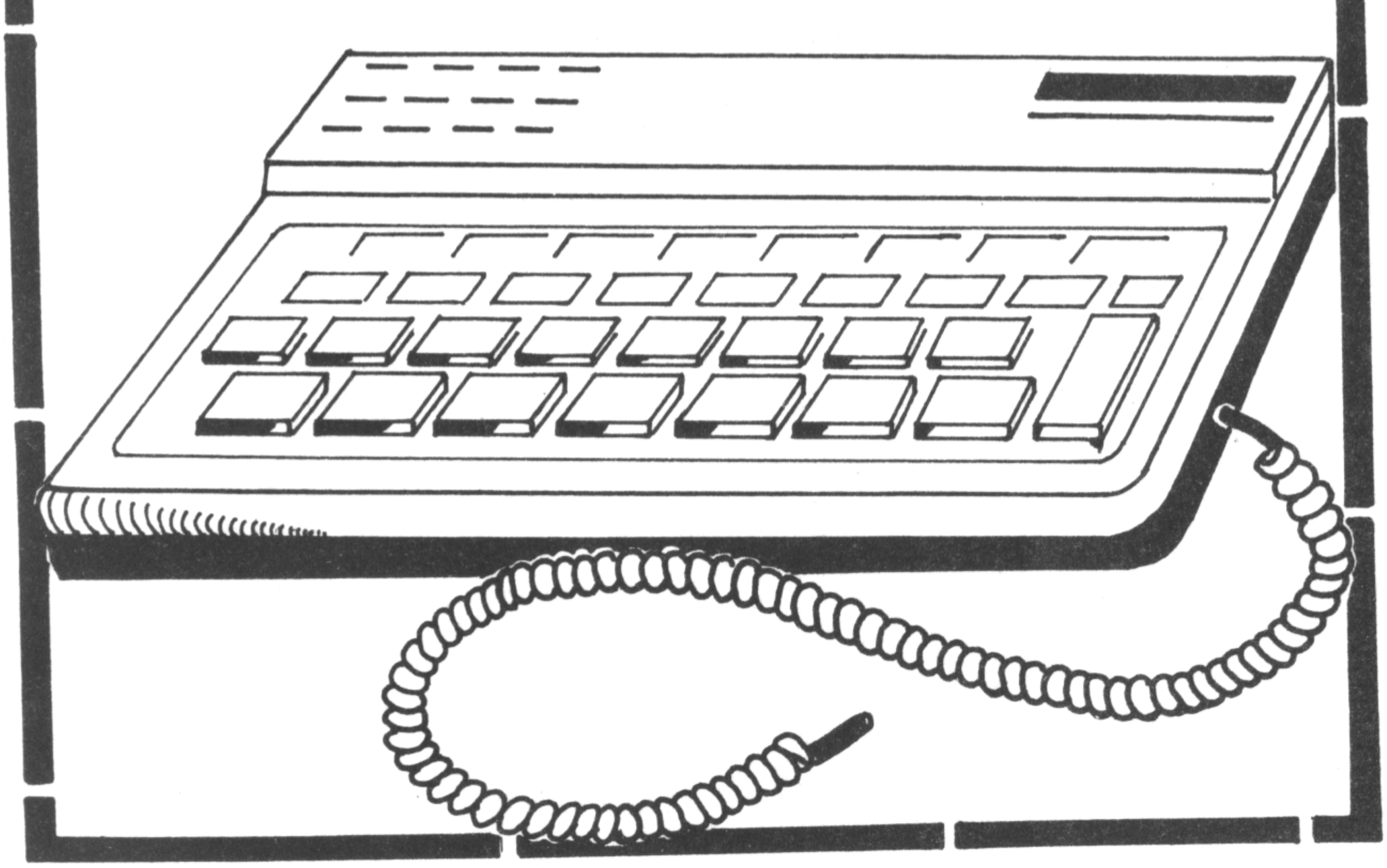
SINCLAIR

609

03

SVAZARM



Náš test:

Síla šachových programů

Herní sílu šachistů nejlépe vyjadřuje systém osobních koeficientů, zavedených mezinárodní šachovou federací (FIDE), podle návrhu profesora Ela. Odtud je zvykem nazývat toto hodnocení jako ELO body (ELO rating). Na jakých základech je vybudován systém ELO ? V podstatě na následujících čtyřech:

1. Osobní koeficient ELO je celé číslo tím vyšší, čím je šachista silnější.
2. Rozdíl ELO bodů soupeřů vyjadřuje pravděpodobnost jejich úspěchu takto:
   -pokud mají stejné ELO (tj. rozdíl je nula), mají oba pravděpodobnost úspěchu 50%,
   -při rozdílu 200 bodů má silnější strana 75% naději na úspěch.
3. Pravděpodobnost úspěchu má vzhledem k rozdílu ELO normální Gaussovo rozdělení s parametry takovými, aby vyhovovalo druhému axiomu. Tomu odpovídá střední hodnota 0 a směrodatná odchylka zhruba 285.
4. Rozdíl 200 bodů odpovídá jednomu výkonnostnímu stupni resp. třídě. Základní hodnota pro stupeň mezinárodního hráče je 2200 ELO bodů.

Axiomy 1. až 3. nepotřebují komentář. Hodnoty v čtvrtém mohly být libovolné. Vzhledem k tomu, že pro určení pravděpodobnosti úspěchu je důležitý pouze rozdíl ELO soupeřů, mohla být místo hodnoty 2200 zvolena libovolná jiná, např. 5000. Rovněž pro rozdíl třídy mohla být stanovena jiná hodnota. V tom případě by se pochopitelně tato hodnota musela objevit i v druhém axiomu. Na základě výše uvedených axiomů jsou vypočteny tabulky technických norem pro získání mezinárodních titulů. Na stejném principu je založena i národní klasifikace ve většině členských států FIDE, včetně naší republiky.

Hodnota 2200 odpovídá u nás průměrné úrovni hráče mistrovské třídy (jinak běžněji kandidát mistra sportu - KM). Průměrná úroveň znamená, že v turnaji s průměrnou hodnotou ELO 2200 stačí zisk 50% bodů ke splnění mistrovské výkonnostní třídy.

