ABC STROJOVÉHO KÓDU

Pro počítače typu:

DIDAKTIK SPECTRUM

díl 1

JH

DIDAKTIK SPECTRUM

díl 1

ABC STROJOVÉHO KÓDU Z-80

Vám, kteří jste se odhodlali zakoupit si toto dílko, srdečně blahopřeji. Rozhodli jste se správně, protože pouze pomocí strojového kódu (lidově "strojáku") se stanete plnými a neomezenými vládci svého Didaktiku (či Spectra). Ty pokročilejší z vás chci upozornit, že tato prvá část je určena pouze naprostým začátečníkům.

Autor je přesvědčen, že nejnázornější metoda pronikání do "strojáku" je ta, při niž můžeme vidět výsledek své činnosti, tj. něco se zobrazí na obrazovce, počítač zahraje melodii, atd.

Začneme tedy tím, co uvidíte při svých pokusech na obrazovce, a nejprve trochu kecání (bohužel nutného).

Při nahrávání obrázku z magnetofonu si dobře všimněte, jak se na obrazovku ukládá. Ti z vás, kteří mají "za sebou" příručku ABC DIDAKTIK vědí, proč tomu tak je, zde si jen zdůrazníme, že všechno, co je vykresleno na obrazovce, je zároveň uloženo v tzv OBRAZOVÉ PAMĚTI, která začíná na adrese 16384 a je dlouhá 6912 bajtů, t.j. paměťových míst, adres.

Vrátíme se tedy k nahrávání obrázku. Nejprve se zobrazí první linka zhora, přesněji nultá linka v nultém řádku (číslování začíná vždy nulou!), pak nultá linka v dalším řádku, atd., dokud není zaplněno všech osm prvních řádků zhora. Pak se stejným způsobem začne zaplňovat dalších osm řádků, dokud obrazovka není vyplněna celá. Nyní zvuk z magnetofonu změní charakter a začne se zaplňovat paměť atributů, čili posledních 768 adresových buněk, čímž se nám obrázek vybarví.

Teď se na to podíváme podrobněji. Víme již, že obrazová paměť začíná buňkou s adresou 16384. Každá jednotlivá buňka

obsahuje jeden bajt, to je osm bitů. Přitom bit 1 nám na obrazovce vytiskne bod, bit 0 nechá své místo neobsazené. Dvojkovou soustavou se budeme zabývat později, ale zatím jen pro představu vložíme do prvé buňky obrazové paměti binární číslo, tedy ve dvojkové soustavě, jehož jedničky budou tvořit body s barvou INK a nuly body s barvou PAPER. Číslo 11001100 tedy vytvoří tohle:

což by se nám na obrazovce ukázalo jako dvě krátké čárky v levém horním rohu obrazovky. Schválně zkuste vložit:

POKE 16384, BIN 11001100

Co se na obrazovce objevilo, to vidíte, ale co jsme to vlastně provedli: Příkaz POKE slouží k uložení jednoho bajtu na určenou adresu, v našem případě tedy na adresu 16384. No a protože touto adresou začíná obrazová paměť, zobrazí se vlevo nahoře, tedy na začátku obrazu, ony dvě čárečky.

Zkuste třeba:

POKE 16384,255

(číslo 255 ve dvojkové soustavě je 11111111). Měla by se objevit jedna dlouhá čárka, přes celý první bajt. Je tam? No sláva, takže něco nám již jde. Vytvořili jsme si právě nejjednodušší program ve strojovém kódu.....

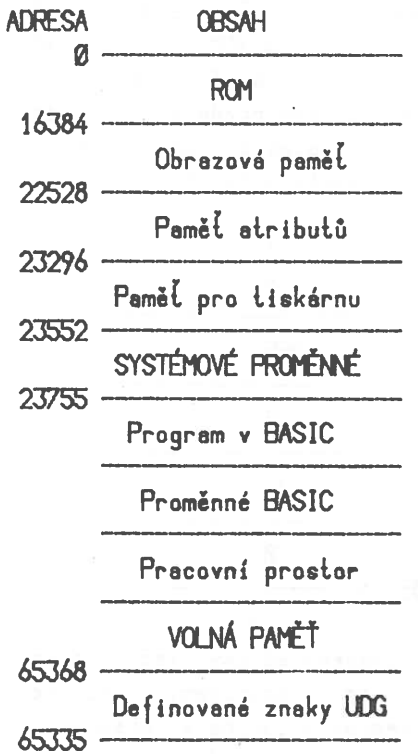
Tímto způsobem tedy můžeme "kreslit" na obrazovku, zkuste si i některé další adresy v obrazové paměti a jiné kombinace nul a jedniček. Upozorňuji, že na tomto principu vznikají i ty hezké obrázky, které vidíme ve hrách, jen je to trošičku komplikovanější, ale klid. Všechno bude.

PAMĚŤ.

Někteří z to již snad znají ze studia BASICu, ale ani těm neuškodí, aby si své znalosti trošku zvopákli. Takže:

Paměť každého počítače se skládá z jednotlivých míst, budeme jim říkat třeba buňky, které jsou očíslovány od nuly až do jejich maximálního počtu. V Didaktiku M (a Spectru) jich je 65536, mají tedy čísla Ø až 65535, těmto číslům (aby se nám to nepletlo) budeme říkat ADRESY, už jsem se zmínil, že třeba obrazová paměť začíná na adrese 16384, tedy tohlenc číslo má prvé buňka paměti, určené pro uložení obrazové informace.

Snad to bude názornější, když si rozdělení paměti namalujeme:

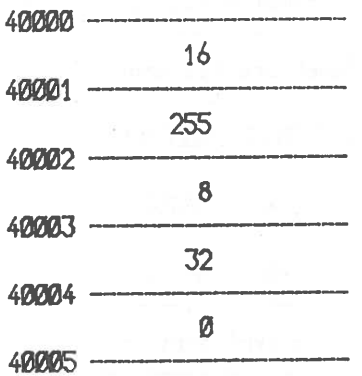


Adresy hranic jednotlivých oblastí mezi BASIC a 65535 jsou pohyblivé, dají se zjistit ze systémových proměnných (později se k tomu dostaneme, viz také ABC DIDAKTIK)

Také hranice na adrese 65368 je pohyblivá, zde je uvedena tak, jak se sama nastaví po zapnutí počítače (nebo po RESET). Tuto hranici můžeme nastavovat příkazem CLEAR s příslušným argumentem tak, jak potřebujeme. Používá se hlavně k určení začátku strojových programů, přes tuto hranici se nám totiž nemůže dostat ani jeden bajt z programu BASIC, který by "stroják" mohl poškodit.

Příklad: CLEAR 39999 nám tuto hranici postaví na adrese 39999, takže od 40000 si klidně můžeme ukládat své strojové programy. Této hranici se říká RAMTOP a její adresa je uložena ve stejnojmenné systémové proměnné (víte kde ji hledat?).

A jedeme dál. Nakreslené rozdělení bylo dost hrubé (jinak by se to do téhle knížky nevešlo), pro bližší orientaci si zde nakreslíme jen kousek paměti s jejím obsahem.



Zde jsou přímo uvedeny adresy jednotlivých buněk paměti, které jsou naplněny různými hodnotami. Takto je názorně vidět, jak jsou určeny (adresou) a zaplněny (obsah), tohle si zkuste zapamatovat, s těmito pojmy se budeme mňňňňc často setkávat.

Adresy s čísly 22528 až 22295 jsou určeny pro uložení atributů tisku, čili o tom, jakou barvu má mít inkoust, jakou papír, zda má určité políčko blikat, atd. Všimněte si, že jsem napsal POLÍČKO, protože zde jsou použity tzv. atributové čtverce, čili jedna adresa platí pro celé políčko 8x8 bodů, podobně, jako u příkazů PRINT.

Tak a teď pokus - co se stane, když na tyto adresy vložíte nějaké číslo. Zkuste třeba POKE 22528,20. Aha! Takhle to je.....

Takže jsme si zopakovali, jak vlastně pracuje obrazová paměť v počítači DIDAKTIK a směle se můžeme pustit do další práce.

První strojový program.

Abychom jen stále zbytečně teoretizovali, zadáme si nějaký opravdový úkol, který se může (a dost často) vyskytnout. Dejme tomu, že potřebujeme si uschovat obrázek z obrazovky do jiné části paměti, abychom ji mohli zatím využít k jiným účelům, třeba vypsat na obrazovku MENU, a podle potřeby si jej opět na obrazovku vrátit. Teď teprve začne to zajímavé. BASIC ovládáme docela suverénně, zkusíme to tedy v něm:

```
 1 REM PRENOS OBRAZU DO PAMETI
 2 REM OD ADRESY 50000
10 CLEAR 49999
20 FOR n=0 TO 6911
30 POKE 50000+n, PEEK (16384+n)
40 NEXT n
```

Zkusíme si tento prográmek rozebrat. Na řádku 10 máme příkaz CLEAR s argumentem, to už víme, že takto se nastavuje hranice RAMTOP. Tím docílíme toho, že přes tuhle hranici nám již nemůže žádný bajt z BASIC emigrovat, aby nepoškodil dále uložený obrázek.

Na řádku 20 začíná stěhovací smyčka, obrázek má 6912 bajtů, čili od 0 do 6911 proběhne smyčka právě 6912 krát.

Na řádku 30 nám příkaz POKE uloží příslušný bajt z obrazové paměti (16384+n) na nově určenou adresu, tedy 50000+n (n je 0 až 6911, samozřejmě to víme....). Stěhovací smyčka končí příkazem NEXT n na řádku 40.

A je to. Teď si můžeme na obrazovku cokoli namalovat, nebo napsat, případně i nahrát obrázek z kazety a po spuštění programu příkazem GO TO 10 (a samozřejmě ENTER, nebudu vám to stále připomínat, to už máte vědět) se jednotlivé bajty obrázku začnou kopírovat do určeného úseku paměti, v tomto případě tedy od 50000 do 50911. Na adresu 50000 se přenese bajt z 16384, na 50001 z 16385, atd., až celá smyčka proběhne 6912 krát a počítač ohlásí OK.

To to trvalo, vidíte? Jo, vono to ještě neskončilo? Tak čekejte dál.....

Už? Tak tedy se můžete pustit do další stránky.

Hotovo? Tak teď stiskněte ENTER, obrázek zmizí a na obrazovce máte výpis programu, který nás tak dlouho zdržoval. Naším úkolem nyní bude obrázek z paměti opět přemístit na obrazovku, tedy přestěhovat jej v paměti z adres 50000 atd. na adresy 16384 atd. Napíšeme si tedy další část programu:

```
 90 STOP
100 REM PRENOS OBRAZU NA OBRAZOVKU
110 FOR n=0 TO 6911
120 POKE 16384+n, PEEK (50000+n)
130 NEXT n
```

Zase jej spustíme, příkazem GO TO 100 a v klidu sledujeme, jak je obrázek postupně přenášen po jednotlivých bajtech a jak se nakonec "vybarví", stejně, jako při nahrávání z magnetofonu. Ale ta doba ..... to to trvá!

Pokud se vám tyto prográmky zdály moc pomalé, tak to se vám vůbec nezdálo, je to fakt. No to víte, BASIC, co vod něj můžeme chtít... Takže jste právě tak zralí k tomu, aby jste si totéž vyzkoušeli ve formě strojových programů.

