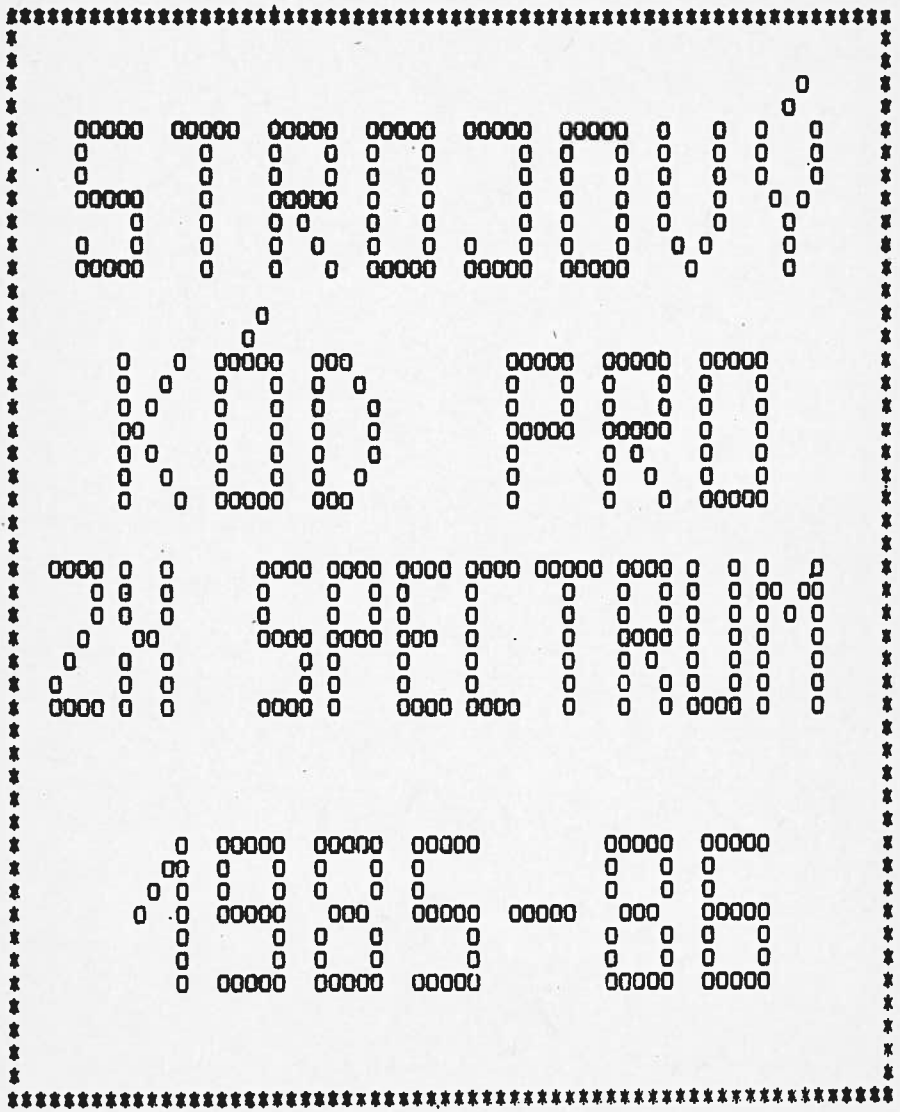


OBSAH

PŘÍLOHY

1. Úvod

Pokud se chcete naučit znát a ovládat strojový kód, pak by nemělo význam, abyste pojednání jen prolistovali a odložili do své knihovny. Ten, kdo chce o svém Spectru vědět o něco více a nezná strojový kód, pro toho má velký význam prostudovat celé pojednání, nahrát si pomocné programy MC 1 a MC 2 a začít programovat ve strojovém kódu.

Abyste se přesvědčili, že námaha, kterou vynaložíte na učení strojového kódu není marná, začneme s několika sledy strojového kódu, který nám na displeji zobrazí nějaké abstraktní obrázky.

Vložte do počítače následující program:

```
10 CLEAR 31999
20 BORDER 0
30 DATA 1,0,8,17,0,88,33,0,0,237,176,201
40 FOR I=0 TO 11
50 READ X
60 POKE 32000+I,X
70 NEXT I
80 PAUSE 0
90 LET Y=USR 32000
```

Dejte pozor na to, abyste program vložili přesně. Pak jej spusťte stisknutím RUN. Obrazovka zůstane prázdná, dokud nestisknete libovolné tlačítko. Jakmile jej stisknete, obrazovka se zaplní čtverečky, z nichž některé blikají (není-li tomu tak, zkontrolujte znovu program). Průběh je rychlý . A nyní program rozšíříme. Smažte řádku 90 a doplňte program takto:

```
100 LET T=0:LET A=0
110 IF T>=255 THEN LET T=T-256*INT(T/256):LET S=S+1
120 POKE 32007,T:POKE 32008,S
130 LET Y=USR 32000
140 LET T=T+1
150 GOTO 110
```

Stiskněte RUN a libovolné další tlačítko. Obdržíte podobný obraz, avšak bude se posouvat doleva. Změňte 140 na 140 LET T=32 a bude se posouvat nahoru, LET T=31 a bude se posouvat diagonálně. Zkuste si i jiné hodnoty pro T. Proveďte LET T=T+1 a změňte DATA takto:

30 DATA 1,0,24,17,0,64,33,0,0,237,176,201 a celé opakujte a měňte řádek 140.

Dáme-li oba sledy dohromady, obdržíme:

```
10 CLEAR 31999
20 BORDER 0
30 DATA 1,0,24,17,0,64,33,0,0,237,176,201
35 DATA 1,0,8,17,0,88,33,0,0,237,176,201
40 FOR I=0 TO 23
50 READ X
```

```
 60 POKE 32000+I,X
 70 NEXT I
100 LET T=0:LET S=60
110 IF T>=256 THEN LET T=T-256*INT(T/256):LET S=S+1
120 POKE 32007,T:POKE 32008,S
130 LET Y=USR 32000
140 LET Y=USR 32012
150 LET T=T+1
160 GOTO 110
```

Stiskněte RUN a nechte program běžet asi 2 minuty. Program se náhle ztrácí. Změňte v řádce 150 na T+32, T+31 atd. Měňte aktualizaci v řádce 100. Podívejte se, co se stane, dáte-li S=0 nebo S=40 . Pro S=63 běží program tak rychle, že jej těžko můžete sledovat. Přiřadíte-li:

```
145 IF INKEY$="" THEN PAUSE 0
```

nebude se dít nic, dokud nestisknete nějaký tastr. Krátkými stisky můžete sledovat postupný vývoj vzorku na obrazovce. Změnou hodnot můžete vytvořit nepřeberné množství vzorků. Nesmíte ovšem měnit řádky 30, 35, 130 nebo 140, ve kterých je zapsán strojový kód.

2. Čísla ve strojovém kódu

Pro práci ve strojovém kódu budeme potřebovat nejen desítkovou soustavu, ale i jiné soustavy. Především binární se základem 2 a dále hexadecimální soustavu se základem 16.

2.1 Desítková soustava

S čísly pracujeme ve známých dekadických pojmech. Jestliže napíšeme číslo 5237, budou je všichni číst jako 5 tisíc 2 stě 37, t.j. početně : (5*1000)+(2*100)+(3*10)+7.

Znamená to ,že první číslo zprava (7) násobíme 10↑0=1,druhé číslo zprava (3) násobíme číslem 10↑1=10,třetí číslo (2) násobíme 10↑2=100 a čtvrté číslo (5) násobíme 10↑3=1000. Vidíme, že násobitel má základ 10 a exponent se zvyšuje o 1 pro každé další číslo ve směru zprava doleva. Proto tuto soustavu nazýváme desítkovou neb dekadickou. Posledně napsaný text "název" budeme nadále používat.

2.2 Binární soustava

První konstruktéři elektrických počítacích strojů použili zpočátku dekadickou soustavu, ale dostali se do nepřekonatelných obtíží, zjistili totiž, že je nesmírně obtížné rozeznávat v elektrických hodnotách rozdíly 10 ti stavů, neboť dekadická soustava užívá čísel 0 - 9. Nejjednodušší, co lze s elektrickým signálem provést, je jeho zapnutí a vypnutí. Toto jsou dva stavy: Ano a Ne. Nebo jak je v binární soustavě uvedeno, stavy 0 a 1. Binární soustava má jenom tato dvě čísla. Binární soustava má tedy základ 2.

Tak např. můžeme číslo v binární soustavě 1101 převést na dekadické podobným způsobem, jako jsme to provedli s číslem 5237 v dekadické soustavě. První číslo zprava (1) násobíme číslem 2↑0=1, druhé číslo zprava (0) násobíme číslem 2↑1=2, třetí číslo (1) násobíme číslem 2↑2=4 a čtvrté číslo (1) číslem 2↑3=8 takže:

```
1  1  0  1
!  !  !  !
!  !  !  !----x 2↑0=1
!  !  !-------x 2↑1=0
!  !----------x 2↑2=4
!  !-------------x 2↑3=8
                 -------
součet dává dekadické číslo      13
```

Píšeme tedy, že číslo 1101 B=13 D, kde B je binární index a D dekadický index.

Obráceně je to rovněž snadné. Máme vyjádřit v binární soustavě číslo 25D. Rozepíšeme mocniny čísla 2:

```
2↑5  2↑4  2↑3  2↑2  2↑1  2↑0      a vypočte se

32   16   8    4    2    1
```

Zkusíme, které nejvyšší číslo, t.j. kterou nejvyšší mocninu dvou můžeme od daného čísla odečíst. Jestliže lze odečíst, píšeme 1, nelze-li, píšeme 0. 32 nelze odečíst, je větší, takže pod 32 píšeme 0. 16 lze odečíst, pod 16 píšeme 1. 25D-16D=9D. Zbytek po odečtení zkoumáme stejně. 8 lze od 9 odečíst, pod 8 napíšeme 1:9D-8D=1. Číslo 4 ani 2 nelze od 1 odečíst, proto pod 4 a 2 píšeme nuly. Jedničku lze od jedné odečíst a proto pod 1 píšeme 1. Bude tedy:

```
32 16  8  4  2  1
 0  1  1  0  0  1
```

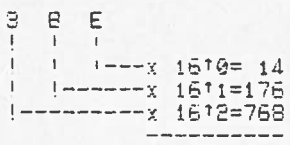
Číslo tedy bude 25D = 11001 B

2.3 Hexadekadická soustava

Binární soustava je výhodná pro malá čísla, ale pro velká čísla je již neúnosná. Např. začíná-li zásobník kalkulátoru pamětí 23651D, pak je v binární soustavě 0101110001100011. S takovým číslem se již obtížně manipuluje nehledě na možnost chyb a proto se používá hexadekadická soustava, která má navíc velkou výhodu ve vztahu k binární soustavě.

Číslo v hexadekadické soustavě je někdy označováno indexem h nebo H. My budeme používat H. V této soustavě je určování poněkud složitější, protože pro čísla 10 - 15 musíme používat písmena a až f (nebo A až F). Opět použijeme raději písmena velká. Bude tedy 10=A, 11=B, 12=C, 13= D, 14=E a 15=F. Hexadekadická soustava používá šestnáct čísel 0 až 9 a A až F. Má tedy tato soustava základ 16.

Číslo z hexadekadické soustavy např. 38E H převedeme do dekadické soustavy stejným způsobem jako u binární soustavy. První číslo zprava (E) násobíme číslem 16↑0=1, druhé číslo zprava (8) násobíme číslem 16↑1=16 a třetí číslo (3) násobíme číslem 16↑2=256. Je potom:



Součet dává dekadické číslo 958, píšeme tedy, že číslo 38E H = 958 D.

Výhodou této soustavy vzhledem k binární je to, že každé číslo hexadekadické soustavy může být nahrazeno binárním číslem stejné hodnoty. Tyto převody ukazuje následující tabulka:

Dek	Hex	Bin	*	Dek	Hex	Bin
0	0	0000	*	8	8	1000
1	1	0001	*	9	9	1001
2	2	0010	*	10	A	1010
3	3	0011	*	11	B	1011
4	4	0100	*	12	C	1100
5	5	0101	*	13	D	1101
6	6	0110	*	14	E	1110
7	7	0111	*	15	F	1111

Poznámka : Ve Spectru můžeme požívat jak malá,tak velká písmena. V tomto pojednání budeme požívat velká písmena.

Převedeme nyní do hexadekadické soustavy číslo 12325 D. Rozepíšeme mocniny čísla 16:

16↑4	16↑3	16↑2	16↑1	16↑0
65536	4096	256	16	1

Zkoumáme, kterou nejvyšší mocninu můžeme odečíst a kolikrát ji můžeme odečíst. Tento násobitel je první z hledaných čísel hexadekadické soustavy. Je tedy
12325-4096-4096-4096=37
16↑3 je v daném čísle obsaženo 3x. Proto pod 16↑3 píšeme 3. Zbytek čísla zkoumáme stejným způsobem. 16↑2 nelze odečíst od zbytku a proto pod něj píšeme číslo 0. 16 je v 37 obsaženo 2x a pod 16↑1 píšeme 2. Zbytek je 5, takže pod 16↑0 píšeme 5. Proto:

65536	4096	256	16	1
0	3	0	2	5

Výsledek:12325D=3025H

Nyní vypišme jednotlivá čísla hexadekadické soustavy v binární formě:

3	0	2	5
0011	0000	0010	0101

Binární vyjádření čísla 123250 obdržíme, když napsaná binární čísla dáme k sobě, takže můžeme napsat:

123250=0011100000100101B

Protože převod z hexadekadické soustavy do binární je snadný, používá se v případě, že potřebujeme čísla zpracovávat v binární soustavě, což potřebujeme právě na počítači. Je to rovněž výhodné v případě udělání chyby. Při dlouhých binárních číslech snadno uděláme chybu při zápisu nul a jedniček.

2.4. Převod pomocí počítače

Pro převod z dekadické do hexadekadické soustavy lze použít následující program:

```
 30 LET P=4
 35 LET H$="0000"
 40 INPUT"DEKAD.ČÍSLO (MAX 65535)";TN
 50 LET N=INT(TN/16)
 60 LET R=TN-16*N
 70 LET H$(P)=CHR$(R+48+7*(R>9))
 80 LET TN=N
 90 LET P=P-1
100 IF TN>0 THEN GOTO 50
110 PRINT "HEX.ČÍSLO=";H$
```

Řádka 70 se stará o vyjádření velkých písmen A až F pro čísla větší než 9. Protože v kódu ASCII, který se v Spectru používá, je A o 7 hodnot výše, musí být pro čísla větší než 9 použito 7*(R>9). Je-li to pravda, pak výraz (R>9)*7 se připočte, není-li to pravda, je výraz roven nule a 7 se nepřipočte. Číslo 48 je nutné proto, že pro nulu má kód ASCII pořadové číslo 48. Výsledek je vyjádřen čtyřmístným číslem, je-li zadáno číslo větší než 65535, program nefunguje .

Pro opačný přepočet je následující program:

```
140 INPUT "4 MISTNE HEX.CISLO";H$
150 LET TN=0
160 FOR P=1 TO 4
170 LET TN=TN*16+CODE H$(P)-48-7*(H$(P)>"9")
180 NEXT P
190 PRINT"DEKAD.CISLO=";TN
```

V řádce 170 je použit stejný, avšak opačný způsob než v řádce 70. Oba dva programy můžeme spojit přidáním následujícího sledu:

```
2 PRINT "CISELNE SOUSTAVY:":PRINT
3 PRINT "1.DEKAD.->HEX
         2.HEX---->DEKAD
         3.KONEC"
5 INPUT "VOL 1,2 NEBO 3";V
8 IF V=1 THEN GOSUB 20
9 IF V=2 THEN GOSUB 140
10 IF V=3 THEN STOP
```

```
 12 PAUSE 0:GOTO 2
120 RETURN
200 RETURN
```

Poznámka : V praxi pak názvy vkládáme bez háčků a čárek.

3. Kladná a záporná čísla

Počítací stroje pracují s binární soustavou s pevně stanoveným počtem binárních čísel 0 a 1. Každé číslo představuje 1 bit. Je-li binární číslo sestaveno ze 4 čísel, t.j. 4 bitů, říkáme mu čtyřbitové slovo. Počítací stroje pracují s 8mi, často se 16 ti případně s 32 bitovými slovy podle vybavení počítače.

Z tabulky na straně 6 vidíme, která čísla dekadické soustavy můžeme vyjádřit 4 bitovým slovem. Je evidentní, že v praxi přicházejí v úvahu větší čísla. Počítací stroj, který by používal jen 4 bitová slova, by byl omezený. V binární soustavě jsou však další dva problémy. Způsob psaní binárních čísel nevyjádří ani zlomek, ani záporná čísla .

S problémem zlomků se nebudeme zabývat,protože strojový kod většinou používá celá čísla, avšak způsob vyjádření záporného čísla je mnohem důležitější.

Metoda vyjádření záporného čísla v binární soustavě je relativně jednoduchá. Chceme-li kladné číslo vyjádřit jako záporné, provedeme dva úkony:

1. Přeměníme každou 0 na 1 a každou 1 na 0
2. k výsledku se připočte 1

Např. chceme vyjádřit -3. Kladné číslo 3D=0011B (4 bitové slovo). Postup je tedy následující :

```
   0011  ...binární vyjádření 3D
+  1100  ...změna bitů
-------
+  0001  ...připočtení 1
-------
   1101  ...výsledek -3D
```

Binární číslo 1101B představující -3D se nazývá "binárně komplementární" k binárnímu číslu 0011B. Přesvědčíme se, zda je to správné. Víme, že součet +3 a -3 musí dát 0. Tedy:

```
   0011  ...3D
+  1101  ...-3D
-------
   0000  ...výsledek
   1111  ...přenos
```

Neobdržíme tedy jen 0000, ale protože nejnižší bity jsou nuly a my pracujeme se 4 bitovým slovem, nejvyšší bit z přenosu jednoduše odpadá. Platí tedy pro součet jakéhokoliv čísla, pokud je pevně stanoven počet bitů. Proto nesmíme zapomenout předsadit v binárním čísle nuly k doplnění úplné délky slova před tím, než použijeme dvojkový komplet.

Z tabulky 4 bitových slov na následující straně, ve které jsou doplněna záporná čísla binární,je okamžitě patrné, že každé

slovo se vyskytuje dvakrát, např. číslo 1001B se rovná 2D, ale také -7D. Proto se musí rozsah platnosti omezit.

dekadické kladné	Binární	binárně komplementární	dekadické záporné
0	0000	0000	0
1	0001	1111	-1
2	0010	1110	-2
3	0011	1101	-3
4	0100	1100	-4
5	0101	1011	-5
6	0110	1010	-6
7	0111	1001	-7
8	1000	1000	-8
9	1001	0111	-9
10	1010	0110	-10
11	1011	0101	-11
12	1100	0100	-12
13	1101	0011	-13
14	1110	0010	-14
15	1111	0001	-15

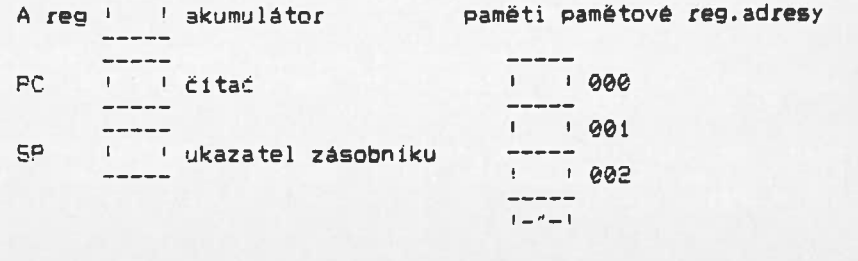
Rozsah platnosti je v tabulce ohraničen čárkovanou linií. V podstatě jsme 16 binárních čísel posunuli směrem do záporných čísel tak, aby byla v poloviné čísel nula.

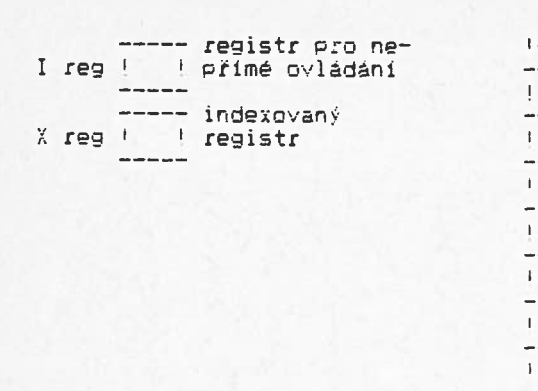
Rozsah čísel, která lze do čtyřbitového slova umístit, je tedy -8 až +7. Při 5ti bitovém slovu by to bylo -16 až +15, při 6ti bitovém -32 až +31, při 8mi bitovém slovu (t.j. náš případ) by dosáhlo hodnoty -32768 až +32767. V příloze číslo 1 je tabulka binárně komplementárních čísel pro 8mi bitová slova.

4. Architektura počítače

Pro snažší výklad činnosti mikroprocesoru zvolíme zjednodušenou verzi počítače se zjednodušeným mikroprocesorem, který je mikroprocesoru Z80 podobný. I ze zjednoušené formy získáme všechny potřebné znalosti o systému práce mikroprocesoru.

Vyjdeme z toho, že náš zjednodušený počítač má paměť pro 16ti bitová slova a určitý počet 16ti bitových speciálních registrů má pro speciální operace:





Podívejme se nejprve do paměti.

Každé paměťové místo v počítači má svou adresu, která je udána číslem, tak jako v našem případě od 000. Paměti jsme očíslovali hexadekadickými čísly. Co může být v pamětovém místě určeno ? Uspořádání 16ti bitového slova. Těchto 16 bitů může zaznamenat cokoliv. Kdybychom chtěli dát do paměti binárně komplementární číslo, bude paměť obsahovat slovo čísla v rozsahu -32768D až 32767D. Kdyby to mělo být celé kladné číslo, bude paměť obsahovat slovo čísla v rozsahu 0 až 65535D. Můžeme také slovo rozdělit do dvou polí po 8mi bitech, z nichž jeden může znamenat alfabetický, interpunkční, nebo grafický symbol.To z toho důvodu, že počítač ukládá 16ti bitové slovo do dvou paměťových míst po 8mi bitech.

Každých 8 bitů tvoří 1 byte (bajt). Jednotlivá paměťová místa jsou vlastně 8mi bitové registry schopné uschovat hodnoty od 00000000B do 11111111B. Maximální hodnota, kterou můžeme do jednoho byte uložit je tedy FFH=255D. Větší čísla se ukladají do dvou byte, v pořadí nižší byte, vyšší byte. U dvoubytového čísla 5FFF H je nižší byte FFH a vyšší 5FH. V tomto pořadí jsou také uloženy do paměťových míst A a A+1. Maximální hodnota čísla ve dvou bytech může činit FFFFH=65535D.

Nyní se obrátíme k registrům. Nejprve jenom k A registru. Tento registr je použitý, je-li prováděno počítání. Výsledek každého výpočtu, který od počítače žádáme, je dán do registru A. (Někdy jej nazýváme akumulator). Většina početních operací pracuje se dvěmi hodnotami. Nemá smysl dát počítači příkaz vypočti 3+. Musíme mu říci, kolik musí ke třem připočítat a tato hodnota musí být v registru ještě před prováděním sčítání. Napíšeme tedy příkaz ke sčítání: ADD (1A3)

Počítač to pochopí takto:

1. Přidej obsah paměti 1A3H k obsahu A registru. (Závorky jsou proto, že má být přičten obsah paměti 1A3 a ne číslo 1A3H.
2. Dej výsledek zpět do registru A.