U nás jsou následující výkonnostní třídy (zkratka VT) s příslušnou průměrnou hodnotou ELO v závorce:
4.VT (1400), 3.VT (1600), 2.VT (1800), 1.VT (2000),
KM (2200), MS - mistr sportu (2400).
V mezinárodní klasifikaci jsou to stupně:
mezinárodní mistr - IM (2400) a
mezinárodní velmistr - IGM (2600).
Kromě těchto byl zaveden titul mistr FIDE - M-FIDE (2350), který poněkud nezapadá do systému. Obdobně náš mistr sportu je vlastně stupeň mezinárodního mistra v národním měřítku.

Aby bylo možno hráče ohodnotit ELO body, musí sehrát určitý počet partií proti hráčům, kteří již ELO mají. To platí jak pro mezinárodní, tak i pro národní klasifikaci.

Pro ilustraci: nejvyšší ELO v současné době má mistr světa Garri Kasparov, a to 2740. Nejlepší žena - mistryně světa Maja Ciburdanidzeová 2550. Nejlepší šachové automaty - 2300.

Sílu šachových programů se pokusíme také vyjádřit pomocí ELO bodů. Pokud programy mezi sebou sehrají dostatečně mnoho

partií, lze na základě dosažených výsledků vypočítat rozdíly jejich ELO bodů. Když určíme nejméně u jednoho programu jeho ELO ve škále výkonnostních stupňů živých šachistů, uvedené výše, získáme ELO i pro ostatní programy. Za tento etalon vzal autor program COLOSSUS 3.0, realizovaný na počítači ATARI 800 XL. V originálním návodu je uvedena jednak síla v ELO (1800) a také výsledky proti některým programům realizovaným na Spectru. Navíc si tento program zahrál i proti dalším testovaným programům; šlo o následující programy na Spectru:

A. COLOSSUS CHESS 4.0
B. SUPERCHESS 3.5
C. SUPERCHESS 3.0
D. CYRUS CHESS 48K
E. PSI CHESS
F. TURK CHESS 1.3
G. SPECTRUM CHESS 2 (od firmy Artic)
H. MASTERCHESS 2

Uvedený soubor zahrnuje nejznámější realizace algoritmu šachové hry na Spectru. Reprezentuje i vývoj tohoto speciálního software v čase. Nové či neuvedené realizace si čtenáři mohou sami ocenit, když je nechají hrát proti těmto "reprezentantům".

Výpočty vzájemných rozdílů ELO byly provedeny na základě výsledků 114 partií. Z toho 64 partií bylo převzato z firemního manuálu k etalonovému programu (viz výše) a 50 bylo sehráno mezi dvěma počítači. Aby partie měly určitou hodnotu a objektivně odrážely sílu programu, byly sehrány vždy až do úplného konce, daného pravidly šachu. Stupně obtížnosti byly vždy voleny tak, aby program měl minimálně 1 minutu na 1 tah. Celý experiment si vyžádal zhruba 300 hodin strojového času. Dohrávání partií až do úplného konce bylo nutné, protože programy hrají koncovky slaběji než zahájení či střední hru. U slabších programů byly některé výkony v koncovkách přímo žalostné a vyžadovaly od obsluhy notnou dávku otrlosti při dohrávání.

Poznámka: Aby soupeřící programy měly stejný čas na rozmyšlenou, byl programu COLOSSUS 4.0 vždy nastaven Prediction=0.

Z hodnocených programů je s převahou nejlepší COLOSSUS 4.0. V následující tabulce je u programu uveden rozdíl ELO od vítěze. V závorce je potom absolutní síla v ELO za předpokladu známého ELO pro COLOSSUS 3.0 na ATARI. Je škoda, že autor neměl možnost nahlédnout do pravděpodobně obdobného manuálu k verzi programu COLOSSUS na Spectru.

1.	COLOSSUS 4.0		(1750)
2.	CYRUS	-199	(1551)
3.	SUPERCHESS 3.5	-207	(1543)
4.	SUPERCHESS 3.0	-290	(1460)
5.	PSI CHESS	-378	(1372)
6.	SPECTRUM 2	-408	(1342)
7.	TURK 1.3	-572	(1178)
8.	MASTERCHESS 2	-646	(1104)

Vyhodnocení: V současné době je na Spectru nejsilnějším šachovým programem COLOSSUS 4.0. Zhruba o jednu VT slabší a prakticky stejně silné jsou SUPERCHESS 3.5 a CYRUS 48K (šance proti COLOSSU cca 24%). O jeden a půl VT slabší je SUPERCHESS 3.0 (šance 16%). Zhruba o dvě třídy slabší jsou PSI CHESS (program s nejlepší šachovou grafikou na Spectru) a starý Artic Chess 2 (neboli Voice Chess) s šancemi cca 9% proti COLOSSU. Nejslabší, cca o 3 stupně, jsou staré programy TURK 1.3 a MASTERCHESS 2 (šance cca 2%).

Tyto relativní odstupy v síle lze považovat experimentem za prokázané. Diskutabilní je zařazení do absolutní škály ELO. Od průměrného hráče 2.VT, to je ohodnocení etalonového programu, by autor očekával přece jenom lepší hru v koncovkách.

dr. P. Tesař

Kopírovací programy:

COPY COPY

Jedním z nejoblíbenějších a nejuniverzálnějších kopírovacích programů je program COPY COPY polského autora T. Wilczeka (v současnosti jednoho z průkopníků a propagátorů počítačových sítí v Polsku). Program sice nemá příliš pohodlné ovládání, umožňuje však více, než ostatní kopírovací programy.

Najednou můžeme překopírovat maximálně 15 souborů. Rozsah paměti je 42496 bytů, výjimečně však můžeme zkopírovat i blok o délce 49152 bytů.

Program COPY COPY nahrajeme do počítače příkazem LOAD "". Ihned po nahrání můžeme začít vkládat příkazy. Ty se vkládají jako ve standartním BASICu Sinclair, tzn. stiskem jednoho tlačítka. Tedy i slova TO a STEP vkládáme pomocí SYMBOL SHIFT a kláves F a D. Jednotlivé příkazy si probereme postupně:

CAT - klávesa C

Příkaz zajistí vypisování hlaviček z probíhajícího magnetofonového pásku, aniž se mění nebo ruší obsah souborů již nahraných do počítače. Po zobrazení patnácti hlaviček je nutné příkaz navolit znovu, chceme-li ve čtení pokračovat.