Doufám, že jste počítač mezitím nevypnuli,takže na adresách od 50000 máte stále ještě uložen váš obrázek. No a začneme tvořit náš první stroják.

Nejprve si vybereme adresu, na kterou (lépe řečeno od které adresy) jej budeme ukládat. Aby nám nepřepsal obrázek, zvolíme si třeba adresu 57000 (místa v paměti máme dost, tak jaképak fraky).

Vložte si do paměti počítače tento program v BASIC, který si podrobněji rozebereme později:

```
200 REM PROGRAM PRO VLOZENI STROJ. KODU
205 REM K PRENOSU 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
260 DATA 1,0,27,237,176,201
```

Odstartujeme jej pomocí GO TO 200 a on nám do paměti od adresy 57000 postupně uloží 12 osmibitových čísel strojového kódu, který bude sloužit ke stejnému účelu, jako náš předešlý program v BASIC, který máme zapsán od řádku 100. Ovšem s jakým rozdílem, to uvidíte za chvilku sami.

Nejdříve si ale tento prográmek pěkně rozebereme (jak již je zvykem), abychom věděli, o co zde kráčí.

Takže na řádku 210 je opět začátek smyčky, která má za úkol dvanáctkrát uložit do dvanácti paměťových buněk s adresami 57000 57001, 57002 atd. dvanáct osmibitových čísel, které máme uloženy v řádcích 250 a 260 v souboru DATA. Tyto čísla nám pak v určené oblasti vytvoří strojový program, který budeme používat.

Nojo, ten chytrák nám tady napsal řadu čísílek, ale jak jen k ním přišel? Hmmmm....

Tak helejte: tyhle čísla jsou strojovým překladem programu napsaného v jazyku zvaném assembler Z80, který je stejný pro všechny počítače s mikroprocesorem Z80 (tj. od Spectra, přes SHARP až po třeba SORD M5). Že v Didaktiku takový mikroprocesor nejni? Ale jo, jenže zde se jmenuje U880D, jinak je to tentýž. No a ten assembler vypadá asi takto:

```
LD HL, 50000    ;nastavení počáteční adresy, v překladu 33,80,195
LD DE, 16384    ;nast. cílové adresy, v překladu 17,0,64
LD BC, 6912     ;nastavení počtu bajtů, v překladu 1,0,27
LDIR            ;provedení přenosu, v překladu 237,176
```

Pěkný zmatek, že jo? Zkusíme se v něm trochu zorientovat. Vpravo za středníkem jsou uvedeny komentáře k jednotlivým řádkům, středník zde zastupuje slůvko REM, které známe z BASIC. Vlastní zápis programu je tedy v levé části, před středníkem.

A pěkně od Adama: Program zapsaný v assembleru se skládá ze sledu příkazů a operandů, to je čísel, se kterými bude pracovat (ale dyk to už znáte z BASIC, že jo?). Každý příkaz má svůj kód, který je uveden v seznamu. Zkusili jste si prolistovat manuál dodávaný k počítači až do konce? Např. "modrý" pro Didaktik Gama od str. 72 dále? Nebo pro Didaktik M příloha E? Zde si můžete zjistit, že třeba příkaz "LD HL, číslo" má kód 33. Aha, tak proto je prvé číslo v seznamu DAT právě 33.....

Tento příkaz (tedy LD HL, 50000) je možno přeložit do "lidštiny" asi takto: vezmi číslo 50000 a schovej jej do registru (šuplíku), který se jmenuje HL (ve skutečnosti to jsou dva šuplíky, H a L, ale o tom později). Nebo třeba LDIR (ten má dva kódy, které musí být použity současně: 237 a 176) počítači říká: na adresy, počínaje tou, která je napsána v šuplíku HL, ukládej bajty z buněk počínaje adresou, která je v šuplíku DE, celkem tolikrát, jaké číslo je v šuplíku (tedy registru) BC, v našem případě tedy 6912 krát.

Dále je příkaz RET (kód 201), který programu řekne: když jsi skončil, vrať se zpět tam, kde tě vyvolali. V našem případě tedy do BASIC (je to něco jako RETURN).

Ale je tady ještě jeden problém. Ten blbec, co tuhle knížku napsal, kecá. Tvrdí, že LD HL, 50000 má kódy 33,80,195. Těch 33 jó, to je v pořádku, ale kde je těch 50000?

Takže vězte moji drazí, že nekecám, hned vám to vysvětlím. Krátce jsem se zmínil o tom, že "šuplík" neboli registr HL, jsou vlastně dva registry, H a L. Do každého registru se vejde jen jedno osmibitové číslo, tedy jeden bajt, čili 0 až 255. Číslo 50000 je šestnáctibitové, musíme si jej tedy rozložit na dvě osmibitová čísla - a bude se zase čarovat.

50000 - 256 * INT (50000 / 256) = 80
INT (50000 / 256) = 195

A už se to sype, už to jede, máme tady dvě osmibitová čísla která dohromady dávají hodnotu 50000: tedy 80 a 195.

****
ZAPAMATUJTE SI: rozklad šestnáctibitového čísla N (to je většího než 255) na dvě osmibitová provedeme pomocí zaklínadla:

N - 256 * INT (N / 256) = vyšší bajt
INT (N / 256) = nižší bajt.

Proč jim říkáme vyšší a nižší bajt se ještě dozvíte, zatím to berte jako fakt.
****

Zrovna tak je to na druhém řádku programu - v tabulce si najdeme, že LD DE, číslo má kód 17. Číslo 16384 si rozložíme na dvě osmibitová, tedy 0 a 64, atd.

Že je to dost komplikované? Jojo, takový je život. Ale aby nám to počítač usnadnil (vždyť to je jeho úkol, usnadňovat nám život), existují programky, které nám zápis programu v assembleru samy přeloží do strojového kódu. Těchto programů je celá řada, ale aby se vám to v hlavě nepletlo, doporučuji se naučit ovládat jeden a toho se stále držet. Všechny totiž "umí" totéž, ale každý se obsluhuje jinak. Z vlastní zkušenosti (letité) mohu doporučit staříčký GENS 3, který nám nabídne tolik možností, že je snad ani všechny nevyužijete (viz příručku GENS - MONS, vydal HELLSOFT, cena 30.- Kčs). Ale to jsme se již dostali moc daleko, vrátíme se zpět na pevnou půdu našeho *programu*.

Takže - v paměti počítače máme uložen strojový program, teď ještě jak jej spustit. Strojový kód můžeme volat mnoha způsoby,

např.: PRINT USR 50000

nebo: LET l=USR 50000

nebo: RANDOMIZE USR 50000

Nejpoužívanější je ten poslední, ale někdy budeme muset použít i některý jiný, pokud budeme potřebovat, aby program provedl něco jiného. Na konci strojového programu máme příkaz RET (ten tam vždy musí být!), který nám po jeho provedení vrátí řízení zpět do BASIC.

V prvém případě provede PRINT, tedy vypíše na obrazovku obsah registrů B a C (tím nám pokazí obrázek - brrrr, tedy zde jej nepoužijeme), ve druhém případě toto číslo uloží do proměnné L (v tomto případě nám to nevadí, jindy jo, někdy to i budeme potřebovat) a v posledním případě vloží toto číslo (obsah BC) do systémové proměnné SEED, kde slouží jako zdrojové číslo při funkci RND. Zde nám to nevadí (jindy třeba jo), takže zde to klidně můžeme použít.

Naťukejte tedy: RANDOMIZE USR 50000 (a ENTER) a koukejte, jak rychle se nám obrázek z paměti dostal na obrazovku. To je ale fofr, co? Kam se hrabe BASIC.....

Takže jsme trochu chytřejší, proč si tedy nezrychlit i přenos obrázku z obrazovky do paměti? K tomu nám poslouží tento prográmek:

ASSEMBLER	KOMENTÁŘ	PŘEKLAD
LD HL, 16384	;nastav. počát. adresu	33,0,64
LD DE, 50000	;nastav cílovou adresu	17,80,195
LD BC, 6912	;počet bajtů k přenosu	1,0,27
LDIR	;proveď přenos	237,176
RET	;a zpět do BASIC	201

Že vám to něco připomíná? No jasně, je to přesně totéž, co minulý prográmek, až na to, že jsou navzájem přehozeny adresy do registrů HL a DE. V tomto případě tedy prográmek provádí přesun opačným směrem, a to je to, co vlastně chceme....

Jeho vložení do počítače už snad zvládnete také, čísla z pravého sloupce (překlad) zapíšete do souboru DATA v našem minulém zaváděcím programu, a je to. Ale hopla! Pozor na to, že jej musíme uložit na jinou adresu! Na adrese 57000 máme přece obsazeno programem pro převod na obrazovku! Zkuste jej uložit třeba od adresy 57050, tam je již volno. Ale bacha na to, že jej nyní budeme spouštět od určené adresy! Tedy:

Přenos obrazu do paměti - RANDOMIZE USR 57050
Přenos obrazu na obrazovku - RANDOMIZE USR 57000

ZASE TROCHU POKECU.

Tak jste se namlsali prvým chodivým programem ve strojáku, je tedy na čase opět trochu teorie která nikoho nezabije, než se dostaneme k dalším programům.

Jak jste si jistě všimli v prvé kapitole tohoto povídání, tak mikroprocesor ve vašem počítači nepracuje s proměnnými podle jejich jména, jak jste byli zvyklí z BASIC, ale s jednotlivými buňkami paměti, které jsou určeny jejich vlastními adresami. Operace se provádí s jednotlivými bajty, nebo někdy i dvoubajtovými celými čísly. (Šimsisi? No eslisisišim....)

Mikroprocesor dále obsahuje několik jedno- a dvoubajtových buněk, kterým říkáme REGISTRY a které se účastní ve většině operací. Jednotlivé příkazy strojového programu se provádí v těchto registrech, takže zde nejsou potřebné takové adresy, jaké používáme při manipulaci s buňkami v paměti. Úplně nám postačí znát (pro začátek) osm jednobajtových registrů, označených písmeny A, F, B, C, D, E, H a L.

Nejvíce možností má registr A, zvaný také Akumulátor. Registry B a C, D a E, stejně jako H a L je možno spojit do dvojic přechovávajících dvoubajtová čísla vcelku.

Mimo uvedených jsou v mikroprocesoru ještě další registry (IX, IY, SP, PC, I, R, A', F', B', C', D', E', H', L'), ale s těmi se zatím nebudeme zatěžovat, až je budeme potřebovat, tak si o nich něco povíme. Zatím budeme dělat, jako že nejsou...

Ale teď se dostaneme k tomu horšímu - stručný výklad činnosti jednotlivých příkazů mikroprocesoru, zatím alespoň těch nejpoužívanějších. Jinak jejich seznam si můžete nalistovat v manuálu k počítači, kompletní seznam (trochu delší) bude ve druhém díle této příručky, máte se na co těšit, je jich opravdu trochu moc...

V dalším výkladu bude písmenem R označován registr, tedy libovolný, místo R si můžete dosadit jméno kteréhokoli (ovšem existujícího) registru z prve uvedených. Písmeny nn budeme označovat jednobajtové číslo (Ø-255, ØØ - FF), nn nn dvoubajtové číslo (Ø-65535, ØØ - FFFF). Připravte si mozkové závity (k dalšímu popisu se ale budete vracet dost často, to vám zaručuji) a jedeme s kopce.

STRUČNÝ PŘEHLED INSTRUKCÍ Z8Ø

INSTRUKCE PRO PŘESUNY.