Máme tedy první příkaz napsaný v úrovni strojového kódu. Podívejme se na jeho obecnou formu. Sestává z operačního kódu ADD a adresy (1A3).

4.1 Program sčítání

V BASICu vyjádříme sčítání následujícím sledem:

```
LET R=B+C
```

Jako první musíme přiřadit pro R, B a C konkrétní adresy. Řekněme, že to bude řada 103H, 104H, 105H. Obsah adresy 104H musíme dát do registru A. Tomuto úkonu dáme příkaz LD (load-naplnění):

```
LD (104)
```

a přičteme obsah adresy 105H

```
ADD (105)
```

Konečně ještě potřebujeme příkaz, kterým obsah v registru A předáme do paměti 103H. Určíme tedy příkaz k předání :

```
ST (103)
```

Nyní máme jednoduchý program v úrovni strojového kódu, který sestává ze tří povelů.

```
LD (104) vlož B do registru A
ADD (105) připočti C k obsahu registru A
ST (103) dej výsledek z registru A do B
```

Jakým způsobem zpracovává počítač příkazy ? Pamatujte si, že slovo uložené v paměti může mít jakýkoliv význam. Nejlépe můžeme uložit příkaz v pamětovém slovu. Z toho plyne, že operační kódy jsou dány sledem bitů. Založme si tedy tabulku operačních kódů vyjádřených 4 bitovým slovem :

operační kód	binární kód
ADD	0000
LD	0001
ST	0002

Každý další příkaz, který budeme probírat v tomto zjednodušeném počítači, zařadíme vždy do tabulky

Vycházíme z toho, že všechny binární kódy jsou 4 bitové . To připouští 16 různých povelů. Je to malý počet, ale vhodný pro náš zjednodušený počítač.

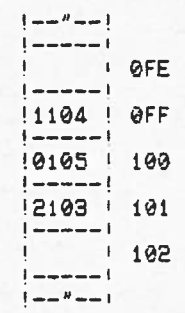
Avšak máme 16ti bitové slovo, takže 12 bitů zůstane pro adresovou část příkazu. LD (104) vyhlíží v počítači takto :

```
--------------------------------
!0!0!0!1!0!0!0!1!0!0!0!0!1!0!0!
--------------------------------
' oper. '----------v----------!
   kód     adresa      104H převedeno
           paměti      do binární soustavy
```

Víte-li jak jsou uspořádány bity, víte vše. Dále budeme psát příkazy v hexadekadickém tvaru.

4.2 Programový čítač

Řekněme, že náš program tří příkazů uložíme od odresy 0FF. Nyní potřebujeme pokyn, kterým bychom počítači řekli : "Začni počítat, vykonej příkaz na 0FFH, pak na 101H..." K tomu účelu nám poslouží PC registr (programový čítač). Dává počítači znamení ke čtení. Spustíme-li program, nasadíme PC na první příkaz. Zatím co počítač příkaz provádí, zvýší se čítač PC o 1, takže systém se vrátí k PC, provede další příkaz atd.



Takto postavený program však není úplný, protože počítač po provedení neví, jak dál. Pro tento účel můžeme použít příkaz :

JP 416 t.j. skoč na 416H

Jakmile je toto provedeno, je programový čítač na adrese 416H. Pak provede příští příkaz obsažený v paměťovém místě 416. Pak jde dále k 417, 418 atd. až k nejbližšímu povelu proveď skok (JP). Za příkazem JP může být jakákoliv adresa.

Tento příkaz se rovná spíše GOTO než IF...THEN. Co nutně potřebujeme, je příkaz, který programový čítač PC pošle na novou adresu jen tehdy, je-li splněna nějaká podmínka. Nejjednodušší podmínka je tato :

JPZ 2A7 - skoč na adresu 2A7H, jestliže A registr obsahuje 0.
JPN 14E - skoč na adresu 14EH, jestliže je obsah A registru záporný.

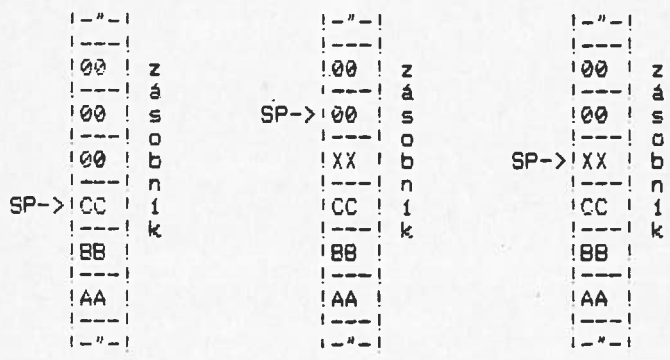
Toto je minimum, které si můžeme dovolit. Nyní můžeme provéřovat kladná čísla (obsah registru A není 0). Dáváme jen pozor na to, kdy program skočí buď na příkaz JPZ nebo na příkaz JPN.

5. Skoky a podprogramy

Dosav. přehled příkazů je skromný. Máme jen LD a ST, které působí v pamětech,ADD je vlastně jen sčítání a pro ukončení máme HLT.Můžeme ještě rozšířit počítání příkazem SUB,kterým odečítáme obsah paměťového místa od A registru, ale to je také vše. Žádné dělení,násobení ani umocňování. Co potřebujeme,je zásoba rozvětvovacích příkazů, odpovídajících v BASICu příkazům IF ... THEN.

5.1 Skoky

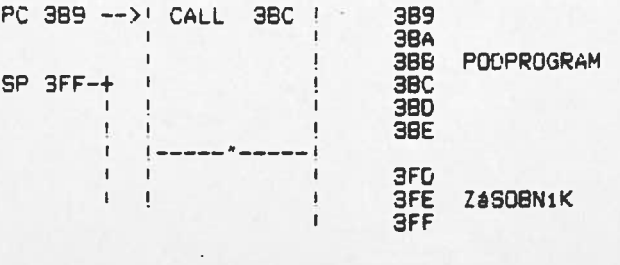
Z normální řady příkazů lze odbocit velice snadno. Musí se jenom pozměnit obsah progr. čítače PC. K tomu máme další příkaz, kterým můžeme uložit obsah registrů do zásobníku. Je to PUSH XX. Příkazem POP XX jej můžeme vrátit tam, odkud jsme jej vzali. Přitom se ukazatel zás. SP zmenší o 1. Vypadá to následovně:



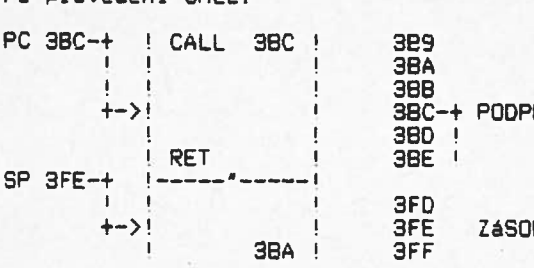
Ukazatel zás. SP ukazuje ve skut. na první nepoužité místo. Zás. si můžeme představit jako stoh knih na sobě. Příkazem PUSH říkáme "Dej knihu na stoh" a příkazem POP "Vezmi shora knihu". Máme-li např. řadu veličin X, Y a Z uchovat v zás. pomocí PUSH provedeme:

PUSH X a chceme-li je opět použít POP Z
PUSH Y musíme jej vyjmout příkazem POP Y
PUSH Z POP v obráceném pořadí POP X

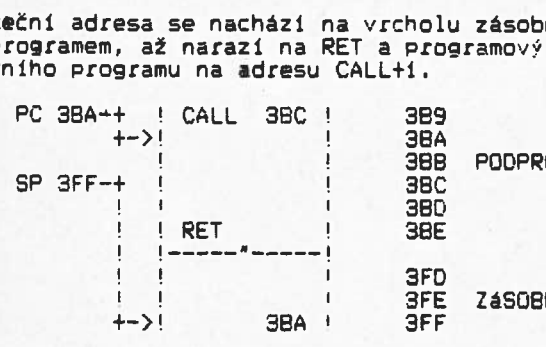
Podpr. používá ukazatel zás.k tomu,aby si zapamatoval zpát. adr. Při provedení CALL je dána co zás.zpáteční adr.(CALL+1). Objeví-li se RET, je adr.ukazatele zás.dána na programový čítač automaticky bez příkazu se stejným efektem jako příkaz POP. Příklad:



Po provedení CALL:



Zpáteční adresa se nachází na vrcholu zásobníku. Program projede podprogramem, až narazí na RET a programový čítač PC se vrátí do hlavního programu na adresu CALL+1.



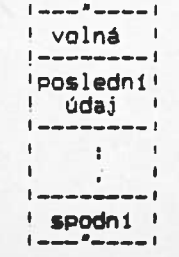
5.2 Podprogramy a zásobník

Jestliže je možné uvnitř programu řídit přenos z jednoho místa na druhé, proč by nemohl existovat příkaz na způsob GOSUB a RETURN jako v BASICu. Takový příkaz existuje:

CALL 205 vyvolá program na 205H

Příkaz CALL způsobí, že programový čítač skočí na adresu 205H. K tomu bychom mohli použít příkaz JP. Ale příkaz CALL má ještě druhou funkci. Ukládá adresu příkazu za CALL do zásobníku tak, že objeví-li se příkaz "RETURN" (správné operační kód RET), je uložená adresa dána do programového čítače, čímž program opět skočí do hlavního programu a pokračuje dalším příkazem.

Zásobník je úsek paměti s pevně určenou "dolní pamětí" a s proměnnou "horní". Ukazatel zásobníku SP obsahuje adresu horní paměti. Adresa příkazu za CALL je přesunuta do zásobníku do paměti, na kterou ukazuje ukazatel zásobníku SP. Po uložení se SP zvýší o 1, takže ukazuje na další volnou paměť v zásobníku.



Vedení příkazu v adrese 101H zvýšil PC o 1 a počítač najde adresu 102H, kde však žádný příkaz není. Ve 102H musí být příkaz k zastavení. Nazveme jej HLT (HALT=stát). Program pak zní takto:

```
 LD (104)
ADD (105)
 ST (103)
HLT
```

Důležitá upozornění:

Protože používáme slova, která mají při různých příležitostech různý význam, musíme sled pečlivě sestavit a určit počátek, neboť počítač jej postupně probírá tak, jak jsme mu jej vložili. Jestliže bychom začali s ADD, připočte obsah připojené paměti do registru A, neboť vychází z toho, že A registr nějaké číslo obsahuje. Počítač to pak nepřezkušuje - protože také nemůže - neboť každé uspořádání bitů může představovat číslo. Stejně tak však může každé uspořádání bitů představovat příkaz. Když tedy programový čítač ukáže na paměťové místo, bude jeho obsah proveden jako příkaz.

Pravidlo zní : Mějte data a program jasně rozdělené. Jestliže to neuděláte, musíte počítat s tím, že budete postaveni před neřešitelné hádanky. Také se vám může stát, že celý program během provádění beze stopy zmizí.

6. Nepřímé ovládání a indexované registry

Jak lze využívat adresy v paměti, aby ukázala na jinou paměťovou adresu. Zbývají ještě dva registry:
I registr pro nepřímé (indirektní) ovládání a index. registr X. Oba mají podobné funkce. Mohou měnit adresovanou část příkazu, zatím co běží program.

6.1 Nepřímé ovládání

Nejprve se podívejme na registr I. Stanovíme nový operační kód LDI (load direct). Stejně jako HLT není spojen s žádnou adresou. Pro počítač je stejný jako LD, jen nejvyšší bit adresového pole je dán na "1". Tento bit se nazývá "nepřímý příznak". Ukazuje počítači jedině to, že je v provozu nepřímé zařízení.
Binární forma příkazu LDI je tedy:

```
 ---------------------------------
 !0!0!0!1!1!0!0!0!0!0!0!0!0!0!0!0!
 ---------------------------------
operační!  adresa nepoužita
  kod   !
        +->nepřímý příznak
```

V hexadecimální soustavě je to číslo 1800.Narazí-li počítač na tento příkaz,použije číslo obsažené v registru I jako adresu. Např.obsahuje-li registr I číslo 1E4 a je proveden příkaz LDI,je jeho působení stejné, jako by příkaz zněl LD 1E4. Jinak řečeno registr I působí jako ukazovatel paměťových míst. S registrem I můžeme snadno pohybovat, což nám usnadní počítání. Víme, že jediné místo, kde se počítá,je registr A a proto si vytvoříme operační kód, který umožní výměnu mezi registry A a I. Bude to op. kód XAI, který provede "výměnu obsahu registru A za obsah registru I".

Samozřejmě můžeme příznak pro nepřímé řízení přidat ke každému příkazu, který má adresovou část. Můžeme tedy použít STI, JPI, ADDI atd. V každém případě budou v poslední části 3 čísla v hexakódu 800.

Příklad, ve kterém je tento způsob použit :

Dejme tomu, že chceme stanovit řadu čísel o počtu 20ti členů obsahující sudá čísla, t.j. 2, 4, 6,..., 40. Bude to tedy následující program v BASICu, který chceme vyjádřit ve strojovém kódu:

```
10 FOR C=1 TO 20
20 LET A(C)=C*2
30 NEXT C
```

V tomto programu se vyskytnou některé hodnoty, které musíme mít předem uloženy někde v paměti. Jsou to 1 (protože musí zvyšovat čítač smyčky o 1), pak 2 (protože je to zvyšování pro obsah řady) a konečně 20 jako test pro ukončení řady. Pro začátek se nebudeme starat o to, kde budou tato čísla uložena a dovolíme si adresy nazvat jménem. Musíme však tato jména změnit v čísla, jestliže dojdeme k vyjádření ve strojním kódu. Stanovme tedy, že čísla, které potřebujeme, jsou k dispozici na místech N1,N2 a N20. Mějme rovněž místo nazvané BAZE, které obsahuje adresu prvního příkazu řady a jméno COUNT, které slouží jako čítač smyčky . Nejprve nastavíme registr I na počátek řady, t.j.sledu:

```
LD BAZE
XAI
```

Pak dáme na adresu COUNT "1":

```
LD N1
ST COUNT
```

Přičtením hodnoty COUNT zdvojíme velikost v registru A a uložíme do místa, které ukazuje registr I. Nazývá se to "ukládání dat registrem I".

```
ADD COUNT
STI
```

Odečteme z registru A hodnotu COUNT a porovnáme registr A s N20. Bude-li výsledek 0 ukončí se proces takto:

```
SUB COUNT
SUB N20
JPZ OUT
```

OUT je další neurčená adresa. Nevíme ještě kde je a proto ji dáme předběžné jméno OUT.
Jestliže není výsledek po porovnání 0 pak k COUNT přidáme 1:

```
        LD COUNT
        ADD N1
        ST COUNT   a zvýšíme registr I o 1:
        XAI
        ADD N1
        XAI
```

Běžné COUNT je nyní také v registru A, takže se lze smyčkou vrátit ke zdvojení:

```
        JP SMYČKA  a tím jsme dali výrazu ADD COUNT
                   symbolickou adresu "SMYČKA". Tomuto
                   příkazu představíme symbolickou adresu
                   následovanou dvojtečkou:
SMYČKA: ADD COUNT.
```

Podobně to můžeme provést při přidělení čísel adresám, které jsme nazvali jmény. Definujeme proto nový op. kód HEX, který slovo na požadovanou hodnotu vrátí. HEX není vlastně op. kód, ale tzv. pseudoinstrukce. Celý program vypdá následovně (prvního a posledního sloupce si zatím nevšímejte):

```
020             LD   BAZE                1 033
021             XAI                      A 000
022             LD   N1                  1 030
023             ST   COUNT               2 032
024  SMYCKA:    ADD  COUNT               0 032
025             STI                      2 800
026             SUB  COUNT               4 032
027             SUB  N20                 4 031
028             JPZ  OUT                 6 047
029             LD   COUNT               1 032
02A             ADD  N1                  0 030
02B             ST   COUNT               2 032
02C             XAI                      A 000
02D             ADD  N1                  0 030
02E             XAI                      A 000
02F             JP   SMYCKA              5 024
030  N1:        HEX  0001                0 001
031  N20:       HEX  0014                0 014
032  COUNT:     HEX  0000                0 000
033  BAZE:      HEX  0000                0 000
```

Jediná nespecifikovaná adresa v druhém sloupci je OUT. Přijde na řadu později.
Forma programu, který jsme napsali se jmenuje assembler. Nyní rozšíříme tabulku op. kódů o další, které jsme doposud užili.

Operační kód	Hexadekadický kód	Binární kód
ADD	0	0000
LD	1	0001
ST	2	0010
HLT	3	0011

SUB	4	0100
JP	5	0101
JPZ	6	0110
JPN	7	0111
CALL	8	1000
RET	9	1001
XAI	A	1010

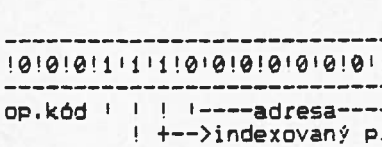
Přidělili jsme op. kódům odpovídající hodnoty v hexadekadickém číslování. Musíme také znát, kde je začátek programu. Určit, na kterou adresu případne začátek programu je více méně libovolné rozhodnutí. My jsme si stanovili, že má být na adrese 020. Protože každý příkaz obsadí jedno slovo, můžeme zachytit adresu každého příkazu písemně. To jsme udělali v prvním sloupci. Dále můžeme nahradit op. kódy a adresy hexadekadickým číslem. Např. LD BAZE číslem 1033, protože LD je identifikováno jako 1 a BAZE jako 033. V prvním sloupci je úplný kód.

Jediný příkaz, který potřebuje komentář je JPZ OUT, kódovaný jako 6047. OUT má adresu 047 proto, že je to první adresa kde může být.

6.2 Indexovaný registr

Při použití registru X je vytvořena pravá adresa tím, že se na adresové pole k obsahu registru I přičítá. Např. když X registr obsahuje 400, má příkaz LDX 005 stejnou působnost jako LD 405. Do adresového pole se nám vkládá ještě jeden bit, který ukazuje, že je v činnosti indexovaný registr.

Příkaz LDX 005 vypadá tedy takto :



V hexadekadické soustavě je to 405. Při indexování vlastně není nic, co by jste nemohli provést nepřímým řízením. Indexování způsobí jen to, že manipulace s adresací je automatická.

7. Mikroprocesor Z-80

Pokud jste prostudovali předchozí stránky a skutečne jste všemu porozuměli, můžete být ujištěni, že porozumíte i mikroprocesoru Z80. Mikroprocesor Z80 je srdce, lépe řečeno mozek vašeho ZX Spectra.

7.1 Architektura Z-80

Predtím, než probereme architekturu mikroprocesoru Z80, si musíme uvědomit několik skutečností:
1. oper. kódy se 4 bitovými slovy dovolí jen 10 povelů. Mikroprocesor Z80 má však 694 příkazů. Aby měl každý příkaz své uspořádání bitů, potřebujeme 8mi bitové slovo (1 byte) a pak vyžaduje použití ještě některých dalších zkrácení.
2. Ve zjednodušeném počítači jsme používali paměť bezstarostně. Mnohé adresy jako HLT, LDI, STI nepotřebují adr. pole, takže jsme u každého takového příkazu promarnili 10 bitů. Mikroprocesor Z80 to řeší různými délkami příkazů. Mnohé příkazy nemají žádné adr. pole a jsou dlouhé 1 byte, jiné mají adresová pole dlouhá 2 byte a tím jsou vlastně dvoubytová. Jsou ale také 3 bytová .
3. Zpracovávat se musí i slova 16ti bitová, což je nepříjemné, manipuluje-li se se znaky, které obsahují 1 byte.
4. často se musíme spokojit s jedním registrem pro více účelů (A registr). Často se stává, že musíme přechodně uschovat mezivýsledky v paměti, zatímco běží další výpočet. Z80 má řadu registrů použitelných pro všechny účely, avšak závisí to na jejich užití.

7.2 Registry a jejich organizace v Z-80

8 bit	8 bit	8 bit	8 bit		16 bit
A	F	A'	F'	registry	IX
B	C	B'	C'	pro	IY
D	E	D'	E'	všechny	SP
H	L	H'	L'	účely	PC
hlavní část		náhradní část			

Náhradní části registrů si zatím nebudeme všímat. Registry můžeme používat jednotlivě, t.j. jako 8mi bitové nebo v párech jako 16ti bitové. Prvním způsobem mohou být využity registry B,C,D,E,H a L, ale jako párové výhradně BC,DE a HL. Registry A a F jsou 8mi bitové a nemohou být slučovány. V 16ti bitových párech má vyšší platnost levý byte (B,D a H). Existují dva indexované registry IX a IY pro ovládání zásobníku a PC (programového ukazatele). Pro nepřímé ovládání může být použit kterýkoliv 16ti bitový registr (BC,DE a HL). Existují dvě sady povelů (příkazů). Jedna pro 8mi bitové operace a jedna pro 16ti bitové. Začneme s 8mi bitovými povely pro "ukládání".

8. Způsoby adresování a LD příkazy

Existuje 5 různých způsobů jak dostat v Z80 příkazy na určenou adresu. Nejsnadnější je ten, kterým můžeme přemístit data z jednoho místa na druhé pomocí příkazu LD (load). Jako příklad 8mi bitové skupiny příkazů jsou právě příkazy pro ukládání LD. Blíží se povelu LD ze zjednodušeného počítače. Kromě přímého, nepřímého a indexovaného adresování jsou ještě dva další a to z registru do registru a poslední, t.zv. bezprostřední ukládání dat.

8.1 Přímé adresování

Je to podobné jako jsme to dělali v předchozí části, jenže nyní musíme udat o který registr se jedná:

```
LD A,(0F1C)
```

ukládá obsah adresy 0F1C do registru A. Všimněte si, že se vždy postupuje zprava do leva, takže napíšeme-li

```
LD (0F1C),A
```

je tím míněno "okopíruj obsah registru A do paměti na adrese 0F1C". Ve skutečnosti je 8mi bitový registr A jediný, který může být adresován přímo.

8.2 Nepřímé adresování

Také toto je normální postup.Chcete-li něco pro nepřímé řízení v registru HL bude příkaz znít

```
LD A,(HL)
```

To znamená "ulož do registru A prostřednictvím registru HL" (to jest to, co je obsažené na adrese v HL registru). Pro opačný příkaz použijeme :

```
LD (HL),a
```

t.j. dej obsah registru A do adresy obsažené v registru HL. (Pro tento příkaz jsou přípustné i jiné registry než A).

8.3 Indexované adresování

V tomto případě se musí uvést, který indexovaný registr se použije a rozsah jeho nasazení t.j. index posunu:

```
LD A,(1A,2E)
```

Všimněte si, že při přímem adresování jsme udávali adresu čtyřmi čísly v hexadekadické soustavě, protože jimi vyjadřujeme 16ti bitovou adresu. Index posunu v příkazu s indexovanou adresou musí být obsažen v jednom bytu, takže je použito jen dvou hexadekadických čísel.

8.4 Z registru do registru

Mezi registry lze data přenášet takto

```
LD D,B
```

což znamená 'ulož obsah . B do D'

8.5 Bezprostřední ukládání

V tomto případě na místo obsahu adresy ukládáme přímo samotná data, takže :

```
LD B,07
```

má význam : "dej číslo 7 do registru B". Všiměte si, že je zde použito dvoumístné hexadekadické číslo, protože hodnota je ukládána do 8mi bitového registru.

Za zmínku stojí, že příkaz "LD" je ve skutečnosti jen "okopírování". Číslo uložené na adrese nebo registru se jenom okopíruje.

