

George K.'s

Rutiny ROM D40

Příručka pro uživatele disketových jednotek:

Didaktik 40, Didaktik 80 a Didaktik Kompakt

© 1993 PROXIMA - software nové dimenze

ÚVODEM

(A radši i na závěr)

Vážení uživatelé,

od doby, kdy **Didaktik Skalica** vyrobil svoji první disketovou jednotku, uplynuly už dva roky. Doposud k ní nebyla vydána žádná literatura a to ani ze strany výrobce, ani ze strany softwarových firem. Teprve koncem minulého roku se do **PROXIMY** dostal zdrojový text MDOSu s poznámkami - bohužel, byl nám na nic, protože Tools 40 i 80 již existovaly, a vše, co jsme o MDOSu potřebovali vědět, jsme si zjistili sami disassemblováním. Jediné, k čemu zdroják MDOSu posloužil, je, že jsem převzal jeho návěští a názvy rutin, což může být užitečné, pokud ho jednou někdo celý okomentuje a vydá.

Moje zkušenosti s **D40 / D80** jsou sice bohaté, ale i tak každou chvíli objevím nějakou novinku, která mi dříve ušla. Berte proto tuto příručku pouze jako stručný popis základních možností disketové jednotky. Základem k pochopení všeho popsaného je vědět něco o programování; na škodu není znát (nebo alespoň mít) komentovaný výpis **ROM ZX Spectra** (to se vždycky hodí).

Příručka je rozdělena do několika částí: práce s bootem, FAT a adresářem, volání služeb MDOSu a využívání vlastních rutin. V přílohách pak najdete něco o systémových proměnných, chybách MDOSu a seznam vybraných rutin. Některé popsané rutiny pochází z **KoZa Commanderu**, ošetření je chyb z Tolstoje, ostatní programy byly napsány zcela nově.

S využitím uvedených poznatků by pro Vás neměl být problém napsat si do svého programu diskové operace, ošetřit u nich chyby, upravit pro disketu dílové hry, atd. Úplný obraz o mechanice může poskytnout jen komentovaný výpis ROM, a ten, dříve či později někdo vydá.

Pro úplnost dodávám, že příručka se vztahuje k **MDOSu verze 1.0**.

George K.

K této příručce existuje soubor zdrojových textů (pro assembler PROMETHEUS), na disketách (zvláštní poděkování J. Flaškovi za jejich přepsání do assembleru a vychytání chyb...). Soubor s názvem "**Rutiny ROM D40 - zdrojové texty**" si můžete musíte objednat na naší adrese.

P. S.: Pokud v programech najdete chybu, napište to na adresu naší firmy, abychom ji mohli do dalšího vydání opravit.

I. Organizace dat na disketě, rutiny pro obsluhu FAT a adresáře

Dříve než vůbec začneme s popisem rutin **MDOSu**, nebude na škodu, říci si něco o tom, jak jsou organizována data na disketách. U každé kapitoly si uvedeme několik vlastních rutin, které můžete použít jak ve svých strojových programech, tak je i volat z basicu.

1.1 Formát

Disketa se formátováním rozdělí na **stopy** a dále na **sektory**. Pro jednu stopu i jeden sektor je na disketě vyhrazeno určité, vždy stejně velké a stejné položené místo. Při větším počtu stop, než je kapacita disku, nedochází k jejich "zhušťování", ale k pokusu vytvořit novou stopu za koncem té poslední (lépe řečeno: za koncem místa pro poslední stopu, protože při devíti (osmi, atd.) sektorech zůstává místo pro desátý (desátý a devátý, atd.) sektor nevyužito). Ne každá čtyřicítková (osmdesátková) mechanika je s to dojet na místo pro jedena-, dva-, třičtyřicátou (osmdesátou) stopu, proto používejte standardní formát 40 x 9 (80 x 9), nebo 40 x 10 (80 x 10).

Při prvním formátování se na disku vytvoří záhlaví stop a sektorů a tento údaj přesně odpovídá stavu diskety. Po druhém formátování to již pravda být nemusí. Naformátujete-li poprvé disketu na 40 x 9 a podruhé na 30 x 5, stopy i sektory "navíc" z prvního formátu na disku už zůstanou a daly by se jednoduše přechíst programem ve strojovém kódu.

V této souvislosti si musíme říci o rozdílu mezi logickými a fyzickými sektory. Nejjednodušeji tento rozdíl ukazuje tabulka, rozdělená do šesti sloupců.

V prvním sloupci najdete číslo stopy.

Druhý sloupec popisuje číslování sektorů ve stopě.

Třetí sloupec je ukázka fyzického číslování sektorů po celém disku - odpovídá to logickému číslování při formátu 40 x 10.

Čtvrtý sloupec ukazuje logické číslování při formátu 40 x 9 (vyplněná ploška značí, že tento fyzický sektor je nevyužit).

Pátý sloupec - formát 40 x 8.

Šestý sloupec - formát 40 x 5 (pozor, takovýto formát Vám ve skutečnosti MDOS neumožní).

Na konci formátování se na disketu uloží boot, FAT a adresář - tři oblasti zaujímavější vždy prvních 14 logických sektorů (např. při formátu 40 x 8 budou tyto oblasti ležet ve fyzických sektorech 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15).

| | | | | | |
|-------|---|----|----|----|----|
| nultá | 0 | 0 | 0 | 0 | 0 |
| | 1 | 1 | 1 | 1 | 1 |
| | 2 | 2 | 2 | 2 | 2 |
| | 3 | 3 | 3 | 3 | 3 |
| | 4 | 4 | 4 | 4 | 4 |
| | 5 | 5 | 5 | 5 | 5 |
| | 6 | 6 | 6 | 6 | 6 |
| | 7 | 7 | 7 | 7 | 7 |
| | 8 | 8 | 8 | 8 | 8 |
| | 9 | 9 | 9 | 9 | 9 |
| první | 0 | 10 | 9 | 8 | 5 |
| | 1 | 11 | 10 | 9 | 6 |
| | 2 | 12 | 11 | 10 | 7 |
| | 3 | 13 | 12 | 11 | 8 |
| | 4 | 14 | 13 | 12 | 9 |
| | 5 | 15 | 14 | 13 | 10 |
| | 6 | 16 | 15 | 14 | 11 |
| | 7 | 17 | 16 | 15 | 12 |
| | 8 | 18 | 17 | 16 | 13 |
| | 9 | 19 | 18 | 17 | 14 |

1.2 Boot

Co do pořadí, je boot prvním sektorem na disku. Co do číslování, jedná se o sektor nultý, protože řadič disketové jednotky čísluje všechno od nuly. Pro něj je boot nultým sektorem v nulté stopě, čili úplným nultým sektorem na disku. My se budeme tohoto číslování držet - pamatujte na to.

Boot je vždy prvním sektorem, který si operační systém přečte před jakoukoliv diskovou operací. To proto, že je zde uložen formát, který MDOSu říká, jak je disk organizován. Formát by ani nemohl být uložen nikde jinde, protože boot je jediný sektor, který vždy leží na stejném logickém i fyzickém místě (představte si formát

40 x 1 a bude Vám to jasné). Dále se v bootu nalézá název diskety a značka, podle které MDOS svoji disketu rozpoznává (např. od disket pro MS DOS).

Jméno diskety najdete v bootu na adrese **192**. Jméno můžete sice zadat pouze při formátování a později ho MDOS neumožňuje změnit, ale jde to a to celkem jednoduše. Bud použijete TOOLS 40 / 80, nebo to provedete programem v Basicu:

```

10 READ *"" ,0,40000      ; přečte boot na adresu 40000
20 FOR a=40192 TO 40201  ; tato smyčka vypíše jméno (a případné
30 PRINT CHR$ (PEEK a);
40 NEXT a                ; otazníky místo CHR$ 0)
50 INPUT a$             ; nové jméno disku (max. 10 znaků)
60 FOR a=1 TO LEN a$    ; uloží nové jméno na místo starého
70 POKE (40191+a),CODE a$ (a TO a)
80 NEXT a
90 FOR a=LEN a$ TO 10   ; je-li jméno kratší než 10 znaků,
100 POKE (40192+a),0   ; bude doplněno nulami
110 NEXT a
120 RESTORE *"" ,0,40000 ; zapíše boot zpět na disk

```

Bezprostředně za jménem (od adr. **202**) následují dva náhodně vygenerované bajty, které sem byly uloženy při formátování, a to z předvídatosti: i pokud byste měli dvě diskety se stejným jménem, půjdou rozlišit, protože pravděpodobnost, že budou mít stejný i náhodný dvojbajt, je velice nízká (1:65535).

Na adresách **204 - 207**, je zapsáno "SDOS", což MDOSu signalizuje, že se jedná o jeho disketu (jak logické).

Nejdůležitější a nejsložitější informací je **formát**. V bootu ho najdete na třech místech, ale jen jedno je to pravé. Od adresy **128** leží kopie systémových proměnných (vztahujících se k formátu); jsou zde údaje o mechanikách, které měl MDOS k dispozici, když disketu formátoval. Z těchto údajů se dá formát diskety vyčíst, ale bylo by zapotřebí testů; proto si veškeré informace bez jakýchkoliv problémů přečteme z adresy **177**.

| adresa | význam obsahu adresy |
|---------|---|
| 177 | 4. bit nastaven při oboustranném formátu 3. bit nastaven při formátování na D40 (nenastaven D80) |
| 178 | počet stop na straně |
| 179 | počet sektorů na stopu |
| 180 | nula |
| 181-183 | viz.177-179 |

Ostatní části bootu jsou za normálních okolností prázdné, ale činnost MDOSu nijak nenaruší, pokud si do nich něco uložíte. Pro jistotu se však nejdříve dvakrát přesvědčte, že ukládáte skutečně do volného místa.

1.3 FAT

FAT je zkratka z anglických slov **File Allocation Table**, čili tabulka umístění souborů. FATka začíná od prvního sektoru (adresa 512) a táhne se až do sektoru pět (poslední adresa ve FAT je 3071). Tabulka obsahuje 1705 položek, každá položka je přiřazena jednomu sektoru (první položka prvnímu sektoru, druhá druhému, atd.).

S FATkou se pracuje následovně: Z adresáře se zjistí číslo prvního sektoru souboru (soubor je zpravidla "rozstrkán" do více sektorů). Z této hodnoty se vypočítá adresa odpovídající položky ve FAT. Na této adrese je uloženo číslo následujícího sektoru, který k souboru patří. Opět se vypočítá odpovídající adresa položky a takhle se to opakuje stále dokola, dokud na místo čísla dalšího sektoru položka neobsahuje značku konce souboru.



dojít k závěru, že do jednoho sektoru FAT (do 512 bajtů) se vejde 341 položek a ještě půl bajte zbyde. Je to skutečně tak - tento půlbajt se nazývá zalomením FATky a obsahuje hodnotu 13 (horní polovina bajte je normálně využita jak ukazuje druhý obrázek). Zalomení jsou na adresách 1023, 1535, 2047, 2559 a 3071. (Všimněte si, že položky sudých sektorů začínají na sudých adresách, horní 13 a položky lichých sektorů na lichých adresách.)