LD A, R	Přenos obsahu registru R do registru A
LD (HL), A	Přenos obsahu registru A do buňky paměti, jejíž adresa je určena obsahem registrového páru HL
LD A, (HL)	Přenos obsahu paměťové buňky, jejíž adresa je v reg. páru HL, do reg. A.
LD A, nn	Přenos jednobajtového čísla do reg. A
LD HL, nn nn	Přenos dvoubajtového čísla do reg. HL
LD HL, (addr)	Přenos obsahu dvou za sebou následujících buněk od adresy addr do reg. HL
LD (addr), HL	Přenos obsahu reg. páru HL do dvou za sebou následujících buněk paměti, počínaje adresou addr.
LD (HL), nn	Přenos jednobajtového čísla nn na adresu, určenou obsahem reg. páru HL. Obsah HL se při tom nemění.

Všimněte si, že pokud je údaj v závorce, znamená to adresu - pokud je v závorce registrový pár, znamená to adresu, která je uložena v tomto registrovém páru - čili tam uložené dvoubajtové číslo je považováno za číslo adresy.

Následující instrukce pro přenosy zatím nebudeme používat, ale pro úplnost si je uvedeme.

LD A, (IX+Ø2)	Obsah adresy o 2 vyšším než určuje obsah registru IX, zapsat do registru A.
LA (IY-Ø2), A	Zapíše obsah registru A na adresu o 2 nižší, než udává obsah registru IY.
LD SP, nn nn	nastaví registr SP na adresu nn nn

LD SP, HL	Do registru SP (zásobníku) se zapíše adresa, uložená v HL
LD SP, (addr)	Do registru SP se zapíše obsah z adresy addr.

To jsou věci.... no prostě něco strašnýho, tomu v životě nebudu rozumět.... Ale klid, vezmeme to pomalounku a vovo to pude.

Zatím si všimněte jedné věci: výraz LD znamená LOAD, čili "nacpat" něco někam. Kam a odkud to vzít, nám říkají další znaky Prvý označuje registr KAM to patří, druhý ODKUD to vzít. Místo názvu registru to může být i jedno- nebo dvoubajtové číslo, adresa (také dvoubajtová) a podobně.

Stále tady pohazuji výrazy jako "adresa", "buňky" atd., pokud nevíte o co jde, přečtěte si znovu kapitolu "PAMĚŤ" a tak si osvěžte svou paměť (čili počítačsky: proveďte refresh). Pokud je vám to jasné, můžete si cpát do hlavy další instrukce.

INC R	Zvýšení obsahu registru R o 1 (increment), t.j. přičtení jedničky.
DEC R	Snížení obsahu registru R o 1 (decrement - pozor! ne exkrement, to je něco jinéhol), t.j. odečíst jedničku

Aby jste to neměli tak jednoduché, budeme používat i příkazy k blokovému přenosu dat, které přenesou z jedné části paměti celý blok bajtů najednou. Nesetkali jsme se již s tím někde?

Předem si ale musíme nastavit obsahy registrových párů:

HL - adresa prvého bajtu, určeného k přenosu
DE - adresa prvé buňky, kam bajty přenést
BC - počet přenášených bajtů

Takže s chutí do toho:

LDI LOAD-INCREMENT, bajt z adresy určené obsahem HL je uložen na adresu určenou obsahem DE. Obsahy registrů HL a DE se zvýší o 1 (increment), obsah registru BC se sníží o 1 (na znamení, že 1 bajt byl přenesen).

LDIR LOAD-INCREMENT-REPEAT, bajt z paměťové adresy určené obsahem HL je uložen na adresu určenou obsahem DE. Obsahy registrů HL a DE se zvýší o 1, obsah registru BC se sníží o 1. Následuje kontrola obsahu registru BC, pokud není nula, celý postup se opakuje. Po vynulování BC se pokračuje následující instrukcí.

LDD LOAD-DECREMENT, stejné jako LDI, ale obsahy HL a DE se snižují o 1 (decrement), pracuje tedy "odzadu", od vyšších adres k nižším.

LDDR LOAD-DECREMENT-REPEAT, jako LDIR, obsahy HL a DE se vždy sníží o 1, jako u LDD.

A následuje další lahůdka: skokové instrukce.

Program nemusí vždy běžet od jedné adresy k druhé, ale může skočit do jiného bloku a pokračovat tam. Ale to už přeci znáte z BASIC, je to známý GO TO..... Zatím nepodmíněné skoky:

JP addr Skok na uvedenou adresu

JP (HL) Skok na adresu, uvedenou v registru HL.

A nyní podmíněné skoky (obdoba IF.....GO TO), zatím jen pro úplnost, podrobněji se k nim dostaneme později.

JP Z, addr Podmíněný skok na adresu addr, je-li Z=1 (v registru F, o něm bude řeč dále).

JP NZ, addr Skok na adresu addr, není-li výsledek operace nulový (JUMP NOT ZERO)

JP PE, addr Podmíněný skok na adresu addr. Dokud obsah registru B není nulový, indikátor P/V (o něm

i o dalších si povíme za chvíli) je ve stavu logické 1, podmínka PE je splněna a skok se provede. Při BC s nulovým obsahem je platná následující podmínka PO.

JP PO, addr totéž, testuje podmínku PO, t.j. je-li obsah BC nulový.

A následuje specialita ve skokových instrukcích - relativní skoky, to si užijete legrace.....
Náhodou, jsou to nejužitečnější skokové instrukce, ale na to jistě časem přijdete sami.

JR nn JUMP RELATIVE, číslo nn v rozsahu od -128 do +127 udává směr (+ nebo -) a délku skoku - čili kolik bajtů se má přeskočit.

Také tyto skoky mohou být i podmíněné (JR Z, JR NZ, atd.), podobně, jako u JP.
Dále existují zahadné:

RST nn kde číslo nn může být Ø, 8, 16, 24, 32, 4Ø, 48. Převede běh programu na adresu v ROM určenou číslem nn. Používá se pro vyvolání základních podprogramů operačního systému počítače, např. RST 16 vytiskne obsah registru A, RST Ø je totéž co v BASIC RANDOMIZE USR Ø, atd.

DJNZ nn Podmíněný relativní skok s opakováním. Počet opakovaných skoků je uložen v registru B.

CALL addr Skok do podprogramu na adresu addr, obdoba GO SUB Podprogram musí být zakončen příkazem RET (jako v BASIC RETURN).

CALL NZ, CALL Z, CALL NC, CALL C, CALL PO, CALL PE, CALL M, podmíněné skoky do podprogramu, podobně, jako u příkazů JP.

RET Příkaz návratu z podprogramu. Také tento příkaz může být vázán na splnění určité podmínky, jako u JP a CALL, třeba RET Z, atd.

STAVOVÉ INDIKÁTORY.

Tak, a teď vám zamotám hlavu dokonale. Ale žádný strach, pokud tomu nyní neporozumíte (není na tom nic, zač by jste se museli stydět), můžete se sem kdykoli vrátit pro rozumy.

Pozornější z vás si všimli, že v popisu registrů se také vyskytuje registr F, zatím ale o něm nebyla žádná bližší zmínka. Takže to napravíme a pustíme se do něj.

Označení F je odvozeno z anglického FLAG, tedy vlajka, v tomto případě se používá pro "vztyčení" nebo "sklopení" vlajky v závislosti na výsledcích operací. Ale držme se techniky, v praxi to je tak, že jednotlivé bity tohoto registru mohou obsahovat buď 1 nebo Ø a tím nám dávají najevo, co se dělo.

Když si tento registr rozkreslíme, vypadá asi takto:

SIGN FLAG	ZERO FLAG	nepouž.	HALF CARRY	nepouž.	P/V FLAG	ADD/SUB FLAG	CARRY FLAG
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit Ø

Takže abychom si rozuměli, jednotlivým bitům místo FLAG budeme raději říkat indikátor, aby to bylo trošičku srozumitelnější. A začneme od kraje:

Stavový indikátor CY (CARRY FLAG), indikátor přenosu. Je ovlivňován především instrukcemi provádějícími součet, odečet, bitovou rotaci a posuv. Při součtu obsahuje 1 vždy tehdy, když dojde k přeplnění registru A (přeteče), t.j. snažíme se do něj vopchat číslo větší, než se tam vejde, tedy nad 255. Při odečítání obsahuje 1 tehdy, je-li výsledek menší

než nula. Nedojde-li při operaci k přenosu, obsahuje CY nulu (Ø).
Přímo se dá ovlivnit příkazy:
SCF - Set Carry Flag, uloží do CY jedničku
CCF - Complement Carry Flag, vloží do CY opačný bit, než jaký obsahoval před provedením příkazu.

Indikátor nuly Z (ZERO FLAG)
Z=1 když výsledkem testované operace je nula (mimo dále uvedené výjimky)
Z=Ø když výsledek je jiný než nula.
Není ovlivněn při odečítání párových registrů!
Ovlivňuje jej DEC R, INC R, BIT, CP atd.
Zda indikátor Z obsahuje 1 nebo Ø, zjišťujeme zařazením podmínky Z nebo NZ v podmínkách volání, skoků a návratů.

Indikátor znaménka S (SIGN FLAG)
Jestliže je testované číslo kladné, obsahuje Ø. Pro záporná čísla obsahuje 1.

Parita nebo aritmetické přeplnění PV (PARITY/OVERFLOV FLAG)
V případě sudé parity obsahuje 1, v případě liché obsahuje Ø. Vztahuje se především k logickým operacím. Aritmetické přeplnění se indikuje tehdy, je-li výsledkem součtu dvou záporných čísel číslo kladné, nebo je-li součet dvou kladných čísel záporný, t.j. přesahuje-li výsledek interval -128 až + 127.

HALF CARRY a SUBSTRACT FLAGS - H a N
Indikátory přenosu a odečítání, pro programátora jsou nepřístupné.

OSMIBITOVÉ ARITMETICKÉ INSTRUKCE

Ovlivňují všechny indikátory, až na výjimku: INC a DEC neovlivňují stav indikátoru CY.

ADD R	Přičte do registru A obsah registru R, výsledek je v registru A.
SUB R	Odečte od obsahu registru A obsah registru R, výsledek uloží do registru A.
ADC R	Totéž co ADD, k výsledku součtu se přičte obsah indikátoru CY. Při CY rovném nule bude výsledek stejný, při CY rovném jedné se k výsledku součtu přičte ještě číslo 1. Používá se při sečítání vícebajtových čísel.
SBC R	Totéž co SUB, od výsledku se odečte obsah CY. Používá se při odečítání vícebajtových čísel.
DAA	Specielní aritmetické instrukce pro práci s BCD čísly (Binary Code Decimal). Převádí binární číslo obsažené v reg. A na formát BCD.
BIT	Test stavu bitu, je-li ve stavu 1 nebo Ø. Např. BIT Ø,A testuje stav nultého bitu registru A. Podle toho, je-li ve stavu 1 nebo Ø, bude ovlivněn indikátor Z. Je-li bit nulový, je Z nastaven na 1 (ZERO, čili NULA je přítomno).
SET	Nastavení bitu do stavu 1. Např. SET 1,B nastaví (nebo ponechá) bit č. 1 registru B do stavu 1.
RES	Vynulování bitu, např. RES 3,(HL) vynuluje třetí bit buňky, jejíž adresa je v reg. HL.

LOGICKÉ OPERACE.