8.6 Hexakódy

Podívejte se jak vypadají jednotlivé příkazy ve vyjádření v hexadekadické soustavě. Kompletní přehled je pak v příloze č. 6.

8.6.1 LD A,(071C)

Nejprve vyhledáme operační kód pro LD A,(nn), přičemž nn je na libovolné adrese. Je to kód 3A. Vezmeme-li jej v úvahu, je příkaz kódován:

```
3A 071C
```

Bohužel,je tu malá komplikace způsobená zpracováním čísel v Z80. Na prvním místě musí být totiž nejméně významný byt v adrese. Musíme tedy adresové byty vyměnit :

```
3A 1C0F
```

Je to bezpodmínečné pravidlo pro dvoubytové číslo v příkazech mikroprocesoru Z80 :

Nejprve nižší byt a pak vyšší

Příkaz LD (nn), A má op. kód 32, takže bude :

```
LD (071C), A kódován kódem 32 1C 0F
```

8.6.2 LD A, (HL)

Toto je jednoduché, neboť v příkazu není adresná část, takže se jedná o jednobytový op. kód a to 7E. Podobný pro LD (HL), A je kód 77.

8.6.3 LD A, (IX+2E)

Příkaz obecně zní : LD A, (IX+d), přičemž d je dvoubytové posunutí a op.kód je DD7E. Příkaz zní DD 7E 2E, přicemž je 2E v tomto případě ono zvolené posunutí.

8.6.4 LD D, B

Zde není žádný problém, op. kód je 50.

8.6.5 LD B, 07

Op. kód je 06, takže příkaz správně zní 06 07.

9. Ukládání, provozování a zajišťování stroj. kódu

Hlavní cíl této kapitoly je vytvoření jednoduchého programu v BASICu, který umožní vkládání a užití str. kódu. Dokonalejší verze takového programu je obsažena v příloze 7 této příručky.

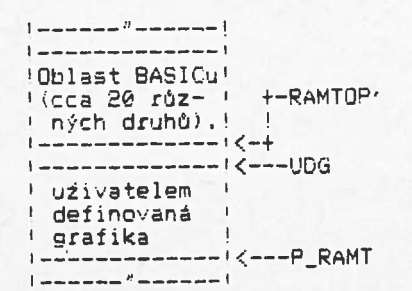
Začneme nejprve paměťmi ZX Spectra. Spectrum má dva druhy pamětí : pevnou, neměnnou, z které můžeme informace jen číst, ale nemůžeme je měnit. Je to paměť ROM (Read Only Memory). Z této paměti přečteme informaci povelem PEEK. Druhá operační paměť, do které můžeme ukládat data, příkazy a kterou můžeme číst i mazat, nazýváme pamětí RAM (Random Acces Memory).

Do RAM se ukládá a je v ní prováděn program ve strojovém kódu. Podívejme se na adresy obou pamětí :

typ	startovací adresa dec	hex	koncová adresa dec	hex	
ROM	0	0	16383	3FFF	
RAM	16384	4000	32767	7FFF	(16K)
			65535	FFFF	(48K)

V paměti RAM může nastat kolize, neboť je v ní ukládán i program v BASICu. Máme-li program ve strojovém kódu uložen na libovolném místě v RAM, vkládaný program v BASICu jej jednoduše přepíše, smaže, nebo nějak jinak zničí. Musíme proto program ve strojovém kódu umísťovat v RAM na místo, které není BASICem upotřebeno, nebo které nemůže měnit.

Vhodné místo v RAM je na jejím konci, kde to vypadá takto :



RAMTOP má adresu 23730 a 23731. UDG 23675 a 23676 a P_RAMT má adresu 23732 a 23733. Jsou to dvoubytové proměnné a hodnoty např. RAMTOP nalezneme, zadáme-li PRINT PEEK 23730+256*PEEK 23731. Zapneme-li počítač poprvé, jsou nastaveny stand. hodnoty:

Proměnná	16K		48K	
	dec	hex	dec	hex
RAMTOP	32599	7F57	65367	FF57
UDG	32600	7F58	65368	FF58
P_RAMT	32767	7FFF	65535	FFFF

Programy v BASICu obsahují spodní část včetně adresy RAMTOP. Od části UDG až k P_RAMT se ukládají data pro užitou grafiku. Tato oblast může být zmenšena nebo zvětšena, jestliže se změní hodnota UDG. Do části UDG BASIC nezasahuje. Zvětšením hranic UDG si vytváříme chráněné místo pro náš strojový program. Místo pro strojový kód vytvoříme, přesadíme-li hodnotu RAMTOP. Např. abychom udělali místo pro 600 bytů a začali jsme se strojovým kódem na adrese 32000, musíme RAMTOP změnit na 32000-1=31999. Toto vytvoří více než potřebné místo pro všechno, o čem se bude v této práci pojednávat a startovací adresa pro strojový kod bude 32000D a 7000H. Zvolili jsme si zaokrouhlená místa (čísla). U verze 48K použitím 32000D promarníte 32Kb, takže by bylo lépe volit 65000, t.j. FA00H.

Abychom přesadili RAMTOP, použijeme příkaz :

CLEAR 31999

Příkaz má následující působnost :

-maže všechny proměnné
-maže display jako CLS
-přesazuje PLOT do levého dolního rohu (0,0)
-zavádí příkazem RESTORE ukazatel dat zpět k počátku
-přesazuje RAMTOP na 31999 a jistí adresu 32000 a výše proti porušení Basicem
-zavádí nový GOSUB ukazatel pod tuto novou hodnotu RAMTOPu

Příkaz CLEAR 31447 může být rovněž použit a RAMTOP bude usazen na 31447D, takže budeme mít více místa. Adresy budou tak zaplňovány od 31446D. Nejvhodnější je zvykat si na příkaz CLEAR. Při zakládání dalšího programu po NEW nebo RANDOMIZE USR 0 nebo po přerušení napájení je RAMTOP usazen na začátek, t.j. na standardní hodnotu.

Důležité upozornění:

Chcete-li najít obsah od adresy "xxxx", použijte CLEAR xxxx-1. Nikoliv pouze CLEAR xxxx. Příkaz CLEAR reaguje na poslední nezajištěnou adresu a nikoliv na první zajištěnou.

9.1 Ukládání strojového kódu

Co potřebujeme,je program, který hexakódy a příslušné byty uloží nad RAMTOP. Později jej ještě v této kapitole doplníme. Tento program nazveme "MC 1".

```
10 CLEAR 31999
20 PRINT "ADRESA BASE:32000"
30 PRINT "KOLIK BYTE PRO DATA:";
40 INPUT D:PRINT D
50 LET B=32000
60 FOR I=0 TO D
70 POKE B+I,0
80 NEXT I
90 LET A=B+D
100 DIM H$(2)
110 PRINT "CODE:"
120 INPUT C$
130 IF C$="A" THEN GOTO 200
140 PRINT C$+" ";
150 LET HS=CODE C$(1)-48-39*(C$(1) "9")
160 LET HJ=CODE C$(2)-48-39*(C$(2) "9")
170 POKE A,16*HS+HJ
180 LET A=A+1
190 GOTO 120
200 POKE A,201
```

Cílem řádky 39 je rezervování požadovaného počtu bytů před strojovým programem, což je někdy účelné. Řádky 60 až 80 naplní rezervovaný prostor nulami. Vlastní data můžeme do takto rezervovaných bytů jenom vložit, pokud je to třeba, pomocí POKE. Řádky 150 a 160 převádí kód hexa na dvě dekadická čísla mezi 0 a 15 tak, že 16*HS+HJ dává skutečné vyjádření hexadekadického kódu v dekadickém. Řádka 170 je ukládá pomocí POKE na správné adresy. Vstup "a" není hexadekadický a slouží jako ukončení a dává počítači na vědomí, že již nebudou následovat žádné kódy. Řádka 200 se stará o to, aby poslední příkaz byl RET, což je skok do BASICu. Příkaz RET má hexadekadický kód C9. Jak je v manuálu k ZX Spectru zdůrazněno, je důležité končit tímto příkazem. Program jej nasazuje automaticky, ale nic se nestane, zadáte-li jej i Vy a bude tak v programu za sebou dvakrát.

9.2 Spouštění str. programu

Vložený kód se spouští příkazem USR. USR je funkce, jejímž argumentem je adresa, na které vložený strojový kód začíná. Začíná-li strojový program na adrese 32000, potřebujeme příkaz LET y=USR 32000. Přitom nás skutečná hodnota y nezajímá. Každý příkaz obsahující USR 32000 a jehož syntaxe budou správné, rozběhne program. Např:

```
RANDOMIZE USR 32000   nebo
RESTORE USR 32000 .
```

Je-li na počátku programu reservován počet bytů d, musí být počátek programu posunut od 32000 o d, aby se zamezilo tomu, že se tato část stane součástí programu. S následujícím rozšířením programu "MC 1", k tomu přizpůsobeným, můžeme provozovat strojový kód. Je nakonec vybaven i volbou pro další rozšíření :

```
300 INPUT "VOLBA ?";O$
310 IF O$="R" THEN GOSUB 400
399 STOP
400 LET Y=USR(B+D)
410 RETURN
```

9.3. Zajišťování stroj. kódu

Program ve strojovém kódu se může zajistit pomocí SAVE, použijeme-li způsob ukládání podle manuálu ze strany 144. Dejme tomu, že má sled 77 bytů, včetně posledního RET. Můžeme je tedy zajistit pod názvem : "m-code" příkazem SAVE "m-code" CODE 32000, 77 kde 32000 je první startovací byte a 77 je délka. S délkou si nemusíme dělat starosti, můžeme třeba napsat 600 (hodnotu kterou jsme uvolnili posunutím RAMTOP). Magnetofonovému pásku to je jedno, jenom to bude trvat déle.

Stejně jednoduché je i opačné nahrání do počítače, příkazem LOAD. Postačí příkaz LOAD "m-code" CODE, nebo zapomeneme-li název, jen LOAD "" CODE. Oba způsoby zabudujeme do našeho programu jako volbu:

```
320 IF O$="S" THEN GOSUB 500
330 IF O$="1" THEN GOSUB 600
500 INPUT "JMENO PRO SAVE ?";N$
510 IF N$="" THEN LET N$="m-code"
520 SAVE N$ CODE 8,600
530 RETURN
600 INPUT "JMENO PRO LOAD ?";N$
610 LOAD N$ CODE
620 RETURN
```

9.4 Přezkoušení a zobrazení

Poslední rozšíření k programu "MC 1" umožní přezkoušet sled programu, eventuelně jej vytisknout, máme-li tiskárnu. Je-li v listingu chyba, korigujeme ji přímo na adrese pomocí POKE. Program v příloze 7 to provádí jiným způsobem.

```
340 IF O$="Z" THEN GOSUB 700
350 IF O$="P" THEN GOSUB 700
700 IF O$="P" THEN PRINT:IF O$="Z"THEN PRINT
705 FOR I=8 TO A
710 LET J=PEEK I:LET JS=INT(J/16):LET JJ=J-16*JS
720 LET H$(1)=CHR$(J$+48+7*(JS>9))
730 LET H$(2)=CHR$(JJ+48+7*(JJ>9))
740 IF O$="P" THEN PRINT I;" "+H$+" ";J
750 IF O$="Z" THEN LPRINT I;" "+H$+" ";J
760 NEXT I
770 RETURN
```

Program ukazuje na displeji a tiskne na tiskárně adresy jak v dekadické tak hexadekad. formě. Všimněte si, že při zadávání kódu v hexadekad. tvaru jsou pro 10 až 15 používána malá písmena, ale po volbě "P" se ukazují velká. Chcete-li malá, musíte změnit v řádku 720 a 730 číslo 7 za 39. Abychom měli možnost volby přezkoušení před i po spuštění programu, zařadíme :

```
370 GOTO 300
```

Pro zastavení programu vložíme ještě :

```
360 IF O$="A" THEN STOP
```

Možnosti volby tedy jsou :

```
A-STOP
S-SAVE
L-LOAD
P-PRINT (NA DISPLEJ)
R-RUN (STROJOVÝ KÓD)
Z-COPY (NA TISKÁRNU)
```

Další část knihy vychází již z toho, že tento program máte nahraný na magnetofonovém pásku a pro další probírané pokyny jej budete používat.

10. Aritmetika

Na počátku probereme několik postupů pro sčítání a odčítání. Klíčem k počítání je existence příkazů ADD a SUB, které používají registru A a mimo přímého adresování může být použito jakéhokoliv způsobu adresování. Napišme program, který má sečíst čísla 4 a 7. Jedno číslo musíme vložit do registru A, druhé do registru B, sečíst je a výsledek někam uložit. Použijeme-li pro výsledek adresu 32000D, musíme rezervovat i byte pro data (v dotazu "kolik byte pro data ?"). Nejdříve tedy mnemoniku :

```
          4 dáti do registru A      LD A,04
          7 dáti do registru B      LD B,07
sečíst  při čemž suma zůstane v registru A
                                    ADD A,B
          uložit výsledek           LD (7D00),A
```

v příloze k tomu vyhledáme operační kódy a obdržíme :

```
 LD A,04      3E 04
 LD B,07      06 07
ADD A,B       80
 LD (7D00),A  32 00 7D
```

Všimněte si, jak se přehodí v hexadek. kódu 7D00 na 007D

Program, který začíná u 7D03H vypadá tedy takto :

```
první číslo uložit do reg.A     LD A,(7D00)
druhé číslo uložit do reg.B     LD B,(7D01)
obě čísla sečíst               ADD A,B
součet dát do 32000             LD (7D02),A
```

Protože v příloze č.6 není hexakód pro LD B,(nn), musíme to obejít. Jedna z cest vede přes registr HL, takže napíšeme místo LD B, (7D01) toto :

```
do reg. HL uložit adresu      LD HL,7D01
do reg. B uložit obs.reg      LD B,(HL)
```

Program má pak tvar :

```
LD A,(7D00)  3A 00 7 D
LD HL,7D01   21 01 7 D
LD B,(HL)    46
ADD A,B      90
LD (7DD2),A  32 02 7 D
```

Zkuste uložit program, při čemž rezervujete 3 byty a pak program stopněte volbou "A", aby jste mohli zadat data.
Vložte :

```
POKE 32000,44 : POKE 32001,33
```

chcete-li sečíst 44 a 33. Pak zadejte GOTO, aby jste se vrátili do programu "MC 1" a volbou "R" spusťte program. Pak zadejte "P" a obdržíte následující výpis, ve kterém bude na prvních třech místech :

```
32000  2C  44
32001  21  33
32002  4D  77
```

A pak následuje další část programu. Výsledek 77 bude na adrese 32002. Změňte pomocí POKE vložená data a podívejte se na výsledek. Především ale zkuste :

```
POKE 32000,240 : POKE 32001,100
```

Uvidíte, že počítač je přesvědčen, že 240+100=84. Bude-li se Vám to zdát zvláštní, máte pravdu, ale vinu nenese počítač. Příkaz ADD zapomněl na přenos. Představte si to v binárním kódu :

```
240...11110000
100..+01100100
--------------
   +->01010100 B=84D -přenos
   1  11              !
                      <-+
```

Nyní můžeme program uložit. Nahrajte program "MC 1" a uveďte jej do činnosti pomocí RUN. Na žádost : "KOLIK BYTů PRO DATA ?" vložte 1. Potom vložte postupně řadu hexadek. kódů ve tvaru :

```
3e 04 06 07 80 32 00 Td
```

a zakončete písmenem "S" jako ukončení. Objeví se "volba ?" a Vy stiskněte "R" pro RUN ve strojovém kódu. Výsledek se na displeji neobjeví. Uvidíte jej však na adrese 32000 po volbě "P":

```
32000  0B  11
32001  3E  62
32002  04   4
32003  06   6
32004  07   7
32005  80 128
```

```
32006  32  50
32007  00   0
32008  7D 125
32009  09 201
```

Počátek programu vidíte na adrese 32001D a také ukončující RET-C9. Výsledek sčítání je na adrese 32000, kde jsme jej chtěli mít. Nežli půjdeme dále, napíšme program ve strojovém kódu pro výpočet

18+66 13+17 23+6

Použijte tentýž program, ale změňte čísla 04 a 07. Výsledek je na adrese 32000.

10.1 Lepší využití rezervovaných dat

Nechceme-li pro každý nový pár čísel psát nový program, změníme čísla na proměnné, které uložíme takto:

```
POKE 32000,m : POKE 32001,n
```

Výsledek uložíme na adresu 32000D. Tím ovšem narušíme počátek předcházejícího sledu. Výhodnější je stanovit čísla tak, že se přičítají jako data. Uloží se jako část programu do registrů, sečtou se a uloží výsledek v datech. Rezervujeme proto 3 datové byte : první číslo 32000D-7D00H, druhé číslo 32001D-7d01H a výsledek : 32002D-7D02H. Součet dává 9tý bit, který nemůže být v 8mi bitovém byte postižen. Zapadl někam dozadu a výsledek je devátý, t.j. o 256 menší. Správné řešení je 256+84=340. Neobjevilo se žádné hlášení, přezkušování nemá úspěch. Co počítač napíše, je hotovou věcí. Co netestuje, nemůže se dozvědět. Existuje metoda co s přenosem nad 8 bitů-viz později (příznaky).

10.2 Odčítání

Abychom mohli odečíst druhé číslo od prvního, změníme následovně program : Místo ADD A,B 80 použijeme jen
SUB A,B 90
Proveďte to, uvidíte že 44-33=11 a stanovte, jak bude působit přenos a co obdržíte (např. : 17-99=?)

11. Výběr příkazů ze Z-80

V následujících dvou tabulkách je uveden výběr asi 30ti příkazových způsobů, které obsáhnou asi 230 konkrétních příkazů. První tabulka je pro 8mi bitové operace, druhá pro 16ti bitové.

* OP.d * KOD * ZPŮS.* ADRES. *	LD	ADD ADC SUB SBC AND OR XOR CP	INC DEC SLA SRA SRL	JR JRC JRNC JRZ JRNZ DJNZ

Z REG DO REG	LD r,s	ADD A,r	INC r	
BEZPROSTŘEDNE	LD r,n			
PŘÍMO	LD A,(nn) LD (nn),A			
NEPŘÍMO	LD A,(HL) LD (HL),A	ADD A,(HL)	INC (HL)	
INDEXOVANÉ	LD A,(IY+d) LD (IY+d),A	ADD A,(IY+d)	INC (IY+d)	JR d

Bohužel nemůžeme využít všechny druhy adresování pro všechny příkazy. Tabulka ukazuje v kterých případech adresování využijete.
Pro 16ti bitové operace platí tato tabulka:

* op. * kód * * * způs.* adres.*	JP	JPZ JPMZ JPC JPNC JPP JPM	LD	ADD ADC SBC	INC DEC PUSH POP
Z REG.DO REG.				ADD HL,r	INC r
Z REG.DO REG.BEZPROSTŘED	JP nn	JPZ nn	LD r,nn		
PŘÍMO			LD HL,(nn) LD (nn),HL		
NEPŘÍMD	JP (HL)				
INDEXOVANÉ					

Vysvětlení k tabulce:

1. "r" nebo "s" je míněn libovolný registr. Zda je 8mi nebo 16ti bitový závisí na tom, ve které tabulce se nalézá. Příklad : V příkazu LD r,s jsou r a s kterýkoliv 8mi bitový registr (A,B,C,D,E,H nebo L), ale v příkazu ADD HL,r je pár registrů, t.j. BC,DE,HL, a AF.
2. "n" je každé 8mi bitové číslo, "nn"každé 16ti bitové.
3. Jestliže registr (např.LD A,(nn)) je výslovně uveden, jedná se o jediný registr, který lze pro tento účel použít.
4. "d" je 8mi bitové číslo indexovaného posunu.

Bližší pohled na některé nové operační kódy

AND - Tato operace bere obsah registru a srovnává jej bit po bitu s jiným 8mi bitovým výrazem. V případě a jen v tom případě, že ve stejném odpovídajícím pořadí jsou "1", vrátí se na tom místě do registru A jednička zpět, pro jiné případy se objeví na stejném místě v registru A nula.
Příklad : příkaz AND A,07 má následující působení:

A reg před příkazem	00110101
07 reg	00000111
A reg po prov.příkazu	00000101

Vidíte, jak byly poslední tři nejnižší bity přeneseny ? Můžeme tedy AND použít k vyvolání části byte.

OR - Je podobný příkazu AND, avšak bit je označen "1", když se jednička vyskytne nejen v obou výrazech, ale i v jednom z nich. Např. : OR A,05 je :

A reg před příkazem	01001011
05	00000101
A reg po prov.příkazu	01001111

Zde je porovnávaný bit bez ohledu na jeho původní hodnotu dosazen na hodnotu "1"

XOR - Zde musí být hodnota porovnávaných bitů různá, aby byl výsledek "1". Např. XOR A,B3 je :

A registr před příkazem	01011010
B3	10110011
A reg po prov.příkazu	11101001

Vhodné je,můžeme-li registr přepnout z 0 na 1 a naopak.Obsahuje-li registr A na počátku 0,bude vždy při provedení přík. XOR A,01 změněna hodnota v registru A (0 na 1.zpět na 0 a zpět na 1 atd.)

CP - Je to příkaz "porovnej" (compare). Obsah A registru bude porovnáván s jiným 8mi bitovým výrazem. Vyvolá to problém jak ukázat výsledek porovnání. V tomto případě je vhodný registr F obsahující tzv.příznaky (flagy). Každý bit F registru obsahuje informaci o působení posledního příkazu, který jej měnil (ne každý příkaz mění příznak). Příznaky, které nás nejvíce zajímají, jsou tzv. přenosové příznaky, nulové, přeplňovací a znaménkový příznak. CP může každý z nich změnit, avšak největší význam má nulový příznak, který je nasazen tehdy, jsou-li obě porovnávané hodnoty stejné. Je-li obsah A registru menší, než ten s kterým je porovnáván, bude nasazen znaménkový příznak. Je to totéž jako : "výsledek je negativní". V tomto okamžiku nemusíte více vědět o příznacích. Vrátíme se k nim v příloze č.5. a kapitole 16.

11.1 Skoky

Všechny podmíněné skoky se provedou nebo neprovedou podle obsahu příznaku. Tak např. JPZ znamená, "skoč je-li nastaven nulový příznak". Není-li nastaven, neskočí. Podívejme se, jak se použije příkaz CP :

Stanovíme třeba, že chceme vědět, zda určitý byte, který je na registru HL, obsahuje číslo 1EH. Jestliže tento případ nastane,nechť nastoupí odbočení na adresu 4478H. Kód bude znít:

```
LD A,1E   3E 1E
CP A,(HL) BE
JPZ 4478  CA 78 44
```

Ostatní skoky se chovají obdobně :

JPNZ značí: "skoč, když výsledek není roven nule"- nulový příznak není nasazen.

JPC značí: "skoč, je-li výsledek pozitivní" - znaménkový příznak není nasazen.

JPM značí: "skoč, je-li výsledek negativní" - znaménkový příznak je nasazen.

JPNC značí: "skoč, není-li žádný přenos" - přenosový příznak není nasazen

Všechny skoky mají společné to, že mají stanovenou adresu skoku, t.j. na kterou adresu má skočit příkaz.