Pro nezasvěcené je organizace FAT více než zmatená, ale určitě úsporná a snadno modifikovatelná, jak se za chvíli sami přesvědčíte. Asi největší neobvyklostí (oproti logice strojového kódu) je ukládání do 1,5 bajte. Na obrázku vidíte tři bajty ze začátku FAT. Naznačený princip uložení se opakuje vždy po třech bajtech. Počítáte-li správně, musíte

Pro příklad si vezměme dvě čísla **A** a **B**, která chceme uložit od adresy 512.

První, co zjistíme, je, že čísla nesmí být vyšší než dvě na dvanáctou minus jedna, tedy 4095, protože jinak by se do dvanácti bitů nikdy nemohla vejít. To nijak nevadí, protože sektorů na disku nemůže být více jak 5×341 (kapacita FAT), tj. 1705, a pro tyto účely omezení vlastně nijak neomezuje. Číslo **A** celočíselně vydělíme 256 - výsledek označíme jako **AH**, zbytek po dělení jako **AL**. To samé provedeme s číslem **B** (dostaneme ... **BH**, **BL**). Hodnotu **AL** uložíme na adresu 512, hodnotu **BL** na adresu 514. Hodnotu **AH** vynásobíme 16 a přičteme k ní **BH**. Výsledek uložíme na 513 - a to je všechno.

Některé základní rutiny pro práci s FAT

Naším prvním cílem bude mít podprogram, který by dokázal přibližně to, co je popsáno v následujícím odstavci, tedy proběhnout celou stezku souboru od začátku do konce:

Soubor leží rozházen v sektorech 19, 311, 124, 67 a 502 (jdou za sebou tak, jak jsem je napsal). Vypočítáme adresu, na které leží položka pro devatenáctý sektor. Z ní si přečteme, že další sektor souboru je 311. Opět vypočteme adresu 311. položky... atd., až 502. položka již nebude obsahovat číslo následujícího sektoru, ale znamená, že tímto sektorem soubor končí.

Budeme potřebovat vědět, jaké hodnoty mohou položky obsahovat:

| Vyšší půlbajt | Nižší bajt | Celková hodnota |
|--|-----------------|-----------------|
| %1101 (13) | %11011101 (221) | 3549 |
| Sektor je nedostupný. Takto jsou označeny sektory, ve kterých je uložen boot, FAT, adresář a logické sektory, které na disku nebyly formátováním vytvořeny (např. při formátu 40 x 9 jsou to sektory 721 - 1704). Pro MDOS je to znamení, že sektor nemůže použít pro uložení souboru. | | |
| %1101 (13) | %11111111 (255) | 3583 |
| Vadný sektor, byl takto označen při formátování a nelze k ničemu použít. | | |
| %1100 (12) | %00000000 (0) | 3072 |
| Prázdný soubor (jeho délka je nulová). | | |
| %0000 (0) | %00000000 (0) | 0 |
| Prázdný sektor. Mohou být do něj zapsána data. | | |

Značka konce souboru je vytvořena takto: Délka souboru je vydělena 512. Nás zajímá zbytek po dělení (0-511), ke kterému přičteme **3584**, čímž jsme dostali koncovou značku souboru (neplatí pro prázdné soubory).

Teď uvedu podprogram **FDBLOCK**, který očekává v **HL** číslo sektoru a vrací v **DE** obsah položky vztahující se k tomuto sektoru a v **HL** adresu této položky. Podprogram pracuje s celou FATkou - musíte si ji načíst (prozatím basicovským **READ ***) do paměti na adresu **XXX**.

```

FDBLOCK  push  bc          ;uschovej původní obsah HC
          ld    bc,xxx-512  ;do BC dej začátek FAT v paměti - 512
          ld    de,341      ;do 1 sektoru FAT se vejde 341 položek
          or    a           ;shoď případné CARRY

```

```

LESS      inc    b           ;posuň se na další sektor
          inc    b
          sbc   hl,de       ;odečti od čísla sektoru 341
          jr    nc,LESS     ;je-li výsledek nezáporný, opakuj smyčku
          add   hl,de       ;vrať do HL, poslední kladnou hodnotu
MORE      ld    e,l        ;a okopíruj ji do DE
          ld    d,h
          srl   d           ;DE vyděl dvěma
          rr   e
          ex   af,af'      ;ulož příznak, bylo-li číslo sudé/liché
          add   hl,de       ;HL = adresa položky od počátku sektoru
          add   hl,bc       ;HL = adresa položky od počátku FAT
          ld    e,(hl)      ;do DE vyzvedni obsah položky
          inc   hl
          ld    d,(hl)
          dec   hl         ;adresa v HL je zachována
          ld    a,l        ;nastav: položka leží na sudé adrese
          ld    (TYPE),a
          ex   af,af'      ;zjisti, je-li to pravda
          jr    nc,NODD    ;ano, skoč
ODD       xor   a         ;změň nastavení (položka na liché adr.)
          ld    (TYPE),a
          ld    a,e        ;okopíruj do A vyšší bajt
          ld    e,d        ;do E dej nižší bajt z D
          jr    BOTH      ;skoč dál
NODD     ld    a,d        ;vyšší bajt do A (nižší je správně v E)
          rrca
          rrca
          rrca
          rrca
BOTH     and   15         ;zachovej spodní polovinu vyššího bajtu
          ld    d,a        ;a tu dej do D
          pop   bc         ;obnov původní obsah HC
          ret            ;vrať se
TYPE     defb 0

```

Příznak TYPE (určuje, zda položka sektoru leží na sudé nebo liché adrese) v tuto chvíli k ničemu nepotřebujeme, ale bude se nám hodit později.

Pomocí podprogramu **FDBLOCK** sestavíme program, který bude umět procházet stezky soubory. Mohl by vypadat třeba takto:

```

PATH     ld    hl,YYY     ;do HL dej číslo prvního sektoru stezky
P0       call  FDBLOCK    ;zjisti následující sektor
          ld    hl,3072   ;je-li to 3072, je soubor prázdný
          sbc   hl,de
          ret   z         ;a proto se vrať
          ld    a,d        ;zkus, jestli to byl poslední sektor
          cp   14
          ret   nc       ;ano, vrať se
          ex   de,hl      ;ne, dej do HL číslo následujícího sektoru
          jr    P0       ;opakuj smyčku

```

Tento podprogram je sice svým způsobem dokonalý, ale má malou vadu - nedělá totiž vůbec nic, co by mělo nějaký hlubší smysl... zato ovšem funguje!

Další operací, která je při práci s FAT zapotřebí, je zápis. Pro něj budeme potřebovat novou rutinu **BCTOBT**, která vyžaduje v HL adresu položky sektoru a BC číslo sektoru, které má být do položky uloženo.

```

BCTOBT    ld    a,(TYPE)        ;sudá/lichá pozice
          or    a
          jr    z,HCODD        ;skoč - lichá
          ld    (hl),c         ;ulož nižší bajt
          inc   hl              ;posuň ukazovátka do FAT
          rlc   b               ;vynásob B šestnácti
          rlc   b
          rlc   b
          rlc   b
          ld    a,15           ;zachovej dolní polovinu společného bajtu
BCCOM     and   (hl)
          or    b               ;jako horní polovinu přidej
          ld    (hl),a         ;ulož do FAT
          ret                    ;vrať se
BCODD     ld    a,240         ;zachována bude horní pol. spol. bajtu
          call  HCCOM
          inc   hl              ;posuň ukazovátka
          ld    (hl),c         ;zapiš nižší bajt
          ret                    ;vrať se

```

Z uvedených subrutin již můžeme skloubit první užitečnou rutinu - **DELPATH** – vymazání souboru z FAT (není v ní ošetřena možnost, že FAT je poškozená). Jako vstupní parametr dejte do HL číslo prvního sektoru souboru.

```

DELPATH   call  FDBLOCK        ;zjisti adr. pol. sektoru a násl. sektor
          ld    a,d            ;jedná se o prázdný soubor?
          cp    12
          jr    nz,DP1         ;ne, skoč
          ld    a,e            ;otestuj ještě nižší bajt
          or    a
          jr    z,DP3         ;soubor je prázdný, skoč
          jr    DP2           ;jinak pokračuj dál
DP1       cp    14            ;je to značka konce souboru?
          jr    nc,DP3        ;ano, skoč
DP2       call  DP3           ;vyčisti položku v tabulce FAT
          ex   de,hl          ;do HL dej číslo následujícího sektoru
          jr    DELPATH       ;opakuj smyčku
DP3       ld    bc,0          ;BC = 0
          jr    BCTOBT        ;ulož na adr. pol. nuly (prázdný sektor)

```

Posledním podprogramem, který si popíšeme, je vyhledání volného sektoru. Bude se jmenovat **FRBLOCK** a nebude potřebovat žádné vstupní parametry.

```

FRBLOCK   ld    bc,13         ;13. sektorem končí adresář,
FRO       inc   bc            ;od 14. sektoru je volné místo
          ld    d,b           ;okopíruj číslo sektoru do DE
          ld    e,c
          or    a              ;shod' možné CARRY
          ld    hl,2*T*S      ;HL=2*počet stop*počet sektorů (formát)
          sbc   hl,de         ;odečtením zjistí, je-li sektor s číslem

```

```

ret   c           ;v DE na disku a když ne, vrať se se s CARRY
ex   de,hl       ;do HL přendej číslo sektoru
call  FDBLOCK    ;spočti adr. položky a vyzvedni obsah
ld   a,d         ;je-li nenulový,
or   e
jr   nz,FRO      ;pokračuj ve smyčce
ret              ;vrať se s číslem volného sektoru v BC

```

Podprogram FRBLOCK prohledává FAT od prvního datového sektoru (č. 14) až do posledního (2*T*S - podle formátu; dvojka značí oboustranný formát, jednička jednostranný). Vrací: buď příznak CARRY, když ve FAT není žádný volný sektor, nebo příznak NC, číslo volného sektoru v BC a adresu jeho položky v HL. Podprogram můžete volat od návěští FRO - pokud nechcete, aby prohledával FAT stále od začátku; potom ale sami musíte hlídat, aby registr BC neztratil svoji hodnotu.

Tímto kapitolu o FAT uzavřeme a podíváme se na poslední důležitou oblast diskety, na adresář.

1.4 Adresář

Adresář leží v šestém až třináctém sektoru, což je mezi adresami 3072 a 7167. Jsou v něm uloženy názvy souborů a vše, co k nim patří (přípona, délka, atributy ...). Oproti FATce má krásné jednoduchou strukturu: pro každý soubor je vyhrazeno 32 bajtů, z čehož snadno spočítáme, že jeden sektor pojme $512 / 32 = 16$ názvů; celkem můžeme do osmi sektorů (vyhrazených pro adresář) uložit 128 názvů souborů.