LOAD - klávesa J

Samotné LOAD pokračuje v nahrávání a ponechá již nahrané soubory v paměti.
LOAD 1 - vymaže nahrané soubory a začne nahrávat.
LOAD n - začne nahrávat od n-tého souboru
LOAD TO m - nahrává od následujícího do m-tého souboru
LOAD n TO m - od n-tého do m-tého souboru
LOAD AT nn - od adresy nn (nn je inicializovaný na 23296). Příkazem LOAD AT 23040 je možné zvětšit volnou paměť na 42496 bytů.
LOAD (nn - nahrání pouze prvních nn bytů bloku. Může se použít též LOAD n (nn. Např. LOAD (6912 nahraje pouze obrazovku.
LOAD (nn TO - nahraje blok mimo prvních nn bytů. Např. LOAD (6912 nahraje blok kromě obrazovky.

SAVE - klávesa S

SAVE - přehraje na pásek všechny nahrané soubory
SAVE n - přehraje soubory počínaje n-tým
SAVE TO m - přehraje soubory od prvního do m-tého
SAVE n TO m - přehraje soubory od n-tého do m-tého
SAVE STEP z - nastaví mezeru v sekundách mezi soubory. Nastavit můžeme 0 až 8. Při z=9 je délka mezery ovládána ručně - na obrazovce se objeví nápis "Start tape, then press any key."
Příkazy můžeme kombinovat, např. SAVE n TO m STEP z.

VERIFY - klávesa V

VERIFY - začne ověřovat správnost všech souborů v paměti.
VERIFY n - začne od n-tého souboru
VERIFY n TO m - od n-tého do m-tého souboru.

LET - klávesa L

Příkaz provede změnu hlavičky. Za příkazem musí následovat parametry v pořadí: název, délka, start,délka programu. Parametry oddělujeme čárkami. Vynecháme-li některý parametr, zůstane beze změny, ale jeho oddělovací čárky musíme napsat, aby počítač věděl, který parametr kam patří. Tak např. příkaz LET 3 = ,500,1,5 nezmění název souboru č.3, ale určí jeho délku 500, start na prvním řádku a prog na 5.

▼ názvu se mohou použít i grafické symboly. LET bez parametrů vypíše hlavičky souborů. Jestliže se objeví nápis "Out of memory", je nutné měnit hlavičku po částech.

LIST - klávesa K

Vypíše 15 bytů paměti ve formátu: adresa, obsah bytu (dekadicky), ASCII kód. Po stisknutí ENTER pokračuje výpis dalšími 15 byty.
LIST nn - provede výpis od adresy nn.
Obsah paměti mezi adresami 23552 a 23754 (systémové proměnné) je přesunut na jiné místo! Ukončení výpisu provedeme stiskem X nebo vložením příkazu.

POKE - klávesa O

POKE x, nn - uloží od adresy x dvoubytové číslo nn
POKE x, n - uloží na adrese x číslo n (max.255).
Této schopnosti kopírovacího programu COPY COPY můžeme využít při vkládání "pouků" do her tak, aby byla zajištěna nesmrtelnost hlavního hrdiny apod. Blok hry nahrajeme tam, kde má skutečně podle hlavičky být, pomocí příkazu LOAD AT nn. Příkazem POKE změníme podle návodu obsah paměti a výsledek nahrajeme na pásek příkazem SAVE. Nahraná hra je tak již upravena.

USR - klávesa U

USR nn spustí strojový program od adresy nn.

RETURN - klávesa Y

Návrat do BASICu. Nemaže nahrané soubory.

COPY - klávesa Z

COPY - umožní zkopírování bloku (bez hlavičky) o délce 49096 bytů. Nebyl-li program COPY COPY dosud použit (po nahrání z pásku), musíme příkaz vložit dvakrát za sebou.
COPY nn - zkopíruje blok dlouhý až 49152 bytů. Vlastní kopírovací program o délce 29 bytů se umístí na adresu nn, tj. někam, kde nebude překážet. Např. COPY 16387 jej uchová v první řádce obrazovky, COPY 23698 jej uloží do MEMBOT.
Po nahrání bloku jej přehrajeme na pásek stiskem CAPS SHIFT. Při použití těchto příkazů se program COPY COPY zničí a je nutné jej znovu nahrát. Příkaz se neprovede, bylo-li předtím použito posledních 200 bytů paměti - COPY COPY pak musíme nahrát znovu.

PRINT - klávesa P

Výpis na tiskárnu typu ZX Printer. V takovém případě musíme soubory nahrát za paměť tiskárny, tj. LOAD AT 23552.

V průběhu své činnosti COPY COPY podává informace o typu souboru. P znamená program v BASICu, B program ve strojovém kódu, A numerické pole a $ pole znaků. Za typem programu se může objevit čtvereček, který znamená, že soubor je bez hlavičky, nebo že jeho délka není shodná s údajem v hlavičce. Napsané a neprovedené příkazy můžeme mazat pomocí DELETE. Příkazy se provedou stiskem ENTER. Samotné ENTER opakuje poslední vložený příkaz, ale bez jeho parametrů. Provádění LOAD, SAVE a VERIFY můžeme přerušit pomocí BREAK. CAPS SHIFT a 7 vymaže poslední nahraný soubor.

(Podle Komputer 1/86 upravil mm)

Náš program:

Rozšiřování polí u Spectra

Operace s jedno- i vícerozměrnými poli naráží na problémy, protože pole je nutné deklarovat před jeho použitím. Později už nejde rozšiřovat, takže buď dimenzujeme pole zbytečně "do rezervy", nebo se nám do něj data později nevejdou a celé pole musíme ručně přepsat . První rozměr pole můžeme zvětšit kdykoli později pomocí následující procedury, kterou pro pohodlnější vkládání uveřejňujeme v BASICu, nic však nebrání jejímu pozdějšímu nahrání na pásek a použití v podobě stroj. kódu.

```
 10 CLEAR 65099: LET A = 65099
 20 FOR K = 100 TO 121
 30 LET S = 0
 40 FOR L = 1 TO 10
 50 LET A = A+1: READ B: POKE A+B: LET S=S+B: NEXT L
 60 READ M: IF M<>S THEN PRINT "CHYBA V ";K
 70 NEXT K
100 DATA 58,129,92,230,31,246,128,24,7,58,1003
101 DATA 129,92,230,31,246,192,50,129,92,50,1241
102 DATA 110,254,42,75,92,126,230,127,202,112,1370
103 DATA 6,203,255,254,0,40,6,205,184,25,1178
104 DATA 235,24,238,205,184,25,235,229,19,237,1631
105 DATA 83,33,255,19,19,213,19,237,83,35,996
106 DATA 255,30,4,205,6,255,42,176,92,245,1310
107 DATA 124,181,202,108,4,241,205,244,254,221,1784
108 DATA 225,221,70,0,5,40,28,221,35,221,1066
109 DATA 35,197,235,221,126,1,205,244,254,213,1731
110 DATA 221,126,2,205,244,254,225,25,218,108,1628
111 DATA 4,235,193,16,228,213,193,237,83,31,1433
112 DATA 255,225,205,85,22,35,30,224,205,6,1292
113 DATA 255,198,31,237,75,31,255,87,114,35,1318
114 DATA 11,120,177,32,249,237,75,31,255,221,1408
115 DATA 42,33,255,205,17,255,221,42,35,255,1360
116 DATA 237,75,176,92,205,17,255,201,6,8,1272
117 DATA 17,0,0,31,48,6,235,25,218,108,688
118 DATA 4,235,41,16,244,201,58,129,92,203,1223
119 DATA 119,62,1,32,1,131,201,221,110,0,878
120 DATA 221,102,1,9,221,117,0,221,116,1,1009
121 DATA 201,0,0,0,0,0,0,0,0,0,201
```

Před použitím procedury ji musíme předat parametry:
POKE 23681,CODE "a$" ,kde a$ je název pole
POKE 23728,x-256 * INT(x/256)
POKE 23729,INT(x/256) ,kde x je počet dodatečných elementů prvního rozměru pole a x nesmí být 0.

Proceduru vyvoláme příkazem RANDOMIZE USR 65100 pro numerická pole a příkazem RANDOMIZE USR 65109 pro znaková pole. Pokusíme -li se aplikovat proceduru na neexistující pole, objeví se chybové hlášení "Variable not found". Jestliže paměť nutná pro umístění dalších elementů přesahuje adresu 65535, počítač oznámí "Integer out of range" a v případě nedostatku místa "Out of memory".

Janusz Jarmoch
Bajtek 4/87

Co s programem...

PRO-DOS 1.1
(Professional Display Operating System)

Program PRO-DOS umožňuje práci s 8 okénky obrazovky, má jednoduchou obsluhu, umožňuje psaní v 8 směrech a 256 velikostech písma, používá celou obrazovku včetně editační zóny, může psát až 64 znaků na řádek, dokáže uložit obrazovku do jiného místa paměti a tak vytvářet rychlé změny obrazu, nepoužívá interrupt a může spolupracovat s Interfacem 1. Asi řeknete, že většinu z toho dokáže třeba BetaBasic 3.1 také. Ale pozor - PRO-DOS zabírá v paměti pouze 3787 bytů!

Před použitím programu je třeba nejprve provést inicializaci příkazem RANDOMIZE USR 60000. Tuto inicializaci musíme provést znovu po každém příkazu RUN nebo CLEAR. Autor proto doporučuje napsat první řádek v programu takto:

```
1 CLEAR 42999: RANDOMIZE 60000.
```

Při každé inicializaci se provede příkaz *NEW (pozor, to není NEW jako v BASICu - viz dále).

PRO-DOS umožňuje pracovat až s osmi nezávislými okénky. Pracuje se v nich pomocí Basicového příkazu PRINT a LIST. Speciální grafické příkazy PRO-DOS pracují vždy s celou obrazovkou o rozměrech 255 x 182 pixelů. Je k dispozici 31 nových příkazů. Na rozdíl od BASICu jim vždy předchází znak "*" a musí se celé vypsat po písmenech. Všechny výstupy PRO-DOS jsou spojeny s kanálem "P". Výstup PRO-DOS na tiskárnu umožní příkaz OPEN#2,"p" (zpět OPEN#2,"s").

Příkazy okének:

*NEW - provede následující sadu příkazů: PRINT INVERSE 0;FLASH 0;OVER 0;BRIGHT 0;INK 0;PAPER 7;AT 0,0;: *WSIZE 0,0,23: *PLOT 0,191: *TPAT 0: *GPAT 255: *DIR 8: *CSIZE 1,1: *NORMAL: *CHR: *SCREEN 16384

*CLS - nuluje okénko a naplní jej *TPAT (*TPAT je rotující binární hodnota bytu, který se zapisuje do okénka).

*CLEAR mod - mod může nabývat následujících hodnot:
0 = příkaz je ignorován
1 = nuluje pixely *TPAT
2 = nuluje atributy
3 = nuluje pixely i atributy (stejně jako *CLS)

*SCROLL mod,směr - mod je stejný jako u *CLEAR. Směr může nabývat těchto hodnot:
5 = vlevo
6 = dolu
7 = nahoru
8 = vpravo
Jsou povoleny i součty směrů. Např. *SCROLL 3,15 provede scroll vpravo a nahoru.

*ROLL mod,směr - provádí rolování okénka. Parametry jsou stejné jako pro *SCROLL. Rolování však funguje pouze pro směry vlevo a vpravo.

*WRAP - vypíná automatický scroll. Jakmile je okénko zaplněno, pokračuje se na pozici AT 0,0.

*NOWRAP - zapíná automatický scroll (horní řádky se samy při zaplnění okénka ztratí).

*CCHR - přepíná na zobrazování 64 znaků na řádek.

*CHR - přepíná na zobrazování 32 znaků na řádek. Pokud by se okénko s 64 znaky nevešlo do okénka s 32 znaky, dostanete chybové hlášení "Out of screen".

*TPAT byte - hodnota byte (0-255) je vzorkem, jehož binární hodnotou je po příkazech *CLEAR nabo *CLS vyplněno okénko. Vzorek rotuje, čímž vzniká šikmý motiv.

*SCREEN adr - adr určuje počáteční adresu uložení kopie obrazovky v RAM. Musí být mezi RAMTOP a 60000. Po tomto příkazu pracují grafické příkazy tak, jako by obrazovka začínala na adrese adr. Délka obrazovky je vždy 6912 bytů, tak si nepřepište PRO-DOS, protože provádění příkazu *SCREEN není nijak hlídáno. Příkaz *SCREEN používáme především při zpracování komplikovaných grafických obrazců, aby nebylo vidět, jak se tvoří.