Chceme-li z jakéhokoliv důvodu provádět program někde jinde než kam směřuje, t.j. v jiných paměťových adresách, než jsme jej původně uložili, musíme změnit i skokové adresy. Z80 to zvládá pomocí "relativních skoků" - JR. Jinak řečeno, můžete odtud, kde se nacházíte, skočit dopředu (nebo dozadu) o stanovený počet bytů. Toto posunutí je obsaženo v jednom byte, takže vzdálenost, která může být přeskočena, je maximálně 128 byte dopředu, nebo 127 byte dozadu.

Posunutí se počítá od adresy příštího příkazu v programu, kam by programový ukazatel přišel, kdyby nebyl prováděn žádný skok.

		- 128	80
		- 3	FD
18		- 2	FE
?		- 1	FF
///		0	00
		1	01
		2	02
		3	03
		.	
		.	
		127	7F
velikost posunu		Dvojkově komplementární	hexadek.kód

Příklad : Chceme každý byte postupně prověřit, až se najde poprvé 1 EH. Nyní předpokládejme,že startovací adresa je již v HL. Píšeme :

```
LD A,1E
CP A,HL  smyčka
INC HL
JRNZ     smyčka
```

Máme zde nový příkaz INC,což je zkrácený název příkazu INCrement = zvýšení o 1. K obsahu daného registru je jednoduše přičtena 1, takže porovnávací byte v následující paměti porovnává byt v následující paměti, protože registr HL je při každé smyčce zvednut o 1. (Existuje také příkaz DECrement - zmenšení o 1).

Není zřejmý žádný rozdíl mezi smyčkou JRNZ a JPNZ. Teprve když příkaz zařadíme do sledu, bude rozdíl patrný.

adresa	příkaz	hexadek.kód
7D00	LD A,1E	3E 1E
7D02 smyčka	CP A,HL	BE
7D03	INC HL	23
7D04	JRNZ smyčka	20 FC

Program jsme ukládali od adresy 7D00, jak jsme zvykli z "MC 1". Zajíma nás však, proč je v adresové části příkazu JNRZ údaj FC ? Vysvětlení je toto : Při provádění příkazu JRNZ je programový ukazatel PC zvýšen o dva byty, protože se v příkazu jedná o dva byty. Ukazatel PC se tak dostane na hodnotu 7DC6H. Chceme skočit na smyčku, která je na adrese 7D02H, o 4 byty vzad. Víme, že 4D= 00000100B. K 4D nalezneme dvojkové komplementární kód podle odstavce 3. přepnutím bitů a přičtením 1:

```
      4D...00000100
přepneme 11111011 a připočteme 1
      +        1
      ----------
Výsledek....   11111100=FCH
```

Poznámka : Příkaz INC nemění příznak.
Tentýž program s absolutním skokem by měl vypadat takto :

adresa	příkaz	hex.kód
7D00	LD A,1E	3E 1E
7D02 smyčka	CP A,(HL)	BE
7D03	INC (HL)	23
7D04	JPNZ smyčka	C2 02 7D

Všimněte si, že příkaz JPNZ má 3 byty, protože obsahuje 16ti bitovou adr. (Nezapomeňte na záměnu adr. 7D02-záznam v hexa kódu 02 7D).

Do skupiny skoků, které jsme ještě neprobrali, patří jeden užitečný příkaz DJNZ. Tento příkaz snižuje reg.str B o 1 a skočí jen není-li výsledek 0. Vezměme opět náš malý program "hledej 1E" a stanovme podmínku, že má prohledat úsek jen 100 bytů, (hex 64) dlouhý a pak smyčku opustit bez ohledu na to, zda byl 1 EH nalezen nebo ne :

```
            LD B,64      06 64
            LD A,1E      3E 1E
smyčka      CP A,(HL)    BE
            JPZ NOVA     CA (NOVÁ ADRESA)
            INC HL
            DJNZ smyčka  10 F9
```

Smyčka byla provedena 100x, pokud není nalezeno 1EH, načež následuje odbočení na "novou adresu". Z toho vidíme, že DJNZ působí stejně jako FOR-NEXT v BASICu. Velikost skoků ve všech příkazech relativních skoků vypočítáme stejným způsobem. Tabulku dvojkově komplementárních hex. kódů naleznete v příloze č.1. Program "MC 2" v příloze č.7 vypočítává relativní skoky automaticky během psaní kódu.

11.2 ADC a SBC

Jsou to příkazy "ADD s přenosem" a "SUB s přenosem", t.j. sčítání s přenosem a odčítání s přenosem. Víme, že v registru F je obsažen i příznak přenosu (indikátor přenosu). Je nastaven na 1, jakmile vznikne aritmetickým příkazem z registru potřeba přenosu. Příkaz ADC působí stejně jako ADD, ale k součtu připočítáváme 1 v případě, že předchozí operací byl nastaven přenosový bit. Příkaz SBC působí stejně, jenže 1 ve stejném případě odečítá.

11.3 Posouvání

Příkazy k posouvání : SLA, SRA a SRL mají stejnou vlastnost a to, že posouvají bitový sled a to takto :

Příkaz SLA provádí aritmetický posuv operandu doleva přičemž sedmý bit operandu se přenese na příznak C a nultý bit se obsadí nulou. Obsahuje-li např. B registr :

```
00101100  bude po příkazu SLA B změněn na
01011000  a příznak C bude obsazen 0
```

Všimněte si, že vpravo je doplňovací nula. Protože :

```
00101100 B=44D  a
01011000 B=88D
```

je působení příkazu jasné : násobí dvěmi :.

Další SLA B dává 10110000 B. Zde je starší bit=1, tedy číslo je považováno za záporné a je nastaven znaménkový příznak.

Příkaz SRA provádí aritmetický posuv operandu doprava, přičemž se nultý bit operandu přenese na příznak C a sedmý bit operandu zůstane zachován. Tato operace je chápána jako dělení dvěmi. Je-li záporné číslo děleno dvěmi musí být výsledek rovněž záporný, takže znaménkový bit zůstává. Je-li v registru 10110000 bude po SRA B v registru 11011000.

Příkaz SRL provádí posun operandu vpravo, přičemž nulty bit operandu se přenese na příznak C a sedmý bit se doplní nulou. Je to logické posunutí doprava,aniž se co změní. Bude-li tedy v registru B 10110000, bude po příkazu SRL B v registru B 01011000.

11.4 Příkazy PUSH a POP

Příkazy nám dovolují přístup do depozitu jiným způsobem než podprogramem. Mohou být použity pro zajištění hodnot některého párového registru po určitou dobu. Řekněme, že máme nějakou hodnotu v registru BC, kterou budeme potřebovat později, neboť teď chceme registr BC využít pro jiný účel. Píšeme tedy :

```
PUSH BC
.............
kódy pro využití registru BC
.............
POP BC
```

Zavedením příkazu PUSH se zásobníkový ukazatel SP sníží o 1 a zavede vyšší byte do zásobníku, pak se opět sníží o 1 a zavede do zásobníku nižší byte.

Příkaz POP přesune 2 byte z vrcholu zásobníku do registrového páru tak, že vezme nižší byte z místa kam ukazuje zásobníkový ukazatel SP, sníží se o 1 a vezme vyšší byte a znovu se sníží o 1, aby ukazoval na následující vrchol zásobníku.

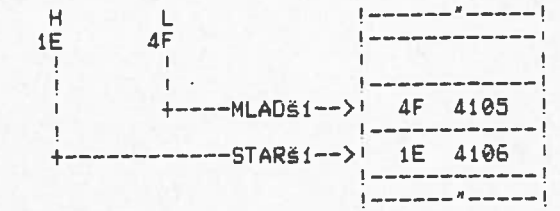
Jsou-li takto zajišťovány 2 nebo více registrů, je nutné jejich návrat do programu provést v opačném pořadí. Je třeba si ještě uvědomit, že registr A je neustále používán, takže nesmí být zajišťován.

Upozornění : Dbejte, aby při sestavování programů ve strojovém kódu byly PUSH a POP stále spolu.

11.5 Vlastnosti 16ti bitových operací

Velmi důležité u 16ti bitových operací (především PUSH a POP) je pořadí, ve kterém jsou přenášeny byty z registrů do pamětí a naopak. Děje se to takto :

LD (4105),HL má následující účinek, jestliže registrový pár HL obsahuje 1E4FH :



Méně významný byte nebo nižší nebo "mladší" byte v registru je uložen na udanou adresu a významný byte do následující adresy.

LD HL,(4105) má přesně opačný účinek. Příkaz má kód 2A 05 41. Zrovna tak pokus naplnit registr HL hoonotou 1000.

LD HL,1000 bude mít kód 21 0010. Ačkoliv se v tomto případě jedná o data a ne o adresu, jsou byty přeneseny stejně.

11.6 Přerušení, zhroucení systému

Jestliže se vložený BASIC program zhroutí, můžete vyváznout pomocí BREAK, aniž by jste ztratili program. Zhroucení ve str. kódu je horší. Buď se všechno z paměti vymaže, nebo zůstane počítač netečný a musíte jej znovu vybudit přerušením napájení. Chcete-li vidět příklad zhroucení, které nezpůsobí NEW ? Zkuste:

```
RANDOMIZE USR 996
```

Hluk, barva a žádná reakce na klávesnici. Ještě horší je :

```
RANDOMIZE USR 1400
```

Při práci ve strojovém kódu je zhroucení nežádoucí. Jediný nejjistější prostředek je přerušení napájení. Několik rad pro snížení nebezpečí zhroucení systému :

1. Prozkoumat pedanticky sled strojového kódu a ujistit se, že jste všechno správně zadali.
2. Nepoužívejte nikdy příkaz HALT (op. kód 76H). Není to totiž stop jako v BASICu. Bezpečně vede ke zhroucení.
3. Dejte pozor, aby vždy byly ve sledu obsaženy společné příkazy CALL a RET, PUSH a POP.
4. Stanovte správnou startovací adresu.
5. Nahrajte si na mgf pásek vše co můžete předtím, než sled poprvé vyzkoušíte, aby jste zbytečně neztratili všechno.

12. Násobení ve strojovém kódu

Strojový kod neobsahuje žádný příkaz pro násobení, Z80 takový příkaz nemá. Přesto však to jde, spojíme-li logiku, aritmetiku a posouvání. Probereme příslušný program, který to dovolí :

Příklad : Budeme pracovat s 8mi bitovým registrem. Máme-li vynásobit např.: 9 a 13 bude to vypadat v binární soustavě takto :

```
  000001001
* 000000101
```

V binární soustavě je násobení jednodušší, než v dekadické. Je-li číslo kterým jsme násobili 1, pak hořejší řádek opíšeme, je-li nula, posuneme pouze násobence o jedno místo doleva. Začíná se vpravo :

```
  00001001.....P
* 00001101.....Q
  ----------------
  00001001
+ 00100100
+ 01001000
  ----------------
  01110101
```

Samozřejmě při každém kroku musíme u P vpravo doplnit nuly, stejně jako při dekadickém násobení. Strojový kód to chápe jako posunutí doprava. Jak je naznačeno, přidělili jsme oběma číslům jména P a Q. Zatímco je P posunuto doleva, doporučuje se posunout současně Q doprava. Tím zkoumáte vždy nejnižší bit, podle kterého se P k součtu připočte nebo nepřipočte.

12.1 Ve strojovém kódu

Dáme-li P do registru D a Q do registru E bude postup následující:

1. Do registru A uložit 0.
2. Je-li v E nejnižší bit 1, připočíst hodnotu D registru do A.
3. D posunout doleva.
4. E posunout doprava.

Kroky 1 až 4 opakovat 8x. Jednotlivé kroky vyjádřené op. kódem.:

První krok je evidentní. V druhém nasadíme registr B jako čítač smyček ve spojení s příkazem DJNZ, který se musí objevit na konci:

```
LD A, 00
LD B, 08
```

Chceme-li dále prozkoušet nejnižší bit z E, můžeme to s dosavadními znalostmi provést jen pomocí maskovacího vzoru (00000001 B) a příkazu AND. Vzor dáme do registru C:

```
LD C, 01
```

Protože příkaz AND je prováděn jen s registrem A, musíme si jeho obsah uschovat v registru L, nebot' by byl jinak smazán :

```
smyčka LD L, A
```

Do registru A dáme maskovací vzor a pak porovnáme. Po porovnání dáme zpět do registru A a uschovanou hodnotu z L :

```
LD  A, C
AND A, E
LD  A, L
```

Byl-li výsledek porovnání s registrem A 0, musíme přeskočit část "připočti hodnotu D do registru A", t.j.musíme skočit o dva kroky dopředu. Je-li výsledek porovnání 1, provede se další příkaz :

```
JRZ posunutí
ADD A, D
```

a nyní příkazy pro posunutí :

```
SLA D
SRA E
```

a ukončíme příkazem pro smyčku, při kterém se současně sníží registr D o 1 :

```
DJNZ smyčka
(RET)
```

12.2 Strojový kód pro násobení

Chcete-li tento program vyzkoušet, musíte se postarat o to, aby registr D a E obsahoval hodnoty, které chcete vynásobit. Dáme si je jako u předchozích programů na adresy 7D00H a 7D01H, které naplníme pomocí POKE před tím, než program spustíme. Předřadíme-li tedy uložení násobence a násobitele na uvedené adresy a jejich uložení do registrů D a E, bude program vypadat takto:

```
             LD HL,7D00          21 00 7D
            LD D,(HL)           56
             INC HL              23
             LD E,(HL)           5E
             LD A,00             3E 00
             LD B,08             06 08
             LD C,01             0E 01
smyčka       LD L,A              6F
             LD A,C              79
             AND A,E             A3
             LD A,L              7D
             JR Z,posun          28 01
             ADD A,D             82
posun        SLA D               CB 22
             SRA E               CB 2B
             DJNZ smyčka         10 F3
             RET
```

Program začíná na adrese 7D02H. Všimněte si, že v programu kromě počátku není přidělena žádná konkrétní adresa. Je to tím, že všechny skoky jsou relativní, takže nezáleží na konkrétní adrese, ale na vzdálenosti skoku.

Výsledek násobení musíme také ukázat na displeji. Po spuštění programu je v registru A. Jednou z možností, pro nás známou, je rezervování výsledku ještě v adrese 7D02 tak, jak jsme to udělali pro součet.

To znamená, že musíme ještě vložit :

```
LD 7D02,A                   32 02 7D
```

a při uložení v programu "MC 1" pak při požádání o počet bytů ohlásíme 3 byty. Pro obdržení výsledku dáme :

```
PRINT PEEK 32002
```

Výsledek lze také přenést z registru A do registru C :

```
LD B,0                      06 00
LD C,A                      47
```

a vyvolat pomocí : PRINT USR 32003

Uvědomíme-li si, že USR je funkce a že při návratu do BASICu je výsledek obsažen v BC registru, pak nám bude jasné, že tento příkaz spouští sled strojového kódu a ukazuje výsledek.

12.3 Bit

Existuje jednodušší způsob, jak evidovat nižší bit v registru E. Umí to příkaz :

BIT 0,2	CB 43

kterým nahradíme 4 příkazy:

LD L,A	67
LD A,C	79
AND A,E	A3
LD A,L	7D

Proč jsme hned neřešili problém tímto novým způsobem ? Chtěli jsme dokázat, že program lze řešit různými postupy a dokonce aniž bychom znali celý seznam příkazů Z80.

13. Obrazovkový displej

Velká část paměti ZX Spectra je věnována jediné úloze. Zobrazení na obrazovce televizoru. Chcete-li využít displej ve strojovém kódu, musíte vědět jak na to.

Informace, které se mají dostat na obrazovku, obsahují dvě oblasti paměti RAM, které se nazývají Display File - oblast displeje a Attribut File - oblast atributů. První řídí jemnou grafiku a vytváření znaků, druhá je pro řízení barev. Začneme jednodušší z obou oblastí :

13.1 Oblast atributů

Obsahuje 24*32=768 bytů na následujících místech :

	dec	hex
začátek oblasti atributů	23526	5B00
konec oblasti	23925	5AFF

Prvních 22*32=704 ovládá obvyklý rozsah povelů PRINT, t.j. řádek 0 - 21 pro příkaz AT. Posledních 2*32=64 jsou poslední dva řádky, použité normálně pro chybová hlášení. Rozsah PRINT končí na adrese 23231D respektive 5ABFH. Spodní dvě řádky začínají u 23232D, respektive 5AC0H.

Jsou-li řádky obvykle číslovány 0 - 21 (poslední dva jsou 22 a 23) a sloupce 0 - 31 a označíme řádku písmenem r a sloupec písmenem c, pak atribut s pozicí r,c má adresu 22526+32*r+c.

Každý atribut má jeden bajt, jehož 8 bitů je rozděleno následujícím způsobem :

FLASH	BRIGHT	PAPER	INK
1/0	1/0	3 BITY PRO BARVU	3 BITY PRO BARVU

POZN.: 1=OBSAZEN, 0=NEOBSAZEN. Bajt 11010101 tedy bude :

```
 -----------------
 !1!1!0!1!0!1!0!1!
 -----------------
  ! !!--+--!-+-!
  ! !    !    !
  ! !    !    +->INK 5 (modrý)
  ! !    +------>PAPER 2 (červený)
  ! +------------>BRIGHT nasazen
  +-------------->FLASH  nasazen
```

Dekadická hodnota tohoto bytu je 213. Potřebujeme-li tento atribut pro pozici na řádce 5 a sloupci 7, musíme jej uložit na adresu :

22528+32*5+7 t.j. 22695

Pro uložení potřebujeme příkaz POKE 22695, 213. Barvu INK neuvidíte, nebudete-li zobrazovat nějaký znak. Proto je lépe :

```
PRINT AT 5,7; "a"
```

čímž se přesvědčíte, že znak "a" má barvu atributu.Protože každá řádka má 32 sloupců, nalezne se adresa pro místo pod danou adresou jednoduchým připočtením hodnoty 32D (20H).

Označíme-li tedy atributy následujícím způsobem :

```
 0.řádka  5800 5801 5802...591D 581E 581F !
 1.-"-    5820 5821 5822...583D 583E 583F !
 2.-"-    5840 5841 5842...585D 585E 585F !   OBLAST PRINT
    .                      .              +-> RESPEKTIVE
    .                      .              !   PRINT AT
    .                      .              !
21.-"-    5AA0 5AA1 5AA2...5A8D 5A8E 5A8F !
-------------------------------------------------------
22.-"-    5AC0 5AC1 5AC2...5ADD 5ADE 5ADF !
23.-"-    5AE0 5AE1 5AE2...5AFD 5AFE 5AFF +->HLášení
```

13.2 Oblast displeje

Tato oblast je komplikovanější než předchozí. Každý znak na displeji je uložen jako seznam 8mi bytů, každý odpovídající jedné řadě bodů ve čtverci 8*8 pixelů. Toto uspořádání má vysokou čitelnost matice znaků. Bohužel, byty nejsou pro znak uloženy v paměti pěkně za sebou.

Jestliže pomocí LOAD SCREEN$ ukládáme do Spectra obrázek, podívejte se,jak se kreslí. Děje se to ve skutečnosti dle řady bytů v oblasti displeje. Nejlépe je to vidět, zaplníte-li obrazovku INK 0, nahrajete ji povelem SAVE na MGF pásek a pak ji přehrajete do počítače povelem LOAD :

```
10 FOR E=0 TO 21
20 PRINT "22 * černý čtvereček"
30 NEXT E
40 SAVE "SMYCKA" SCREEN$
```

nahrajte na pásek a pak zadejte povel :

```
CLS: LOAD "SMYCKA" SCREEN$
```

a dívejte se jak je obrazovka začerňována.
Adresy jsou:

	Dekad.	Hexadek.
začátek oblasti displeje	16384	4000
konec oblasti	22527	57FF

Celkový počet bytů je 32*24*8=6144D t.j. 1800H.

Nejlépe si představíme oblast displeje tak, že obrazovku rozdělíte na 3 vodorovné stejné pásy tak, že:

```
blok 1 ... řádka 0 až 7 obrazovky
blok 2 ... -"-   8 až 15 -"-
blok 3 ... -"-  16 až 22 -"-
```

Vidíte, že blok 3 zahrnuje i dvě řádky 22 a 23 pro hlášení. Každý blok obsahuje 6144/3=2048 bytů (800H). Prvních 2048 bytů naplní první blok, další dávka druhý blok a poslední třetí blok a v každém bloku je stejné uspořádání. Hexad. je to takto:

blok	začátek	konec
1	4000	47FF
2	4800	4FFF
3	5000	57FF

Abychom porozuměli uspořádání uvnitř bloků, stačí probrat stav bloku 1. Ostatní dva jsou identické. (jenom nezapomeňte, že ve třetím bloku jsou dvě řádky pro hlášení). Proces se dá snadno popsat pomocí pravoúhlé mřížky čitelné grafiky s 256*176 body. Řádky jsou číslovány od 0 do 175, sloupce od 0 do 255 a řádkou 175 nahoře a sloupcem 0 vlevo, jak je obvyklé u příkazu PLOT, sloupec, řada. Blok 1 sestává potom z řádek 175 až 112,dohromady 8*8 řádek po 256 bodech.

Prvních 32 bytů oblasti displeje obsahuje informace pro řadu 175. Každý byt určuje úsek 8mi sloupců. První byt obsahuje sloupce 0 až 7 a používá pro 1 bit jeden bod (pixel).Obsahuje-li tedy např. adresa 4000 byt 01101100, pak se prvních 8 bodů na obrazovce zobrazí takto :

```
        +--------------------
        !0110110000000000000
        !0000000000000000000
        !0000000000000000000
        !0000000000000000000
        !0000000000000000000
```

s prázdným bodem pro "0" a černým čtverečkem pro "1".

Další byt naplní sloupce 8 - 15, další 16 - 23 atd. až po 32 bytech je naplněna celá řada 175. Abyste to mohli pozorovat, zadejte CLS a pak POKE 16384, BIN 10010011 a uvidíte v hořejší řadě dvě krátké čárky. Proveďte POKE 16384, BIN 10101010 a uvidíte 4 body. Změňte pak hodnotu 16384 na 16385 a 16386 až 16415 a vyplníte řadu 175. Následující byt se nachází na adrese 16416. Zavedete-li POKE 16416, BIN 10101010, zjistíte, že to není další řada, tedy 174, která se vyplňuje. Je to prvních 8 sloupců řady 167. Je to tedy 175-8=167, takže 8 řádků grafiky je přeskočeno. Tento byt a 31 bytů za ním vyplní celou řádku 167. Pak jde systém dále k řadě 168-8=159 a po naplnění skočí o dalších 8 řad až se vyplní celý blok 1. Řady tedy mají v blocích po 32 bytech následující sled :

175 167 159 143 135 127 119

Kdyby bylo dalších 8 přeskočeno, dostali bychom se na řádek 111, který však již není v bloku 1. Na místo toho skočí Spectrum na řadu bezprostředně pod hořejší v bloku 1 a naplňuje ji po blocích 32 bytů ve sledu :

174 166 158 150 142 134 126 118

Toto se dále opakuje znovu, takže další je :

173 165 157 149 141 133 125 117

a tak dále až dosáhne nejspodnější řadu bloku 1

168 160 152 144 136 128 120 112

Tím je blok 1 uzavřen. Dalších 2048 vyplňuje identicky blok 2 a pak blok 3. Pořadí v blocích je stejné. Adresy se posouvají směrem nahoru. Uspořádání obrazovky :

```
32 bytů pro hořejší řadu každého znaku  ř=0
-----------------------"--------------- ř=1
                       :
-----------------------"--------------- ř=7
32 bytů pro druhou řadu každého znaku   ř=0
-----------------------"--------------- ř=1
                       :
-----------------------"--------------- ř=7
```

atd pro třetí řadu, čtvrtou až poslední k 8mé řadě.