Co se přesně obsahuje oněch 32 bajtů pro jeden soubor? Najdete to v následující tabulce.

| bajt | význam |
|-------------|---|
| 0 | Přípona souboru (v ASCII kódu: P, C, N, B, S, Q). |
| 1 - 10 | Jméno souboru. 1e-li kratší než 10 znaků, je doplněno nulami. |
| 11, 12 | Délka souboru (0-65535). |
| 13, 14 | Počáteční adresa souboru (u programů startovní řádek). |
| 15, 16 | Nemá význam (výjimka: u programy délka basicu bez proměnných). |
| 17, 18 | Číslo prvního sektoru souboru (vstupní bod do FAT). |
| 19 | Nula. |
| 20 | Atributy. Každý bit odpovídá jednomu z příznaků HSPARWED. |
| 21 | Ještě jeden bajt pro uložení délky u souborů delších než 65535 bajtů. |
| 22 - 31 | Zaplněno hodnotou 229 |

Prvních sedmnáct bajtů hlavičky je stejných, jako u standardních hlaviček na pásce. Jediný rozdíl je v příponách - zde jsou uloženy přímo ASCII kódy velkých písmen, odpovídající jednotlivým typům souborů: programy - "P", číselná pole - "N", znaková pole - "C", bajty - "B", snapshoty - "S" a sequence - "Q"

Právě sequence jsou příčinou existence 21. bajtu - protože jejich délka může přesáhnout 65535 bajtů, nestačí pro její uložení pouhé dva bajty, ale je zapotřebí bajtů tří. Proto při práci s délkou souboru budete občas muset používat trojbajtovou aritmetiku (je to o trochu složitější).

Atributy souboru se skládají podle jednoduchého systému: je-li atribut nastaven, je nastaven i k němu příslušný bit. K atributu Hidden patří 7. bit, k System 6. bit, atd. Platí pořadí atributů HSPARWED.

Po zformátování je celý adresář zaplněn hodnotou 229. Postupně, jak jsou na disk zapisovány soubory, jsou obsazována místa pro jejich hlavičky. Při vymazání souboru není smazána celá hlavička, ale pouze přípona souboru - na její místo je uloženo oblíbené 229. Tento fakt umožňuje psát programy, které občas dokáží obnovit nějaký ten smazaný soubor.

Podprogramy pro práci a adresářem

Protože teď již známe význam všech čtrnácti základních sektorů (bootu, FATu, adresáře), můžeme podprogramy pro práci s ním1 vzájemně provázat, čímž získáme vlastní rutiny pro některé funkce MDOSu. Zatím jsme si však neřekli nic o čtení, zápisu a sektorů, proto ještě občas použijeme basic. Opište si následující program, který načítá do paměti boot, FAT a adresář, a to od adresy 58000 (změňte si ji podle libosti, ale nezapomeňte na CLEAR).

```
10 CLEAR 57999: FOR a=0 TO 13
20 READ *"", a, 58000+a*512
30 NEXT a
```

Opisujete-li si zdrojáky do assembleru (nejlépe PROMETHEA instalovaného na adresu 25000), vymažte rutinu PATH (je k ničemu), na začátek zdrojáku dopište

```
DIR equ 58000
```

(musí se shodovat s adresou uvedenou v basicu) a opravte druhý řádek v rutině FDBLOCK, na takto:

```
ld bc, DIR
```

Založte do mechaniky nějakou zaplněnou disketu, nechte proběhnout basic a vraťte se do assembleru. Disketu zase vyjměte.

A teď už několik rutin pro práci s adresářem.

CNTFLS. Nemá žádné vstupní parametry. Vrací v A počet souborů v adresáři.

```
CNTFLS ld hl, DIR+3072 ;do HL začátek adresáře
ld de, 32 ;velikost místa pro hlavičku souboru
ld bc, #8000 ;B = 128 (max. počet souborů), C = 0
ld a, 229 ;hodnota, která určuje, že místo je prázdné
CF0 cp (hl) ;je místo prázdné ? - porovnej
jr z, CF1 ;ano, je - obskoč přičtení
inc c ;není - započítej soubor
CF1 add hl, de ;posuň se na další místo pro hlavičku
djnz CF0 ;opakuj B krát
ld a, c ;dej počet souborů do A
ret ;a vrať se
```

FRROOM. Nemá žádné vstupní parametry. Najde volné místo v adresáři. Není-li místo nalezeno, vrací CARRY, jinak NC a v HL adresu místa.

```
FRROOM ld hl, DIR+3072 ;počáteční nastavení stejné jako v CNTFLS
ld de, 32
ld b, 128
ld a, 229
FM0 cp (hl) ;je tu volné místo?
ret z ;ano, vrať se
add hl, de ;ne, zkus další
djnz FM0 ;opakuj B krát
scf ;adresář je obsazený, nastav CARRY
ret ;a vrať se
```

FINDEL. Hledá soubor. Vstupním parametrem je jedenáct bajtů, na které ukazuje registr IX. Těchto jedenáct bajtů obsahuje příponu a jméno hledaného souboru. Pokud se soubor nenajde, vrací CARRY, jinak NOT CARRY a HL ukazuje na hlavičku souboru.

```

FINDFL    ld    hl,DIR+3072    ;začátek adresář
          ld    de,32        ;délka místa pro hlavičku
          ld    b,128        ;maximálně 128 pozic bude prohledáno
FF0       ld    c,11         ;porovnej 11 bajtů
          push hl            ;ulož ukazovátka
          push ix
FF1       ld    a,(ix+0)      ;přečti bajt z hledané hlavičky
          xor   (hl)         ;porovnej ho
          jr   nz,FF2        ;při nerovnosti skoč pro další hlavičku
          inc  hl            ;posuň ukazovátka
          inc  ix
          dec  c             ;opakuj C krát
          jr   nz,FF1
          pop  ix            ;hlavička nalezena, obnov ukazovátka
          pop  hl
          ret                ;vrať se
FF2       pop  ix            ;obnov ukazovátka
          pop  hl
          add  hl,de         ;posuň se na další hlavičku
          djnz FF0           ;opakuj B krát
          scf                ;hlavička nenalezena, nastav CARRY
          ret                ;vrať se

```

Ted' dokončíme dříve započatý podprogram na vymazání souboru. Bude se jmenovat ERASE a vstupní parametr bude stejný jako pro FINDFL. Vracet se bude CARRY, když soubor nebude nalezen, jinak rutina odstraní z adresáře hlavičku a z FATky stezku - dělá tedy v podstatě to samé, jako basicový příkaz ERASE (krom skupinových operací).

```

ERASE     call  FINDFL       ;hledej v adresáři, pokud je nenalezen,
          ret   c            ;vrať se, jinak HL ukazuje na hlavičku
          ld   (hl),229      ;označ hlavičku jako vymazanou
          ld   de,17        ;posuň se na sedmnáctý bajt hlavičky
          add  hl,de         ;a přečti si číslo prvního sektor souboru
          ld   e,(hl)
          inc  hl
          ld   d,(hl)
          ex  de,hl         ;přendej ho do HL
          jp   DELPATH      ;skoč na vymazání stezky.

```

Podprogram pak budete volat například takto:

```

START     ld    ix,HEAD      ;ukazuj na hlavičku
          call  ERASE        ;zkus vymazat soubor
          ret   nc          ;vrať se, pokud byl vymazán
          ld   a,27         ;do A dej číslo hlášení "s File not
          jp   ERROR        ;found" a skoč na vlastní ošetření chyb
HEAD      defb  "P"         ;přípona souboru
          defm "TELEFONY"   ;jméno souboru
          defb  0,0         ;zbytek místa na jméno doplněn nulami

```

Uvedený podprogram ERASE maže i soubory s nenastaveným atributem "D" (soubory, jejichž smazání je blokováno). Chcete-li k tomuto atributu při mazání přihlídnout, použijte podprogram ERASE2. Na vstupních

parametrech se nic nezměnilo, výstupní příznaky jsou trochu složitější: ZERO a CARRY znamená, že soubor nebyl nalezen, NOT ZERO a CARRY, že soubor je označen jako nesmazatelný.

```

ERASE2    call  FINDFL      ;hledej v adresáři, pokud je nenalezen,
          ret   c           ;vrať se, jinak HL ukazuje na hlavičku
          ld   (hl),229     ;označ hlavičku jako vymazanou
          ld   de,17       ;posuň se na sedmáctý bajt hlavičky
          add  hl,de        ;a přečti si číslo prvního sektor souboru
          ld   e,(hl)
          inc  hl
          ld   d,(hl)
          inc  hl
          inc  hl          ;HL ukazuje na atributy
          ld   a,(hl)      ;přečti je do A
          cpl                ;udělej komplement (= xor 255)
          and  1           ;testuj atribut „delete protect“
          scf                ;nastav CARRY
          ret  nz          ;vrať se s NZ a C, je-li soubor chráněn
          ex  de,hl        ;přendej číslo sektoru z DE do HL
          jp   DELPATH     ;skoč na vymazání stezky.

```

Volání rutiny provedeme následovně:

```

START     ld   ix,HEAD     ;ukazuj na hlavičku
          call ERASE       ;zkus vymazat soubor
          ret  nc          ;vrať se, pokud byl vymazán
          ld   a,27        ;hlášení "S File not found"
          jp   z,ERROR     ;případný skok na ošetření chyb
          ld   a,48        ;hlášení "h File is delete protected"
          jp   ERROR      ;skok na ošetření chyb
HEAD      defb  "P"        ;přípona souboru
          defm "TELEFONY"  ;jméno souboru
          defb  0,0        ;zbytek místa na jméno doplněn nulami

```

Může se stát, že budete potřebovat porovnat nejen jméno a příponu souboru, ale i jeho délku, startovní adresu, atd. - to pro případ, aby nebyl vymazán (načten, přepsán) špatný soubor. Vyřešíme to malou úpravou před voláním podprogramu FINDFL a rozšířením vstupního parametru:

```

          ld   ix,HEAD     ;IX ukazuje na rozšířenou hlavičku
          ...
          ld   a,13        ;bude se porovnávat přípona, jméno, délka
          ld   (FF0+1),a   ;celkem 13 bajtů - ulož hodnotu do rutiny
          call FINDFL      ;zavolej porovnání
          ld   a,11        ;vrať původní počet porovnávaných bajtů
          ld   (FF0+1),a
          ...
          ret  c           ;soubor nenalezen - vrať se
          HEAD defm "PTELEFONY" ;rozšířená hlavička - přibyla navíc délka
          defw 0,3240      ;souboru (3240 bajtů)

```

Takto upravená rutina bude hledat nejen program s názvem "TELEFONY", ale navíc takový program "TELEFONY", jehož délka je 3240 bajtů.