*SWAP - kopíruje obsah obrazovky uložené v paměti RAM zpět do obrazovkové paměti na adrese 16384.

*WINDOW x - hodnota "x" v rozsahu 0-7 určuje, na které okénko se budou vztahovat následující příkazy PRINT a LIST. Na grafické příkazy PRO-DOS nemají okénka žádný vliv.

*WSIZE xod,yod,xdo,ydo
- příkaz definuje velikost okénka určeného příkazem *WINDOW a v závislosti na počtu znaků na řádku takto:

	*CHR	*CCHR
xod = první sloupec	(0-31)	(0-63)
yod = první řádek	(0-23)	(0-23)
xdo = poslední sloupec	(0-31)	(0-63)
ydo = poslední řádek	(0-23)	(0-23)

Musí platit, že xod je menší nebo rovno xdo a yod je menší nebo rovno ydo. Nepřípustné hodnoty vedou k chybovému hlášení "Out of screen". Např. *WSIZE 0,0,0,0 vytvoří okénko o velikosti jeden znak. Scroll je ale pak až do vyvolání CLEAR neúčinné.

*WPOKE proměnná,hodnota
- příkaz mění systémové proměnné PRO-DOS se všemi důsledky, které mohou vyplynout z jeho neuváženého použití. Proměnná je v rozsahu 0 - 19. Hodnota = nová hodnota systémové proměnné (0 - 255). Každé z okének má vlastní systémové proměnné o délce 20 bytů. Proto příkaz *WPOKE funguje na ty proměnné, které jsou adresovány zvoleným *WINDOW. Začátek proměnných je na adrese 23728/29.

Proměnná	Význam
0	xod okénka

```
1           yod okénka
2           xdo okénka
3           ydo okénka
4           x-ová souřadnice PRINT
5           y-ová souřadnice PRINT
6           x-ová souřadnice PLOT
7           y-ová souřadnice PLOT
8           TPAT
9           GPAT
10          výška (*CSIZE)
11          šířka (*CSIZE)
12          textové přepínače:
            bit 7 - nevyužit
            bit 6 - 0 = CHR
                    1 = CCHR
            bit 5 - 0 = *NORMAL
                    1 = *LARGE
            bit 4 - nevyužit
            bit 3 - nevyužit
            bit 2 - 0 = *NOWRAP
                    1 = *WRAP
            bit 1 - nevyužit
            bit 0 - nevyužit
13          přepínače *LARGE:
            bit 7 - nevyužit
            bit 6 - nevyužit
            bit 5 - nevyužit
            bit 4 - nevyužit
            bit 3 - vlevo
            bit 2 - dolu
            bit 1 - nahoru
            bit 0 - vpravo
14          ATTR_T
15          MASK_T
16          PF_FLAG
17          LSB DISPLAY FILE
18          MSB DISPLAY FILE
19          nevyužito
Systémové proměnné na adresách 23679, 23695,
23696 a 23697 jsou využívány jen krátkodobě.
```

Příkazy grafiky

*LARGE - přepíná na zvětšené znaky, jejichž velikost je určena příkazem *CSIZE. Pozor na to, že se zvolenou velikostí spolupracuje i příkaz LIST. Pokud byste měli potíže, musíte použít BREAK.

*NORMAL - přepíná zpět na normální velikost písma.

*CSIZE výška, šířka
- určuje zvětšení normálního znaku (8 x 8) jako násobek jeho původní velikosti (musí být aktivní volba *LARGE).

*DIR směr - určuje směr psaní. Výchozím bodem je vždy hodnota PRINT nebo PLOT. Přípustných je 8 směrů podobně jako v příkazu *SCROLL. Např. *DIR 15 bude psát doprava nahoru.

*GPAT byte - určuje vzorek bytu, který bude použit v příkazu PLOT. Byte (0 - 255) je typ vzorku, přičemž normálnímu PLOT Spectra odpovídá 255. Rovněž zde dochází k rotaci binární hodnoty bytu,

čímž při zobrazování vzniká šikmý motiv. Příkaz *GPAT ovlivňuje všechny příkazy, které budou v tomto návodu následovat.

*PLOT x,y - odpovídá normálnímu PLOT,pouze souřadnice y může nabývat hodnot 0 - 191. Souřadnice *PLOT 0,0 je umístěna v levém dolním rohu.

*DRAW x,y - odpovídá normálnímu DRAW. Souřadnice y může být opět až do 191.

*LINE x1,y1,x2,y2
- příkaz nakreslí čáru z bodu A(x1,y1) do bodu B(x2,y2). Druhý oddělovač může být pro větší přehlednost nahrazen klíčovým slovem, např. TO.

*BOX x1,y1,x2,y2
- příkaz nakreslí obdélník (čtverec), jehož hrany jsou rovnoběžné s osami x a y. Body A(x1,y1) a B(x2,y2) jsou souřadnicemi úhlopříčky.

*FBOX x1,y1,x2,y2
- příkaz pracuje stejně jako *BOX, ale vytvořený obdélník je navíc vyplněn vzorkem, který byl zadán příkazem *GPAT.

*TRIANGLE x1,y1,x2,y2,x3,y3
- nakreslí trojúhelník zadaný třemi body.

*ELLIPSE x,y,a,b
- nakreslí elipsu se středem S (x,y) a s poloosami a a b. Pokud a=b, pak bude nakreslena kružnice. Elipsa je aproximována pomocí třiceti přímek.

*FILL x,y - příkaz vyplní uzavřený obrazec barvou inkoustu. Souřadnice bodu A(x,y) musí ukazovat někam dovnitř tohoto obrazce a modifikují také způsob, jak bude obrazec vyplňován. Příkaz není závislý na *GPAT a jeho realizace vyžaduje asi 500 bytů volné paměti. Délka volné paměti není testována.

*PAINT x,y - příkaz pracuje podobně jako *FILL, ale pro vyplňování je použit vzorek *GPAT. Rutina je opět náročná na volnou paměť a tak v případě, že se zobrazí chybové hlášení "Out of memory", uvolněte příkazem CLEAR nejméně dalších 100 bytů.

*HATCH x,y,adr - příkaz pracuje podobně jako *FILL, ale provádí šrafování uzavřeného obrazce vzorkem, který je uložen v osmi bytech na zadané adrese "adr". Je-li vzorek uložen např. v prvním znaku UDG, pak může příkaz vypadat takto: *HATCH x,y, USR "a". Příkaz pracuje tak, že nejdříve provede *FILL a potom teprve šrafování.