14. Oblast atributů

Můžeme naráz obarvit celou obrazovku, nebo ji zhasínat a rozsvěcovat, nechat rolovat jednotlivé barvy atd. ve strojovém kódu. Nejjednodušeji se začíná tím, že píšeme ve strojovém programu způsob vyvolání čtverečku určité barvy v hořejším levém rohu. Použijeme datový byt 7D00H pro barvu (byt atributu). K tomu se použije prog. (MC 1). Op. kódy budou :

```
        LD A,ATRIBUT            3A 00 7D
        LD (5800),A             32 00 58
```

Vložte do stroje a dejte "a" a doplňte data: POKE 32000, 32 číslo 32=4*8 a je to atribut PAPER 4. Teď dejte GOTO 300 a vložte "r".

Zkuste si pár cvičení :

1. čtvereček nemá být zelený, ale červený
2. má být fialový
3. vytvořte jej modrý a nechte blikat
4. at' je žlutý a BRIGHT
5. umístěte jej do řady 0, sloupce 1
6. ----------"-------- 2, -"- 7

Pro 1-4 změníte jen POKE 32000, pro 5 a 6 změníte adresu 5800H pro správné místo atributu.

Jako další vyplňte celý horní řádek zeleně. Potřebujete k tomu smyčku s čítačem, který je = 32 a při každém průchodu zmenšen o 1. Při průchodu zkoušíme je-li 0 a není-li skáčeme :

```
        LD HL,ATRIBUT           21 00 58
        LD B,32 DEC.            06 20
SMYCKA  LD (HL),32 DEC          36 20
        INC HL                  23
        DEC B                   05
        CP B                    B8
        JRNZ SMYCKA             20 F9
```

Tentokrát nebudou žádné datové byty. Dejte přímo "r" při objevení "volba". Můžete vyplnit bez námahy 2 či více řádků, zvětšíte-li čítač na větší číslo. Provedeme-li :

```
        LD B,64                 06 40
```

budou dva řádky zelené, při :

```
        LD B,96                 06 60
```

budou zelené 3 řádky atd. až

```
        LD B,224                06 E0
```

až se obarví 7 řad. Přidáme-li do registru B o 20H více, znamená to, že začneme od nuly. První snížení registru B přijde na 255 a to je přesně totéž, jako by se začalo u registru B na 256. A skutečně, změníme-li druhý řádek na :

```
        LD B,0                  06 00
```

vyplní se zelenou barvou 8 řádků. Dále jít nemůžeme, protože jsme v 1 bytu. Můžeme to obejít. Užití B jako čítače je u povelu DJNZ automatické a uspoří 1 byt.

Takže :

```
                LD HL,ATRIBUT           21 00 58
                LD D,0                  06 00
SMYCKA          LD (HL),32 DEC.         36 20
                INC HL                  23
                DJNZ SMYCKA             10 FB
```

14.1 Barvy v režimu bezprostředního ukládání

Abychom mohli měnit barvu ve všech 24 řádkách displeje, můžeme vzíti jako čítač registry BC a zavést povel CP C s další smyčkou DJNZ. Abychom viděli, že je BC nulové, musíme B a C přezkušovat. Lze to provést i pro jinou dvojci registrů. Lze dát též 3 kopie programu dohromady :

```
                LD HL,5800              21 00 58
                LD B,0                  06 00
SMYCKA 1        LD (HL),32              36 20
                INC HL                  23
                DJNZ SMYCKA 1           10 FB
                LD HL,5900              21 00 59
                LD B,0                  06 00
SMYCKA 2        LD (HL),32              36 20
                INC HL                  23
                DJNZ SMYCKA 2           10 FB
                LD HL,5A00              21 01 5A
                LD B,0                  06 00
SMYCKA 3        LD (HL),32              36 20
                INC HL                  23
                DJNZ SMYCKA 3           10 FB
```

Pro požadavek na počet bytů (MC 1) dejte 0. Po volbě "r" se celý displej vyplní i se 2 spodními řádkama. Aby se tomu zabránilo, změňte třetí LD B, 0 na :

```
                LD B,192                06 00
```

aby při poslední smyčce prošlo jen 6 řádek. Existují také rychlejší cesty, avšak tato dovoluje měnit barvy. Provede se to v druhém průchodu z 3620H hexkódu (3610H) a ve třetím 3630H nejlépe pomocí POKE 32016, 32: POKE 32026, 48 a obdržíte 3 barevné pruhy :

```
+------------------+
|     zelený       |
+------------------+
|     červený      |
+------------------+
|     žlutý        |
+------------------+
```

Změnou požadavku na počet bytů (MC 1) můžeme vyvolat jiné podobné efekty.

Nyní použijeme přídavnou smyčku. Utvořme D jako čítač smyček. Pro změnu ponechme displej fialový :

```
            LD HL,ATRIBUT          21 00 58
            LD D,03                16 03
VNEJSI      LD B,0                 06 00
SMYCKA      LD (HL),24             36 18
            INC HL                 23
            DJNZ SMYCKA            10 F8
            DEC D                  15
            CP 0                   BA
            JRNZ VNEJSI            10 F5
```

Všimněte si, že HL ve vnější smyčce není zvyšován o 1, protože jsou odkrokovány vnitřní smyčkou. Protože registr B neslouží jako čítač, musí být použito pro snižování registru D.

14.2 Jak vypínat a zapínat blikání (FLASH)

Následující sled povelů prochází oblasti atributů a vypíná každý FLASH, ale jinak nic nemění. Bit pro FLASH je v levé poloze bytu atributu. Musíme jej změnit na 0 a ostatní bity nechat v původním stavu. Lze to pomocí masky. Jestliže dáme atribut do registru A a povelem AND jej svážeme s bitovou maskou 01111111, přejde první bit na 0 a ostatní zůstanou nezměněny. Vycházíme-li z uvedeného postupu, bude sled vypadat následovně :

```
            LD BC,768              01 00 03
            LD HL,ATRIBUT          21 00 58
SMYCKA      LD A,(HL)              7E
            AND 127                E6 7F
            LD (HL),A              77
            INC HL                 23
            DEC BC                 0B
            LD A,0                 3E 00
            CP B                   B8
            JRNZ SMYCKA            20 F5
            CP C                   B9
            JRNZ SMYCKA            20 F2
```

registrový pár BC je použit jako čítač smyček. Číslo 76 8D je přirozená délka oblasti atributů. Jak registr B tak registr C musí být přezkušován na 0.

Vložte sled s nulovým datovým bytem a zadejte "a". Potřebujeme něco, s čím bychom mohli srovnávat. Vložte :

```
1000 CLS:FOR I=1 TO 704: PRINT FLASH (INT(2*RND));
     CHR$ (65+I-20*INT(I/20));: NEXT I
1010 GOTO 300
```

Použijte GOTO 1000. Obrazovka se zaplní písmeny,z nichž některé blikají. Při "volbě" (MC 1) zadejte "r" a blikání zmizí. Dejte INK a PAPER na různou barvu a opakujte program. Vidíte, že se nemění.

Opačný problém je v případě blikání celé obrazovky (FLASH). Místo masky AND 127 dáme: OR 129 (128D=10000000B). OR mění první bit na 1 a ostatní ponechává beze změny. Potřebujeme tedy totéž jako v předchozím sledu povelů, jen z řádky AND 127 (E67FH) uděláme :

```
OR 128                          F6 90
```

Použijte opět GOTO 1000. Co se stane použijete-li XOR 129 ?

14.3 Příkazy SET a RES

Hořejší změny v bitové masce můžeme docílit také přímým způsobem. Povel SET dává libovolný bit v libovolném registru na "1", povel RES zase na "0". Bity uvnitř bytu jsou uspořádány :

```
-----------------
!7!6!5!4!3!2!1!0!
-----------------
```

Nejvyšší číslo je na nejvyšším bitu. Povel :

```
SET 4,F zadá bit 4 v registru D na "1"
RES 6,E zadá bit 6 v registru E na "0"
```

Místo AND 127 tedy můžeme použít :

```
RES 7,A                         CB BF
```

a místo OR 128

```
SET 7,A                         CB FF
```

Nevzniká žádná nutnost relativních skoků, neboť obsahují stejný počet bytů a také když provádíme změnu, program běží.

14.4 Posun sloupců (scroll)

Nyní napíšeme sled, kterým odrolujeme sloupec atributů. Zvolíme sloupec 31, t.j. úplně vpravo. Čtverečky atributů (8*8) jsou umístěny bezprostředně vždy pod jednou ze 32 hořejších adres. Musíme použít indexovaný registr se vzdáleností 32. V podstatě musíme atribut horního čtverečku v registru D přenést do hořejšího čtverečku. Provedeme to smyčkou celkem 21x. Vede to k sledu :

```
            LD BC 32 DEC.       01 20 00
            LD IX,START         DD 21 IF 58
            LD A,21 DEC.        3E 15
SMYCKA      LD D,(IX+32)        DD 56 20
            LD (IX),D           DD 72 00
            ADD IX,BC           DD 09
            DEC A               3D
            CP B                B8
            JRNZ SMYCKA         20 F4
```

Upozornění : Hodnota IF v druhém příkazu je číslo sloupce 31 napsané hexadekad. S jiným sloupcem se změní. Potřebujeme opět něco k testování. Vložte program, zadejte "a" a vložte :

```
1000 FOR D=0 TO 97: PRINT PAPER D-7*INT(D/7); "32x
     MEZERU";: NEXT D
1010 GOTO 300
```

Zadejte GOTO 1000 a obdržíte na obrazovce barevné pruhy. A nyní zadejte "r" a uvidíte, jak se v 31 sloupci posouvají čtverečky o jedno místo nahoru. Opět opakujte "r". Chcete-li aby se posouvaly čtverečky v jiném slupci, musíte změnit ve strojovém programu číslo sloupce v hex. vyjádření.

Kromě toho můžeme pozorovat, že spodní čtvereček zůstává nezměněný a má stále stejnou barvu. I v případě, že dosazení do "r" opakujete vícekrát.

Aby spodní čtvereček zůstával bílý,doplníme strojový kód za JRNZ takto:

```
LD IX,56 dec          DD 36 00 38
```

Pozn.: 550 je 8*7 byt atributu 56.

Zkuste si sami: navrhněte, jak scrolovat celý blok sloupců např. 5 až 18 s použitím tohoto sledu strojového kódu, doplněného smyčkou s inkrementací při každém návratu ke startovací adrese IX.

14.5 Generátor vzorků

Následující program vytváří barevný vzorek, jehož vzor si objasněte sami jako cvičení. V programu (MC 1) zadejte požadovaný počet bytů 0.

```
LD B,0                  06 00
LD D,0                  16 00
LO HL,5800              21 00 58
LD (HL),D               72
INC HL                  23
-------------------------------------
! LD A,7                82          !
! LD D,A                57          !
! DJNZ ?                10 F6       +-> II
! LD HL,5900            21 00 59    !
! LD (HL),D             72          !
! INC HL                23          !
-------------------------------------
LD A,7                  3E 07
ADD A,D                 82
LD D,A                  57
DJNZ ?                  10 F8
LO HL,5900              21 00 59
LD (HL),D               72
INC HL                  23
LD A,7                  3E 07
ADD A,D                 82
LD D,A                  57
DJNZ ?                  10 F8
LD HL,5A00              21 00 5A
LD (HL),D               72
INC HL                  23
LD A,7                  3E 07
ADD A,D                 82
LD D,A                  57
DJNZ ?                  10 F8
```

Aby jste docílili jiné vzory, změňte řádku LO A,7 např. na LO A,8. nebo na LD A,32. Místo 07 deklarujte 08.IF, respektive 1E. Můžete použít jakékoliv hex. číslo.

Aby jste mohli smazat displej, zadáte při volbě "a" a pak CLS a pak GOTO 300 a volbu "r". Rozpoznáte co program obsahuje ?

15.Oblast displeje

Oblast displeje se používá pro svou komplikovanou strukturu poněkud obtížněji, avšak výsledky jsou daleko bohatší. Především si musíte uvědomit, že oblast displeje začíná na 4000H a je dlouhá 1800H bytů. Je sestavena ze 3 bloků, které mají rozsahy 4000H až 47FFH, 4800 až 4FFF a 5000 až 57FF. Cit pro velikost a polohu oblasti si vytváříme zapamatováním. Proveďme změnu v počáteční oblasti displeje a podívejme se co z toho vzejde.

Rekněme třeba, že do všech adres dáme stejnou hodnotu. Abychom byli konkrétní, dáme tam byt 10101110B nebo 174D. K tomu můžeme použít registrový pár HL k zavedení adresy a registrový pár BC jako čítač smyček. Program ve strojovém kódu bude pak vypadat takto :

```
                LD HL,DISPLEJ          21 00 40
                LD BC,1800             01 00 18
SMYCKA          LD (HL),174D           36 AE
                INC HL                 23
                DEC BC                 08
                CP B                   B8
                JRNZ SMYCKA            20 F9
                CP C                   B9
                JRNZ SMYCKA            20 F6
```

Tímto obdržíte vzorek kolmých pásků. Jsou vytvořeny obrazem bitů v binárním vyjádření čísla 174D. Černý pás šířka 1, pak bílý šířka 1 atd. Celé se to opakuje 32x a zaplní se vzorem dle vloženého bytu. Uspořádání je svislé a tvoří pásy. Vezmete-li místo 174D jiné číslo, uvidíte jiný vytvořený vzorek. Především vyzkoušejte 1,15 a 170D, tedy 01H, 0FH a AAH.

15.1 Vodorovné linky

Následující sled kreslí celé vodorovné linie. Pro celou řádku je použito 255D.

```
                LO A,3                 3E 03
                LD B,0                 06 00
                LD HL,DISPLEJ          21 00 40
SMYCKA          LD (HL),255 DEC.       36 FF
                INC HL                 23
                DJNZ SMYCKA            10 F8
                DEC A                  3D
                CP B                   B8
                JRZ PRESKOK            28 08
                PUSH AF                F5
                LD A,7                 3E 07
                ADD A,H                84
                LD H,A                 67
                POP AF                 F1
                JR SMYCKA              18 EF
PRESKOK         RET                    C9
```

15.2 Vzorky na obrazovce

Zadáváme-li různé byty do oblasti displeje nebo jeho části, lze vyrobit různé hezké vzory. V ukázkovém sledu je použit registr C, který začíná u FF, pak FE FD atd. Postupně je snižován až k 00. Při pečlivé analýze vzorku na displeji je možné v některých případech rozeznat uspořádání v oblasti displeje. V praxi se však vzorky tak prolínají, že to většinou není možné.

```
                LD BC,768 DEC.          01 00 19
                LD HL,DISPLEJ           21 00 40
SMYCKA          LD (HL),C               71
                INC HL                  23
                DEC BC                  0B
                CP B                    B8
                JRNZ SMYCKA             20 FA
                CP C                    B9
                JRNZ SMYCKA             20 F7
```

Zkuste takovou změnu, aby smyčka byla v registru B

```
SMYCKA LD (HL),B
```

Bude to jiný vzorek. Projde smyčkou 256x a mění hodnotu registru B, což odpovídá 8mi řádkám v režimu čitelné grafiky. Na vzorku uvidíte, že každá horejší řádka má různé hodnoty, takže prvních 256 bytů neodpovídá řádkám 175-169. Způsob jak je vzorek rozdělen na 8 řádků ukazuje, že prvních 256 bytů je ve skutečnosti v řádkách 175, 167, 159 atd.

Struktura vzorku je opět sestavena ze 3 bloků.

15.3 Šrafování

Tento sled má přednost, protože zde není důležitá struktura oblasti displeje. Prochází sice oblastí a nahrazuje každý nulový byt 00000000B bytem 10101010B (AAH) a způsobí, že prázdná místa obrazovky jsou překreslena svislými tenkými čárkami.

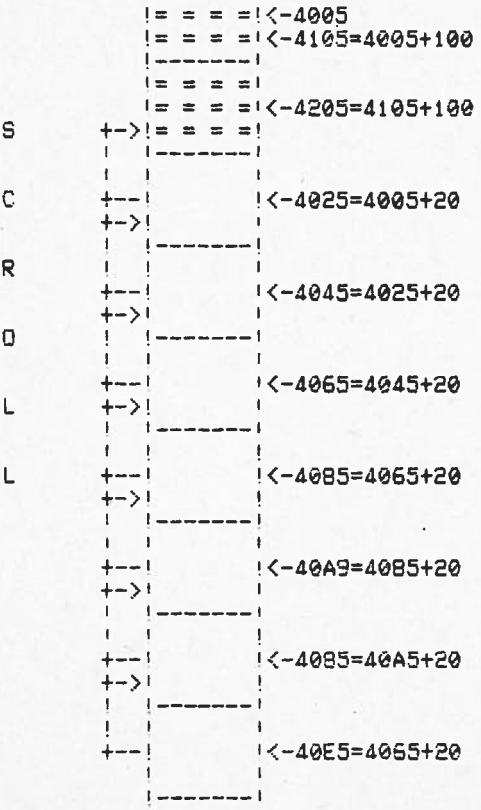
```
                LD HL,DISPLEJ           21 00 40
                LD BC,769 DEC.          01 00 18
                LD A,0                  3E 00
SMYCKA          CP A,HL                 BE
                JR NZ PRESKOK           20 02
                LD HL,AA HEX            36 AA
PRESKOK         INC HL                  23
                DEC BC                  0B
                CP B                    B8
                JRNZ SMYCKA             20 F6
                CP C                    B9
                JR NZ SMYCKA            20 F3
```

Příkaz LD A,0 není vlastně nutný. Registr A obsahuje vždy 0, není-li ničím obsazen. Průběh programu je však jasnější. Nepotřebujeme-li některý povel v programu strojového kódu, můžeme jej vypustit následujícím způsobem: Pomocí POKE 22006,0,

které změní příkaz na 0000. Kód 00 znamená NOP (NO operation), t.j. žádná operace a nezpůsobí nic než spotřebovaný čas. Je to ideální způsob mazání příkazů, aniž bychom museli na místo zadávat jiný kód.

15.4 Skrolování řádků v režimu čitelné grafiky

Tento úkon je komplikovaný. Jednotlivé sloupce obrazovky se mohou nechat odrolovat pryč. Abychom si záležitost mohli zjednodušit, necháme jeden blok displeje scrolovat, t.j.jen 8mi řádkový úsek sloupce. Vezmeme blok 1 a sloupec 5. Sestává z 8mi řádkového úseku, ve kterém má každý svůj znak a odpovídá vlastně místu PRINT AT. Každá řádka je na své adrese vložena do paměti.



Abychom se vyvarovali omylu, nazýváme každý obdélník z 8mi řádek řadou (používá se při povelu PRINT AT) a jedn. řádku - řádkou.

Nejvyšší řádka řady 0 je obsažena v oblasti displeje (paměťová) na adrese 4005H a za ní o 32D je horní řádka řady 1 atd. až k řadě 7. Abychom obdrželi druhou řádku v řadě 0, musíme připojit 256D (100H) k původní adrese 4005H.

4005+100=4105

Přidáním 20H obdržíme druhé řádky dalších 7mi řad. Pak provedeme totéž pro třetí řady a nakonec osmé řady. Toto je stavba sloupce (blok 1). Pro vyrolování bytu o jednu řadu musíme dát do adres o 32D (20H) míst více než předchozí.

Chceme, aby nám odrolovala hořejší řada a až do sedmé řady zůstalo prázdno. Je to obdoba skrolingu atributů, který byl popsán ve stati 14.4, jen nyní musíme projít 8x smyčkou, aby se všech 8 řádek v řadě posunulo. Sled, který by odroloval jen nejhořejší osminu každého znaku, není žádoucí. Musíme tedy počítat s hodnotou 0 nebo 32D.

Ve slabé naději o jasnější výklad předkládáme nejprve sled v BASICu:

```
2000 LET IX=256*64+5
2010 FOR I=1 TO 8
2020 FOR A=1 TO 7
2030 LET B=PEEK (IX+32)
2040 POKE IX,B
2050 LET IX=IX+32
2060 NEXT A
2070 POKE IX,0
2080 LET IX=IX+32
2090 NEXT I
```

Použili jsme proměnnou IX, která odpovídá registru IX ve strojovém kódu. Smyčka A posunuje všechny horní řádky nahoru. Pomocí POKE v 2070 se tvoří prázdný prostor v sedmé řadě a smyčka I opakuje proces pro 2., 3. až 8 řádek každého znaku. Zkuste to provést použitím GOTO 2000.

Až to vyzkoušíte, zjistíte že to chodí, ale pomalu. Znaky se viditelně posouvají nahoru. Jdeme však správnou cestou. Je to totiž velmi vhodný způsob: Nejprve napsat sled v BASICu, aby se vyzkoušelo, že je postup správný. Teprve po odstranění chyb přejdeme na str. kód :

```
                LD DE,0020          11 20 00
                LD IX,4005          DD 21 05 40
                LD L,08             2E 08
SMYCKA 1        LD A,07             3E 07
SMYCKA 2        LD B,(IX+20)        DD 46 20
                LD (IX),B           DD 70 00
                ADD IX,DE           DD 19
                DEC A               3D
                CP D                BA
                JRNZ SMYCKA 2       20 F4
                LD IX,00            DD 36 00 00
                ADD IX,DE           DD 19
                DEC L               2D
                PUSH AF             F5
                LD A,0              3E 00
                CP L                BD
                POP AF              F1
                JRNZ SMYCKA 1       20 E4
```

Pro testování zadejte v BASICu :

```
1000 FOR i=0 TO 21: FOR j=0 TO 31: PRINT CHR$(65+i);: NEXT j:
     NEXT i
```

Potom GOTO 1000 a program se rozběhne. Horejší blok sloupce 5 roluje nahoru v pozorovatelném kroku. Jestliže nasadíte strojový kód, obdržíte přijatelné rychlé rolování. Zadejte :

```
3000 FOR K=1 TO 9: RANDOMIZE USR 32000: NEXT K
```

Jestliže to provedete ve smyčce strojového kódu, běží to tak rychle, že sotva vidíte jak znaky mizí. Jestliže nám tento sled ukázal rolování jednoho sloupce, nebude jistě obtížné rozšířit program tak, aby roloval celý komplet sloupců.

15.5 Vícesloupcové rolování

K tomu, aby roloval celý úsek sloupců, je nutná ještě příprava. Dejte stranou 4 byty (7D00 až 7D03). Budou obsahovat údaje o sloupci, bloku, šířce rolujícího úseku a slepou nulu potřebnou pro vyčištění. Pomocí POKE uložíme na tyto 4 adresy :

```
7D00 sloupec (např.05 pro start 5.sloupce)
7D01 blok (40H nebo 48H nebo 50H)
7D02 šířka (např. 11H pro šířku 17D sl.)
7D03 00 (slepá nula)
```

Strojový kód, který začíná od 7D04 a jehož střední část je jen opakováním toho, co jsme před tím vytvořili.

```
            LD BC,(7D02)          ED 4B 02 7D
            LD DE,0020            11 20 00
            LD IX,(7D00)          DD 2A 00 7D
SMYCKA 3    PUSH IX               DD E5
            LD L,08               2E 08
SMYCKA 2    LD A,07               3E 07
SMYCKA 1    LD B,(IX+20)          DD 46 20
            LD (IX),B             DD 70 00
            ADD IX,DE             DD 19
            DEC A                 3D
            CP D                  BA
            JRNZ SMYCKA 1         20 F4
            LD (IX),00            DD 36 00 00
            ADD IX,DE             DD 19
            DEC L                 2D
            PUSH AF               F5
            LD A,0                3H 00
            CP L                  BD
            POP AF                F1
            JRNZ SMYCKA 2         20 E4
            POP IX                DD E1
            DEC C                 0D
            PUSH AF               75
            LD A,0                3E 00
            CP C                  B9
            POP AF                F1
            JRZ PRESKOK           28 04
            INC IX                DD 23
            JR SMYCKA 3           18 D2
PRESKOK     RET                   C9
```

Nežli se spustí program, musí se uložit do datových bytů pomocí POKE příslušná čísla. Na to zadejte GOTO 1000 pro malý test. program v BASICu, aby se uložilo to, co chceme odrolovat. Program je na str. 51. Pak použijeme buď GOTO 200 s volbou "r" nebo příkaz RANDOMIZE USR 22004.

