acit4)
00310        add hl,de      ;pricteni obsahu citace 4
00320        ld a,(hl)      ;jako PEEK HL --> ulozi do A
00330        ld de,(acit44)
00340        ld hl,65368    ;poc. adresa prvni UDG grafiky
00350        add hl,de      ;soucet obsahu cit.44 + 65368
00360        ld (hl),a      ;jako POKE HL,A
00370        ld de,(acit4)
00380        ld hl,256
00390        add hl,de
00400        ld (acit4),hl  ;zvysi obsah cit.4 o 256
00410        ld hl,acit44
00420        inc (hl)       ;zvyseni cit.44 o 1
00430        ld a,7         ;zacatek testovani cit.44
00440        cp (hl)        ;odecteni
00450        jp m,konec     ;porovnani, skok nebo ne
00460        jp pokea       ;skok na adresu POKEA
00470 navrat push af        ;adresa navratu z Basicu (2.start)
00480        push hl
00490        push de
00500        ld hl,acit3
00510        inc (hl)       ;zvyseni citace 3 o 1
00520        ld a,31        ;zacatek testovani citace 3
00530        cp (hl)        ;odecteni
00540        jp p,zcit4     ;porovnani, bud skok, nebo ne
00550        ld hl,(acit2)
00560        ld de,32
00570        add hl,de
00580        ld (acit2),hl  ;zvyseni citace 2 o 32
00590        ld hl,(acit2)  ;muze se vypustit
00600        ld de,253      ;zacatek testovani citace 2
00610        sbc hl,de      ;odecteni
```

```
00600          jp m,zcit2          ;porovnani, skok, nebo na
00610          ld hl,(aciti1)      ;zacatek zvetseni cit.1
00620          ld de,2048
00630          add hl,de
00640          ld (aciti),hl       ;zvyseni cit.1 o 2048
00650          ld hl,(aciti1)      ;muze se vypustit
00660          ld de,6000          ;zacatek testovani cit.1
00670          sbc hl,de           ;odecteni
00680          jp m,zcit2          ;porovnani, skok, nebo na
00690          ld hl,0             ;zacatek nulovani cit.1
00700          ld (aciti1),hl      ;vlozeni nuly
00710 konec    pop de              ;vybrani zasobniku
00720          pop hl
00730          pop af
00740          ret                 ;navrat do Basicu
```

Pri zkouseni tohoto programu jsme museli provest nekolik zmen, jak je videt na vypisu provedeneho pomoci programu HEXMON.

Tak predevsim jsme adresy citacu presunuli az na konec programu, a vypustili jsme zbytecne radky . Nulovani citace 1 jsme presunuli na zacatek, pred nulovani citace 2. Potom jiz program pracoval perfektne - a i pri preruseni a opetnem znovuuvedeni do chodu, to jest pri novem startu, byly vzdy vsechny citace spravne vynulovany atd...

Adresy citacu jsme museli dat az na konec proto , protoze pri preruseni programu nedoslo k vynulovani citacu a pri novem startu od pocatecni adresy nam jejich ciselne obsahy privadely pocitac do rozpaku. Pri novem startu od zacatku musi byt obsahy vsech citacu nuly.

Vypustili jsme take POP a PUSH AF.

CAST CTVRTA - NECO O R O M

ROM je trvala pamet, to znamena, ze ji nelze zmenit a ze i po vypnuti pocitace v ni zustava stale ulozen puvodni program.

Je to program slouzici k "rozbehnuti" Spectra, obsahuje predevsim jazyk Basic a take ruzne podprogramy, z nichz nektere se daji vyuzit i pri tvorbe nasich vlastnich programu. Ty nas budou zajimat predevsim.

Je velmi jednoduche si tento program v ROM prohlednout na obrazovce pomoci nejakeho disassembleru, nebo si jej i vytisknout na tiskarne. Ale porozumet celemu programu ulozenemu v ROM je velmi nesnadne, proto se o to ani nebudeme pokousset. Disassembler je opak assembleru - to jest, preklada strojovy program zpet do assembleru. Tomu je jiz lepe rozumet. Pomoci zvlastnich programu (napr. MONS3 nebo HEXMON) si muzeme potom takto prelozene programy prohlizet, sledovat, kam vedou jejich jednotlive vetve a pod.

Ale vratme se k ROM. Zabira nejnizsi cast adres od 0 az po obrazovkovou pamet, t.j. konci adresou 16383. Jestlize vlozime do pocitace RANDOMIZE USR 0, zacne mikroprocesor provadet program od adresy 0. Je to stejne, jako kdyz Spectrum vypneme. Pameti RAM se vynuluji, do nekterych se ulozi patricne informace z ROM a na obrazovce se objevi znama zprava, ze pocitac je pripraven k provozu.

Tento prikaz - RANDOMIZE USR 0 - se da tedy vhodne pouzit. Napr. k vynulovani celeho pocitace misto vypinani napajeciho napeti, nebo k zabudovani do programu jako ochrana pred nezadouci manipulaci a pod.

Podprogramy v ROM

	HEXADECIMALNE	DEKADICKY	
COPY	0EAC	1196	
NEW	11B7	4535	
CLS	0D6B	3435	
SCROLL	0DFE	3582	
RANDOMIZE	1E4F	7759	
PLOT	22E5	8933	(Y do B; X do C)
BEEP	03B5	949	(HL-delka; DE-vyska)

Podprogramy v ROM pro klavesnici

SCAN KEYS	028E	654	
KEY CODE	031E	798	

Vsechny podprogramy se volaji : CALL adresa (instrukce CALL je volani podprogramu - jako GOSUB v Basicu. Na konci podprogramu musi byt RET, nebo RET I.)
Nektere pri vyvolani nevyzaduji nic dalsiho, napr. CLS, SCROLL, klavesnice. Jine vyzaduji, aby v urcitych registrech byly jiste hodnoty.

Poznamka :
(Uvedene podprogramy jsou napr. pouzivany i v ukazce strojoveho kodu - v programu SNAKE k Edit/Assembleru).

Vidime, ze nektera z techto podprogramu jsou, nebo mohou byt pro nas zajimave.
Napr. CLS - nemusime tvorit vlastni strojovy program pro CLS, ale jednoduse zaradime do naseho programu v assembleru instrukci CALL 0D6B (hexa) a podprogram v ROM to ucini za nas - provede CLS a vrati se zpet do naseho strojniho programu k dalsi instrukci. Vyzkousejte si to klidne i bez assembleru. Co se stane, kdyz vlozime :