Malá poznámka na závěr: Těm, kteří by si chtěli do svých programů zabudovat výběrová okénka na principu Toolsu (mají je i Orfeus, Pressor VI, CrackShot I, II), bych doporučil, aby nejdříve - prošli celý adresář

a sestavili si tabulku odkazů na hlavičky. Veškeré uživatelské operace jako jízda kurzorem, označování souborů, výběr, ale i některé jiné rutiny půjdou přepsat tak, že se budete pohybovat pouze v tabulce odkazů a nemusíte neustále operovat se samotným adresářem. Vhodná je tabulka o velikosti 3*128 bajtů - pro každý soubor jsou v ní vyhrazeny tři bajty: první je informační (může obsahovat informaci o tom, že soubor je spustitelný, označený, atd.); v druhém a třetím je uložena adresa hlavičky v adresáři.

1.5 Závěrem...

Možná, že Vám přijde nudné zabývat se pouze vymazáváním souborů nebo vyhledáváním hlaviček... v tuto chvíli zatím o ničem jiném mluvit nešlo, protože jsme si ještě neřekli, jak číst a zapisovat sektory. Dostaneme se k tomu v následující kapitole.

Další operace, které se odehrávají pouze v paměti a nepotřebují disk, jsou změna atributů, přípony, startovní adresy, jména souboru a jména diskety - určitě je budete se stávajícími informacemi schopni napsat sami. Dále by šlo napsat obnovování souborů (není zase tak složité), mapování souborů (ve kterých sektorech se nalézají), katalog a informace o sektorech na disku (vadné, obsazené, prázdné). A to bude poslední rutina, kterou si v první kapitole uvedeme. Bude se jmenovat **INFO** a nebude mít žádné vstupní parametry.

```

INFO      ld    hl,0           ;nejprve vynulujeme počítadla
          ld    (BAD+1),hl   ;špatných,
          ld    (GOOD+1),hl  ;dobrých,
          ld    (FREE+1),hl  ;a volných sektorů
          ld    bc,1705      ;maximální počet sektorů do HC
I00       dec   bc          ;začni od sektoru 1704 a postupně snižuj
          ld    h,b         ;číslo sektoru kopíruj do HL
          ld    l,c
          call  FDBLOCK     ;vzvedni obsah položky sektoru do DE
          or    a           ;shoď případné CARRY
          ld    hl,3583     ;odečtením testuj vadný sector
          sbc   hl,de
          jr    nz,I01      ;není vadný, skoč dál
BAD       ld    hl,0       ;zvyš počet vadných sektorů
          inc   hl
          ld    (BAD+1),hl
          jr    I02
I01       or    a           ;pokračuj ve smyčce
          ld    hl,3549     ;odečtením testuj nedostupný sektor
          sbc   hl,de
          jr    z,I02       ;je-li nedostupný, pokračuje ve smyčce
          ld    a,d         ;teď testuj, jestli je sektor prázdný
          or    e           ;(potom DE = 0)
          jr    nz,GOOD     ;není, skoč (je alespoň dobrý)
FREE      ld    hl,0       ;zvyš počet prázdných sektorů
          inc   hl
          ld    (FREE+1),hl ;prázdný sektor je zároveň i dobrý, takže
GOOD      ld    hl,0       ;zvyš. počet dobrých sektorů
          inc   hl
          ld    (GOOD+1),hl
I02       ld    a,b         ;když je BC > 0
          or    c
          jr    nz,I00      ;opakuj smyčku,
          ret              ;jinak se vrať

```

Po proběhnutí podprogramu si můžete z adres BAD+1, GOOD+1 a FREE+1 vyzvednout údaje o počtu vadných, dobrých (volných + použitých = použitelných) a volných (nepoužitých) sektorů. Vynásobíte-li počet

volných sektorů 512, získáte údaj o zbývajícím volném místě na disku v bajtech (tak, jak ho vypisuje příkaz CAT). Připojíte-li k tomu výpis souborů a hezky to všechno vytisknete na obrazovku, získáte nepochybně kvalitnější katalog, než nabízí MDOS.

UPOZORNĚNÍ:

Všechny doposud uvedené rutiny budou bezchybně pracovat pouze v případě, že bude od adresy DIR nahráno z disku první čtrnáct sektorů (zajišťuje program v Basicu). Protože k rutinám není „připojen“ žádný kontrolní tiskový výstup (myslím na obrazovku), přesvědčte se o jejich správné funkci monitorem (DEVAST, PROMETHEUS...) a ihned uvidíte, jestli pracují tak, jak mají; nebo - připište si do programu ladící výpisy.

Budete-li si chtít adresář (i s úpravami) uložit zpět na disk, nahraďte v basicovém programu na řádce 20 příkaz READ* příkazem RESTORE*.

II. Popis vybraných rutin MDOSu, jejich volání

Podobné rutiny, jaké jsem napsal do 1. kapitoly, má v sobě i MDOS. MDOSovské rutiny ovšem nepracují s bootem, FAT a adresářem najednou (mají k dispozici pouze 1,5 kilobajtu pracovního prostoru) - tahají si je po částech, a proto jsou (zvláště skupinové operace) pomalejší.

Nejčastější diskovou operací, kterou potřebujete, je buď nahrání nebo uložení souboru. Není-li Váš program přímo nadstavbou MDOSu (jako Tools), spokojte se s rutinami, které Vám poskytuje sám MDOS a nepoužívejte vlastní (práci by Vám to spíš zkomplikovalo než zjednodušilo, navíc to stojí daleko více místa).

2.1 Komunikace MDOSové ROM a normální ROM

Abyste vůbec mohli MDOS použít, musíte ho dokázat zavolat. Z basicu to nic není - napíšete si LOAD* a o ostatní se postará interpret; ve strojovém kódu to chodí trochu jinak, ale v žádném případě není pravda, že je to nějak zvláště komplikované a náročné a buhvícoještě, jak občas někdo tvrdí.

Běží-li strojový program, je normálně od nuly do 16383 klasická ROM. Abyste mohli zavolat rutinu MDOSu, musíte přestránkovat. Uděláte to takto:

```
SHADE      rst 0          ;přestránkuj
           di            ;zakaž přerušení
           . . .
```

Každý asi namítne, že se program po RST 0 zhroutí - a má pravdu, pokud ale nepřipojíme krátký basic:

```
10 POKE #247,79: RANDOMIZE USR START
```

Pro příště již nemůžete spouštět strojové programy přímo z assembleru, ale musíte se vrátit do basicu a zadat GOTO 10. Namísto START si napište adresu, kde Váš program začíná.

Teď si vysvětlíme, co jsme vlastně všechno provedli. Začneme Basicem: příkaz POKE # ukládá do stínové paměti RAM v disketové jednotce. Na relativní adrese 247 (absolutně 16119) je normálně uložena hodnota 32; podívejme se, co její změnou dosáhneme: po RST 0 se program nejdříve dostane do ROM disketové jednotky (dále jen DROM) a až teprve po nějaké době pokračuje v normální ROMce (dále jen NROM). Po chvíli běhu v DROM dojde až sem:

```
           push hl        ;ulož obsah HL
           ld  hl,16111,SYSMRK ;do HL začátek kontrolní tabulky
           ld  b,8        ;do B délku tabulky
00122
CHCOLD    ld  a,h         ;dej H do A
           xor  l         ;vyxoruj s L
           cp  (hl)       ;porovnej s obsahem tabulky
           jp  nz,180,COLD ;nerovná-li se, skoč na reset DRAM
           inc hl         ;další bajt v tabulce
           djnz 122,CHCOLD ;opakuj B krát
           ld  a,(hl)     ;čti první bajt za tabulkou (adr. 16119)
           ld  (hl),32    ;na 16119 ulož 32
           cp  79         ;pokud je A = 79
           jp  z,319,ROMRET ;skoč
           cp  69         ;otestuj ještě 69
           jp  z,320,ERRCOD ;a kdyžtak skoč
           pop hl        ;obnov HL
           . . .
```

Tato část programu probíhá po každém přestránkování do DROM. Nejprve je zjištěno, jestli není narušena kontrolní tabulka a pokud ano, bude potřeba zinicilizovat MDOS (bývá to po zapnutí počítače nebo po úmyslném narušení tabulky). Je-li tabulka v pořádku, testuje se obsah adresy 16119 (a vždy je tam vrácena hodnota 32). Program může pokračovat tudy (to právě potřebujeme):

```
00314
ROMRET    pop    hl            ;obnov registry, které jsi někdy na
          pop    af            ;začátku testů uložil na zásobník PoP bc
          ex     (sp),hl
          ei                     ;povol přerušení
          ret                     ;vrať se
```

Převratnost tohoto programu spočívá v tom, že při návratu nepřestránkuje a tudíž od adresy 0 zůstane DROM. Z toho vyplývá závěr: je-li na adrese 16119 v DRAM hodnota 79, program se po instrukci RST 0 (můžete použít i CALL 0) nezhroutí, ale pouze přestránkuje a vrátí se. Ovšem pozor - používáte-li zrovna přerušení IM 2, které má vektor ukazující na tabulku v NROM (na adr. 14592), přerušení se Vám zhroutí (nedáte-li ihned po po přestránkování DI), protože přestránkováním tabulka „zmizí“! I vzhledem k upraveným ROMkám, stodvacetsmičce (+, 2+, 2A, 3...) a jim podobným tuto tabulku radši nepoužívejte, vyhnete se zbytečným komplikacím.

Program v DROM testuje nejen hodnotu 79, ale i 69. Zkuste:

```
10 POKE #247,69                ;na 16119 dej 69
20 POKE 16384,243              ;na 16384 dej kód instrukce DI
30 POKE 16385,118              ;na 16385 dej kód instrukce HALT
90 RANDOMIZE USR 16384
```

Po spuštění se program kousne - vyplývá to ze zakázání přerušení a následného nekonečného čekání na něj. Stiskněte RESET (majitelé Spectra „gumák“ mají smůlu) a... počítač vypsal 0 OK, 40:1 (pokud ne, je mi líto, ale na mém plusku i firemním „emku“ to dělal). To co jste viděli, bylo elegantní softwarové NMI.

Pokud Vás zajímá, jak po instrukci POP HL program v DROM pokračuje - je zjišťováno, zda-li má být proveden některý z příkazů MDOSu; když ne, skočí se na reset.