*MATCH adr - provede nové šrafování obrazce zadaného předchozím příkazem *HATCH, ale lze použít jiný vzorek, např. *MATCH USR "p".

Rutiny *HATCH a *MATCH vyžadují asi 6 kB RAM a tato třetí obrazovka je automaticky ukládána na adresu 53000. Proto je

nutné vlastní obrazovky ukládané pomocí *SCREEN umístit na adresy nižší o 6912 bytů. Pokud výše uvedené příkazy nebudete používat, lze samozřejmě tuto oblast bez problémů využít. Zkuste experimentovat a uvidíte, že, lze dosáhnout zajímavých efektů i tehdy, když zadáte *SCREEN 53000, čímž se vlastně obě obrazovky překryjí.

Barvy

Příkazy pro barvy vkládejte pomocí PRINT např. takto:
PRINT INVERSE 1; OVER 1; *PLOT x,y
Jako řídicí znaky lze používat CHR$ 8 až CHR$ 13, CHR$ 16 až CHR$ 23.

Součástí programu je demonstrační část v BASICu. Podrobně ji prostudujte. Tak zjistíte nejlépe, jak lze předváděné efekty pomocí PRO-DOS nejlépe naprogramovat.

Překlad programů z počítače ZX-81

Protože BASIC počítače Sinclair ZX-81 je jednodušší než u Spectra a jeho množina příkazů je vlastně podmnožinou příkazů Spectra, zdálo by se, že s překladem nemohou být vážnější problémy. Byly dokonce pokusy tento překlad automatizovat. Skutečně, v BASICu nás může překvapit pouze příkaz SCROLL, který Spectrum nemá. Nahradíme ho však snadno příkazem RANDOMIZE USR 3582. Tam, kde náš program pracuje s náhodnými čísly, použijeme raději proměnnou, která není jinak v programu použita, např. LET q = USR 3582.

Příkaz PLOT x,y v programech pro ZX-81 musíme nahradit příkazem PRINT AT 21-y/2,x/2. (Tiskneme příslušný grafický symbol.) Podobně příkaz UNPLOT y,x nahradíme PRINT AT 21-y/2,x/2. (Tiskneme mezeru.)

Pokud překládaný program obsahuje volání rutin z paměti ROM, nemusíme ztrácet hlavu. Většinou je najdeme i v ROMce Spectra. Pokud ne, snadno je přepíšeme do paměti RAM.

Skutečné potíže nastanou teprve tehdy, provádí-li překládaný program změny systémových proměnných příkazem POKE, nebo vyvolává-li jejich obsah příkazem PEEK. Dnes si ukážeme několik příkladů, jak při překladu postupovat. V příštím čísle uvedeme kompletní tabulku odpovídajících systémových proměnných u obou počítačů.

Příkazem POKE 16510,0 se u ZX-81 vytvořil řádek číslo 0. U Spectra tedy použijeme POKE 23756,0.

Při měření času v sekundách (v proměnné T) bylo u ZX-81 nutné použít sekvenci POKE 16436,255:POKE 16437,255: LET T=(65535-PEEK16436-256*PEEK 16437)/50. U Spectra ji nahradíme: POKE 23672,0:POKE 23673,0: LET t=(PEEK 23672+256*PEEK 23673)/50. Z toho plyne, že u ZX-81 šlo takto měřit čas pouze asi do 10 minut. Na Spectru můžeme nad 10 minut použít i byte 23674.

Narazíte-li ve starých programech pro ZX-81 na sekvenci příkazů POKE 16388,X-256*INT(X/256): POKE 16389, INT(X/256), vězte, že takto se nastavoval RAMTOP. Nám postačí CLEAR X.

Ale, upřímně řečeno, málo programů stojí za tu práci!

-mm-

První kroky ve strojovém kódu

část 2 - práce s registry

Dříve, než se pustíme do dalšího výkladu, musíme odbočit. Při práci s počítači používáme jiné číselné soustavy - hexadecimální (zkráceně hex).

Hexadecimální soustava

V normálním životě používáme decimální soustavu. Máme 10 číslic (0 až 9), které kombinujeme, abychom definovali číslo. Decimální soustavu si snadno osvojíme, protože můžeme počítat na deseti prstech.

Pro počítač je tato naše soustava nepohodlná, protože má osm "prstů"; to je počet bitů v jednom registru nebo v jednom bajtu paměti. Používáme-li BASIC, překladač v paměti ROM převádí čísla do decimální (desetinné) soustavy, aby nám usnadnil práci. Když ale použijeme strojový kód, je jednodušší přizpůsobit se počítači a hexadecimální (šestnáctkové) číselné soustavě. To je soustava, která má 16 číslic (dvě "ruce" po osmi "prstech") .

Decimální číslice 0 až 9 jsou tytéž jako v hexadecimální soustavě (hex). Ale decimální číslo 10 je v hex "A", jedenáct je "B" a tak dále až do 15, což je "F" hex. Pak následuje šestnáctka, což je "10" hex, decimálních 26 je "1A" hex. Nejvyšší hodnota, jakou můžeme vyjádřit s pomocí dvou hexadecimálních číslic je "FF", tedy 255 v decimální soustavě. To je také nejvyšší hodnota, kterou můžeme uchovat v registru, nebo v bajtu paměti. Hodnota každého bajtu může tedy být definována dvoumístným hexadecimálním číslem (např. 0A pro deset, FA pro 254).

Další výhody hexadecimální soustavy oceníme v průběhu tohoto kursu. Abychom Vám usnadnili seznamování s hexadecimální číselnou soustavou, připojujeme krátký převodní program:

```
 10 LET A$ = "0123456789ABCDEF"
100 INPUT B$
110 IF B$(1) = "$" THEN GO TO 300
120 LET Z = VAL B$
130 LET A = Z
140 LET H$ = ""
150 IF A = 0 THEN GO TO 400
160 LET R = INT (A/16)
170 LET S = A - 16 * R
180 LET H$ = A$(S+1) + H$
190 LET A = R
200 GO TO 150
300 LET H$ = B$(2 TO)
310 LET Z = 0
320 FOR J = 1 TO LEN H$
330 FOR I = 1 TO 16
340 IF H$(J) = A$(I) THEN GO TO 370
350 NEXT I
360 STOP
370 LET Z = Z + (I - 1) * 16 ^ (LEN H$ - J)
380 NEXT J
400 PRINT Z, H$
410 GO TO 100
```

Program převádí desetinná a šestnáctková čísla. Po jeho spuštění (RUN) vložte decimální číslo, nebo hexadecimální číslo s prefixem "$". Podle něj program (v řádku 110) zjistí, jaké je vkládané číslo. Výsledek se vytiskne na obrazovce.