Dále následují příkazy (instrukce), které ZATÍM nebudete potřebovat, ale jsou velice užitečné, jsou to t.zv. LOGICKÉ OPERACE. Jejich pomocí se často dá dlouhatánský strojový program podstatně zkrátit a zjednodušit, některé funkce bez nich ani nejdou naprogramovat. Experti tvrdí, že kvalita programátora se pozná podle počtu použitých logických operací, tak bacha na to. Ostatně - až se jimi budeme zabývat blíže, vysvětlíme si jejich funkce podrobněji, zatím jen jako přehled.

XOR R	Provede logickou operaci XOR mezi reg. A a R.
XOR A	Provede logickou operaci XOR v registru A, výsledek je ten, že reg. A se vynuluje, a tedy i indikátor Z se nastaví na 1. Používá se pro zkrácení a zrychlení programu místo LD A,Ø.
OR R	Logická operace OR mezi registry A a R. Indikátor Z se nastaví na 1 tehdy, je-li výsledek operace Ø, čili obsahy obou registrů jsou nulové.
OR A	Používá se jako test obsahu registru A, indikátor Z se nastaví na 1, jestliže registr A obsahuje Ø.
CPL	Komplementace registru A, je to v podstatě provedení logické operace NOT.
NEG	Totéž, rozdíl je v chování stavových indikátorů.
AND	Po provedení se indikátor přenosu CY vždy nastaví na nulu. Pracuje s registrem A. Indikátory S a Z jsou výsledkem logické operace ovlivněny, indikátor parity obsahuje 1 při sudé paritě a Ø při liché paritě.

INSTRUKCE PUSH a POP.

Je to vlastně přenos obsahu mezi registry a zásobníkem, jakési "úschova" obsahu registrů. Příkaz PUSH uloží obsah registru do zásobníku, POP jej přenese ze zásobníku do registru.

1. Do zásobníku ukládané 2 bajty jsou uloženy tak, že napřed je uložen vyšší a pak nižší bajt párového registru.
2. Po uložení dvou bajtů se obsah registru SP i spodní adresa zásobníku sníží o 2.
3. Po odběru se obsah registru SP i spodní adresa zásobníku zvýší o 2.
4. Ze zásobníku jsou bajty odebírány systémem LIFO (Last In, First Out), registry se plní postupně nejprve nižší bajt, pak vyšší.

PUSH AF	Obsah registrů A a F do zásobníku.
PUSH HL	Obsahy registrů H a L do zásobníku
POP AF	Obsah zásobníku do registrů A a F
POP HL	Obsah zásobníku do registrů H a L

Ještě kratičká poznámka: pokud bychom v programu použili tento sled instrukcí PUSH a POP, dojde vlastně k výměně obsahu obou registrových párů AF a HL. Někdy to je užitečné, jindy to může způsobit havarii, takže pozor na pořadí!

Příkazy PUSH a POP se používají k dočasnému uschování číselných parametrů, jejichž hodnoty budeme ještě v programu potřebovat.

Uffff, to byl maraton, a to ještě zdaleka není všechno, ještě nám zbývají takové lahůdky, jako rotace, posuvy, atd. Ale to si necháme zase na později, vím dobře, že už toho je na vás dost. A proto kvůli povzbuzení pozornosti najedeme na

DALŠÍ PROGRÁMEK.

Abychom opět mohli jeho činnost sledovat na obrazovce, bude jeho úkolem zaplnit paměť atributů bajtem s udanou hodnotou, např. 71 = BIN Ø1ØØØ111 (černý papír, bílý inkoust, zvýšený jas, bez blikání). Zvolím zde poněkud neobvyklý způsob zápisu, tentýž program je uveden v BASIC, v assembleru a v kódech, spolu s komentářem. Tak se koukněte:

1Ø LET HL=22528	LD HL, 22528	33,Ø,88	;adresa začátku paměti atributů
2Ø LET BC=768	LD BC, 768	1,Ø,3	;počet bajtů do BC
3Ø POKE HL,71	LD (HL), 71	54,71	;konstantu do buňky s adresou uvedenou v HL
4Ø LET HL=HL+1	INC HL	35	;zvyš obsah HL o 1
5Ø LET BC=BC-1	DEC BC	11	;obsah BC zmenšit o 1
	LD A, B	12Ø	;obsah reg. B vlož do reg. A
	OR C	177	;proveď logický součet reg. A a C
6Ø IF BC<>Ø THEN GO TO 3Ø			
	JR NZ, -8	32,248	;není-li výsledek nula, opakuj.
	RET	2Ø1	;je-li konec, vrať se do BASIC

K BASICovému programku: proměnné HL a BC samozřejmě nemají nic společného s registry mikroprocesoru, jejich označení bylo zvoleno pouze pro názornost příkladu. Zkusíme tento program v BASIC naťukat do počítače a spustit - jeho provedení trvá kolem 12 sekund.

Ještě si rozebereme podrobněji program v assembleru: příkaz LD HL, 22528 chápe registry H a L jako jeden šestnáctibitový registr. Údaj, který je do něj vepsán, je adresa prvé buňky paměti atributů (0+256*88=22528).

Obdobně příkaz LD BC, 768 vepíše do registrů B a C, braných jako celek, hodnotu 768. Příkaz LD (HL), 71 umožňuje zapsat do vybrané buňky paměti hodnotu, uvedenou v příkaze (v našem případě tedy 71). Adresa buňky paměti je uvedena obsahem registrového páru HL. Jelikož HL obsahuje v této chvíli číslo 22528, tak do buňky s touto adresou je vložena určená hodnota 71.

Jednobajtový příkaz INC HL zvětší o 1 (inkrementuje) číslo, uvedené v HL. Původně tam bylo 22528, nyní tedy bude 22529.

Obdobně příkaz DEC BC zmenší o 1 (dekrementuje) obsah BC. Původně tam bylo 768, nyní tedy 767.

Příkaz LD A, B je použit z toho důvodu, že další operace, které budeme provádět, fungují pouze s rgistrem A. Tento příkaz vloží do registru A hodnotu, obsaženou v registru B a příkaz OR C provede logický součet registrů C a A. Logický součet se provádí samostatně s každou dvojicí odpovídajících bitů, asi tak

```
   01001000          00000000          00000000
OR 00100100       OR 00000001       OR 00000000
   --------          --------          --------
   01101100          00000001          00000000
```

Vidíme názorně, že výsledek operace bude nulový pouze tehdy jestliže oba registry A a C budou mít nulový obsah. Jelikož jsme do registru A pokaždé vložili obsah registru B, příkaz OR C vlastně zjišťuje, kdy bude obsah registrového páru BC nulový.

Příkaz JR NZ, -8 označuje podmíněný skok. Podmínkou provedení skoku je nenulový výsledek předešlé operace (tedy v BASIC : IF BC<>0 THEN GO TO 30). Skok se provede nazpět o 9 buněk paměti programu, čili na příkaz LD (HL), 71. Od tohoto bodu se celý postup stále opakuje, dokud nebyla konstanta 71 vložena do všech 768 buněk paměti atributů.

Jinými slovy - příkaz k podmíněnému skoku bude platný tak dlouho, dokud obsah registrů B a C se postupným odečítáním jedničky (dekrement) nezmenší na nulu. Tímto způsobem jsme ve strojovém programu vytvořili smyčku, která proběhne celkem 768 krát. Po každém průběhu smyčkou se hodnota uložená v reg. páru HL zvětší o 1 (inkrementuje), takže hodnota 71 bude postupně vepsána do 768 následných buněk paměti, počínaje buňkou s adresou 22528.

Již předem jsme si tento programek odzkoušeli v BASIC, proč tedy (jakožto experti) si jej nezkusit i ve strojáku? Zvolíme tedy opět nejprve adresu, od které jej budeme do paměti ukládat. Vzhledem k tomu, že je poměrně krátký, zvolíme si např. úsek označovaný jako PRINTER BUFFER, neboli stručně česky - vyrovnávací paměť pro tiskárnu. Tento úsek začíná od adresy 23296, náš program bude dlouhý 15 bajtů, čili konec bude na adrese 23310.

Sestavíme si tedy jednoduchý programek v BASIC, který nám strojový kód zavede do paměti (vzpomeňte si - již jsme něco podobného používali!). Začneme třeba řádkem 100, abychom mohli v paměti nechat uložený zároveň i odpovídající program v BASIC, pro srovnávání jejich rychlosti.

```
100 FOR a=23296 TO 23310
110 READ x: POKE a,x
120 NEXT a
130 DATA 33,0,88,1,0,3,54,71,35
140 DATA 11,120,177,32,248,201
```

Spustíme zaváděcí program (víte jak? Napovím naposledy: GO TO 100), čímž se nám uloží do paměti kódy strojového programu. No a můžeme si jej vyzkoušet - spusťte jej příkazem RANDOMIZE USR 23296.

Sakra, to byl ale fofr..... Provedení celého programu trvalo kolem 10 milisekund. Pro srovnání: program v BASIC běžel kolem 12 sekund, jak jsme si již zjistili.

K čemu takový prográmek ale může sloužit? Je to dosti univerzální rutina, kterou stojí za to si někam poznamenat, můžeme ji použít i jindy ve svých vlastních programech. Např. k zaplnění libovolného úseku paměti RAM libovolnou hodnotou, zcela postačí jen změnit počáteční adresu, počet bajtů určených k zaplnění a hodnotu konstanty, která tyto bajty zaplní. Pro ilustraci: autor (tedy moje maličkost) používá tuto rutinu k vymazání textu ve svém textovém editoru, před započetím psaní nového textu, nebo před jeho nahráváním. Text se ukládá od adresy 32000, čili počáteční adresa je 32000, no a konečná - dle délky textu. To už si zjišťuje jiná rutina, i když by to šlo i bez ní - prostě do celého rozsahu paměti, určeného k ukládání textu, vložit kód 32 - což je, jak známo mezera, tedy prázdnota.

Když máme tento prográmek v paměti, zkusíme si s ním ještě zaexperimentovat, abychom si zjistili, co všechno umí. Třeba zaplnit celou obrazovou paměť bajtem 51 = BIN 00110011 (připomíná vám to něco? Esli ne, tak jste nečetli dost pozorně začátek! Rychle to napravte!).

Nejprve zapíšeme novou adresu do příkazu LD HL, obraz, jak známo, začíná adresou 16384. Když si ji rozložíme na dvě jednobajtové čísla, podle již uvedeného vzoru... Ale raději zopakuji. Nižší bajt = adresa - 256 * (INT (adresa / 256)), vyšší bajt = INT (adresa / 256). Pokud vám vyšly čísla 0 a 64, počítali jste správně.

Nyní si je vložíme do strojového programu:

POKE 23297,0: POKE 23298,64

A podobně postupujeme i s příkazem LD BC, délka nyní bude 6144, tedy po rozložení 0, 24.

POKE 23300,0: POKE 23301,24

Nakonec ještě vložíme nový bajt, kterým budeme paměť zaplňovat, tedy 51:

POKE 23303,51

Pokud máte ještě v paměti zaváděcí program, můžete pozměnit údaje v seznamu DATA a spustit jej znova, výsledek bude stejný, jako při změnách pomocí POKE. V tom případě:

130 DATA 33,0,64,1,0,24,54,51,35

řádek 140 zůstane beze změny. Zaváděcí program spustíme znova, a máme v paměti nový strojový program.