Podprogram na přestránkování trochu upravíme:

```
SHADE     call    IM1            ;nastav přerušení IM1 (není vždy nutné)
          rst     0              ;přestránkuj
          di                     ;zakaž přerušení
          ld     a,79            ;na 16119 dej znovu 79 (přepíše hodnotu
          ld     (16119),a      ;32, kterou tam dal MDOS)
          ret                     ;vrať se

IM1       ld     iy,23610
          ld     a,63
          ld     i,a
          im     1
          di
          ret
```

Rutina SHADE zajišťuje, že "příště" budete moci přestránkovat bez použití POKE # (tj. bez návratu do Basicu). Teď můžete napsat třeba takový program:

```
STANDROM  equ     #1700
START     call    SHADE          ;přestránkuj do DROM
          ld     hl,0            ;přenes DROM do RAM na adr. 40000
```

```

ld    de,40000
ld    bc,14336
ldir
call  STANDROM      ;přestránkuj zpátky do NROM
...
ret                    ;vrať se

```

Zastavím se u adresy #1700, STANDROM (5888 dekadicky). Je na ní instrukce RET (a to jak v DROM, tak v NROM) - ze softwarového hlediska CALL STANDROM nedělá nic, ale zato hardware při skoku na 5888 automaticky přestránkuje zpátky normální ROMku.

Chcete-li, aby Váš program běžel při NROM a jen občas si volal DROM, stránkujte tam a zpět vždy pomocí SHADE (kdyby se v něm neukládala na 16119 hodnota 79, tak by již při druhém zavolání rutina nepřestrátkovala, ale zhroutila se) a STANDROM. Chcete-li, aby program běžel po celou dobu "ve stránce" (při aktivní DROM), přestránkujte jenom jednou (na začátku). Podprogramy z NROM pak zavoláte takto:

```

...
rst   #28                ;restart pro volání NROM
defw  ADDRESS            ;adresa podprogramu v NROM
...

```

Musíte mít neustále na paměti, že "dole" je jiná ROMka a že nemůžete použít ani normální přerušení IM1, ani znakovou sadu, atd. - protože v DROM nejsou k dispozici. Můžete volat pouze restarty 16, 24, 36 (dekadicky) - fungují stejně jako v NROM. Všechno ostatní je jinak.

Programy, které běží "ve stránce" (Tools 80, Heroes '92), nejdou „snapnout“ - na tlačítko SNAP buď nereagují vůbec nebo udělají něco nepředvídatelného. Občas se i velice špatně vymazávají, stisk RESETu neberou vážně apod. - proto do všech programů dávejte funkci QUIT (vymazání) - majitelé "gumáků" aspoň nemusí neustále vytahovat přívod proudu a program působí solidněji.

```

QUIT      rst   0
          rst   0

```

Na závěr si uvedeme program, který na obrazovku vytiskne aktuální mechaniku

```
10 POKE #297,79: RANDOMIZE USR 4e4
```

Od adresy 40000 přeložte stroják:

```

START     ld    a,2                ;otevři kanál pro tisk na obrazovku
          call  #1601
          rst   0                    ;přestránkuj do DROM
          di                    ;zakaž přerušení
          ld    a,(16042)           ;přečti aktuální drive
          call  #1700               ;přestránkuj do NROM
          rst   #10                 ;vytiskni aktuální drive (A, B)
          ei                    ;povol přerušení
          ret                    ;vrať se

```

Šlo by to napsat i jinak, třeba:

```

START     rst   0                    ;přestránkuj
          di                    ;zakaž přerušení
          ld    a,2                ;kanál obrazovky
          rst   #28                 ;volej rutinu v NROM na adrese #1601
          defw  #1601
          ld    a,(16042)           ;přečti aktuální drive

```



```

rst #10 ;vytiskni drive
ei ;povol přerušení
jp #1700 ;přestránkuj zpět a zároveň se vrať

```

První program si do stránky skáče pouze pro číslo drivu a jinak běží při NROM, zatímco druhý funguje ve stránce a NROM vrací až na závěr.

2.2 RESET MDOSu a inicializace připojených mechanik

Jistě důvěrně znáte tu chvíli, kdy obraz zčervená a mechanika zahrčí... RESET.

```

00180
COLD      ld hl,0 ;odkud (DROM)
          ld de,14336 ;kam (DRAM)
          ld bc,2048 ;kolik (velikost DRAM)
          ldir ;přenes kus paměti
          ld hl,0 ;nastav stejné parametry
          ld de,14336
          ld bc,2048

00200
RAMTES    ld a,(de) ;porovnávej, jestli je v paměti to,
          inc de ;co se do ní přeneslo
          cpi ;(test na vadnost paměti)
          jr nz,303,RAMERR ;chyba - skoč
          jp pe,200,RAMTES ;opakuj EC krát
          ld hl,14336 ;začátek DRAM
          ld d,h ;okopíruj do DE
          ld e,l
          inc de ;posuň o bajt dál
          ld bc,2048 ;velikost DRAM (stačilo by LD B,8)
          ld (hl),0 ;zapln nulu (lépe: LD (HL),C)
          ldir ;celou DRAM
          ld hl,16111,SYSMRK ;zde vytvoř kontrolní tabulku
          ld b,8 ;o délce osm bajtů

00227
SETMRK    ld a,h ;H okopíruj do A
          xor l ;vyxoruj s L
          ld (hl),a ;ulož do tabulky
          inc hl ;další bajt
          djnz 227,SETMRK ;opakuj B krát
          ld (hl),32 ;na 16119 dej 32
          ld a,127 ;testuj řadu kláves B N M CS SP
          inc a,(254) ;když je současně stisknuto B M SP
          rra ;tak dej na 15968 nenulovou hodnotu
          jr c,257,NODEB ;(zapomenuté ladící tisky...)
          rra
          jr nc,257,NODEB
          rra
          jr c,257,NODEB
          rra
          jr nc,257,NODEB
          rra
          jr c,257,NODEB
          ld (15968),a ;nastav sys. pro. DEBUG

```

00257

```

NODEB      ld    de,15872      ;nastav systémové proměnné
           ld    hl,3832      ;pro mechaniky A, B
           ld    bc,24        ;(stačilo: LD C,24)
           ldir
           ld    a,65         ;aktuální je drive A
           ld    (16042),a
           ld    hl,22528     ;začátek atributů ve videoram
           ld    de,22529     ;nastav červený papír a inkout
           ld    bc,768       ;(maže se o jeden bajt navíc!)
           ldir
           ld    a,2          ;červený border
           out   (254),a
           ld    sp,16389     ;vrchol zásobníku na konec DRAM
           call  8726,HWINIT  ;zinitializuj mechaniky
           di                ;zakaž přerušování
           ld    sp,4121      ;vrchol zásobníku ukazuje na 4121
           jp    5888         ;přestránkuj a pokračuj resetem v NROM

```

inicializace MDOSu sice funguje, ale každý zkušený programátor by ji asi napsal lépe... navíc ten zapomenutý ladící výpis při každém šáhnutí na disk působí amatérsky (zkuste stisknout B M SP, držet je během resetu a pak dát třeba katalog disku). Zajímavý moment je až na konci podprogramu: příkaz LD SP,4121 směřuje vrchol zásobníku do NROM, což je celkem neobvyklé. V NROM jsou na tomto místě bajty 1 a 0 - tedy adresa 1. Při skoku na 5888 dojde k přestránkování a instrukce RET si "vyzvedne" ze zásobníku návratovou adresu (musí to být "jedna" a ne "nula", protože při skoku na nulu dojde k přestránkování do DROM (program by se nechutně zacyklil)) - a pokračuje klasickým RESETem RAMky.

Při špatně provedené inicializaci DRAM program skáče na adr. 303 (možná, že už se Vám někdy stalo, že jste nedozastrčili kabel mezi počítačem a mechanikou a v borderu se rozjely barevné proužky a počítač „vrčel“ - to má na svědomí právě tahle část programu):

```

00303
RANIERR   xor    a            ;A = 256 00304
CYCLE     dec    a            ;sniž A
           out   (259),a      ;obarvi border
           ex   (sp),hl       ;tyto dvě instrukce jsou zde kvůli
           ex   (sp),hl       ;časové prodlevě (jedna trvá 19T)
           jr   nz,304,CYCLE  ;opakuj A krát
           jp   0             ;zkus znovu RESET

```

Než skončíme s inicializacemi, zastavíme se ještě u podprogramu na adrese 8726, HWINIT (je volán před koncem inicializace MDOSu), který při RESETu roztáčí motory a nastavuje správné hodnoty systémových proměnných mechanik. Nebude na škodu, když si o systémovkách nejdříve něco málo řekneme:

Začátek systémových proměnných MDOSu je na adrese 15872. Zde je vyhrazeno 48 bajtů pro 4 připojitelné mechaniky (pro každou 12). Bohužel, některé rutiny (včetně inicializačních) jsou „opraveny“ tak, že pracují pouze se dvěma disketovými jednotkami (škoda). Oněch dvanáct bajtů pro každou mechaniku (resp. jen pro A: a B:) se nastavuje na adrese 257 (viz. výše) a dále upravuje podle skutečnosti právě v podprogramu HWINIT na adrese 8726 (viz. níže). Význam jednotlivých bajtů je shodný s tím, co jsem uvedl v kapitole 1.2 u popisu uložení formátu v bootu. Při formátování se na disketu kopírují informace právě odsud.

| bajt | význam |
|-------------|--|
| 0 | 0. bit je nastaven, je-li mechanika připojena |
| | 7. bit je nastaven, je-li drive aktuální (nastavený příkazem MOVE) |
| 1 | 4. bit - 1 = oboustranný formát, 0 = jednostranný formát |
| | 3. bit - 1 = D40, 0 = D80 |
| | 2. bit - I = mechanika B, 0 = mechanika A |

| | |
|------|--|
| 2 | počet stop na straně (když je zde nula, drive není definován - C:, D:) |
| 3 | počet sektorů ve stopě |
| 4 | číslo stopy, kam byla naposled nastavena hlava |
| 5-7 | jako 1-3 |
| 9-11 | nuly |

Mělo by platit, že v bajtech 1, 2, 3 jsou parametry diskety a v bajtech 5, 6, 7 parametry mechaniky. Pokud toto rozdělení porušíte a nesprávně přepíšete bajty 5, 6, 7, nebude to vadit v tom případě, že budete používat své vlastní rutiny. Zavoláte-li ovšem za těchto okolností MDOS (nebo se vrátíte do Basicu) můžete se dočkat spousty chybových hlášení...