```
RANDOMIZE USR 1196      (= 0EAC HEXA)
RANDOMIZE USR 4535      (= 11B7   "  )
RANDOMIZE USR 3435      (= 0D6B   "  )
RANDOMIZE USR 3582      (= 0DFE   "  )
```

Udelejte to ovsem jen tehdy, jestli nemate ve Spectru cenny program. Mohli byste o nej prijit.

Priklad k podprogramu SCAN KEYS a KEY CODE (v ROM)

Nejdrive neco uvodem : jestlize chceme vyuzit nektere casti ROM, ucinime to prikazem : "jdi na adresu tu a tu, proved podprogram a vrat se zpet". Cili neco podobneho, jako je GO SUB a RETURN v Basicu.
V Basicu muzeme k tomu pouzit instrukci RANDOMIZE USR adresa - a po provedeni podprogramu se nam mikroprocesor vrati zpet do Basicu.
Ve strojovem kodu pouzijeme instrukci CALL I. (adresa, nebo navesti). Instrukce CALL provadi volani podprogramu. Navratova adresa se ulozi do zasobnikove pameti tak, jako by se ukladala instrukci PUSH a do programoveho citace se zavede adresa, dana operandem I. Operandem I. muze byt navesti, nebo primo cislo adresy. Na konci podprogramu musi byt instrukce RET,nebo RET I., ktera zpusobi navrat zpet na dalsi instrukci hlavniho programu

(jako RETURN). Podobne podminena instrukce RET I. provede podmineny navrat z podprogramu - a to jen tehdy, je-li splnena podminka dana operandem I. Jinak se pokracuje dalsi instrukci, ktera je za RET I. Operandem I. je nektera z podminek C, NC, Z, NZ, PE, PO, M - viz nasledujici tabulku podminek podminenych instrukci JP, CALL a RET.

I.	Indikator	Stav indikatoru	Stav, splnujici podminku
NZ	Z	nulovan	vysledek nenulovy
Z	Z	nastaven	vysledek nulovy
NC	C	nulovan	bez prenosu
C	C	nastaven	s prenosem
PO	P/V	nulovan	licha parita
PE	P/V	nastaven	suda parita
P	S	nulovan	vysledek nezaporny
M	S	nastaven	vysledek zaporny

Tabulka 4.11.-i Tabulka podminek instrukci JP, CALL a RET

Tuto tabulku jsme vyuzivali i v predchazejici casti u instrukci skoku - JP.
Poznamka :
U podprogramu v ROM jsou samozrejme jiz instrukce RET zabudovany.

Nasleduje ukazka testovani klavesnice (za pouziti podprogramu v ROM - SCAN KEYS a KEY CODE) :