Instrukce přenosu dat - LOAD

Už minule jsme poznali příkaz assembleru LOAD. Je to nejpoužívanější příkaz vůbec. Má také mnoho variací. Ty jednodušší a nejčastěji používané si teď ukážeme.

Nejjednodušší ze všech příkazů LOAD jsou ty, které překopírují obsah jednoho registru do druhého. Je to podobné jako když v BASICu použijeme příkaz LET B = C. V assembleru pak píšeme LD B,C. Původní registr zůstává beze změny, ale hodnota, kterou obsahoval (číslo mezi 00 a FF hex) je zkopírována do jiného registru. Tuto operaci provede jediná instrukce strojového kódu (operační kód - opcode). V tabulce uvádíme operační kódy pro všechny přenosy mezi registry (LD r,r):

		z registru (r)						
		A	B	C	D	E	H	L
do registru (r)	A	7F	78	79	7A	7B	7C	7D
	B	47	40	41	42	43	44	45
	C	4F	48	49	4A	4B	4C	4D
	D	57	50	51	52	53	54	55
	E	4F	58	59	5A	5B	5C	5D
	H	67	60	61	62	63	64	65
	L	6F	68	69	6A	6B	6C	6D

Neexistuje obdobná instrukce, která by provedla zkopírování obsahu registrového páru do jiného páru registrů. Tohoto výsledku ale můžeme dosáhnout použitím dvou přenosů, např.

```
LD  H,B
LD  L,C
```

abychom zkopírovali obsah registrového páru BC do HL.

Někdy je samozřejmě nutné přenést do registrů hodnoty, které se nacházejí mimo procesor, tedy zvnějšku. Jedním způsobem je přenos dat z programu. Tak například příkaz

```
LD  B,n
```

naplní registr B hodnotou n. Ve strojovém kódu se LD B,n objeví jako dvoubajtová instrukce. První bajt je 06 hex, což je operační kód pro LD B,n. Následující hexadecimální číslice představuje hodnotu, která má být přenesena do registru B. Této číslici říkáme operand. Jestliže tedy dva bajty (operační kód a operand) jsou např. 06FB, pak bude do registru B přenesena hodnota FB. Všechny ostatní registry mohou být plněny tímto způsobem a příslušné operační kódy uvádíme v tabulce:

LD r,n		LD dd,nn	
LD A,n	3E	LD BC,nn	01
LD B,n	06	LD DE,nn	11
LD C,n	0E	LD HL,nn	21
LD D,n	16		
LD E,n	1E		
LD H,n	26		
LD L,n	2E		

Pamatujte, že pro tuto instrukci musíte mít druhý bajt!

Pokud chcete, můžete naplnit registrový pár jedinou instrukcí. Od minule si pamatujeme, že registry H a L, B a C a D a E mohou být spojeny do párů, takže mohou obsahovat čísla od 0 do 65535 (00 až FFFF hex). C, E a L představují nižší bajt, zatímco B, D a H představují vyšší bajt páru.

Pro instrukci, která přímo naplní registrový pár, jsou nutné tři bajty. První z nich je operační kód, druhý je hodnota nižšího bajtu a třetí je hodnota, která bude umístěna ve vyšším bajtu. Ve dvoubajtovém čísle je jako první vždy uvedena hodnota nižšího bajtu. Instrukce, které přenášejí dva bajty do registrového páru, naleznete ve druhém sloupci předchozí tabulky.

Přímý přenos čísla opdpovídá např. příkazům BASICu jako LET B = 5, nebo LET BC = 1225. Větší možnosti v programování bychom ale měli, kdybychom mohli do registru přenést nějakou obdobu proměnné (tj. obsah určité adresy z paměti). To skutečně můžeme udělat a hned několika způsoby.

Registr A se v mnohém liší od ostatních (to ještě poznáme). Je to například registr, ve kterém se provádí veškerá jednobajtová aritmetika. Proto musí být více možností, jak plnit tento registr, než u ostatních registrů. Můžeme plnit registr A obsahem určité adresy v paměti pomocí instrukce LD A,(nn). Písmena "nn" představují adresu paměti. Od posledně si pamatujeme, že závorky znamenají "obsah adresy, jejíž číslo je v závorkách". LD A,(mm) je tříbajtová instrukce strojového kódu. První bajt (3A) je operační kód, druhý a třetí bajt představují nižší a vyšší bajt adresy, jejíž obsah má být přenesen do registru A.

Podobně můžeme přímo přenést obsah určité adresy z paměti do registrového páru. K tomu slouží instrukce LD dd,(nn), kde "dd" zastupuje HL, DE nebo BC. Je-li dd HL, postačí jednobajtový operační kód (2A), následovaný dvoubajtovým operandem. K přenosu do BC nebo DE je nutný dvoubajtový operační kód, následovaný adresou operandu. Adresa operandu ukazuje na bajt, jehož hodnota bude přenesena do nižšího registru v páru. Bajt v paměti s adresou o 1 vyšší bude přenesen do vyššího registru v páru.

Pro ještě větší možnosti v programování můžete použít registrový pár HL, abyste adresovali místo v paměti, jehož obsah bude přenesen do registru. K tomu slouží instrukce LD r,(HL), kde r představuje jakýkoli registr. Tato instrukce požaduje pouze jeden bajt (operačního kódu), protože vždy vybírá hodnotu pouze z registrového páru HL. Na adrese, kterou tato hodnota udává, vyzvedne číslo a to překopíruje do registru. Registr A umožňuje i registrovým párům BC a DE, aby mu sdělily adresu, ze které se má naplnit - např. instrukcí LD A,(BC).