Upravený stroják spustíme opět pomocí RANDOMIZE USR 23296, a ejhle - zase je na obrazovce něco jiného, samozřejmě opět bleskurychle.

A BUDEME SI ČMÁRAT....

Protože již pomalu začínáme být ostřílení "strojákoví" borci, musíme se ještě naučit různé fígle, jak si usnadnit práci a program co nejvíce zkrátit. Zkušení programátoři používají co nejvíce již hotové rutiny, které jsou uloženy v paměti ROM, proč to nezkusit také? Příslušný úsek programu pak nemusíme pracně vymýšlet, ani jej zdlouhavě vkládat a ověřovat, vždyť takových je již spousta v počítači od výrobce! No a abychom vše názorně viděli na obrazovce, jak se snažíme již od začátku, zkusíme si kreslit.

Základní příkaz pro kreslení v BASIC je PLOT. Je to ten nejjednodušší příkaz, který již jistě dokážete bravurně a s přehledem používat. My ale chceme pracovat ve strojáku, tak se na něj podíváme z jiné strany. Podprogram obsluhy tohoto příkazu je v ROM na adrese 8933.

Vstupními parametry k příkazu PLOT jsou souřadnice bodu, který se má vytisknout (NOJO PORÁT, TO PŘECE FŠICHNI VÍME.....) Bez udání souřadnic (po zapnutí počítače) to jsou souřadnice Ø,Ø Souřadnice vodorovné osy vložíme do registru B, souřadnice svislé osy do registru C.

Například:

```
LD B, 128
LD C,  88
```

Ale my chceme být odborníci, proto si (i tento) úsek zkrátíme a ušetříme nějaký ten bajt paměti (zároveň se program zrychlí, ale to je neměřitelné). Jak? No přeci použijeme registry B a C jako jeden registrový pár, čili šestnáctibitový registr BC. Do něj vložíme: BC = B + 256 * C, tedy

```
LD BC, 22656
```

Ušetřili jsme sice jen jeden bajt (zatím), ale při psaní delšího programu to již bude móóóc bajtů.

Řekli jsme si, že podprogram obsluhy PLOT začíná na adrese 8933. Použijeme tedy příkaz CALL adresa, o kterém jsme si řekli, že je to obdoba BASICovského GO SUB. Po provedení programu se musíme vrátit do BASIC, nezapomeňte tedy na RET! Právě vymyšlený úsek programu tedy bude vypadat následovně:

```
LD BC, 22656
CALL 8933
RET
```

Po jeho proběhnutí by se na souřadnicích 128,88 měla objevit tečka. Kdo si to chce vyzkoušet již nyní, použije opět již známý zaváděcí prográmek, nebo (jelikož je to krátké) uloží strojový kód pomocí příkazů POKE.

Zvolíme si počáteční adresu, třeba 30000. Aby se nám do toho nepletl ten lump BASIC, určíme mu hranici, až kam může:

```
CLEAR 29999
```

A zkusíme vkládat program (pro názornost jeho prvou verzi):

```
LD B má kód 6, čili POKE 30000,6
do B vložíme 128, čili POKE 30001,128
LD C má kód 14, čili POKE 30002,14
do C vložíme 88, čili POKE 30003,88
CALL má kód 205, tedy POKE 30004,205
číslo 9833 si rozložíme na dvě jednobajtové: 229 a 34
                              (víte jak? To už tady bylo!)
Vložíme je do paměti:
prvé -    POKE 30005,229
druhé -   POKE 30006,34
RET má kód 201, čili POKE 30007,201
```

Prográmek je v paměti (i když poněkud neúhledně a komplikovaně), můžeme jej spustit:

```
RANDOMIZE USR 30000
```

Našli jste na obrazovce tečku? Jó? Tak můžeme pokračovat.....

Dalším grafickým příkazem je DRAW. Zhruba by se dal přeložit jako "táhni čáru", slouží (jak jistě víte) k vykreslení úsečky, přesněji tedy vektoru, (abych nepohoršil ty vzdělanější) s počátkem tam, kde se nachází poslední použitý bod (jeho souřadnice jsou uloženy v systémové proměnné COORDS, t.j. na adr.

23677 a 23678), vedoucí do bodu, který určují hodnoty použité při DRAW, tedy souřadnice vektoru. Uff, to byla dlouhatánská věta, až jsem se zapotil. Doufám, že jste ji přežili, abychom mohli pokračovat.

Podprogram DRAW, použitý v ROM vyžaduje jako vstupní parametry čtyři údaje, které čte z registrových párů BC a DE.

V registrovém páru BC jsou uloženy souřadnice vektoru v absolutní hodnotě (bez znaménka, tedy jako kladná čísla) a v DE údaj o vodorovné souřadnici v kódu U2 (nejvyšší bit označuje polaritu, pokud má hodnotu 1, jedná se o záporné číslo, Ø značí kladné číslo). V registrovém páru DE je tedy hodnota staré známé funkce z BASIC, tedy SGN (signum, znaménko) souřadnic X a Y, obsažených v registrovém páru BC. Je to trochu komplikované, ale co se dá dělat. Snad vám bude útěchou, když prozradím, že bude ještě hůř.....

Podprogram DRAW začíná od adresy 9402 a má jednu nepříjemnou vlastnost. Mění totiž obsah registrového páru H'L' (zatím jsem vám jeho existenci zatajil), který proto musí být před návratem do BASIC obnoven. K tomu použijeme příkazy EXX (zamění mezi sebou hlavní a vedlejší registry) a PUSH (uschová obsah registru).

Takže takový program pro DRAW bude vypadat třeba takto:

```
LD (23677), 255          ;souřadnice X
LD (23678), 175          ;souřadnice Y
EXX                      ;zaměnit sady registrů
PUSH HL                  ;alternativní pár H'L' uschovat
EXX                      ;zpět původní sada registrů
LD BC, 22656             ;souřadnice 128, 88
LD D, 255                ;zaplnit registr D
LD E, 1                  ;znaménko pro záporné číslo
CALL 9402                ;vyvolat podprogram DRAW
```

```
EXX                 ;zaměnit sady registrů
POP HL              ;obnovit obsah H'L'
EXX                 ;zpět původní sada
RET                 ;návrat do BASIC
```

Posledním z grafických příkazů je CIRCLE, slouží (jak jistě víte) ke kreslení kružnic. Začátek jeho podprogramu je na adrese 11555.

Způsob předávání vstupních parametrů je poněkud netypický, ale zato méně namáhavý. Spočívá v uložení třech hodnot do zásobníku tzv. kalkulátoru (nehledejte jej v šuplíku, je ve vašem počítači), které si pak podprogram CIRCLE odebírá jako souřadnice středu kružnice a jejího poloměru. Potřebné hodnoty se do zásobníku kalkulátoru uloží pomocí podprogramu STACK-A, který začíná na adrese 11560. Tento podprogram převezme hodnotu z registru A a uloží ji na vrchol zásobníku.

Před vyvoláním podprogramu CIRCLE opět musí být vedlejší registrový pár H'L' uschován, takže:

```
EXX                 ;zaměnit sady registrů
PUSH HL             ;reg. pár H'L' uschovat
EXX                 ;zpět původní sada registrů
LD A, 128           ;souřadnice X středu kružnice
CALL 11560          ;uložit do zásobníku kalkulátoru
LD A, 88            ;souřadnice Y středu kružnice
CALL 11560          ;uložit do zásobníku kalkulátoru
LD A, 80            ;poloměr kružnice
CALL 11560          ;uložit do zásobníku kalkulátoru
```

```
CALL 11555          ;podprogram CIRCLE
EXX                 ;zaměnit sady registrů
POP  HL             ;obnovit registry H'L'
EXX                 ;zpět původní sada registrů
RET                 ;návrat do BASIC
```

Pokud si tento program chcete vyzkoušet, tak vás nechám trochu potrápit. Najděte si v seznamu příkazů jednotlivé kódy, tyto kódy uložte do seznamu DATA, ve smyčce je načtěte a uložte od zvolené adresy..... atd. Stačí? Schválně, jestli se vám to povede.

A my ostatní pojedeme dál ve stínu divokých skal. Jak jistě víte, v BASIC se u příkazu DRAW může použít ještě třetí parametr čímž se dají docílit velmi zajímavé efekty. Ve strojovém kódu se toto dá realizovat pomocí zvláštního podprogramu na adrese 9108.

Všechny tři vstupní parametry musíme uložit do zásobníku kalkulátoru ještě před vyvoláním tohoto podprogramu. Podobně, jako u podprogramů DRAW a CIRCLE musíme i zde dbát na uschování vedlejšího registrového páru H'L'.

Tak už jsem vás dost potrápil teoretizováním, pokud jste u toho neusnuli, můžete si nyní popsané programy vyzkoušet. Když bych měl tento prográmek rozepsat do jednotlivých příkazů assembleru, zabralo by to hodně místa, proto uvádím pouze kódy jednotlivých příkazů uložené do souboru DATA zaváděcího programu.

Až se naučíte (doufám že brzy) používat program MONS 3, či DISSASEMBLER, můžete si jejich pomocí hotový prográmek přeložit do assemblerovštiny a srovnat s předcházejícími popsanými úseky. Je to prakticky shodné, liší se jen zadáním jiných souřadnic pro kreslení.

```
  5 REM  - X I C H T -
 10 BORDER 0: PAPER 0: INK 7: OVER 1: CLS
 20 CLEAR 29999
 30 FOR f=0 TO 131
 40 READ a: POKE 30000+f,a
 50 NEXT f
100 DATA 217,229,217,62,100,205
110 DATA 40,45,62,100,205,40,45
120 DATA 62,59,205,40,45,205,45,35
130 DATA 62,68,205,40,45,62,116
140 DATA 205,40,45,62,16,205
150 DATA 40,45,205,45,35,62,132
160 DATA 205,40,45,62,116,205,40,45
170 DATA 62,16,205,40,45,205,45,35
180 DATA 62,100,50,125,92,62,80
190 DATA 50,126,92,1,0,32,17,1,1,205
200 DATA 186,36,62,96,205,40,45
210 DATA 62,32,205,40,45,62,2
220 DATA 205,40,45,62,64,50,125
230 DATA 92,62,64,50,126,92,205
240 DATA 148,35,1,68,116,205,229
250 DATA 34,1,69,117,205,229
260 DATA 34,1,132,116,205,229
270 DATA 34,1,133,117,205,229
280 DATA 34,217,225,217,201
```

Tak - a je to. Povedlo se vám jej zavést do počítače? Tak již jsi jej můžte spustit známým příkazem RANDOMIZE USR 30000, schválně, co se objeví - místo obrazovky máte teď zrcadlo.....

HEXADECIMÁLNÍ ČÍSLA.

HEXE znamená německy čarodějnice, ale nebojte se, čarovat sice budeme, ale nic špatného z toho nevzejde, spíše nám to pomůže v dalším snažení o zvládnutí strojového jazyka.

Postupné vkládání bajtů programu do paměti ze souboru DATA je jednoduché, na to nám stačí obyčejná smyčka. Často se ale vyskytne potřeba zavést do paměti několik set i více bajtů. Pak se zápis dat v desítkové soustavě ukáže být velmi neekonomický. Při přepisování programu, např. z časopisu, může také velmi snadno dojít k chybě, jejíž hledání ve spoustě dat spotřebuje mnoho času a hlavně nervů. Jen si vzpomeňte na minulý program XICHT.... Je tedy na čase zajímat se o lepší a ekonomičtější metodu vkládání strojových programů do paměti počítače.