Pokuste se pracovat v BASICu se smyčkou do strojového kódu. Je to velmi efektní. Tento sled můžete použít pro rolování u bloku 2 a obdržíte pěkný program pro hrací automaty.

Poslední upozornění:

Pomíchají-li se Vám řádky trochu do sebe,zkuste dát pak JRZ před POP AF. Obrazovka Vám odroluje, ale může přijít chybové hlášení "C Nonsens in BASIC : 1". Znamená to, že se něco dostalo do smyčky se zásobníkem a při kódu v této řadě pak přeskočí poslední JR přeskok přes onen povel POP AF. Chceme-li pak přivést Spectrum zpět na BASIC, usadí se zpětná adresa nad zásobník zbývajících bitů registru AF a systém se tak dostane do zmatku.

16. Více o příznacích

Ve strojovém kodu nejsou žádné příkazy FOR/NEXT pro smyčky, nebo IF/THEN pro podmínky. Abychom je obdrželi, musíme ve strojovém kódu použít příznaky (indikátory). Jako příklad je uveden program ve strojovém kódu pro přečíslování řádků.

Dosud jsme se úspěšně vyhýbali technickým detailům F registru a při každém podmíněném skoku jsme použití příznaku taktně zamlčeli. V žádném případě nebudeme uvádět detaily jak F registr pracuje, neboť je to jeden z nejkomplikovanějších popisů činnosti Z80. S uspořádáním funkcí F registru se nyní seznámíme. F registr má 8 bitů, využito je z nich 6. Jsou to :

C.....přenosový příznak
Z.....nulový příznak
S.....znaménkový příznak
P/V.....paritní(přeplň.)příznak
H.....poloviční přenosový příznak
N.....odečítací příznak

V registru jsou seřazeny:

S	Z	X	H	X	P/V	N	C

Při čemž "X" jsou nevyužité bity.

Přenosový příznak je ovlivněn především sčítáním, odčítáním, rotací a posunem.

Nulový příznak je ovlivněn téměř vším. Zhruba řečeno: když někde něco, mimo LD, INC a DEC mění obsah registru A, je nulový příznak nasazen na "1" v případě, že v A registru je "0". Jinak zůstává nezměněn (=0). Bit nasadí příznak, je-li daný bit=0. CP nasadí příznak, nebo jej zruší podle výsledku porovnání.

Znaménkový příznak ukládá znaménkový bit výsledku každé operace, která byla naposled provedena: 1 pro negativní, 0 pro pozitivní.

Paritní, přeplňovací příznak působí jednak ve vztahu k aritmetice a jednak k logice. Při aritmetice bude nasazen jako "1", jestliže dojde v dvojkové komplementární aritmetice k přeplnění (např.suma dvou pozitivních čísel na konci akumulátoru se přeplní a vede k negativnímu výsledku). Při logických operacích je usazen na "0", jestliže byt v registru A je sestaven ze stejného počtu nul a jedniček a změněn na "1", je-li počet nul a jedniček v bytu různý. Tato nepřímá/přímá vlastnost bitů se nazývá paritou bitů. Například :

A obsahuje 01101100-přímá parita. Příznak P/V=0
A obsahuje 10010001-nepřímá parita. Příznak P/V=1

Příznaky H a N jsou použity jen pro binární vyjádření dekadických čísel. Jsou většinou platné pro buzení hardware připojeného k počítači.

Jak jsou ovlivňovány příznaky C a Z různými operacemi, je uvedeno v příloze 5. Pro začátečníky v programování strojovým kódem jsou to nejpotřebnější příznaky.

Pro nulový příznak Z jsme použili např. v každém JRZ, JRNZ nebo DJNZ. Pravidelně předchází příkaz jako :

```
CP B
```

který nasazuje příznak tak, jako by to byl příkaz SUB A,B ,CP B, ale nemění obsah registru A. Jestliže registry A i B obsahují tytéž byty, dalo by odečtení nulu, je nasazen nulový příznak Z na "1". Jestliže jsou A a B různé, je nulový příznak vypnut.

Přenosový příznak byl použit méně. Je nutný jako čítač smyček, když neví, došla-li poslední smyčka. Lze se o tom snadno přesvědčit. Řekněme, že chceme vědět, zda je číslo v registru HL větší než číslo v registru BC. Příkaz: SBC HL,DC ED 42 nasadí přenosový příznak, je-li BC větší než HL a zruší jej, je-li BC menší nebo rovno HL. Uzavřete podmíněným skokem :

```
JR C kamkoliv 38 ??
```

a skok se provede, když je BC větší nebo rovno HL.

```
JRNC kamkoliv 30 ??
```

a skok se provede, je-li BC rovno nebo menší než HL. Pro určování velikostí se užijí pozitivní čísla - ve dvojkově komplementární aritmetice.

16.1 Přečíslování řádků

Přečíslování řádků můžeme provádět pomocí jednoduchého sledu v BASICu, který mění čísla řádků do pravidelného pořadí 10, 20 atd. v desítkových krocích až na konec programu. Musíme znát uložení řádků v BASICu. Program je uložen mezi adresami v PROC a VARS. Bez použití floppy disku je PROC na adrese 23755. Každá řádka má tvar :

NS	NJ	LJ	LS	Kód pro řádku	ENTER

NS a HJ jsou zde vyšší a nižší byty čísla řádku, LJ a LS nižší a vyšší byty počtu znaků v řádku a ten je zakončen ENTER. Všimněte si, že v čísle řádku je představen vyšší byt, zatímco v počtu znaků je představen nižší byt. Například program :

```
1 REM 12345678901
2 PRINT a
```

který je uložen v paměti následovně :

adresa	obsah	poznámka
23755	0	vyšší byt čísla prvního řádku
23756	1	nižší ----------"----------
23757	13	---"------ délky řádku
23758	0	vyšší byt délky řádku
23759	234	kód pro REM
23760	49	---"--- 1
22761	50	---"--- 2
23762	51	---"--- 3
23763	52	---"--- 4
23764	53	---"--- 5
23765	54	---"--- 6
23766	55	---"--- 7
23767	56	---"--- 8
23768	57	---"--- 9
23769	48	---"--- 0
22770	49	---"--- 1
23771	13	---"--- ENTER
23772	0	vyšší byt čísla druhého řádku
23773	5	nižší ----------"----------
23774	8	----"----- délky druhé řádky
23775	0	vyšší ------------"--------
23776	245	kód pro PRINT
23777	97	---"--- a
23778	13	---"--- ENTER
23779		počátek oblasti proměnných

Lze to snadno přezkoušet pomocí PEEK a smyčky.

Tímto rozborem jsme si přiblížili cestu k přečíslování řádků. Je třeba najít a projít obsah programu a hledat byty pro ENTER (kód 13). Další dva byty za ním jsou čísla řádků, které můžeme měnit. Takový způsob však není jistý, protože kód 13 se může vyskytnout i jako délka řádku. Pak by nastal zmatek. Je lepší cesta: Byty délky řádků sdělí přesně až kam se musí jít, aby se dosáhlo k bytům příštího řádku. Toto vede k stroj. kódu :

16.2 Program

Jde o to začít u obou prvních bytů programové oblasti s číslem prvního řádku, tento přečíslovat, použít následujících obou bytů délky řádku ke skoku na příští číslo řádku a to celé

opakovat, až je dosaženo oblasti VARS, kde se končí. Jako první musíme uložit adresy. Použijeme BC pro VARS, kde končíme. Pak použijeme BC pro HL pro program (PROG) kde začínáme. Pak DE pro nová čísla řádků. Registry připravíme takto :

```
LD BC,VARS          ED 4B 4B 5C
LD HL,PROG          2A 53 5C
LD DE,10 DEC.       11 0A 00
```

Adresy systém. proměnných nalezneme v příl. 3. Jako další musíme stanovit, zda registr HL převyšuje hodnotu v reg. BC. Jestliže převyšuje, tak začneme. Nyní přijde na řadu přenosový příznak. Zkouška :

```
PUSH HL             E5
SBC HL,BC           ED 42
POP HL              E1
JRNC KONEC          30 15
```

Posledních 15 není ve skutečnosti vypočteno, dokud není celý program sestaven. Je dáno jen předběžně. Následuje vlastní přečíslování :

```
LD (HL),D           72
INC HL              23
LD (HL),E           73
```

Potom musíme přečíst oba násl. byty, pro nalezení délky řádku. Výsledek se musí někam uložit. Registr HL je použit pro přepočet. Lze použít registr DC. Lze to obejít tak, že hodnotu v něm uložíme, abychom ji později vyzvedli zpět.

```
PUSH BC             C5
INC HL              23
LD C,(HL)           4E
INC HL              23
LD B,(HL)           46
```

Posuneme HL nahoru k nejbližšímu číslu řádku :

```
ADD HL,DC           09
INC HL              23
```

a vrátíme starou hodnotu do registru BC

```
POP BC              C1
```

Nakonec dáme do registru DE správnou hodnotu pro následující řádku přičtením hodnoty 10D. K tomuto musíme obnovený registr posunout na zásobník :

```
PUSH HL             E5
LD HL,10 DEC.       21 0A 00
ADD HL,DE           19
LD D,H              54
LD E,L              5D
POP HL              E1
```

A nyní smyčkou zpět ke kroku "zkouška" pro odzkoušení, zda není HL příliš veliké :

JR zkouška	18 E5

a nakonec :

RET	C9

Celý sled vložte v naznačené formě,jako obyčejně se zakončujícím RET. Program v BASICu, který máte vložen, bude přečíslován, zadáme-li :

```
RANDOMIZE USR 32000
```

Samozřejmě, že se nemění čísla řádků po příkazech GOTO a GOSUB. Princip je však jasný.

17. Hledání paměťového bloku

Existuje několik účinných příkazů, které nám naráz přesunou celý paměťový blok. Některé sledy, které jsme dosud probrali, nebyly sestaveny nejvhodnějším způsobem. Bylo to pro zjednodušení a pochopení. V této kapitole se to musí zlepšit.

17.1 Prohledávání paměť. bloku

K tomu jsou velmi účinné příkazy jako CPDR (porovnej, sniž o 1 a opakuj - compare, decrement and repeat). Podobně CPIR znamená : porovnej, zvyš o 1 a opakuj.

Příkaz CPIR:

Tento příkaz provádí odečtení paměťové lokace, jejíž adresu obsahuje registrový pár HL, od obsahu registru A. Registrový pár HL se zvýší o 1 a současně se sníží o jednu registrový pár BC. Když je obsah registrového páru od 0 různý a zároveň výsledek posledního čtení nebyl roven 0, pak se celá činnost opakuje. Je-li obsah registrového páru BC roven 0, nebo byl-li výsledek posledního odečtení roven 0, pokračuje se instrukcí následující bezprostředně. Příklad: Máme nalézt první znak "E" v bloku dat počínaje adresou 20H a konče adresou 100H. Nenalezne-li se znak, provede se skok na návěstí NENI :

```
LD A,"E"              3E 0E
LD HL,20              26 20
LD BC.100-20+1        06 B1
CPIR                  ED B1
JRNZ,NENI             20 NENI
```

Prvním příkazem nastavíme srovnávací znak "E" do registru A, pak vložíme do registru HL první adresu od které hledáme, a nastavíme počítadlo na 100H-20H+1=81H. Po spuštění postupuje program podle popisu činnosti.

Příkaz CPDR:

Tento příkaz provede odečtení pam. místa, jehož adresu obsahuje reg. pár HL, od obsahu reg. A. Sníží reg. pár HL a DC o 1. Tuto činnost opakuje, dokud není obsah reg. páru BC roven 0 nebo donení výsledek odečítání roven 0. Jestliže se objeví některý z těchto případů, pokračuje instr. bezprostředně následující. Příklad: Hledejme poslední znak "D" v operační paměti. Pokud žádný nenalezneme, provedeme skok na návěstí ERROR. Sled vypadá takto :

```
LD A,"D"            3E 0D
LD HL,0FFF          26 FF 0F
LD BC,0             06 00
CPDR                ED B9
JPNZ,ERROR          C2 ERROR
```

Příklady, kde bylo použito skoků a smyček, lze snadno řešit i jinak. Použili jsme skoky, abychom se s nimi seznámili.

17.2 Přenos celého bloku

Příkazy pro přenos dat LDIR a LDDR přesunují bloky dat uvnitř paměti. Např.: příkaz LDIR provede :

1. Uloží do registrového páru HL adresu prvního bytu bloku, který má být přenesen.
2. Uloží do registrového páru DE adresu prvního bytu, kam má být blok přenesen.
3. Uloží do reg. páru BC počet přesouvaných bytů. Příkaz LDIR potom přenese první byt, zvýší HL a DE o 1, sníží BC o 1 a pokračuje dál, dokud není obsah reg. páru BC roven nule.

Je zajímavé, že jednotlivé kroky provádějí zvýšení reg. párů HL a DE, ale není patrný žádný přenos. To obstarává příkaz LDIR. Na konci přenosu obsahuje HL byty bezpr. za přenášeným blokem a DE hořejší konec přenášené oblasti. Prostudujte si program ve stati 17.5.

Příkaz LDDR je podobný, snižuje ale registrové páry HL a DE o 1 a snižuje i pár BC, který slouží jako čítač kroků.

17.3 Přenášení

Přenos bloku užijeme pro případ, že program je uložen mimo oblast atributů a je třeba jej přenésti do oblasti atributů. K tomu se potřebuje 704 datových bytů. Výsledkem je změna v ukládacím programu MC 1.

```
10 CLEAR 21599
```

čímž lze získat dostatek prostoru. Pak se spustí program příkazem GOTO 10 a sdělíme, že chceme rezervovat 704 datových bytů :

```
LD HL,21600 DEC     21 70 78
LD DE,ATRIBUTY      11 00 59
LD BC,704 DEC       01 C0 02
LDIR                ED B0
```

Nový počátek oblasti dat je 31600D.Sled sám má startovací adresu 32304D a je od této adresy uložen.

Nyní si musíme uložit na správné místo data, která chceme přenést. Např.:

```
5000 FOR W=31600 TO 32303
5010 IF W<31900 THEN POKE W,48
5020 IF >=319900 THEN POKE W,32
5030 NEXT W
```

Zadejte GOTO 5000 a počkejte až se to provede.Pak smažte displej a zadejte RANDOMIZE USR 32304. Obdržíte žluté a zelené plochy stejně jako v programu v BASICu.

17.4 Rolování v oblasti atributů - stranou

Změňte počátek programu v řádce 10 opět na CLEAR 31999. Zde nepotřebujeme tolik místa. Pěkné užití instrukce LDIR se naskytne v programu, kde se řada atributů posouvá vlevo a ty co vlevo odchází, se opět objeví vpravo. Pro jednoduchost se volí jen řada 0.

```
            LD HL,ATRIBUT          21 00 58
SMYCKA      LD D,H                 54
            LD E,L                 5D
            INC HL                 23
            LD BC,31 DEC           01 1F 00
            LD A,(DE)              1A
            LDIR                   ED B0
            LD (DE),A              12
```

Program testujte v BASICu :

```
6000 FOR S=0 TO 21 :PAPER S/6:PRINT AT 0,2;"_";: NEXT S
```

a pak RANDOMIZE USR 32000. Následuje rolování. Abychom se přesvědčili, že jsme to provedli správně, dáme :

```
7000 FOR T=1 TO 500:RANDOMIZE USR 32000:NEXT T
```

a uvidíte, co se bude dít. Necháte-li program proběhnout od startovací adresy 22x, může přerolovat celou oblast atributů. Program však musíte doplnit smyčkou.

17.5 Rolování displeje

Zadejme si úkol, nechat odrolovat jednu řádku oblasti displeje stranou. Sled v programu je pak následující:

```
            LD A,8                 3E 08      ;ČÍTAČ SMYČEK
            LD DE,ZACATEK          11 00 40   ;ZAČ.ŘÁDKY
SMYCKA      LD H,D                 62
            LD L,E                 63
            INC HL                 23
            PUSH DE                D5         ;ZAJISTIT ZAČ.Ř
            LD BC,31 DEC           01 19 00
```

```
            PUSH AF             F5          ;ZAJISTIT ČÍTAČ
            LD A,(DE)           1A          ;ZAJISTIT 1 BYT
            LDIR                ED B0       ;ROLOVAT VLEVO
            LD (DE),A           12          ;NAPLNIT 1 BYT
            POP AF              F1          ;OBNOVIT ČÍT.SM
            DEC A               3D          ;SNÍŽENÍ ČÍTAČE
            CP 8                B8          ;ZKOUŠKA NA KONEC
            JRZ,PRESKOK         29 04
            POP DE              D1
            INC D               14          ;DALŠÍ ŘÁDEK
            JR SMYCKA           16 EB
PRESKOK     POP DE              D1
```

Vložte smyčku, nebo smycky, blok po bloku stejné jako v 8mi řadovém bloku a budete mít sled, kterým můžete odrolovat celý displej. Jsou možné i různé obměny.

18. Některé další pokyny

Příkazy mikroprocesoru Z80, které jsme dosud probrali, se netýkaly periferií. Uveďme 3 z nich.

18.1 Přepínání bitů

Příkaz, který provádí inverzi příznaku C v reg. F, je CCF.

Je-li "1", nastaví jej na 0; je-li "0", nastaví jej na "1".

Příkaz, který provede nastavení příznaku C na "1" je : SCF.

Je-li příznak na 0,nastaví jej na "1",je-li na "1",zůstává.

K nulování C v registru F se použije nejprve SCF a pak CCF.

Pro přepnutí všech bitů v registru A použijte příkaz CPL, kterým je každý bit změněn z "1" na "0",nebo z "0" na "1". Je-li obsah registru A 01101100, bude po příkazu CPL 10010011.

18.2 Mnemonika

Je třeba upozornit, že někt. jiné prameny používají mnem. op.kódy v pozměněné formě. Tam kde my píšeme LD a,(nn). užívají LD (nn). Vychází z toho, že reg. A je jediný registr, který může být přímo naplňován, takže se zápis nemusí brát tak přísně.

18.3 Alternativní registry

Víme již, že existuje ještě podružná řada registrů. Jejich hl. úloha tkví v tom, že zajišťuje obsah hl. reg. po dobu provádění jiných sledů a obsah hl. registrů musí být při tom měněn. Děje se to výměnou obsahů hlavních a vedlejších registrů před prováděním sekundárního sledu:

```
EX AF,AF'          DC  ;záměna AF na AF'
EXX                D9  ;záměna z BC,DE,HL
                       ;do BC',DE',HL'
CALL               CD  ;vyvolání sekund.
                        sledu
EX AF,AF'          08  ;obnovení obsahu
EXX                D9  ;-------"-------
```

Tyto příkazy působí stejně, jako bychom pro zajištění obsahu registrů před vyvoláním CALL použili PUSH k přesunu do zásobníku a POP k jeho vyjmutí.

18.4 Podprogramy v ROM

Řada programů, o které se budete pokoušet, je obsažena ve str. kódu v ROM. Proč je tedy nevyvolat přímo. Ušetří se i paměť. To je vše správné, ale my jsme si dali za cíl seznámit se s příkazy str. kódu Z80 a umět je použít pro ZX Spectrum.

18.5 SAVE BASICu a MC

Máte-li program v BASICu, který využívá pro část ve str. kódu posunutého RAMTOPu, můžete jej nahrát povelem :

```
SAVE "program"
SAVE "M-kod" CODE 32000,600
```

kde 32000 je počáteční adresa a 600 je délka strojového kódu, t.j. počet bytů. Chcete-li nahrát z magnetofonu do počítače data, pak :

```
LOAD "program"
LOAD "M-kod" CODE
```

Nezapomeňte na CLEAR 32000.
Lépe je, dáte-li do sledu v BASICu na koncovku první volenou řádku (např.9900)

```
9800 LOAD "M-kod" CODE
```

a při nahrávání do magnetofonu zadáte:

```
SAVE "program" LINE 9800
SAVE "M-kod" CODE 32000,600
```

Tímto způsobem je pak LOAD "program" nahrán současně jako program v BASICu i ve strojovém kódu. Můžete také nahrát několik sledů strojového programu za sebou, máte-li stejné uspořádané programy na mgf pásku. Pak můžete v BASIC programu dát za sebou:

```
-----------------------------------
LOAD "" CODE: LOAD "" CODE:... atd.
-----------------------------------
```

podle počtu nahraných programů ve strojovém kódu.Postupně budeme nahrávat : Program 1...Program 2. atd.

18.6 Využití strojního kódu

Řekli jsme, že jsou ještě jiné důvody pro použití str. kódu, než jenom zvýšení rychlosti oproti BASICu vyvolaná nutností interpretace. Pokusíme se to vysvětlit : V prvním případě vezměme smyčku FOR/NEXT :

BASIC	STROJOVÝ KÓD
10 FOR I=20 TO 1 STEP -1	LD B,14
-------	------
50 NEXT I	DJNZ SMYCKA

V obou případech je při smyčce zmenšováno o 1. Tento proces je však v BASICu mnohem komplikovanější než ve strojovém kódu. Je to proto, že BASIC pracuje nepřetržitě s dekadickými hodnotami včetně desetinných míst, takže snižování provádí o 1.000000000 což není jednodušší než např. 1.587145678. Použitý způsob je proto v BASICu komplikovaný a náročný na čas. Strojová verze má však k tomu jeden určitý povel. Proto je průběh smyčky až 100x rychlejší.

Druhý bod, který nepotřebuje objasnění, je ten, že strojový kód potřebuje méně paměťového prostoru než program v BASICu. Opět příklad :

BASIC	STROJOVÝ KÓD	
30 IF B=P AND P=Q THEN LET P=W	LD HL,5000	3
	LD A,(HL)	1
	LD HL,5001	3
	SUB A,(HL)	1
	JRNZ,PRISTI BYT	2
	LD HL,5002	3
	LD A,(HL)	1
	LD HL,5003	3
	SUB A,(HL)	1
	JRNZ,PRISTI BYT	2
	LD HL,5001	3
	LD A,(5003)	3
	LD (HL),A	1
	celkem	27

Strojový kód vychází z toho, že R,P,Q a W jsou obsaženy v bytech 5000, 5001, 5002 a 5003. V praxi to není tak jednoduché, protože každé číslo obsadí 5 bytů a SUB je ve skutečnosti CALL do ROM pro sled k odečtení čísel s pohyblivou desetinnou tečkou. V každém případě budeme potřebovat více než 27 bytů. Stejný program v BASICu spotřebuje jen 18 bytů pro každý symbol. Jeden další pro číslo řádky a jeden byt pro omezení řádky. Čím je v BASICu příkaz komplexnější, tím je větší spotřeba bytů oproti strojovému jazyku.