Výčet a popis některých důležitých systémových proměnných MDOSu najdete na konci této příručky v příloze A. A už teď se vrátíme k nakousnuté inicializační rutině...

```

08726
HWINIT      ld      a,208
            out     (129),a
            xor     a                ;první drive (A:) má číslo 0

08731
HWINI0     push    af                ;uschovej číslo drivu
            call   8620,DRVCOMP      ;vypočti začátek oblasti proměnných drivu
            push   ix                ;přes zásobník adresu zkopíruj do HL
            pop    hl
            inc    hl                ;posuň se na 1. bajt (počítáno od nuly)
            ld     e,1                ;okopíruj adresu do DE
            ld     d,h
            inc    hl                ;posuň se na 5. bajt
            inc    hl
            inc    hl
            inc    hl
            ld     bc,3                ;kopíruj obsahy bajtů 5, 6, 7 do 1, 2, 3
            ldir
            res    0,(ix+0)           ;zruš` signály: drive připojen,
            res    7,(ix+0)           ;drive aktuální
            ld     a,(ix+2)           ;zjistí, zda je drive vůbec definován
            and    a
            jr     z,8817,HWINI1      ;není, zkus další
            pop    af                ;obnov číslo drivu
            push   af
            call   9547,DSELECT       ;vyber mechaniku
            call   9035,HOME          ;hlavu pošli na začátek disku
            out    (135),a
            and    4
            jr     z,8817,HWINI1      ;skoč, když drive není připojen
            set    0,(ix+0)           ;nastav signál: je připojen
            ld     (ix+4),0           ;vynuluj syst. proměnnou
            ld     a,54
            ld     d,16
            call   9024,SEEK
            ld     a,02
            ld     d,16
            call   9024,SEEK
            and    4
            ld     a,40                ;čtyřicet stop do A
            jr     nz,8808,TRK40      ;skoč při mechanice D40
            ld     a,80                ;oprav počet stop na osmdesát

```

```

08808
TRK40      ld      (ix+6),a      ;zapiš počet stop do syst. prom.
           ld      (ix+3),a
           call   9035,HOME    ;hlava na začátek disku

08817
HWINI1    pop     af          ;obnov číslo disku
           inc     a          ;zvyš ho
           cp     2          ;jsou povoleny dva (A:, B:)
           jr     nc,8731,HWINI0 ;takže smyčka proběhne pouze 2x
           call  9526,DSKSTP   ;vypni motory
           ld     a,136       ;inicializuj interface v disk. jednotce
           out    (127),a     ;inicializace obvodu 8255 (špatně!)
           ld     a,15        ;nastavení 7. bitu portu C
           out    (127),a
           ld     a,11        ;nastavení 5. bitu portu C
           out    (127),a
           in     a,(95)      ;čtení portu C
           and    240         ;zachovej pouze 5. a 7. bit
           cp     160        ;byla zjištěna jiná 8255 - nastav ZERO
           ld     a,155       ;všechny porty nastav jako vstupní
           out    (127),a
           ret     z          ;vrať se při jiném interface (ZERO)
           ld     a,32        ;odblokuj vnitřní interface
           out    (145),a
           ret                ;vrať se

```

V inicializaci obvodu 8255 je chyba. Správně by měl program dělat toto:

- 1) zjistí, jestli už náhodou není jedna 8255 připojená (platí třeba pro Gamu)
- 2) je - vrať se (RET Z)
- 3) není - odblokuj vnitřní interface v disketové jednotce

Chyba je hned v první instrukci LD A,136, která nastavuje vrchní polovinu portu C jako vstupní, a proto do něj není zapsána testovací hodnota - inicializace vnitřního interface pak proběhne vždycky. Namísto LD A,136 mělo v programu být LD A,128.

MEZIKAPITOLA O OŠETŘENÍ CHYBOVÝCH HLÁŠENÍ

Při volání rutin MDOSu budete zajisté chtít, aby se program při chybě vracel do Vašich rutin a nikoliv, aby skončil hláškou v basicu nebo náhlým zhroucením. Následující nastíněné ošetření chyb neřeší hlášení typu „(R=Retry)“ - to proto, že takováto hlášení se vypisují přímo v rutině pro práci se sektorem a nespádají mezi klasické chyby (jak se jich zbavit se dozvíte ve třetí kapitole).

Opište si následující program; rutiny v něm uvedené budeme používat jednou provždy, proto se k nim v dalším textu již nebudu vracet, ale budu se na ně pouze odkazovat názvem.

```

START      jp     G0          ;poprvé skok na začátek G0
           ld     (RETURN+1),sp ;ulož vrchol zásobníku
           ...          ;nějaký Program...
RETURN     ld     sp,0        ;obnov zásobník
           ret                ;vrať se
ERROR      call  IM1         ;nastav IM1 (pro jistotu)
           call  737         ;vypni motory a přestráknij zpět
           ld     a,(iy+0)    ;přečti si číslo chyby
           ld     (MESSAGE+1),a ;a ulož je pro pozdější zpracování

```

```

        ld    (iy+0), -1    ;nastav: žádná chyba
        ld    a, 1         ;ukazuj na 1. příkaz
        ld    (23620), a    ;na řádku č. 10
        ld    hl, 10       ;(7e tam ten POKE# & spol.)
        ld    (23618), hl
        ld    sp, (23613)   ;vyzvedni vrchol "chybového zásobníku^
        ei                    ;povol přerušení
        jp    7030, STMT-RET ;proved' "GO TO 10"

RESERR  ld    de, 0        ;hodnota, uložená podprogramem SETUP
        call IM1           ;nastav IM1
        ld    hl, (23613)  ;na vrchol "chybového zásobníku"
        ld    (hl), e      ;ulož původní hodnotu
        inc  hl
        ld    (hl), d
        ld    (iy+0), -1   ;signál: žádná chyba
QDONE   ld    a, 0         ;je-li A = 0, byla disková operace chybná
        or    a            ;a v tom případě nastav ZERO
        ld    a, 0         ;vynuluj A a neznič příznaky (!)
        jr    QYD          ;skoč dopředu
YDONE   ld    a, 1         ;A = 1 ...disková operace proběhla OK
Qy0     ld    (QDONE+1), a ;ulož na Q1bNE+1
        ret                ;vrať se

SETUP   ld    (START+1), hl ;HL = návratová adr. po diskové operaci
SETERRS ld    hl, (23613)  ;čti vrchol "chybového zásobníku"
        ld    e, (hl)      ;hodnotu na něm uloženou
        inc  hl
        ld    d, (hl)
        ld    (RESERR+1), de ;přendej na RESERR+1
        ld    (hl), ERROR/256 ;a nahraď ji adresou rutiny ERROR
        dec  hl
        ld    (hl), ERROR?256
        ret                ;vrať se

SETERR  call SETERRS      ;nastav chybový zásobník
SET1    call SHADE        ;přestránkuj
SET2    call 7841         ;vyber aktuální drive a roztoč ho
        call NAMER        ;nastav jméno souboru
        call 8491         ;zjistí, jestli už soubor existuje
        ret  nz           ;pokud ne, vrať. se
        ld    (15986), hl  ;ulož si relativní adresu jeho hlavičky
        ret

SHADE   call IM1          ;nastav IM1
        rst  0            ;přestránkuj
        di                ;zakaž přerušení
        ld    a, 1         ;nastav: staré soubory jsou přepisovány
        ld    (15970), a
        ld    a, 79        ;hodnota pro "další^ přestránkování"
        ld    (16119), a
        ret                ;vrať se

NAMER   ld    hl, 16042    ;nastav systémové proměnné

```

```

        ld    de,16000
        call LD10
        ld    hl,FILENM+1    ;přenes do nich i jméno souboru
        call LD10
        ld    a,(FILENM)    ;přečti příponu souboru
        ld    (de),a        ;a také ji ulož
        ld    a,(15979)    ;nastav systémové proměnné
        inc  a
        ld    (15985),a
        ret                ;vrať se

MOVER   ld    de,FILENM    ;přesuň hlavičku soubozv z HL na DE
        ld    bc,17        ;délka hlavičky
        ldir
        ret                ;vrať se

LD10    ld    bc,10        ;z HL na DE přenes 10 bajtů
        ldir
        ret                ;vrať se

FILENM  defb  0            ;přípona souboru
        defs 10            ;jméno souboru
        defw 0,0,0        ;délka, st. adr., třetí parametr (Basic)
MESSAGE ld    a,0          ;číslo chyby je v A (0 = OK),
MESSAG2 ld    sp,(RETURN+i) ;záleží na Vás, jak s ním naložíte...
        ...

```

Tuto spoustu rutin upotřebíme postupně podle potřeby. Co přesně dělají, vysvětlím hned: Celý chybový systém je postaven na jednoduchém principu - na stejném, jako jsou dělány rutiny typu ON ERROR, ON BREAK (jejich popis naleznete třeba v programu Supercode nebo nějaké příručce o Spectru). Na vrchol chybového zásobníku se uloží namísto adresy 8 (standardní ošetření chyb Basicem) adresa vlastní rutiny - v našem případě je to ERROR. Nastane-li tedy při spolupráci s disketovou jednotkou chyba, program "doskáče" až do ERRORu; ošklivé je, že při tom skákání interpret uloží na 16119 hodnotu 32 a nám by se už příště nepovedlo přestránkovat. Proto se ERROR vrací do Basicu na řádek 10, kde provede POKE #247,79 (#247 = 16119) a znovu spustí stroják od adresy START. A tím se dostáváme k zdánlivě nelogickému START JP GO; kdo pozorně četl popis rutiny SETUP, nemohlo mu ujít, že na START+1 se něco ukládá - to něco, to je adresa, kde má program po chybě pokračovat.

Abych to toho vnesl jasno, uvedu krátký příklad použití rutin:

```

START   jp    GO            ;prozatím skok na GO
GO      ld    (RETURN+1),sp ;ulož zásobník
        ...
        ld    hl,R1        ;návrát po chybě
        call SETUP        ;nastav ošetření chyby
        call SET1         ;přestránkuj, roztoč motor, hledej soubor
        call ???          ;volej diskovou operaci (SAVE, LOAD)
        call YDONE        ;nastav: operace proběhla v pořádku
R1      ld    sp,(RETURN+1) ;obnov zásobník
        call RESERR       ;zruš ošetření chyb a zjisti průběh akce
        jp    z,MESSAGE   ;skoč, pokud nastala při operaci chyba
        ...
RETURN  ld    sp,0         ;obnov zásobník
        ret                ;vrať se

```

Upozornil bych jenom na zásobník - pokud nevíte, v jaké jeho úrovni se právě nacházíte, použijte raději tento postup:

```

...
ld    (R1+1),sp
call  ???
call  YDONE
R1    ld    sp,0
...

```

Je sice o pár bajtů delší, ale zato vždy použitelný.

To bylo skoro všechno, co bych Vám k ošetření chyb chtěl říci. Podprogram MESSAGE jsem úmyslně nedokončil a nechal na Vás, abyste si ho zpracovali podle vlastního uvážení - můžete v něm vypisovat hlášky v češtině nebo jakoukoliv chybu ignorovat... to už závisí jen na Vás.