```
NAV 1   CALL 654  (028EH)   ;odskok do podprogramu v ROM, kde
                             testuje dve klavesy
        RET NZ              ;jsou-li stisknuty 2 klavesy, nastavi
                             se Z=0 a program skoci zpet do toho
                             programu, odkud se sem dostal
        CALL 798  (031EH)   ;skok do ROM, kde testuje 1 klavesu
        JP NC, NAV 1        ;neni-li stisknuta zadna klavesa, je
                             stav indikatoru  C=0  a program se
                             vrati na NAV 1. Je-li stisknuta
                             nejaka klavesa, jde na dalsi instr.
        CP N                ;porovna kod stisknute klavesy s N
        JP Z, NECO          ;je-li kod klavesy = N, tak skoci na
                             navesti NECO. Neni-li kod klavesy=N,
                             pokracuje dalsi instrukci
        JP NAV 1            ;jdi  na  NAV  1  -  vratime jej na
                             pocatek tohoto programu k dalsimu
                             testovani.
NECO    ;po stisknuti prislusne klavesy provede program  ulozeny
         od adresy NECO. Na konci  tohoto  programu  muzeme  dat
         opet JP NAV 1 a tak znovu a znovu testovat klavesnici.
```

Tech instrukci CP N a JP Z, NECO JINEHO muzeme samozrejme za sebou seradit celou radu a testovat tak tolik klaves, kolik chceme. Podobne jako v Basicu pomoci :
IF INKEY$ = "a" GO TO (nebo GO SUB), a tak dale.

Na miste hodnoty N v CP instrukci muzeme napsat rovnou pismeno uvedene na klavese, napr. takto :

```
CP 'a'   ;testuje klavesu  "a"  (Pozor, je rozdil mezi "A" a "a")
CP 'k'   ;testuje klavesu  "k"
CP '4'   ;testuje klavesu 4, a tak dale.
```

Z uvedeneho prikladu vyplyva, ze :

1. Podprogram v ROM ma vlastne za ukol ulozit do registru A hodnotu stlacene klavesy (napr.pismeno k), protoze porovnavaci instrukce CP provadi ve skutecnosti odecteni operandu (k) od registru A a dle vysledku nastavi hodnoty stavovych indikatoru (Z) ve vyse uvedene tabulce 4.ii.-i. Nasledujici JP Z, NECO pak dle hodnoty indikatoru Z provede bud odskok na NECO, nebo pokracuje dal.

2. Dale z prikladu vyplyva, ze za hodnoty 8-bitovych cisel (operandu) v instrukcich, ve kterych muzeme uvadet primo 8-bitova cisla, muzeme tato cisla uvest ve forme pismen, nebo cislic (tedy znaku) v jednoduchych uvozovkach. Napiseme-li napr. LD A, 'x' ;mikroprocesor ulozi do registru A zdanlive pismeno x, ve skutecnosti vsak ulozi do registru A jen 8-bitove cislo, ktere je podle kodu ASCII prirazeno pismenu x, tedy tam ulozi jeho CODE. Timto zpusobem se dali do registru (a ovsem i do pameti) ukladat pismena - a vsechny znaky vubec, a opet je z nich "vybirat". Je to stejne, jako ukladani cisel, jen misto cisel muzeme psat rovnou pismena v jednoduchych uvozovkach.
Ale a k cemu toto testovani klavesnice pouzit? Krome jineho napr. i k preruseni jakehokoliv strojniho programu, probihajiciho ve smycce. Staci jen vlozit toto testovani dovnitr smycky (nebo citace) a po stisknuti patricne klavesy program ze smycky vystoupi a pujde tam, kam jej instrukce JP Z, NECO posle. Nebo take pomoci RET do Basicu. Do te smycky vsak nemusime vlozit cely tento testovaci program, staci tam jen vlozit instrukci skoku do naseho testovaciho podprogramu (napr.CALL) a pomoci RET NZ a podobne, se zase vracet do smycky hlavniho programu, nebo.....atd.

Priklad pro vyuziti podprogramu PLOT na adr. 22E5H

Chceme nakreslit bod o souradnicich PLOT 200, 150 (souradnice x = 200, y = 150).
V assemblery to muzeme udelat takto :

```
00030       LD  B, 150
00040       LD  C, 200
00050       CALL 22E5H
00060       RET
```

Po prekladu do strojoveho kodu a jeho spusteni se objevi bod, ktery ovsem diky posuvu kursoru poskoci o osm linek nahoru.

Jak psat (v Basicu) na spodni dva radky (AT 22 a AT 23):

Zkuste napsat tento program :

```
1 PRINT #0; "Toto je radek AT 22, 0"
2 PRINT #1; "Toto je radek AT 23, 0"
3 GO TO 2
```

Radek GO TO 3 je tam proto, aby nam na dolni radek nenapsal pocitac hlaseni O.K. o tom, ze splnil program.
Zkuste si zaexperimentovat, dalsi podrobnejsi informace naleznete napr. v "Programovani ve str. kodu" I. a III. dil.

Napsal F.Jochec asn. + jun. pro ZO SVAZARM KAROLINKA.