Poslední typ instrukce LOAD umožňuje umístit určitou hodnotu v paměti RAM bez toho, že bychom ji museli protahovat přes některý z registrů procesoru. Je to instrukce LD (HL),n. Má dva bajty - druhý z nich (n) představuje hodnotu, která bude umístěna na adrese, obsažené v registrovém páru HL.

z paměti		do paměti	
LD A,(nn)	3A	LD (nn),A	32
LD A,(BC)	0A	LD (BC),A	02
LD A,(DE)	1A	LD (DE),A	12
LD A,(HL)	7E	LD (HL),A	77
LD B,(HL)	46	LD (HL),B	70
LD C,(HL)	4E	LD (HL),C	71
LD D,(HL)	56	LD (HL),D	72
LD E,(HL)	5E	LD (HL),E	73
LD H,(HL)	66	LD (HL),H	74
LD L,(HL)	6E	LD (HL),L	75
LD BC,(nn)	ED4B	LD (nn),BC	ED43
LD DE,(nn)	ED5B	LD (nn),DE	ED53
LD HL,(nn)	2A	LD (nn),HL	22

Všude, kde je možné naplnit registr (nebo registrový pár) z adresy v paměti, je také možné zkopírovat hodnotu registru do paměti. Tak např. LD (HL),A překopíruje obsah registru A na adresu v paměti, určenou obsahem registrového páru HL.

Později se v příkladech k těmto instrukcím vrátíme, zatím se snažte je pochopit alespoň podle tabulky, která shrnuje instrukce, které kopírují čísla z a do paměti.

Jednoduchá aritmetika

Počítače by nám asi nebyly mnoho platné, kdyby uměly jen přenášet data z jednoho místa na druhé. Proto bude většina našeho seriálu věnována operacím s registry. Později si vysvětlíme sčítání a odečítání ve strojovém kódu. Jestliže ale postačí, abyste přičetli (nebo odečetli) jedničku k obsahu registru, nebo registrového páru, pak Vám procesor Z 80 nabízí jednoduchý způsob, jak to provést.

Instrukce INC a DEC přičtou, resp. odečtou jedničku k jakémukoli registru nebo páru. Všechny tyto instrukce jsou dlouhé pouze jeden bajt a jejich operační kódy jsou uvedeny v tabulce:

	A	B	C	D	E	H	L	BC	DE	HL
INC	3C	04	0C	14	1C	24	2C	03	13	23
DEC	3D	05	0D	15	1D	25	2D	0B	1B	2B

Tak například, jestliže hodnota registru A je 3A, pak INC A ji zvětší na 3B, zatímco DEC A ji zmenší na 39 hex. Bude-li hodnota A FF (nejvyšší možná), pak INC A ji přemění na nulu. Podobně, je-li hodnota A nula, pak DEC A z ní udělá FF.

U registrových párů pracují operace INC a DEC s kombinovanou hodnotou, obsaženou v tomto páru. Takže pozor, INC HL není totéž jako INC H a INC L. A ještě něco: neplet te si instrukci DEC (decrement) s označením desetinného čísla dec!

Jednoduché smyčky

S tím, co jsme se naučili, můžeme odstartovat program ve strojovém kódu (funkcí USR), provést řadu po sobě jdoucích instrukcí a pomocí instrukce RET se vrátit do BASICu. Z BASICu víme, jak užitečné a výkonné jsou smyčky typu FOR... NEXT. Podobné smyčky ale můžeme organizovat i ve strojovém kódu. Nejjednodušším způsobem je použití instrukce DJNZ n.

Název instrukce je zkratkou anglických slov "Decrement B and Jump if Not Zero" (zmenši registr B o jedničku a skoč, když není nula). Instrukce používá registr B jako počítadlo, podobně jako používáme proměnnou ve smyčce FOR...NEXT. Jakmile se tato instrukce začne provádět, je registr B zmenšen o jedničku a zkoumán, zda jeho hodnota není po tomto odečtení nula. Jestliže není, pak čítač instrukcí skočí na hodnotu, uvedenou v operandu n (DJNZ je dvoubajtová instrukce). Je-li hodnota B nula, přičte se k čítači instrukcí jednička, takže počítač začne s prováděním nejbližší další instrukce za DJNZ. Jednou z velkých výhod při psaní strojových programů za pomocí dobrého assembleru je to, že assembler vypočítá hodnotu "n" sám. Jestliže ale počítáte

tyto tzv. relativní skoky sami, pak pamatujte, že výchozím bodem pro výpočet je adresa operačního kódu instrukce, která následuje za DJNZ. To právě je místo, kam se dostanete, kdybyste zadali DJNZ 0.

Použití si ukážeme na příkladech. Pamatujte ale, že při každém použití DJNZ musíme naplnit registr B (počítadlo) příslušnou hodnotou, podobně jako je tomu u příkazu FOR ve smyčce FOR....NEXT.

A ještě jedna instrukce. Uvidíte ji v některých příkladech, je to instrukce NOP. Znamená " No Operation" - nedělej nic! Je to způsob, jak napsat ve strojovém kódu PAUSE. NOP pouze zpomaluje průběh programu o zlomky sekund, takže má-li být vidět nějaký efekt, potřebujete spoustu těchto instrukcí (ve smyčce).

Příklad

Nejprve se podíváme na výpis našeho demonstračního programu v assembleru. Všechny příklady používají adresy v paměti, které leží na obrazovce, takže jejich změna je dobře viditelná.

Příklad 1:

```
        LD HL,22528     210058   začátek atributů
        LD DE,30100     119475   adresa pro uchování 1 bajtu
        LD A,22         3E16     čítač smyčky
JEDNA   LD (DE),A       12       uchováno na adr. 30100
        LD B,32         0620     čítač sloupců
DVA     LD (HL),127     367F     hodnota atributu umístěna v řádce
        INC HL          23
        DJNZ DVA        10FB
        LD A,(DE)       1A       obnovení čítače smyčky
        LD B,A          47       kopie do B
        DEC A           3D       zmenši čítač smyčky o jednu
        LD (DE),A       12       přemísti čítač smyčky
        DJNZ JEDNA      10F3     skoč zpět na JEDNA, dokud není  vše
        RET             C9             hotovo....
```

Příklad 2 je v podstatě stejný, jen první tři řádky změníme takto:

```
        LD HL,16384     210040   začátek obrazovky-pixelů
        LD DE,30100     119475   skladovací adresa
        LD A,200        3EC8     čítač smyčky
```

V příštím pokračování si uvedeme další příklady a naučíme se je vkládat do Spectra.

Podle ZX Computing upravil mm