18.7 Kde může být MC uložen

Hlavní nevýhodou ukládání kódu za RAMTOP je, že jej nelze nahrát přímo jako součást programu v BASICu. Předností je, že tam lze uložit i jiné programy. Jsou různé alternativy :

Oblíbeným trikem je uložení programu ve strojovém kódu do příkazu REM a dát jej jako první řádek programu BASICu. Potom program začíná:

```
1 REM xxxxxxxxxxxxxxx.....xxxxxxxxxx
```

s tolika x, kolik má strojový program bytů. Do paměti se strojový kód ukládá pomocí POKE. Kód je pak přístupný pomocí příkazu RANDOMIZE USR 23700 a může být nahrán na mgf pásek. Mimo to není příkazem RUN smazán.

Jiné místo, kam lze strojový kód uložit, jsou řetězcové proměnné (za systémovou proměnnou VARS).

Zadáme-li:

```
1 DIM a$(79)
```

může obsahovat 79 bytů. Pak můžeme vložit strojový kód postupně do a$(1), a$(2), a$(3), atd. Nevýhodou je, že se může smazat při RUN nebo CLEAR. A navíc se snadno můžeme zmýlit ve startovací adrese.

18.8 Debugging - odvšivení

Ve strojovém kódu není žádný příkaz pro postupné procházení programem. Program MC 2 Vám umožní redakci kódu, ale debugging to není. V této fázi je pro Vás nejlépe míti program ve strojovém kódu napsán na papíru, aby jste jej mohli lehce prohlížet.

Na základě zkušeností Vám doporučujeme navrhnout nejprve požadovaný program v BASICu, odstraníte všechny chyby a pak teprve sestavíte program ve strojovém kódu. Přitom v programu BASICovém používáme příkazy, které odpovídají příkazům strojového kódu. Jde to sice pomalu, ale nejjistěji dosáhnete cíle.

Pro praktické použití by bylo nejvhodnější doplnit pomocný program "MC 2" jednak sledem, který by dovolil prohlížet příkaz po příkazu a jednak o sled, který by dovolil zobrazení registrů na obrazovce. K tomu musíte :

a.)Napsat sled ve strojovém kódu,ve kterém všechny registry uložíte do zásobníku pomocí příkazu PUSH a ve formě hexadekadických čísel je zobrazíte na obrazovce.

b.)Mezi každý řádek vložíte CALL pro výpis napsaného sledu.

c.)Vypracujete způsob, jak se dostat zpátky do BASICu, jestliže si to ovšem přejete.

Příloha 1

Převod z hexadekadické soustavy do dekadické a naopak:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	128	-127	-126	-125	-124	-123	-122	-121	-120	-119	-118	-117	-116	-115	-114	-113
9	112	-111	-110	-109	-108	-107	-106	-105	-104	-103	-102	-101	-100	-99	-98	-97
A	-96	-95	-94	-93	-92	-91	-90	-89	-88	-87	-86	-85	-84	-83	-82	-81
B	-80	-79	-78	-77	-76	-75	-74	-73	-72	-71	-70	-69	-68	-67	-66	-65
C	-64	-63	-62	-61	-60	-59	-58	-57	-56	-55	-54	-53	-52	-51	-50	-49
D	-48	-47	-46	-45	-44	-43	-42	-41	-40	-39	-38	-37	-36	-35	-34	-33
E	-32	-31	-30	-29	-28	-27	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17
F	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
	až potud jsou dvojkově komplementární, od 0 obyčejná															
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Příloha 2

Tabulka pro rezervování paměťových míst :

Pozn.: Pro rezervování 100 bytových míst (bloků) na horejším konci RAM použijte následující adresy. Nezapomeňte dát před požadovanou adresou příkaz CLEAR.

počet rezervovaných bytů	16Kb adresa dec	16Kb adresa hex	49Kb adresa dec	49Kb adresa hex
100	32500	7EF4	65269	FEF4
200	32400	7E90	65169	FE90
300	32300	7E2C	65069	FE2C
400	32200	7DC8	64969	FDC8
500	32100	7D64	64869	FD64
600	32000	7D00	64769	FC90
700	31900	7C9C	64669	FC9C
800	31800	7C38	64569	FC38
900	31700	7BD4	64469	FBD4
1000	31600	7B70	64369	FB70
1100	31500	7B0C	64269	FB0C
1200	31400	7AA8	64169	FAA8

Příloha 3

Adresy systémových proměnných :

systémová proměnná	dec	hex
CHANS	23631	5C4F
PROG	23635	5C53
VARS	23627	5C4B
E-LINE	23641	5C59
WORKSP	23649	5C61
STKBOT	23651	5C63
STKEND	23653	5C65
RAMTOP	23730	5CB2
UDG	23675	5C7B
P-RAMT	23732	5CB4
CHARS	23606	5C36
BORDCR	23624	5C48
DFSZ	23659	5C6B
COORDS	23677-8	5C7D-E
DFCC	23684	5C84
DFCC-S	23685	5C85
SCR-CT	23692	5C8C
ATTR-P	23693	5C8D
pevné adresy		
DISPLAY FILE	16384	4000
ATRIBUT FILE	22528	5800
16 K RAMTOP	32599	7F57
48 K RAMTOP	65367	FF57
16 K UDG	32600	7F58
48 K UDG	65368	FF58
16 K RAMT	32767	7FFF
48 K P-RAMT	65535	FFFF
PRINTER BUFER	23296	5B00
SYSTEM VARIABLES	23552	5C00
MICRODRIVE MAPS	23734	5CD5
CHARACTER SET	15360	3C00

Příloha 4

Rozsah působení povelů Z-80

Seznam operačních kódů (mnemonika) s rozsahem jejich působení. Příkazy, které byly v pojednání,jsou označeny pouze stručně. Působení na příznaky je vypuštěno.

ADC	Sčítá včetně přenosového příznaku. Ukládá se v A nebo v HL.
ADD	Sčítá a nechává příznak přenosu nepovšimnut. Ukládá se v A nebo HL.
BIT	BIT b,r nastaví Z příznak odpovídající hodnotě b-tého bitu registru r.
CALL	Vyvolá podprogram. Podmíněné volání je signalizováno přídavným písmenem C (volej, když je přenosový příznak nastaven); M (když je znaménkový příznak nastaven - výsledek porovnávání je záporný); NC (když C příznak není nastaven); P (když není nastaven znaménkový příznak - výsledek je kladný); PE (když je nastaven příznak parity); PO (když není nastaven příznak parity); Z (když je nulový příznak nastaven); NZ (když není nulový příznak nastaven).
CCF	Komplement příznaku C (změní jeho bit v opačnou hodnotu).
CP	Porovnávání. Nastaví příznak jako odečítání od registru A, který nechává nezměněný.
CPD	Porovnání a dekrementace. Porovnání s HL a nato snížení registrů HL a BC o 1.
CPDR	Porovnání, dekrementace a opakování. Vyhledávání bloku programu. Jako u CPD, ale opakuje až je výsledek = 0 nebo registr BC dosáhne 0.
CPI	Jako CPD jen zvyšuje HL o 1. Registr BC o 1 snižuje.
CPIR	Jako CPDR, ale registr HL zvyšuje o 1.
CPL	Komplement registru A. Změní se každý bit v opačnou hodnotu.
DAA	Používá se při práci s binárně kódovanými čísly.
DEC	Dekrementace. Snížení o 1.
DI	Zamaskování přerušení.

DJNZ	Dekrementace a skok, není-li 0. Snižuje B o 1 a skočí relativně tehdy, je-li nastaven nulový příznak. Používá se jako smyčka FOR/NEXT v BASICu.
EI	Uvolnění přerušení.
EX	Výměna hodnot mezi AF a AF', HL a DE, HL a (SP), IX a (SP), IY a (SP).
EXX	Výměna registr. párů BC, DE, HL a BC', DE', HL'.
HALT	Přerušení činnosti CPU.
IM	Nastavení režimu přerušení.
IN	Přenos bytu z jiného zařízení.
INC	Inkrementace. Zvýšení o hodnotu 1.
IND	
INDR	
INI	Provádí přenos odpovídající LDD, LDDR, LDR, LDI, LDIR.
INIR	
JP	Skoky. Přidáním písmene C, M, NC, NZ, P, PE, PO a Z se mění na podmíněné skoky podobně jako CALL.
JR	Relativní skoky s následným posuvem podle údaje (1 byt). Podmíněné jsou jen s písmeny C,NC,NZ,Z.
LD	Ukládání. Lze použít všechny způsoby porovnání.
LDD	Pozor ! Není totéž jako LD D. Ukládá do registrového páru HL obsah adresy HL a sníží v HL, DE, BC obsah o 1.
LDDR	Ukládá, snižuje a opakuje. Pro přenos bloku. Provede LDD, jestliže BC dosáhne 0. Kopíruje paměťový blok, jehož délka je uložena v BC, přenáší z DE do HL.
LDI	Jako LDD, jen zvyšuje HL a DE. BC je snižováno.
LDIR	Jako LDDR, jen zvyšuje HL a DE.
NEG	Negace. Mění znaménko obsahu A registru.
NOP	Žádná operace. Nepřidává nic, jen spotřebuje čas.
OR	Logické OR na shodné bity. Je uloženo do A.
OTDR	
OTIR	

OUT	Blokové výstupy.
OUTD	
OUTI	
POP	Předání ze zásobníku do určeného registru.
PUSH	Předání z registru do zásobníku.
RES	Dá bit na nulovou hodnotu.
RET	Vrátit se z podprogramu do hlavního programu. Je možný podmíněný návrat odpovídající možnosti příkazu CALL.
RETI	Návrat z přerušení, respektive z maskovaného přerušení.
RETN	
RL	Rotace doleva. Něco jako posuv včetně příznaku C, jako by to byl osmý bit.
RLA	Rotace akumulátoru doleva. Jako RLA, ale s jiným působením na příznaky.
RLC	Pozor, nejedná se o RL C (rotace doleva)! Zde se 7-mý bit přenese do příznaku C a nulový bit je 0.
RLCA	Jako RLC A, ale se stejným rozdílem v příznaku jako RLA.
RLD	Užívá se pro binárně kódovaná dekadická čísla .
RR	Jako RL, ale doprava.
RRA	Jako RLA.
RRC	Jako RLC.
RRCA	Jako RLCA.
RRD	Jako RLD.
RST	Jako CALL, ale jen pro adresy 0H, 6H, 12H, 18H, 20H, 28H a 38H. U Spectra jsou všechna v ROM. RST 0 působí stejně, jako když se na okamžik vypne napájení počítače.
SBC	Odečítá s ohledem na přenosový příznak. Ukládá do A nebo HL.
SCF	Nasazuje přenosový příznak na "1".
SET	Nasazuje jeden bit na "1".

SLA Aritmetický posuv vlevo. Posouvá všechny bity o jedno místo doleva. Bit 0 je 0 a bit 7 se přenese do 0 nebo C.

SRA Aritmetický posuv doprava.Všechny bity se posunou o jedno místo doprava.

SRL Logický posun doprava. Všechny bity se posunou o jedno místo doprava. Sedmý bit = 0.

SUB Odečítání. Příznak se nemění. Ukládá se v A. Neexistuje příkaz SUB HL, r. Potřebujete-li jej, sesaďte přenosový příznak a použijte SBC.

XOR Exklusivní OR pro každý bit. Ukládá se v A.

Příloha 5

Nulové a přenosové příznaky.

V této příloze naleznete, kterými příkazy a jak je ovlivněn nulový příznak Z a přenosový příznak C. Symboly mají náaledujícíí význam :

- \- nezměněno
- X ovlivněno výsledkem operace
- 1 nasazen na "1"
- 0 nasazen na "0"
- \+ nasazen na "1",je-li A=(HL),jinak 0.

Příkaz	C	Z	Příkaz	C	Z	Příkaz	C	Z
ADC	X	X	IND	-	X	RLT	-	-
ADD(8-BIT)	X	X	INDR	-	1	RETI	-	-
ADD(16BIT)	X	-	INI	-	X	RETN	-	-
AND	0	X	INIR	-	1	RL	X	X
BIT	-	X	JP	-	-	RLA	X	-
CALL	X	X	JR	-	-	RLC	X	X
CCF	X	-	LD A,1	-	X	RLCA	X	-
CP	X	X	LD A,H	-	X	RLD	-	X
CPD	-	+	LD	-	-	RR	X	X
CPDR	-	+	LDD	-	-	RRA	X	-
CPI	-	+	LDDR	-	-	RRCA	X	-
CPIR	-	+	LDI	-	-	RRCA	X	-
JPL	-	-	LDIR	-	-	RRD	-	X
DAA	X	X	NEG	X	X	RST	-	-
DEC	-	-	NOP	-	-	SBC	1	-
DI	-	-	OR	0	X	SCF	1	-
DJNZ	-	-	OTBR	-	1	SET	-	-
EI	-	-	OTIR	-	1	SLA	X	X
EX	-	-	OUT	-	-	SRA	X	X
EXX	-	-	OUTD	-	X	SRL	X	X
HALT	-	-	OUTI	-	X	SUB	X	X
IM	-	-	POP	-	-	XOR	0	X
IN A,(N)	-	-	PUSH	-	-			
IN R,(C)	-	X	RES	-	-			

Příloha 6

Operační kódy Z-80

Příloha obsahuje kompletní seznam kódů Z80. V seznamu N znamená 1 byt, NN 2 byty. Symbol G znamená posun o 1 byt ve dvojkově komplementárním režimu. Dejte pozor na to, že nižší byt je před vyšším.

Příklad : LD BC, má op.kod 01nn, takže LD BC,732F je kódováno takto: 012F73. Nebo: LD A,(IY+d) má op.kod FD7E, takže LD A,(IY+07) je kódováno FD7E67.

Tabulka op.kódů je dle ZILOG Inc. Očíslování kódů obsahuje příloha A manuálu ZX Spectra, kde jsou pro memoniku použita malá písmena.

ADC A,(HL)	8E	ADD A,L	85	AND N	E6N
ADC A,(IX+G)	DD8EG	ADD A,N	86N	BIT 0,(HL)	CB46
ADC A,A	8F	ADD HL,DE	19	BIT 0,(IY+G)	FDCBG46
ADC A,(IY+G)	FD8EG	ADD HL,BC	09	BIT 0,(IX+G)	DDCBG46
ADC A,B	88	ADD HL,HL	29	BIT 0,A	CB47
ADC A,C	89	ADD HL,SP	39	BIT 0,B	CB40
ADC A,D	8A	ADD IX,BC	DD09	BIT 0,C	CB41
ADC A,E	8B	ADD IX,DE	DD19	BIT 0,D	CB42
ADC A,H	8C	ADD IX,IX	DD29	BIT 0,E	CB43
ADC A,L	8D	ADD IX,SP	DD39	BIT 0,H	CB44
ADC A,N	CEN	ADD IY,DC	FD09	BIT 0,L	CB45
ADC HL,BC	ED4A	ADD IY,DB	FD19	BIT 1,(HL)	CB4E
ADC HL,DE	ED5A	ADD IY,IY	FD29	BIT 1,(IX+G)	DDCBG4E
ADC HL,HL	ED6A	ADD IY,SP	FD39	BIT 1,(IY+G)	FDCBG4E
ADC HL,SP	ED7A	AND (HL)	A6	BIT 1,A	CD4F
ADD A,HL	8D	AND (IX+G)	DDA6G	BIT 1,B	CB48
ADD A,(IX+G)	DD86G	AND (IY+G)	FDA6G	BIT 1,C	CB49
ADD A,(IY+G)	FD86G	AND A	A7	BIT 1,D	CB4A
ADD A,A	87	AND B	A0	BIT 1,E	CB4B
ADD A,B	80	AND C	A1	BIT 1,H	CB4C
ADD A,C	81	AND D	A2	BIT 1,L	CB4D
ADD A,D	82	AND E	A3	BIT 2,(HL)	CB56
ADD A,E	83	AND H	A4	BIT 2,(IX+G)	DDCDG56
ADD A,H	84	AND L	A5	BIT 2,(IY+G)	FDCBG56
BIT 2,A	CB57	BIT 5,L	CB6D	CP C	B9
BIT 2,B	CB50	BIT 6,(HL)	CB76	CP D	BA
BIT 2,C	CB51	BIT 6,(IX+G)	DDCBG76	CP E	BB
BIT 2,D	CB52	BIT 6,(IY+G)	FDCBG76	CP H	BC
BIT 2,E	CB53	BIT 6,A	CB77	CP L	BD
BIT 2,H	CB54	BIT 6,B	CB70	CP N	FEN
BIT 2,L	CB55	BIT 6,C	CB71	CP D	EDA9
BIT 3,(HL)	CB5E	BIT 6,D	CB72	CPDR	EDB9
BIT 3,(IY+G)	FDCBG5E	BIT 6,H	CB74	CPIR	EDB1
BIT 3,A	CB5F	BIT 6,L	CB75	CPL	2F
BIT 3,B	CB58	BIT 7,(HL)	CB7E	DAA	27
BIT 3,C	CB59	BIT 7,(IX+G)	DDCBG7E	DEC (HL)	35
BIT 3,D	CB5A	BIT 7,(IY+G)	FDCBG7E	DEC (IX+G)	DD35G

BIT 3,E	CB5B	BIT 7,A	CB7F	DEC (IY+G)	FD35G
BIT 3,H	CB5C	BIT 7,B	CB79	DEC A	3D
BIT 3,L	CB5D	BIT 7,C	CB79	DEC B	05
BIT 4,(HL)	CB66	BIT 7,D	CB7A	DEC BC	0B
BIT 4,(IX+G)	DDCBG66	BIT 7,E	CB7B	DEC C	0D
BIT 4,(IY+G)	FDCBG66	BIT 7,H	CB7C	DEC D	15
BIT 4,A	CB67	BIT 7,L	CB7D	DEC DE	1B
BIT 4,B	CB60	CALL C,NN	DCNN	DEC E	1D
BIT 4,C	CB61	CALL M,NN	FCNN	DEC H	25
BIT 4,D	CB62	CALL NC,NN	D4NN	DEC HL	2B
BIT 4,E	CB63	CALL NN	CDNN	DEC IX	DD2B
BIT 4,H	CB64	CALL NZ,NN	C4NN	DEC IY	FD2B
BIT 4,L	CB65	CALL P,NN	F4NN	DEC L	2D
BIT 5,(HL)	CB6E	CALL PE,NN	ECNN	DEC SP	3B
BIT 5,(IX+G)	DDCBG6E	CALL PO,NN	E4NN	DI	F3
BIT 5,(IY+G)	FDCBG6E	CALL Z,NN	CCNN	DJNZ G	10G
BIT 5,A	CB6F	CCF	3F	EI	FD
BIT 5,B	CD66	CP (HL)	BE	EX (SP),HL	E3
BIT 5,C	CB69	CP (IX+G)	DDBEG	EX (SP),IX	DDE3
BIT 5,D	CB6A	CP (IY+G)	FDBEG	EX (SP),IY	FDE3
BIT 5,E	CB6B	CP A	BF	EX AF,AF'	08
BIT 5,H	CB6C	CP B	B8	EX DE,HL	ES
EXX	D9	JP M,NN	FANN	LD (IY+G),L	FD75G
HALT	76	JP NC,NN	D2NN	LD (IY+G),N	FD36GN
IM 0	ED46	JP NN	C3NN	LD (NN),A	32NN
IM 1	ED56	JP NZ,NN	C2NN	LD (NN),DC	ED43NN
IM 2	ED5E	JP P,NN	F2NN	LD (NN),DE	ED53NN
IN A,(C)	ED78	JP PE,NN	EANN	LD (NN),HL	22NN
IN A,(N)	DBN	JP PO,NN	E2NN	LD (NN),IX	LD22NN
IN B,(C)	ED40	JP Z,NN	CANN	LD (NN),IY	FD22NN
IN C,(C)	ED49	JR C,G	38G	LD (NN),SP	ED73NN
IN D,(C)	ED50	JR G	18G	LD A,(BC)	0A
IN E,(C)	ED58	JR NC,G	30G	LD A,(DE)	1A
IN H,(C)	ED60	JR NZ,G	20G	LD A,(HL)	7E
IN L,(C)	ED68	JR Z,G	28G	LD A,(IX+G)	DD7EG
INC (HL)	34	LD (BC),A	02	LD A,(IY+G)	FD7EG
INC (IX+G)	DD34G	LD (DE),A	12	LD A,(NN)	3ANN
INC (IY+G)	FD34G	LD (HL),A	77	LD A,A	7F
INC A	3C	LD (HL),B	70	LD A,B	78
INC B	04	LD (HL),C	71	LD A,C	79
INC BC	03	LD (HL),D	72	LD A,D	7A
INC C	0C	LD (HL),E	73	LD A,E	7B
INC D	14	LD (HL),H	74	LD A,H	7C
INC DE	13	LD (HL),L	75	LD A,I	ED57
INC E	1C	LD (HL),N	36N	LD A,L	7D
INC H	24	LD (IX+G),A	DD77G	LD A,N	3EN
INC HL	23	LD (IX+G),B	DD70G	LD B,(HL)	46
INC IX	DD23	LD (IX+G),C	DD71G	LD B,(IX+G)	DD46G
INC IY	FD23	LD (IX+G),D	DD72G	LD B,(IY+G)	FD46G
INC L	2C	LD (IX+G),E	DD73G	LD B,A	47
INC SP	33	LD (IX+G),H	DD74G	LD B,B	40
IND	EDAA	LD (IX+G),L	DD75G	LD B,C	41
INDR	EDBA	LD (IX+G),N	DD36GN	LD B,D	42
INI	EDA2	LD (IY+G),A	FD77G	LD B,E	43
INIR	EDB2	LD (IY+G),E	FD70G	LD B,H	44