Místo zápisu v desítkové nebo dvojkové soustavě (ta je obvzvláště "nelidská") se používá šestnáctkové (hexadecimální) soustava, ve které je každý bajt zakódován do dvou hexadecimálních číslic. V soustavě o základu 16 potřebujeme 16 různých číslic, sadu desítkových 0 - 9 tedy doplníme šesti prvými písmeny abecedy: A až F. Hodnoty těchto znaků jsou následující:

HEX číslice	Dekadická hodnota	Dvojková hodnota
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101

6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Přechod z dvojkové (binární) do šestnáctkové, hexadecimální soustavy je jednoduchý, postačí čtyřbitové skupině přiřadit odpovídající hexadecimální číslice. K zápisu jednoho bajtu v hexadecimální soustavě potřebujeme 2 znaky a právě tolik bajtů paměti programu v jazyku BASIC, pro totéž číslo v desítkové (dekadické) soustavě potřebujeme 8 bajtů - při větším počtu čísel je úspora zřejmá.

A teď - jak se zabezpečíme proti chybám při přepisování programů? Všechny další strojové programy, které si zde budeme uvádět, budou zapsány následujícím způsobem:

Skupina bajtů je uvedena jako řetězec hexadecimálních čísel po dvou číslicích na bajt. Všechny bajty, tvořící jednu skupinu jsou sečteny a zbytek z dělení tohoto součtu číslem 256 nám představuje tzv. kontrolní součet, doplněný do řetězce jako poslední dva bajty.

Schválně si teď zkuste představit, že při opisování třeba změníte hodnotu některé číslice, nebo zaměníte mezi sebou dva sousední znaky. Zaváděcí program postupně čte hodnoty bajtů a sečítá je. Součet se vydělí 256 a zbytek po dělení je porovnán s posledním, tzv. kontrolním bajtem. Pokud se výsledek neshoduje se zde uvedeným číslem, určitě jsme někde udělali chybu. Zaváděcí program pak ohlásí číslo vadného řádku, což nám značně usnadní hledání chyby.

Před vlastním blokem dat se nachází dvě desítková čísla. Prvé je adresa prvého bajtu v paměti, od kterého se program bude ukládat, druhé uvádí, od kterého řádku programu začíná blok dat (pro případ ohlášení chyby).

Konec bloku dat rozezná podle prázdného řetězce " ". K zavedení strojového programu do paměti stačí jednoduše zadat příkaz RESTORE s argumentem ukazujícím na prvý řádek bloku dat, a zaváděcí program spustit příkazem GO SUB 9990. Tímto postupem je umožněno uložit do různých částí paměti více samostatných programů současně.

Tak si konečně uvedeme univerzální zaváděcí program, který budeme nadále používat. V dalším textu uvedu vždy pouze blok dat k tomuto "zavaděči", česky "loudru"..... (čti LOADER).

```
9990 REM LOADER STROJOVYCH PROGRAMU
9991 READ a,n
9992 READ l$: LET l= LEN l$: LET s=0: LET k=2: LET n=n+1
9993 IF l=0 THEN RETURN
9994 LET o2= CODE l$(k): LET o1= CODE l$(k-1)
9995 LET c=o1-48-7*(o1>64)+16*(o2-48-7*(o2>64))
9996 IF k<l THEN POKE a,c: LET s=s+c: LET k=k+2: LET a=a+1:
     GO TO 9994
9997 IF s-256*INT (s/256) <> c THEN PRINT "CHYBA NA RADKU";
     n: STOP
9998 GO TO 9992
```

A BUDEME SI PSÁT.....

Teď již víme, co jsou to hexadecimální čísla, tak se můžeme pocvičit v jejich používání. Minule jsme si zkoušeli čmárat na obrazovku, ale častěji než kreslení ve strojovém kódu, budeme určitě potřebovat vypisovat různé zprávy, nápisy, výsledky atd. Takže se podíváme na to, jak by jsme to mohli provést, pro zjednodušení opět použijeme rutiny z ROM.

Nejprve vám ale musím prozradit, že v našem superpočítači existují tzv. "kanály", kterými proud dat teče tam, kam je námi (nebo programem) nasměrován. Budeme-li si chtít proud dat nasměrovat na obrazovku, musíme si nejprve otevřít kanál č. 2 (číslo 3 vede na tiskárnu, atd). Jako prvý krok se naučíme smazat obrazovku, to budeme určitě používat velmi často.

Prohlédněte si následující program v assembleru, jak se to dělá. Zde je použit opět jiný způsob zápisu, na začátku každého úseku programu si uvedeme jeho název (tzv. LABEL, lejbl), aby jsme jednak věděli co má dělat, a hlavně nm nahrazuje (prozatím) jeho adresu. Při vkládání programu do počítače pomocí programu GENS 3 pak místo CALL 63025 můžeme napsat třeba CALL smaz a je to jednodužší. Ale k věci:

```
SMAZ:   CD 6B 0D    CALL 0D6B    ;podprogram CLS v ROM
        3E 02       LD A, 02     ;kanál č. 2
        CD 01 16    CALL 1601    ;tento otevřít (v ROM)
        C9          RET          ;a zpět
```

Dokázali by jste tento prográmek uložit do souboru DATA pro zavedení do počítače? Vždyť je to přece tak jednoduché, hele:

```
10 DATA "CD6B0D3E02CD0116C9"," "
```

Aha, tak moc jednoduché to zase není, jak se zdálo, chybí nám zde kontrolní součet na konci řetězce. Ale abychom si (zatím) příliš nekomplikovali život, vypustíme v loaderu řádek 9997. Později si ukážeme, jak si kontrolní součet snadno a jednoduše vytvoříme a budeme jej moci používat.

Tak dál: Navrhneme si tedy program PRINT, který si přečte v paměti data (kódy znaků) a vypíše je na obrazovku. Opět si zjednodušíme život (proč ne?) použitím hotového programu z ROM na adrese 203C (hexadecimálně).

```
PRINT:  CALL SMAZ       ;příprava obrazovky a kanálu tisku
        LD DE, TEXT     ;adresa začátku textu v paměti
        LD BC, DELKA    ;a jeho délka (bajtů k vytištění)
        CALL 203C       ;zavoláme podprogram v ROM
        RET             ;a vrátíme se do BASIC.
```

Následuje seznam bajtů k vytištění, použijeme kódy znaků podle seznamu v manuálu k počítači.

```
TEXT:   0D 0D 0D 0D     ;4krát ENTER - 4 prázdné řádky
        48 20 45 20     ;a jednotlivé kódy
        4C 20 4C 20     ;oddělené kódem 20 - mezerou
        53 20 4F 20     ;za každým písmenem se tedy
        46 20 54 20     ;vytiskne mezera
```

Pro lepší názornost si celý prográmek sepíšeme jak v kódech tak i v assembleru a mimo názvů bloků si budeme uvádět i přímo konkretní adresy. Pro jeho uložení si zvolíme třeba adresu 64000 to je hexadecimálně FA00 (aby se nám to lépe psalo). Takže:

```
SMAZ:   FA00  CD 6B 0D        CALL 0D6B
        FA03  3E 02           LD A, 02
        FA08  C9              RET
PRINT:  FA09  CD 00 FA        CALL SMAZ
        FA0C  11 16 FA        LD DE, TEXT
        FA0F  01 14 00        LD BC, DELKA
        FA12  CD 3C 20        CALL 203C
        FA15  C9              RET
TEXT:   FA16  0D 0D 0D 0D
        FA1A  20 48 20 45
        FA1E  20 4C 20 4C
        FA22  20 53 20 4F
        FA26  20 46 20 54
```

Tak, a teď jsme zvědavi, jak (a jestli vůbec) to bude fungovat. Vložíme si příslušné bajty do paměti pomocí uvedeného zaváděcího programu (zatím vynecháme řádek 9997), který doplníme těmito řádky:

```
10 DATA 64000,20
20 DATA "CD6B0D3E02"
30 DATA "CD0116C9"
40 DATA "CD00FA1116FA"
50 DATA "011400CD3C20"
60 DATA "C90D0D0D0D20"
70 DATA "482045204C20"
80 DATA "4C2053204F20"
90 DATA "462054"," "
```

Taky jste si toho už všimli? Schválně jsem v předešlém výkladu zamlčel jednu věc, a čekám, kdy na to přijdete sami..... Prostě - adresy a vůbec všechny dvoubajtová čísla se do příkazů LD, CALL atd. píšou obráceně! Nejprve druhý bajt a potom prvý. Takže CALL 1601 zapíšeme jako CD 01 16. Zatím si to zapamatujte jako fakt, až k tomu bude vhodná chvíle, vysvětlím proč. Teď by vás to ještě popletlo.

Ale k našemu programku. Je v něm obsažen jeden chyták, asi jste na něj přišli, pokud jste se jej pokoušeli spustit. Smazal obrazovku - a dost. To víte, autor dělá ze sebe chytrého a nechává vás, aby jste mu na to vždycky naletěli..... O co zde vlastně jde: strojový program se nemusí vždy spouštět na počáteční adrese prvého bajtu, ale třeba někde jinde. Já jsem ze začátku neměl o tom žádnou literaturu a taky jsem koukal jak blbej, co mi programky, které jsem si pečlivě opsal, vlastně dělají.

Začátek programu a jeho startovní adresa - to jsou (dosti často) dvě rozdílné věci! Pokud jste se jej tedy pokoušeli spustit RANDOMIZE USR 64000, tak se pouze vymazala obrazovka a nic víc. Prohlédněte si program v assembleru - vono to vlastně je tak napsaný, nojo...

V tomto případě potřebujeme spustit program PRINT, ten ale začíná na adrese FA09, to je 64009! Čili znova: RANDOMIZE USR 64009 - a ejhle, heleho - vono to funguje (pokud jste neudělali někde chybu).

Budete-li tento program PRINT používat později ve svých vlastních programech jako podprogram, tak tedy vězte, že pokud bude podprogram PRINT volán jen jednou, s jedním textem, je možno SMAZ vynechat a příkazy CALL 0D6B, LD A,02 a CALL 1601 umístit těsně před CALL 203C, např. takhle:

```
LD DE, TEXT
LD BC, DELKA
CALL 0D6B
LD A, 02
CALL 1602
CALL 203C
   atd
   atd
```

V tom tvaru, jak jsme si program uvedli předtím, je univerzálnější - dá se volat buď samotná rutina SMAZ, nebo PRINT. Rutina PRINT může být pak v programu použita i vícekrát, s jinými údaji v DE a BC, podle toho, kolik různých textů budeme chtít vypisovat. Přitom texty nemusí být v programu uloženy za rutinou PRINT, ale třeba až úplně na konci programu, třeba i všechny najednou, aby se nám nepletly mezi příkazy strojového kódu.

Ještě jedna rada: pokud bude rutina PRINT použita v rozsáhlejším programu, bude vhodné před jejím spouštěním obsahy registrů DE a BC uschovat a po provedení opět obnovit, aby se mohly používat dále s původním obsahem. Samozřejmě že k tomu použijeme příkazy PUSH DE, PUSH BC a na konci rutiny opět POP BC a POP DE.