2.3 Formátování

Podprogram basicového příkazu FORMAT začíná na adrese 4896. Nejprve se zjišťuje, je-li jméno a vše co s ním souvisí syntakticky v pořádku (nepříliš zajímavá pasáž) a teprve po dalších testech a nastaveních se začne formátovat (okomentuji jen kousek):

```

. . .
04945    ld    a,(ix+5)        ;okopíruj systémové proměnné
          ld    (ix+1),a
          ld    a,(ix+6)
          ld    (ix+2),a
          ld    a,(ix+7)
          ld    (ix+3),a
          ld    a,(16020)      ;testuj, jde-li se o jednostranný formát
          cp    "S"
          jr    nz,4974,RF0401 ;ne, obskoč
          res   9,(ix+1)      ;vymazání příslušného bitu

04974
RF0401   call  8571,KILFAT    ;čisti sytémovky
          push hl              ;ulož registry
          push de
          ld    de,993,DOSMES ;tabulka hlášení
          ld    a,63+128      ;invertované číslo hlášení
          call  8639,SYSRQ    ;vypiš „All data...” a čekej na stisk
          pop  de              ;obnov registry
          pop  hl
          ret  nc              ;vrať se, nebylo-li stisknuto P (R)

04990
FOR_ME   ld    a,(15979)      ;aktuální drive
          call  9547,DSELCP    ;vyber mechaniku
          call  9035,HOME      ;hlava "domů"
          ld    c,(ix+2)      ;do C počet stop
          bit  4,(ix+1)        ;testuj formát
          jr    z,5010,RFORMS  ;při jednostranném obskoč
          rlc  c ;vynásobení počtu stop dvěma

05010
RFORMS   ld    b,0            ;H = 0

05012
RFORM6   push  bc              ;ulož počítadlo
          ld    de,256         ;jedna stopa

```

```

ld    a,(15979)      ;aktuální drive
call  8860,BFORM    ;formátuj stopu
pop   bc            ;obnov počítadlo
inc   b             ;zvyš počet zformátovaných stop
ld    a,b           ;přendej do A
cp    c             ;porovnej s C
jr    nz,5012,RFORM6 ;nejdou všechny - opakuj dokud B<>C
. . .

```

Podprogram dále pokračuje podobnou smyčkou, ve které kontroluje a počítá zformátované sektory; potom vytvoří boot (uloží formát, jméno, náhodný dvojbajt, značku MDOSu) a FAT a nakonec vypíše informace o výsledku FORMATu.

Vlastní zformátování stopy provádí podprogram BFORM (8860) - jeho detailní popis a i popis celého FORMATu se vymyká rozsahu této příručky a hodí se spíše do komentovaného výpisu ROM D40/80. Když už jsme u těch komentovaných výpisů: co já vím, má snad každý programátor, který něco pro disketovou jednotku něco většího udělal, svůj vlastní (výpis samozřejmě). Ale on je rozdíl mezi poznámkami, ve kterých se vyzná pouze jejich autor a mezi něčím, co by se mohlo publikovat asi proto žádný komentovaný výpis zatím nevyšel (jó, kdyby byl extra zájem... dalo by se... hmmm).

Programátor většinou potřebuje ve svých programech použít FORMAT tak, aby umožnil buď uživateli nebo sobě přímo zadat parametry diskety a nebyl navíc obtěžován neestetickým výpisem "All data..." do editační zóny. Proto začátek rutiny napíšeme sami a do MDOSovského formátu vskočíme až ve vhodnou chvíli.

```

FOR_ME  equ  4990      ;naš vstupní bod do MDOSovského formátu
START   jp    GO
GO      ld    (RETURN+1),sp
. . .
ld    (R1+1),sp      ;ulož současnou hodnotu SP
call  FORDIS        ;volej zformátování disku
R1      ld    sp,0     ;nastav SP
call  RESERR        ;odpoj ošetření chyb a testuj, byla-li
jp    z,MESSAGE     ;chyba - ano, skoč
. . .
RETURN  ld    sp,0
ret
FORDIS  ld    hl,R1    ;při chybě skákej na R1
call  SETUP        ;nastav ošetření chyb
call  SHADE        ;přestránkuj
ld    hl,DKNAME    ;jméno disku
ld    de,16010     ;přenes do syst. Proměných
call  LD10
call  8609         ;nastav IX na zač. dat. oblastí drivu
ld    a,(ix+5)     ;okopíruj hodnoty
ld    (ix+1),a
ld    a,(ix+6)
ld    (ix+2),a
ld    a,(ix+7)
ld    (ix+3),a
call  8571         ;čisti systémovky
call  FOR_ME      ;volej FORMAT v DROM
ld    a,79        ;obnov obsah 16119 pro příště
ld    (16119),a
call  737         ;vypni motory a nastránkuj NROM
JP    YDONE       ;s nastavením: „operace OK" se vrat

```



```
DKNAME      defm  "NONameDisk"
```

Takto napsaný podprogram formátuje diskety podle nadefinovaných parametrů v systémových proměnných. Upravíme jej, aby formátovl podle parametrů námi zadaných; přepíšeme jen tu část, která je mezi voláním SET IX a K_FAT.

```
...
ld    a, TRACK
ld    (ix+2), a
ld    a, SECT
ld    (ix+3), a
ld    a, OPT
ld    (ix+1), a
...
```

Parametr OPT (option) si vypočtete na základě tabulky z kap. 2.2 - musíte vzít v úvahu, pracujete-li na mechanice D40 / D80 A nebo B a chcete-li oboustranný nebo jednostranný formát. Namísto TRACK napište počet stop na stranu, namísto SECT počet sektorů ve stopě (viz. tabulka).

Následující tabulka Vám usnadní přehled o možných maximálních a minimálních formátech...

| | <u>D40</u> | | <u>D80</u> | |
|-----------------|-------------------|---------|-------------------|---------|
| | stopy | sektory | stopy | sektory |
| standard | 40 | 9 | 80 | 9 |
| maximum | 43 | 10 | 83 | 10 |
| minimum | 1 | 8 | 1 | 8 |

Maximální hodnoty závisí na hardwarových možnostech jednotlivých mechanik (ne každá mechanika je má právě takovéto). Sektorů musí být na stopě alespoň 6 (při jakémkoliv počtu stop), jinak by se Vám disketu nepodařilo zformátovat. Těžko říct, jestli se to dá považovat za chybu - můžou za to rutiny pro práci s FAT, které vyžadují, aby se celá FATka nacházela na jedné stopě.

Před volání podprogramu můžete připojit uživatelskou nadstavbu - zadání výše uvedených parametrů z klávesnice (samozřejmě i s ošetřením hodnot - nezapomeňte na automatickou detekci D40/80 a potvrzení volby).

Snad jedinou nevýhodou tohoto "vlastního" FORMATu je, že ničí obrazovku a vyžaduje existenci basicových systémových proměnných (používá tisk do kanálu obrazovky, CLS, kalkulačku). Pokud byste se chtěli všeho toho zbavit, nezbylo by Vám nic jiného, než celou rutinu opsat (i s některými podprogramy) a opravit (příjemnou zábavu).

2.4 LOAD souboru

Příkaz LOAD* začíná na adrese 5892 a pokračuje pasáží, která je do určité míry stejná jako pro LOAD z pásky. Podprogram posuzuje nejen syntaxi LOADu, ale i SAVE a MERGE. Jsou odchyceny nesmysly jako MERGE DATA, SCREEN\$, CODE, atd. a posouzeno jméno souboru a případné jméno disku.

Z programátorského hlediska jsou zajímavé vyřešeny vstupní body rutiny:

```
05889
RSAVE      ld    a, 0
           defb  33

05892
RLOAD      ld    a, 1
           defb  33

05895
RMERGE     ld    a, 3
           ...
```

Do registru A je ukládáno číslo požadované operace (SAVE = 0, LOAD = 1, MERGE = 3; VERIFY* neexistuje). Zavoláte-li RSAVE, bude program vlastně vypadat takto:

```
05889
RSAVE      ld    a,0
           ld    hl,318
           ld    hl,830
           ...
```

Tímto způsobem se ušetří místo, které by bylo potřeba k obskakování ostatních vstupních bodů (vypadalo by to takhle):

```
05889
RSAVE      ld    a,0
           jr    ROPER

05893
RLOAD      ld    a,1
           jr    ROPER

05897
RMERGE     ld    a, 3
ROPER
```

Dále už se touto nadstavbou zabývat nebudeme (vytváří hlavičku, kontroluje parametry, atd.) a přejdeme rovnou k samotnému nahrání souboru. Oproti magnetofonu je zde rozdíl: zatímco páskový blok dat nemusel mít hlavičku (hlavička je definována jako blok dat o délce 17 bajtů s identifikačním bajtem nula), soubor na disku hlavičku mít musí. Při psaní MDOSovských rutin se autor snažil zachovat jak vstupní, tak výstupní parametry rutin pro LOAD i SAVE (jenom škoda, že ještě nezachoval stejné vstupní adresy) - proto se např. přípony musí přepočítávat z čísel 0, 1, 2, 3 na P, N, C, B, atd.

První, co potřebujete při LOADu souboru udělat, je zjistit, je-li soubor vůbec na disketě. To zajišťuje podprogram na adrese **6507** - abych se přiznal, dodnes jsem ho ještě nepoužil. IX ukazuje za hlavičku (hlavička je "pásková" a je potřeba k ní přidat správnou příponu), HL není pro běh rutiny důležitý a v BC musí být hodnota 17:

```
06507
LOAR01     push hl                ;schovej HL
           push ix                ;ulož ukazatel za hlavičku
           ld    a,(ix-17)        ;přečti si páskový typ souboru
           ld    (ix+0),a
           call 6608,FINTYP       ;nastav příponu
           call 7311,SETACT       ;vyber a rozběhni mechaniku
           call 8491,SEAFIL       ;hledej soubor v adresáři
           jr    nz,6549,TSTSNP   ;nenalezen, skoč dál
           ld    (15986),hl       ;ulož si rel. adr. hlavičky v adresáři
           pop  ix                ;obnov ukazatel za hlavičku
           push ix                ;okopíruj ho do DE
           pop  de
           ldir                    ;přenes hlavičku do adresáře
           ld    l,(ix-17)        ;příponu souboru místo páskového typu
           ld    (ix+0),1
           xor  a                  ;nuluj systémovku
SYSLD      ld    (15978),a
           pop  hl                ;obnov HL
           ret                    ;vrať se
```

Když soubor nalezen nebyl, nemusí to nutně znamenat, že na disku není. Hlavička je pořád zpracovávána jako pásková, ale protože se čte z disku, nemusí LOAD *"HELLO" chtít nahrávat program "HELLO" (přípona P), ale snap "HELLO" (přípona S). Toto uvažuje zbytek podprogramu:

```

06549
TSTSNP    ld    a,(16020)      ;vypadala hlavička jako od programu?
          cp    "P"
          jp    nz,8113,NOTFND ;ne, skoč na chybu „File not found“
          ld    a,"S"          ;ano, zkus tedy, jestli to není snap
          ld    (16020),a
          call  8491,SEAFIL    ;znovu hledej soubor
          jp    nz,8113,NOTFND ;nenalezen (skoč) - nebyl to snap
          ld    (15986),hl     ;ulož adresu hlavičky
          jp    916,SNPLOA    ;skoč na load snapu

```

Další podprogram nahrazuje přímo rutinu LDBLOCK (adr. 1366) z NROM. I volací parametry jsou téměř stejné: IX = místo v paměti, kam nahrát data; DE = délka dat:

```

06579
LOABLK    ld    (15988),ix    ;ulož IX na LOADIX
          ld    (15990),de    ;ulož DE na LOALFN

6582
LOAB21    call  7311,SETACT   ;vyber a rozběhni mechaniku
          ld    hl,(15988)    ;přečti si kam
          ld    de,(15990)    ;kolik
          call  8101,LOAFND   ;a nahraj data ze souboru

06595
LOAFIN    ld    ix,(15988)    ;vyzvedni IX
          ld    de,(15990)    ;vyzvedni DE
          add  ix,de          ;přičti
          xor  a              ;nastav ZERO a A = 0
          scf                ;nastav CARRY
          ret                ;vrať se

```

Rutina LOABLK nahraje od adresy IX DE bajtů ze souboru, který byl nastaven podprogramem LOAR01. Na závěr nastaví IX, A a příznaky stejně, jako je vrací rutina LDBLOCK v NROM, když byl blok dat nahrán bez chyby.