JP (HL)	E9	LD (IY+G),C	FD71G	LD B,L	45
JP (IX)	DDE9	LD (IY+G),D	FD72G	LD B,N	06N
JP (IY)	FDE9	LD (IY+G),E	FD73G	LD BC,(NN)	ED4BNN
JP C,NN	DANN	LD (IY+G),H	FD74G	LD BC,NN	01NN
LD C,(HL)	4E	LD H,(IY+G)	FD66G	NOP	00
LD C,(IX+G)	DD4EG	LD H,A	67	OR (HL)	B6
LD C,(IY+G)	FD4EG	LD H,B	60	OR (IX+G)	DDB6G
LD C,A	4F	LD H,C	61	OR (IY+G)	FDB6G
LD C,A	48	LD H,D	62	OR A	B7
LD C,C	49	LD H,E	63	OR B	B0
LD C,D	4A	LD H,H	64	OR C	B1
LD C,E	4B	LD H,L	65	OR D	B2
LD C,H	4C	LD H,N	26N	OR E	B3
LD C,L	4D	LD HL,(NN)	2ANN	OR H	B4
LD C,N	0EN	LD HL,NN	21NN	OR L	B5
LD D,(HL)	56	LD I,A	ED47	OR N	F6N
LD D,(IX+G)	DD56G	LD IX,(NN)	DD2ANN	OTDR	EDBB
LD D,(IY+G)	7D56G	LD IX,NN	DD21NN	OTIR	EDB3
LD D,A	57	LD IY,(NN)	FD2ANN	OUT (C),A	ED79
LD D,B	50	LD IY,NN	FD21NN	OUT (C),B	ED41
LD D,C	51	LD L,(HL)	6E	OUT (C),C	ED49
LD D,D	52	LD L,(IX+G)	DD6EG	OUT (C),D	ED51
LD D,E	53	LD L,(IY+G)	FD6EG	OUT (C),E	ED59
LD D,H	54	LD L,A	6F	OUT (C),H	ED61
LD D,L	55	LD L,B	68	OUT (C),L	ED69
LD D,N	16N	LD L,C	69	OUT (N),A	B3N
LD DE,N	11NN	LD L,E	6B	OUTI	EDA3
LD E,(HL)	5E	LD L,H	6C	POP AF	F1
LD E,(IX+G)	DD5EG	LD L,L	6D	POP BC	C1
LD E,(IY+G)	FD5EG	LD L,N	2EN	POP DE	D1
LD E,A	5F	LD SP,(NN)	ED7BNN	POP HL	E1
LD E,B	58	LD SP,HL	F9	POP IX	DDE1
LD E,C	59	LD SP,IX	DDF9	POP IY	FDE1
LD E,D	5A	LD SP,IY	FDF9	PUSH A	F5
LD E,E	5B	LD SP,NN	31NN	PUSH BC	C5
LD E,H	5C	LDD	EDA8	PUSH DE	D5
LD E,L	5D	LDDR	EDB8	PUSH HL	E5
LD E,N	1EN	LDI	EDA0	PUSH IX	DDE5
LD E,N	66	LDIR	ED80	PUSH IY	FDE1
LD H,(IX+G)	DD66G	NEG	ED44	RES 0,(HL)	CBC6
RES 0,(IX+G)	DDCBG86	RES 3,H	CB9C	RES 7,C	CB89
RES 0,(IY+G)	FDCBG86	RES 3,L	CB9D	RES 7,D	CB8A
RES 0,A	CB87	RES 4,(HL)	CBA6	RES 7,E	CB8B
RES 0,B	CB80	RES 4,(IF+G)	DDCBGA6	RES 7,H	CB8C
RES 0,C	CB81	RES 4,(IY+G)	FDCBGA6	RES 7,L	CB8D
RES 0,D	CB82	RES 4,A	CBA7	RET	C9
RES 0,E	CB83	RES 4,B	CBA0	RET C	D8
RES 0,H	CB84	RES 4,C	CBA1	RET M	F8
RES 0,L	CB85	RES 4,D	CBA2	RET NC	D0
RES 1,(HL)	CB8E	RES 4,E	CBA3	RET NZ	C0
RES 1,(IX+G)	DDCBG8E	RES 4,H	CBA4	RET P	F0
RES 1,(IY+G)	FDCBG8E	RES 4,L	CBA5	RET PE	E8
RES 1,A	CB8F	RES 5,(HL)	CBAE	RET PO	E0
RES 1,B	CB88	RES 5,(IX+G)	DDCBGAE	RET Z	C8
RES 1,C	CB89	RES 5,(IY+G)	FDCBGAE	RETI	ED4D

RES 1,D	CB9A	RES 5,A	CBAF	RETN	ED45
RES 1,E	CB8B	RES 5,B	CBA8	RL (HL)	CB16
RES 1,H	CB8C	RES 5,C	CBA9	RL (IX+G)	DDCBG16
RES 1,L	CB8D	RES 5,D	CBAA	RL (IY+G)	FDCBG16
RES 2,(HL)	CB96	RES 5,E	CBAB	RL A	CB17
RES 2,(IX+D)	DDCBG96	RES 5,H	CBAC	RL B	CB10
RES 2,(IY+G)	FDCBG96	RES 5,L	CBAD	RL C	CB11
RES 2,A	CB97	RES 6,(HL)	CBB6	RL D	CB12
RES 2,B	CB90	RES 6,(IX+G)	DDCBGB6	RL E	CB13
RES 2,C	CB91	RES 6,(IY+G)	FDCBGB6	RL H	CB14
RES 2,D	CB92	RES 6,A	CBB7	RL L	CB15
RES 2,E	CB93	RES 6,B	CBB0	RLA	17
RES 2,H	CB94	RES 6,C	CBB1	RLC (HL)	CB06
RES 2,L	CB95	RES 6,D	CBB2	RLC (IX+G)	DCCBG06
RES 3,(HL)	CD9E	RES 6,E	CBB3	RLC (IY+G)	FDCBG06
RES 3,(IX+G)	DDCBG9E	RES 6,H	CBB4	RLC A	CB07
RES 3,(IY+G)	FDCBG9E	RES 6,L	CBB5	RLC B	CD00
RES 3,A	CB92	RES 7,(HL)	CDBE	RLC C	CD01
RES 3,B	CB98	RES 7,(IX+G)	DDCBGBE	RLC D	CD02
RES 3,C	CB99	RES 7,(IY+G)	FDCBGBE	RLC E	CB03
RES 3,D	CB9A	RES 7,A	CBBF	RLC H	CB04
RES 3,E	CB9B	RES 7,B	CBB9	RLC L	CB05
RLCA	07	SBC A,D	98	SET 2,C	CBD1
RLD	ED6F	SBC A,C	99	SET 2,D	CBD2
RR (HL)	CB1E	SBC A,D	9A	SET 2,E	CBD3
RR (IX+G)	DDCBG1E	SBC A,E	9B	SET 2,H	CBD4
RR (IY+G)	FBCDG1E	SBC A,H	9C	SET 2,L	CBD5
RR A	CB1F	SBC A,L	9D	SET 3,(HL)	CBDE
RR B	CB18	SBC A,N	DEN	SET 3,(IX+G)	DDCBGDE
RR C	CB19	SBC HL,BC	ED42	SET 3,(IY+G)	FDCBGDE
RR D	CB1A	SBC HL,DE	ED52	SET 3,A	CBDF
RR E	CBAB	SBC HL,HL	ED62	SET 3,B	CBD8
RR H	CB1C	SBC HL,SP	ED72	SET 3,C	CBD9
RR L	CB1D	SCF	37	SET 3,D	CBDA
RRA	1F	SET 0,HL	CBC6	SET 3,E	CBDB
RRC (HL)	CB0E	SET 0,(IX+G)	DDCBGC6	SET 3,H	CBDC
RRC (IX+G)	DDCBG0E	SET 0,(IY+G)	FDCBGC6	SET 3,L	CSDD
RRC (IY+G)	FDCBG0E	SET 0,A	CBC7	SET 4,(HL)	CBE6
RRC A	CB0F	SET 0,B	CBC0	SET 4,(IX+G)	DDCBGE6
RRC B	CB08	SET 0,C	CBC1	SET 4,(IY+G)	FDCBGE6
RRC C	CB09	SET 0,D	CBC2	SET 4,A	CBE7
RRC D	CB0A	SET 0,E	CBC3	SET 4,B	CBE0
RRC E	CB0B	SET 0,H	CBC4	SET 4,C	CBE1
RRC H	CB0C	SET 0,L	CBC5	SET 4,D	CBE2
RRC L	CB0D	SET 1,(HL)	CBCE	SET 4,E	CBE3
RRCA	0F	SET 1,(IX+G)	DDCBGCE	SET 4,H	CBE4
RRD	ED67	SET 1,(IY+G)	FDCBGCE	SET 4,L	CBE5
RST 0	C7	SET 1,A	CBCF	SET 5,(HL)	CBEE
RST 10H	D7	SET 1,B	CBC9	SET 5,(IX+G)	DDCBGEE
RST 18H	DF	SET 1,C	CBC9	SET 5,(IY+G)	FDCBGEE
RST 20H	E7	SET 1,D	CBCA	SET 5,A	CDEF
RST 28H	EF	SET 1,E	CBCB	SET 5,B	CBE8
RST 30H	F7	SET 1,H	CBCC	SET 5,C	CBE9
RST 38H	FF	SET 1,L	CBCD	SET 5,D	CBEA
RST 8	CF	SET 2,(HL)	CBD6	SET 5,E	CBEB

SBC A,(HL)	9E	SET 2,(IX+G)	DDCBGD6	SET 5,H	CBEC
SBC A,(IX+G)	DD9EG	SET 2,(IY+G)	FDCBGB6	SET 5,L	CBED
SBC A,(IY+G)	FD9EG	SET 2,A	CBD7	SET 6,(HL)	CBF6
SBC A,A	9F	SET 2,B	CBD0	SET 6,(IX+G)	DDCBG76
SET 6,(IY+G)	FDCBGF6	SLA C	CB21	SRL H	CB3C
SET 6,A	CBF7	SLA D	CB22	SRL L	CB3D
SET 6,B	CBF0	SLA E	CB23	SUB HL	96
SET 6,C	CBF1	SLA H	CB24	SUB (IX+G)	DD96G
SET 6,D	CBF2	SLA L	CB25	SUB (IY+G)	FD96G
SET 6,E	CBF3	SRA HL	CB2E	SUB A	97
SET 6,H	CBF4	SRA (IX+G)	DDCBG2E	SUB B	90
SET 6,L	CBF5	SRA (IY+G)	FDCBG2E	SUB C	91
SET 7,(HL)	CBFE	SRA A	CB2F	SUB D	92
SET 7,(IX+G)	DDCBGFE	SRA B	CB29	SUB E	93
SET 7,(IY+G)	FDCBGFE	SRA C	CB29	SUB H	94
SET 7,A	CBFF	SRA D	CB2A	SUB L	95
SET 7,B	CBF8	SRA E	CB28	SUB N	D6N
SET 7,C	CBF9	SRA H	CB2C	XOR HL	AE
SET 7,D	CBFA	SRA L	CB2D	XOR (IX+G)	DDAEG
SET 7,E	CBFB	SRL (HL)	CB3E	XOR (IY+G)	FDAEG
SET 7,H	CBFC	SRL (IX+G)	DDCBG3E	XOR A	AF
SET 7,L	CBFD	SRL (1Y+G)	FDCBG3E	XOR H	A8
SLA (HL)	CB26	SRL A	CB3F	XOR C	A9
SLA (IX+G)	DDCBG26	SRL B	CB38	XOR D	AA
SLA (IY+G)	FDCBG26	SRL C	CB39	XOR E	AB
SLA A	CB27	SRL D	CB3A	XOR H	AC
SLA B	CB20	SRL E	CB3B	XOR L	AD
				XOR N	EEN

Příloha 7

Pomocný program MC 2

Tento program má vícenásobné využití při redakci, zápisu a provozování programů ve strojovém kódu. Je upraven s ohledem na vlastnosti počítače ZX Spectrum.

Postupujte takto:

1. Vložte do Spectra program a zajistěte jej nahráním na MGF pásku, pomocí : SAVE "MC 2" LINE 5.
2. Spusťte tento program pomocí RUN. Je třeba rezervovat počet paměťových míst podle délky programu ve strojovém kódu. Standardní adresa daná programem je 32000. Chcete-li jinou startovací adresu, vložíte ji na dotaz "Chcete jinou startovací adresu ?". Všechno mimo ENTER je vyloženo jako odpověď.
3. Dále je vyžádán počet rezervovaných bytů na počátku programu. Na tato místa musíte vložit hodnoty pomocí POKE.
4. Pak se ukáže skutečná startovací adresa, použitá pro program ve strojovém kódu.
5. Nato se vyžádá MC 2 hexakód a vytvoří se červený kurzor jako blikající značka >.
6. Nyní můžete vkládat strojový program v hexakódech a manipulovat s programem podle dále uvedených povelů.Při zadávání každé kódové skupiny vypouští program mezery. Můžete je dělat, chcete-li, dělí kódy po dvou a zobrazuje je v 10 sloupcích. Každá kódová skupina je ohraničena dvěma hvězdičkami, kurzor je na konci poslední vložené. Např. při vložení "--al-e23---4" se ukáže :

al e2 34 **>

stejně jako když vložíte "ale234" nebo "al E2 34".

7. Nyní může být vložena další skupina kódů. Bude bezprostředně navazovat na kurzor. Dvě hvězdičky mezi skupinami slouží jen pro pohodlí uživatele a zůstávají počítačem nepovšimnuty. Jsou-li kódy vkládány přes RAMTOP, nemusí skupiny povelů Z80 odpovídat, úpravu provedete při přezkoušení.
8. Konečně mohou být uplatněny ovládací povely, a to prvním písmenem. Další znaky za ním zůstávají nepovšimnuty, kromě dále popsaných případů:

G (GO)	spusť sled strojového kódu
L (LOAD)	ulož sled strojového kodu
M (MOVE)	pohyb kurzoru dopředu
N (negative)	pohyb kurzoru dozadu
P (PRINT)	ukaž seznam hexakódů
S (SAVE)	nahrání programu
X (EXCISE)	vymazání ze seznamu hexakódů

Popis povelů :

M,N Povel MA, při čemž A je číslo, které posune kurzor o A míst dopředu, NA jej pouze posune o A míst zpět. Při posunu se seznam kódů neruší. Není-li dáno číslo A bude A=1.

X Povel XA, kde A je číslo větší nebo rovné 0, maže za kurzorem A nejbližší pár hexkódů včetně hvězdiček. Zobrazení zůstává nedotčeno, dokud není stisknuto P. Chybí-li údaj A je A=1.

P Zobrazí seznam hexkódů a dává kurzor na počátek.

S Uloží program na MGF pásek, včetně seznamu hexkódů. Volba Vám dovoluje pojmenovat program podle přání. Abyste mohli program zajistit, nahrajte jej na pásek pomocí SAVE předtím než zvolíte "F" nebo "G". Pak jej nahrávejte pomocí LOAD v BASICu, poté zadejte GOTO 200 (ne RUN !) a pak můžete zadávat "F" nebo "G".

L Uloží program strojového kódu bez omezení hvězdičkami za RAMTOP počínaje hodnotou, kterou jsme pro RAMTOP zvolili. Automaticky vkládá RET pro návrat.

G Spustí sled a vrátí se do pomocného programu MC 2, pokud není ve strojovém kódu vada.

R Je užitečný povel, ulehčující relativní skoky. Dělat je ručně je únavné, neboť musíte posun vypočítat a dosadit v hexasoustavě dvojkově komplementárně. Pro program je to zjednodušení. Dělá se to takto :

a. Vložte program ve strojovém kódu a nasaďte všechny hodnoty relativních skoků na 00.
b. Abyste mohli nasadit správnou hodnotu pro relativní skok, dejte kurzor bezprostředně před 00 a vymažte stiskem "X". Vložte nyní RN, přičemž číslo N je pozice cíle (dekadicky) odečtené z obrazovky a to následovně :
Očíslujte řady a sloupce hexkódů od nuly takto:

```
  0  1  2  3  4  5  6  7  8  9
0

1

2

:
```

a zadejte N=XY pro řadu X a sloupec Y. (T.j. pro byt v řadě 17 a sloupci 5 dáte např. r175.) Skutečnost, že řada má 10 sloupců, vám ulehčí odečítání. Nyní můžete pozměnit program tak, že dáte kurzor na cíl a skok se provede odsud. V praxi to trvá déle, protože musíte trvale pohybovat kurzorem.

c. Nyní stiskněte "P" a obdržíte revidovaný seznam, včetně správných hodnot skoků.
d. Nastavte kurzor na další relativní skok a postup opakujte.
e. Dejte jenom pozor na to, že program bere automaticky ohled na dělení dvěmi hvězdičkami a pracuje v dvojkově komplementárním kódu. V případě, že skok jde mimo takto stanovenou hranici, program MC 2 to ohlásí.
f. Třeba to pro Vás zní komplikovaně a proto se použije následující názorný příklad skrolování sloupce z kapitoly 14. Vložte řadu skupin hex-kódů. Výsledek by měl vypadat takto :

```
RAMTOP:              31999
Rozsah stroj.kódu:32000
Data:                32000 do 31999
RUN USR:             32000
Hexkódy:

   01 20 ** dd 21 1f 58 ** 3c 15 ** dd 56 20
   ** dd 72 00 ** dd 09 ** 3d ** b8 ** 20 00 **
                                        --
```

Podtržená 00 je velikost relativního skoku, která se má vypočítat. (Můžete použít i jiný příklad).

Pomocí "N" usadíte kurzor před poslední dvojicí 00 takto: dd 09 ** 3d ** b8 ** 20 00 **
a stiskněte "X", čímž 00 smažete. Protože příkaz zní: JRNZ SMYČKA a smyčka začíná u kódu dd (dd 56 20) ve druhé řadě kódů. Je to tedy řada 1, sloupec 2, takže musíme zadat "r12". Proveďte to. Krátce nato zůstane obrazovka chvíli prázdná a pak se vytvoří nový seznam. 00 je nahrazeno f4, což je správný relativní skok.

Program MC 2

```
5 REM MC 2 (HEX EDITOR,LOADER AND PARTIAL ASSEMBLER)
7 POKE 23609,50
10 INPUT "CHCETE JINOU STARTOVACI ADRESU (A/N) ";A$
20 IF A$="N" THEN LET RT=32000:GOTO 40
30 IF A$<>"" THEN INPUT "STARTOVACI ADRESA ?";RT
40 CLEAR RT-1
50 LET RT=PEEK 23730+256*PEEK 23731+1
60 PRINT "RAMTOP:";RT-1
70 PRINT "OBLAST STROJOVEHO KODU:";RT
80 INPUT "POCET REZERVOVANYCH BYTU";D
90 PRINT "DATA:     AZ    ";RT+D-1
97 LET B=0
100 PRINT "RUN USR: ";RT+D
110 PRINT PAPER 6;"HEXKODY:"
120 LET CI=0
130 LET H$=""
140 GOSUB 400
```

```
150 DIM F(20)
151 LET F(1)=700
152 LET F(6)=800
153 LET F(7)=900
154 LET F(8)=1000
155 LET F(10)=1100
156 LET F(11)=1200
157 LET F(12)=1300
158 LET F(13)=1400
159 LET F(14)=1500
160 LET F(18)=1600
200 INPUT I$
210 LET A=0
220 LET A=A+1
230 IF A>LEN I$ THEN GOTO 300:GOSUB 700
240 IF I$(A)<>" " THEN GOTO 220
250 LET I$=I$ (TO N-1) + I$ (N+1 TO)
260 GOTO 230
300 IF CODE I$(1)=103 THEN GOTO 500
310 LET I$=I$+"**"
320 LET H$=H$ (TO 2*CI)+I$+H$ (2*CI+1 TO)
330 GOSUB 450
340 GOSUB 600
350 GOSUB 450
360 GOTO 200
400 REM UKAZAT KURZOR
410 PRINT AT  5+INT(CI/10),3*(CI-10*INT(CI/10));FLASH  1;INK  2;
    ">";
420 RETURN
450 REM MAZANI KURZOREM
460 PRINT AT 5+INT(CI/10),3*(CI-10*INT(CI/10));" ";
470 RETURN
500 REM SLEDY Z KLAVESNICE
510 GOSUB F(VODE I$(1)-102)
520 GOTO 200
600 REM ZOBRAZENI VLOZENEHO TEXTU
610 FOR J=1 TO LEN I$(2)
620 PRINT I$(2*J-1 TO 2*J)+"-";
630 LET CI=CI+1
640 IF CI=10*INT(CI/10) THEN PRINT "      ";
650 NEXT J
660 RETURN
700 REM BEZ KODU
705 IF B=0 THEN PRINT "NEBYL VLOZEN KOD":RETURN
710 CLS
720 LET Y=USR (RT+1)
730 RETURN
800 REM UKLADANI ZA RAMTOP
810 LET H$=H$+"C9"
820 LET J=RT+D-1
830 LET I=-I
840 LET J=J+1
850 LET I=I+2
860 IF I>LEN H$ THEN LET B=J-RT:RETURN
870 IF H$(I)="*" THEN GOTO 850
880 POKE J,16*(CODE H$(I)-49-39*(H$(I)>"9"))+CODE  H$(I+1)-48-39
    *(H$(I+1)>"9")
```

```
890 GOTO 840
900 REM POHYB KURZOREM DOPREDU
910 GOSUB 450
920 IF LEN I$=1 THEN LET CM=1
930 IF LEN I$>1 THEN LET CM=VAL I$(5 TO)
940 LET CI=CI+CM
950 IF CI>LEN H$(2) THEN LET CI=LEN H$(2)
960 GOSUB 400
970 RETURN
1000 REM POHYB KURZOREM VZAD
1010 GOSUB 450
1020 IF LEN I$=1 THEN LET CM=1
1030 IF LEN I$>1 THEN LET CM=VAL I$(2 TO)
1040 LET CI=CI-CM
1050 IF CI<0 THEN LET CI=0
1060 GOSUB 400
1070 RETURN
1100 REM ZOBRAZENI
1110 PRINT AT 5,0;
1120 FOR R=1 TO 17:PRINT "32xMEZERU";:NEXT R
1130 PRINT AT 5,0;" ";
1140 LET CI=0
1150 FOR J=1 TO LEN H$(2)
1160 PRINT H$(2*J-1 TO 2*J)=" ";
1170 LET CI=CI+1
1180 IF CI=10*INT (CI/10) THEN PRINT " ";
1185 NEXT J
1190 GOSUB 400
1195 RETURN
1300 REM RELATIVNI SKOKY
1310 LET JCI=VAL D$(2 TO)
1320 LET JS=JCI-CI-1
1325 GOSUB 2000
1330 IF JS>=-128 AND JS<=127 THEN GOSUB 1355
1340 INPUT "NEPLATNA VELIKOST ,POKRACUJ S ENTER";A$
1350 RETURN
1355 IF JS<0 THEN LET JS=JS+256
1360 LET X1=INT (JS/16)
1365 LET X0=JS-16*X1
1367 LET X$=CHR$(X1+48+35*(X1 >9))+CHR$(X0+48+39*(X0>9))
1370 LET H$=H$(TO 3*CI)+X$+H$(2*CI+1 TO)
1375 LET CI=CI+1
1380 GOSUB 1100
1390 RETURN
1400 REM NAHRANI NA MGF PASEK
1410 INPUT "SAVE-NAZEV-";N$
1420 IF N$="" THEN LET N$=" MC 2 "
1430 IF B=0 THEN PRINT "NEBYL VLOZEN KOD !":RETURN
1460 RETURN
1600 REM SMAZANI
1610 IF LEN I$=1 THEN LET K=1
1620 IF LEN I$>1 THEN LET K=VAL I$(2 TO)
1630 LET H$=H$(TO 2*CI)+H$(2*CI+2*K+1 TO)
1640 RETURN
2000 REM PRIZPUSOBENI
2010 IF J$<0 THEN LET W$=H$(2*JCI+1 TO 2*CI)
2020 IF J$>=0 THEN LET U$=H$(2*CI+1 TO 2*JCI)
```

```
2030 LET SC=0
2040 FOR T=1 TO LEN W$
2050 IF W$(T)="*" THEN LET SC=SC+1
2060 NEXT T
2070 IF JS<0 THEN LET JS=JS+SC/2
2080 IF JS>0 THEN LET KS=JS-SC/2
2090 RETURN
9800 RESTORE 9970
9810 FOR F=1 TO 10:READ P$
9820 FOR N=0 TO 7
9830 READ A:POKE USR P$+N,A
9840 NEXT N:NEXT F
9970 DATA "I",9,16,0,48,16,16,56,0
9980 DATA "T",16,40,84,62,68,,68,56,0
9890 DATA "A",8,16,56,4,60,68,69.0
9900 DATA "O",8,16,56,68,68,56,0
9910 DATA "C",40,16,60,64,64,64,60,0
9920 DATA "D",40,16,124,8,16,32,124,0
9930 DATA "E",9,16,56,69,124,64,60,0
9940 DATA "J",40.16,56,69,124,64,60,0
9950 DATA "R",40,16,56,64,64,64,64,0
9960 DATA "L",8,16,68,68,69,69,4,56,:GOTO 5
```

Poznámka : Písmena pod DATA "X" zadávat v grafickém módu.

Natištěno pro klub výp.techniky při ZO SVAZARM Karolinka

- = K O N E C = -

Vytiskly MTZ 22 Nový Jičín