Když jsme již u toho čarování s obrazovkou, popíšeme si jinou rutinu pro její mazání. Místo obyčejného CLS, což umí (v BASIC) každý blbec, si my, jakožto experti vytvoříme něco jiného, lepšího.

```
SMAZ2:  06 08      LD B, 08        ;jeden bajt je 8 bitů
SMYC2:  21 00 40   LD HL, 4000     ;adr. obraz paměti 16384
        11 00 18   LD DE, 1800     ;délka obr. paměti 6144
SMYC1:  CB 3E      SRL (HL)        ;posuň bity v bajtu,
                                    který ukazuje adresa
                                    v reg. HL
        23         INC HL          ;HL zvětši o 1, adresa
                                    dalšího bajtu
        1B         DEC DE          ;DE zmenši o 1, bajt
                                    byl upraven
        7A         LD A, D         ;obsah reg. D zapiš do A
        B3         OR DE           ;zkontrolij obsah
        20 F8      JR NZ,SMYC1     ;není-li výsledkem nula,
                                    opakuj smyčku 1
        10 F0      DJNZ SMYC2      ;celé opakuj 8 krát (B!)
        CD 6B 0D   CALL 0D6B       ;zbytek vymaže ROM
        C9         RET             ;a zpět.
```

Průběh této rutiny SMAZ2 je již poněkud efektnější, obraz se posune o 1 znak doprava a pak teprve zmizí. Umíte si tento program převést do souboru DATA pro zavedení do paměti?

Nu dobrá, ukecali jste mne. Ještě naposledy vám pomůžu. Takže: Zvolíme si adresu, od které budeme program ukládat, třeba 60000 (klidně můžeme hazardovat místem, máme ho dost).

```
100 DATA 60000,110
110 DATA "0608210040"
120 DATA "110018CB3E"
130 DATA "231B7AB320"
140 DATA "F810F0CD6B"
150 DATA "0DC9"," "
```

Pokud si jej budete chtít zařadit k předešlému programu PRINT místo původního SMAZ, musíte pak změnit v programu volání CALL SMAZ jeho novou adresou! Místo FA00 je nyní EA60!

Jo, vy nevíte jak na to? Že hexadecimální čísla jsou pro vás stále ještě kouzlo (HEXE)??? No tak to je nejvyšší čas, abych vám prozradil další fígly -

PŘEVODY ČISELNÝCH SOUSTAV.

Jestliže máte při sestavování strojového programu volný počítač, můžete si do něj vložit následující program v BASIC, který dokáže převádět čísla z libovolné do libovolné soustavy, třeba z šestnáctkové do dvojkové, z pětkové do osmičkové, nebo (což právě potřebujeme) z desítkové do šestnáctkové a naopak.

Vložte si do počítače:

```
10 REM Prevod ciselnych soustav
20 POKE 23658,0
30 INPUT "Prevadene cislo: ";a$: PRINT a$;" (";
40 IF a$="" THEN STOP
50 LET a=0: INPUT "Jeho zaklad: ";z: PRINT z;")",
60 FOR i=1 TO LEN a$
70 LET a=a*z+(CODE a$(i))-48-39*(a$(i)>"9")
```

```
 80 NEXT i
 90 LET a$="": INPUT "Novy zaklad: ";z
100 LET i=a-z*INT (a/z): LET a=INT (a/z)
110 LET a$=CHR$ (i+48+39*(i>9))+a$
120 IF a>0 THEN GO TO 100
130 PRINT a$;" (";z;")"
140 GO TO 50
```

Obsluha tohoto programu je doufám jasná. Zeptá se na převáděné číslo, vložte 60000. Dále chce vědět jeho základ - převádíme z desítkové soustavy, tak mu zadáme 10. Na otázku "Nový základ" odpovíme samozřejmě 16, protože chceme převádět do šestnáctkové soustavy.

Na obrazovce budeme mít:

60000 (10) EA60 (16)

Tento prográmek se nám bude ještě mockrát hodit, třeba pro převod do dvojkové soustavy, pro rozdělování dvoubajtového čísla na dvě jednobajtová, atd. Cože? Ptáte se jak?

No přeci - třeba 16384 potřebujeme rozložit na dvě jednobajtová čísla. Využijeme toho, že v šestnáctkové soustavě je každý bajt složen ze dvou znaků a převedeme si je do šestnáctkového tvaru, objeví se nám 4000 (16). Číslo 00 převádět zpět do desítkové soustavy nemusíme, to umíme z hlavy (no přeci 0, sakra), a 40 (16) = 64 (10) a už máme dvě jednobajtová čísla.

No a pokud nemáme právě po ruce počítač, nebo je momentálně zaneprázdněn jinými úkoly (třeba nám dělá slohový úkol do školy) pomůžeme si následující tabulkou (doporučuji si ji opsat na kousek tvrdšího papíru a uložit na noc pod polštář).

HEX	1	2	3	4
Ø	Ø	Ø	Ø	Ø
1	4Ø96	256	16	1
2	8912	512	32	2
3	12288	768	48	3
4	16384	1Ø24	64	4
5	2Ø48Ø	128Ø	8Ø	5
6	24576	1536	96	6
7	28672	1792	112	7
8	32768	2Ø48	128	8
9	36864	23Ø4	144	9
A	4Ø96Ø	256Ø	16Ø	1Ø
B	45Ø56	2816	176	11
C	49152	3Ø72	192	12
D	53248	3328	2Ø8	13
E	57344	3584	224	14
F	6144Ø	384Ø	24Ø	15

A teď jak ji používat - ganz einfach, jak říkali staří římané. Nebo to byli řekové? Už se mi to plete. Prostě: dejme tomu, že potřebujeme převést číslo 12345 do šestnáctkové (hex) soustavy. Začneme od nejvyšších čísel v prvém desítkovém sloupci. Najdeme číslo, které je nejblíže vyšší, než dané.

```
  12345
- 12288   u tohoto čísla je ve sloupci HEX číslo 3
-------
   57
```

Hledáme další číslo, menší než 57: jeden sloupec jsme vynechali, zapamatujeme si Ø. A dále:

```
  57
- 48  ve sloupci HEX je číslo 3
 ---
   9  ve sloupci HEX je číslo 9
```

Složíme si tedy dohromady 3039, to je číslo 12345 v HEX.soustavě

A do desítkové soustavy: třeba číslo FBAC.

Vedle písmene F je v prvém sloupci	61440
Vedle písmene B je ve druhém sloupci	2816
Vedle písmene A je ve třetím sloupci	160
Vedle písmene C je ve čtvrtém sloupci	12
Podtrhneme, sečteme:	64428

No a to je výsledek v desítkové soustavě.

Zpočátku vám to bude připadat příliš komplikované, ale po několika cvičných převodech to již budete zvládat - no, alespoň tak jako já.....

TEST PRO VÁS.

Chacha, teď se přesvědčíme, jak vám to jde (to jsem ale škodolibej, žejo?). Napíšu vám krátký prográmek v assembleru (prozradím jen tolik, že má provést inversi obrazovky) a vy si jej převedete do souboru DATA pro zaváděcí prográmek. Počáteční (a tedy i startovací) adresu vám určím na 61000 (aby se nepletl s dosud vloženými programy). Koukněte:

```
INVERT:  21 00 40      LD HL, 4000
         01 00 18      LD DE, 1800
SMYC:    3E FF         LD A, FF
         96            SUB (HL)
         77            LD (HL), A
         23            INC HL
         0B            DEC BC
         78            LD A, B
         B1            OR C
         20 F6         JR NZ, SMYC
         C9            RET
```

A včil ukažte co umíte. Schválně jsem nevypisoval ani komentář, abych moc nepomáhal.

Že to bylo jednoduché? A chodí to? Tak to si na vás musím vymyslet něco jiného. Další prográmek bude tajemnější, neprozradím vám ani co má dělat (to by jste měli uvidět), ani jeho kódy, ty si musíte vyhledat v seznamu sami. Hečl!

```
START:   LD B, 00
SMYC2:   PUSH BC
         LD B, 00
SMYC1:   LD A, B
         OUT A, (FE)     ;tady vám napovím: D3 FE
         DJNZ SMYC1
         POP BC
         DJNZ SMYC2
         LD A, (5C48)    ;nezapomeňte! Nejdřív druhý, pak
                          prvý bajt!
         SRL A
         SRL A
         SRL A
         OUT A, (FE)
         RET
```

A hin sa hukáže..... Nemusí to sice trvat do roka a do dne, ale těm, kteří toto zvládnou se prográmek odmění zvláštními efekty.

TUČNÉ PÍSMO.

Než si to ti loudavější dají dohromady, mám tady typ pro ty z vás, kteří vlastní ZX Spectrum. Zaujalo vás písmo na Didaktiku a chtěli by jste jej mít také? Je značně výraznější, oproti původnímu spektráckému. Ale my, vládci strojového kódu, si je přeci umíme vyrobit taky!

Jak na to: Program kopíruje do buněk paměti počínaje adresou 56576 a dále, 96 znaků (t.j. 768 bajtů) a přesune je o jeden bod doprava. Spouští se RANDOMIZE USR 57786, zpět k původním znakům se dostaneme po příkazu POKE 23607,60 (systémová proměnná CHARS, hex. adresa je 5C37).

Program vám tentokrát uvedu i s adresami, ale převést do loudru si jej musíte již sami.

```
TUCPIS:  E1BA  11 00 DD   LD DE, DD00  ;adresa začátku
                                       znakové sady
         E1BD  D5         PUSH DE      ;uschovat
         E1BE  01 00 03   LD BC, 0300  ;768 znaků
         E1C1  2A 36 5C   LD HL,(5C36) ;tabulka znaků
                                       CHARS
SMYC:    E1C4  24         INC H        ;další znak
         E1C5  7E         LD A, (HL)   ;ulož do reg. A
         E1C6  A7         AND A
         E1C7  1F         RRA          ;posuň o 1 bit
                                       doprava
         E1C8  B6         OR (HL)      ;přidej další bod
                                       t.j. zdvojit
         E1C9  12         LD (DE), A   ;nový znak schov.
         E1CA  23         INC HL       ;připrav další
                                       znak
         E1CB  13         INC DE       ;připrav adresu
                                       pro další znak
         E1CC  20 F6      JR NZ, SMYC  ;a celé opakuj
```

```
E1CE   1Ø F4      DJNZ SMYC     ;dokud není 768
                                 bajtů hotovo
E1DØ   E1         POP HL
E1D1   25         DEC H
E1D2   22 36 5C   LD(5C36),HL  ;do CHARS novou
                                 adresu znaků
E1D5   C9         RET           ;a zpět do BASIC.
```

OPAKOVÁNÍ.

Jestliže jste se ve svých pokusech dostali až sem (beze ztráty desítky), jste dobří, ale zase ne tolik. Určitě jste přeskakovali, nevšimli si něčeho důležitého, atd. No a protože opakování je matkou moudrosti (jen aby to nebyla macecha.....), zopakujeme si třeba nejpoužívanější rutiny strojového kódu - to je pro tisk na obrazovku. No a hned si vše podrobně vysvětlíme.

Pokud jste předešlý text sledovali pozorně, jistě vám neuniklo (ale asi jó) že skok na adresu 1Ø, t.j. RST 1Ø vyvolá podprogram v ROM, který se jmenuje PRINT OUT.

Tento podprogram vypíše na obrazovku znak, jehož CODE je uloženo v registru A, na to místo obrazovky, které je právě na řadě. Je to jeden z nejuniverzálnějších a tím pádem také nejčastěji používaných podprogramů v ROM. Používá se nejen k psaní příslušných znaků, ale také třeba ke vkládání řídicích znaků pro obrazovku, nebo ke změně kanálů. Takže nyní se budeme delší dobu zabývat tímto podprogramem.

Umíte si do počítače vložit následující program? (pro jistotu uvádím i strojové kódy):

```
3E 41       LD A, 41     ;vlož CODE znaku "A" do
                          registru A.
D7          RST 10       ;vypiš znak na obrazovku
C9          RET          ;návrat zpět do BASIC
```

Tak se vám vše podařilo, program jste úspěšně zavedli do počítače a spustili jej od jeho počáteční adresy (jistě jste si ji vybrali předem). A hele - ve spodní části obrazovky na řádku 22 se objevilo písmeno A a hned zmizelo, protože se tam vypsalo hlášení o úspěšném ukončení programu. Takže znova - zkuste jej spustit touto posloupností příkazů: RANDOMIZE USR start: PAUSE 0 No a jelikož počítač provádí PAUSE 0, čili nic, zůstalo na dolním řádku písmeno A jako AHA.

AHA, ale proč? Proč právě písmeno A je vám jasné, jeho kód jste vložili do registru A a pomocí RST 10 nechali vytisknout. Ale - (vždycky je tady nějaké to ALE) - proč na spodní řádek? Přeci chceme začít psát od prvého řádku nahoře?

Tak tedy vězte, vážení adepti strojové můzy, že počítače řady Didaktik mají po nastartování (zapnutí nebo RESET) otevřený tzv. kanál K (jako KEYBOARD, klávesnice), který slouží právě k obsluze dolní části obrazovky. My chceme znak vypsat nahoře, proto musíme nejprve otevřít kanál S (jako SCREEN, obrazovka). On ještě existuje kanál P (jako PRINTER, tiskárna), který je určen k obsluze tiskárny, ale to jen tak na okraj, ještě se k němu vrátíme, až jej budeme potřebovat.

A jak kanály otevírat? Představte si takovýhle sled příkazů

```
3E 02       LD A, 02     ;číslo kanálu S do reg.A
CD 01 16    CALL 1601    ;otevři kanál S
```

Příkazem CALL 1601 jsme vyvolali podprogram v ROM na adrese 1601 který provede požadovanou činnost - jestliže registr A obsahuje číslo 2, otevře kanál číslo 2.

Jde to sice i jinak, ale..... no, posuďte sami.

```
AF          XOR A         ;vynulovat registr A
32 3C 5C    LD 5C3C, A    do syst. proměnné TVFLAG
                          vložit Ø
```

Vrátíme se ale zpět k našemu předešlému programu pro tisk na obrazovku. Zkuste si třeba tohle:

```
3E Ø2       LD A, Ø2      ;do reg. A číslo kanálu
CD Ø1 16    CALL 16Ø1     ;otevřít kanál S
3E 41       LD A, 41      ;do reg. A kod znaku "A"
D7          RST 1Ø        ;vypiš znak na obrazovku
C9          RET           ;a zpět do BASIC
```

Tak co, kde se nyní objevil znak "A"? Určitě v levém horním rohu (za předpokladu, že jste tento program zavedli do počítače a spustili). Znak se objevil přesně na tom místě, kde by se ukázal po BASICovském PRINT "A" (při prázdné obrazovce).

No a už zase něco umíme, tak si pro porovnání vyzkoušíme stejný program v BASIC a ve strojovém kódu. Zkuste třeba:

```
1Ø PRINT "A"
2Ø GO TO 1Ø
```

A jak by to vypadalo ve strojáku:

```
3E Ø2       LD A, Ø2
CD Ø1 16    CALL 16Ø1     ;otevřít kanál "S"
3E 41       LD A, 41      ;znak "A"
D7          RST 1Ø        ;vypiš na obrazovku
18 FB       JR FB         ;opakuj od LD A, 41
```

OVŠEM POZOR! TENTO PROGRAM SE DÁ ZASTAVIT POUZE POMOCÍ RESET!
(Hádejte proč.....)

No tak to bychom měli, a jede se dál. Určitě se nespokojíte s tím, že umíte vypsat znak do levého horního rohu obrazovky (a to je také správné, protože nespokojenost hýbe světem i dějinami). Budeme předpokládat, že jej chcete vytisknout na obrazovku na desátý řádek zhora, patnácté místo zleva (přibližně střed obrazovky).

Můžeme si vybrat několik možností, jak na to. Tak třeba použít řídicí znaky obrazovky, z BASIC známe příkaz PRINT AT. Zalistujeme v seznamu kódů.... aha. Řídicí znak AT má kód 16, začneme tedy psát:

```
3E Ø2       LD A, Ø2      ;tohle znáte.....
CD Ø1 16    CALL 16Ø1     ;a tohle taky....
3E 16       LD A, 16      ;řídicí znak AT
D7          RST 1Ø        ;proveď AT
3E ØA       LD A, ØA      ;dekadicky řádek 1Ø
D7          RST 1Ø        ;proveď
3E ØF       LD A, ØF      ;dekadicky sloupec 15
D7          RST 1Ø        ;proveď
3E 41       LD A, 41      ;znak "A"
D7          RST 1Ø        ;proveď
C9          RET           ;a zpět do BASIC
```

Jestli jste si toto vyzkoušeli, objevil se znak "A" na požadovaném místě obrazovky.

Vždycky ale máme více možností, jak něco udělat. Můžeme třeba použít podprogram v ROM na adrese ØDD9, který nastaví požadovanou tiskovou pozici na obrazovce podle údajů v registrech B a C. V registru B je uloženo číslo řádku, v C je číslo sloupce (pozice z levé strany).

Ale abychom se nesplétli! V Didaktiku je skřítek, který se na obrazovku dívá z druhé strany, a tedy vidí vše obráceně (je to původně angličan a tam se jezdí vlevo....). V příručce ABC DIDAKTIK jsem to vysvětloval v kapitole o systémových proměnných takže nyní jen krátce a stručně - číslo řádku bude ØE a číslo sloupce 12. Náš program by pak mohl vypadat třeba takhle:

```
Ø6 ØE       LD B, Øe      ;desátý řádek
Øe 12       LD C, 12      ;patnáctý sloupec
CD D9 ØD    CALL ØDD9     ;PRINT AT
```

a dále již normálně:

```
3E 41       LD A, 41      ;znak "A"
D7          RST 1Ø        ;vypiš
C9          RET           ;návrat do BASIC
```

Jako samozřejmost předpokládám, že jste si nejprve otevřeli kanál "S" ! Čili:

```
3e Ø2       LD A, Ø2
CD Ø1 16    CALL 16Ø1
```

Samozřehmě, že i tento prográmek se dá zkrátit a zrychlit, třeba tím, že registry B a C budeme považovat za dvojitý registr (tedy šestnáctibitový), pak mu říkáme (to už tady bylo!) registrový pár BC. No a místo LD B, ØE a LD C, 12 můžeme použít

```
Ø1 12 ØE    LD BC, ØE12
```

a výsledek bude stejný. Ušetřili jsme ale 1 bajt! (a také rychlost provádění programu je maličko větší).

Jenže my jsme tvorové nároční, nespokojíme se pouze s vypsáním jediného znaku, budeme jich chtít vypsat více, třeba "ZDRAVI VAS DIDAKTIK", "STRC PRST ZKRZ KRK", "STISKNI ENTER" a podobně. Jeden takový prográmek jsme si již dříve uvedli, zde jej zkusíme napsat pomocí příkazů RST 1Ø.

Dávejte pozor:

```
LD A, Ø2                ;víte proč Ø2?
CALL 16Ø1               ;co to má být?
LD A, 41                ;jo, to známe, je to "A"
```

```
RST 10                  ;no přeci tisk
LD A, 48                ;znak "H"
RST 10
LD A, 4F                ;znak "O"
RST 10
LD A, 4A                ;znak "J"
RST 10
RET
```

Vidíte sami, jaký je postup v tomto programu (blbý, že jo?). Do registru A vždycky vložíme kód znaku, určeného k vytištění, vytiskneme jej, znovu vložíme kód dalšího znaku..... atd až do zblbnutí (v případě potřeby delšího textu). Je to takové začátečnické, ale my jsme přece již skoro experti, proto se pokusíme o něco lepšího.

Máte-li k dispozici výpis ROM, můžete zjistit, že na adrese 203C je podprogram PR-STRING (pokud výpis nemáte, musíte mi holt věřit), který je určen pro výpis řetězců. Ano, správně, je to ten, který jsme již jednou použili, aniž bychom o tom něco věděli. Teď už jsme ale chytřejší, a proto si o něm můžeme povídat zasvěceněji.

Před vyvoláním podprogramu musí být v registrovém páru DE adresa prvého znaku řetězce, určeného k výpisu na obrazovku, a v BC počet znaků, které se mají vypsat. Předešlý program proto můžeme nahradit novým. Pro zajímavost zde uvádím i adresy, aby jste si jej mohli vyzkoušet. Aby to však nebylo tak kór moc jednoduché, jsou uvedeny hexadecimálně.....

```
        5B00    11 06               ;kódy pro PAPER 6
        5B02    10 02               ;kódy pro INK 2
        5B04    12 01               ;kódy pro FLASH 1
        5B06    41 48 4F 4A         ;kódy písmen AHOJ
START:  5B0D    3E 02     LD A, 02  ;víte proč?
        5B0F    CD 01 16  CALL 1601 ;otevřít kanál "S"
        5B12    11 00 5B  LD DE,5B00 ;adresa prvého znaku
```

```
5B15    01 0A 00  LD BC,000A  ;10 znaků
5B18    CD 3C 20  CALL 203C   ;podprogram PR-STRING
5B1B    C9        RET         ;to MUSÍTE už znát....
```

Podprogram PR-STRING je takové šikovné udělátko, které za vás mockrát (tedy BC krát) opakuje LD A, N a RST 10, které jsme použili dříve. Vždy si vytáhne bajt z adresy určené v DE, pak jej vytiskne, adresu DE zmenší o 1, BC taky, a celý postup opakuje BC krát, pak skončí.

↑↑

No a my ZATÍM budeme končit také, pokud se vám zdá, že toho umíte ještě moc málo, tak to se vám nezdál V tom případě vůbec neváhejte a koukejte si zakoupit další díl této příručky, ve kterém vás potrápím ještě víc.
Zatím si PEČLIVĚ vše ještě jednou odzkoušejte, aby jste pak byli schopni strávit další porce. Nervově labilnější povahy se již asi vzdají, ale my, kteří chceme svým počítačům skutečně vládnout bez omezení, VYDRŽÍME!

Mnoho úspěchů vám všem přeje

HELLSOFT.

↑↑

V edici příruček HELLSOFT dosud vyšlo:

100+1 PROGRAM - rozebráno
MANUÁLY 1 - rozebráno
Můj přítel DIDAKTIK - mnoho tipů a triků, 48.- Kčs
KLADIVO NA PROGRAMY - jak se "vloupat" do programů... 35.- Kčs
TEXTOVÉ EDITORY - podrobné popisy použití D-TEXT a Tasword 20.-
GENS/MONS - podrobné popisy těchto mocných pomocníků, 30.- Kčs
MANUÁLY 2 (40.-), MANUÁLY 3 (38.-), MANUÁLY 4 (38.-)
Popisy programů, jak užitkové, tak i hry.