Potřebujete-li nahrát ze strojového kódu soubor, uděláte to např. takto (netvrdím, že je to jediný správný postup, ale je fungující):

```

START    jp    G0
G0       ld    (RETURN+1),sp
        ...
        ld    (R1+1),sp
        ld    ix,HEAD        ;hlavička souboru
        call  LOAFIL         ;nahraj soubor
R1       ld    sp,0
        call  RESERR         ;zruš ošetření chyb a testuj jestli
        jp    z,MESSAGE      ;k nějaké došlo - pokud ano? skoč
RETURN   ld    sp,0
        ret
LOAFIL   push  ix            ;ulož ukazatel na hlavičku
        ld    hl,R1          ;při chybě skákej na R1
        call  SETUP
        call  SHADE
        pop  hl              ;ukazatel na hlavičku

```

```

call MOVER          ;přesuň ji na FILEL~R4
call SET2           ;hledej soubor
ld a,27             ;nenalezen - dej do A číslo hlášení
jp nz,MESSAG2      ;"File not found" a skoč

```

Soubor byl na disku nalezen a teď máme několik možností, jak provést jeho načtení. Známe-li bezpečně délku souboru (uložili jsme ji do hlavičky HEAD) a chceme-li soubor také načíst na adresu udanou v naší hlavičce, budeme pokračovat takto:

```

ld de,(FILENM+11)  ;přečti délku
ld ix,(FILENM+13)  ;kam nahrát
call 6574           ;zavolej LOABLK
call 737            ;vypni motory a přestránuj
jp YDONE           ;vrať se s hláškou: operace proběhla OK

HEAD               defm "BfiletoLoad"
                  defw 6912,16384,32768

```

Známe-li pouze název souboru a nevíme ani jeho délku, ani kam ho načíst, využijeme parametrů z nalezené hlavičky:

```

ld ix,(15986)      ;adresa hlavičky do IX
ld e,(ix+13)       ;načti startovní adresu ld d,(ix+14)
push de            ;ulož ji
ld e,(ix+11)       ;načti délku
ld d,(ix+12)
pop ix             ;vyzvedni startovní adresu
call 6574          ;viz. výše...
call 737
jp YDONE

```

Do rutiny samozřejmě můžete vsunout testy, které podle hlavičky (jména, přípony, atributů, délky, startovní adresy, třetího parametru) samy poznají, jestli nalezený soubor doopravdy "patří" k programu - to pro případ, aby nebyl nahrán cizí soubor (většinou to vede k zhroucení).

2.5 SAVE souboru

Vstupní bod SAVE* je na adrese **5889** (popsáno u LOAD*). Přejdeme rovnou k vlastní nahrávací rutině; její volání je stejné jako u rutiny LOAFIL:

```

SAVFIL            push ix
                  ld hl,R1
                  call SETUP
                  call SHADE
                  pop hl
                  call MOVER
                  call SET2

```

Až sem se LOAFIL a SAVFIL v ničem neliší (můžete z této pasáže udělat podprogram). Příznak ZERO je nastaven v případě, že soubor již na disku existuje. Je na Vás, jak se v takovém případě zachováte: jestli soubor rovnou přepíšete nebo se na to zeptáte uživatele, atd. (rutina SHADE nastavuje, že MDOS se již na přepsání souboru ptát nebude).

Pozor! V hlavičce, kterou jste si vytvořili od návěští HEAD, musí být zapsána správná délka souboru, protože MDOS si ji tam odsud bude zjišťovat. Přenesení hlavičky na FILENM a následně do pracovních oblastí MDOSu zajistí podprogramy MOVER a NAMER. Zbytek rutiny SAVFIL už je jednoduchý:

```

ld ix,FILENM      ;IX ukazuje na hlavičku

```

Dále zde musí být naplnění registru HL adresou, odkud se budou data ukládat. Bud Vám stačí údaj v hlavičce (tj. za předpokladu, že je pravdivý) a potom můžete použít

```
ld    l,(ix+13)
ld    h,(ix+14)
```

V jiném případě (potřebujete ukládat data od jiné adresy, než je zapsáno v hlavičce) si musíte naplnit registr HL odpovídající hodnotou sami.

```
call 6650          ;uloč soubor
call 737
jp   YDONE

HEAD defb 3        ;hlavička pro SAVE
defm "filetosave"
defw 6912,16384,32768
```

Aby byl popis uložení souboru kompletní, okomentuji ještě rutinu z MDOSu, která se o to stará:

```
06650
SAVROU    ld    (15988),hl    ;ulož odkud ukládat data
          call 6622,TRAHDR    ;parametry z hlavičky do cyst. Prom. (!)

06656
SAVRUN    push de            ;ulož délku (získána v TRAHDR)
          call 7311,SETACT    ;připrav disk
          call 8491,SEAFIL    ;hledej soubor
          jr   nz,6707,NEWFIL  ;nenalezen (skoč) - vytvoř nový
          call 9739,PROFET    ;čti atributy souboru
          bit  2,a            ;testuj "write protect" atribut
          jr   nz,6677,SAVR01  ;není nastaven - skoč

06672
WPRTER    ld    a,96         ;číslo chyby
          jp   516,ERRR       ;hlaš "File is write protected"

06677
SAVR01    ld    a,(16110)     ;ukládáš snap?
          and  a
          jr   nz,6704,NOCONF  ;ne, skoč dál
          ld    a,(15970)     ;ptát se na přepis? and a
          jr   nz,6704,NOCCINF ;ne, skoč dál
          push hl            ;ulož registry
          push de
          ld    de,943,DOSMES  ;Tabulka hlášení
          ld    a,62+128      ;invertované číslo hlášení
          call 8639,YSRQ      ;Piš "Rewrite..." a čekej na klávesu
          pop  de            ;obnov registry
          pop  hl
          jp   nc,4121,BADFN   ;nestisknuto P (R), hlaš "Bad file name"

06704
NOCONF    call DELENT        ;vymaž starý soubor

06707
NEWFIL    pop  de            ;délka souboru
          ld    hl,(16988)     ;začátek dat
          call 8262,SAVFIL    ;ulož soubor
          jp   6596,LOAFIN     ;konec jako u LOABLK
```

Vhodným voláním této rutiny můžete dosáhnout duplikování souborů (i se stejným jménem) na disk, ignorování atributů i parametrů v hlavičce (na to ovšem pozor).

Podobné rutiny, jaké jsem zde pro čtení a ukládání souborů popsal, používají i programy Heores '92, TOLSTOJ, Prometheus - nejedná se tedy o žádnou žhavou novinku, ale o vyzkoušené postupy, do kterých se mohla chyba dostat pouze při přepisování.

2.6 SAVE, LOAD a spouštění SNAPSHOTů

Přeložíme-li si anglický termín "snapshot" do češtiny, zjistíme, že to není nic víc (a ani nic míň) než "fotografie" nebo "snímek". Tato dvě slova (podle mne) vystihují základní vlastnost snapu - totiž, že přesně zachytil obsah celé paměti RAM (48k) ve chvíli, kdy jste stiskli stejnojmenné tlačítko.

MDOS chápe snapshot jako soubor s příponou "S", začínající na adrese 16256 a mající délku 49280 bajtů - tedy přesně o 128 bajtů víc, než je délka klasické 48k RAM. V oněch 128 bajtech "navíc" (nahrávají se do sítinové RAM) jsou uloženy hodnoty všech registrů, adresa vrcholu zásobníku a stav a mód přerušení.

Nejprve se budeme věnovat ukládání snapshotu. Po stisku tlačítka SNAP (NMI) dojde ke skoku na nulu a přestránkování. V tabulce IORTAB je vyhledána odpovídající adresa - 743, kde se nachází podprogram **SNAPR**.

```

00743
SNAPR      ld      (16382),sp      ;ulož vrchol zásobníku
           ld      sp,16382      ;ukazuj do oblasti, kam budou postupně
           push   af              ;schovány hodnoty všech registrů,
           push   bc
           push   de
           push   hl
           exx                    ;včetně druhé banky
           ex     af,af'
           push   af
           push   bc
           push   de
           push   hl
           push   ix
           push   iy
           ld     bc,(16108)      ;a stavu přerušení
           push   bc
           im     1                ;nastav IM 1
           ld     a,255           ;signál: ukládám snap, nevypisuj chyby
           ld     (16110),a      ;při práci s diskem
           ld     hl,16042
           ld     de,16000
           ld     bc,10
           ldir
           ld     hl,932,SNAPNM  ;přenes jméno snapu
           ld     de,16010      ;do oblasti pro jméno souboru
           ld     bc,11         ;včetně přípony
           ldir
           ld     a,(15969)      ;čti číslo stávajícího snapu,
           inc    a              ;zvětši ho
           ld     (15969),a      ;a ulož pro příště
           dec    a              ;číslo do původního stavu
           ld     b,0
00808

```

```

DECLOP    sub    10          ;děl číslo snapu deseti
          jr     c,815,DECOK ;při zbytku < 10 odskoč
          inc   b           ;inkrementuj výsledek
          jr     808,DECLOP  ;pokračuj v dělení

00815
DECOK     add    a,58        ;ASCII kód číslice je zbytek po dělení
          ld    (16019),a    ;ulož do jména
          ld    a,b         ;výsledek dělení do A
          add   a,48        ;opět vytvoř správnou číslici
          ld    (16018),a    ;a ulož ji do jména
          ei                    ;povol přerušení
          ld    hl,16256    ;začátek snapu
          ld    (15988),hl  ;dej na LOADIX
          ld    de,49280    ;délka snapu
          call  6656,SAVRUN ;ulož soubor (viz. kap. 2.5)
          call  9526,DSKSTP ;zastav motory

```

Tím končí uložení snapshotu na disk. Následuje "rozeběhnutí" snapu, což může být zapotřebí ve třech různých situacích:

- 1) snap byl uložen na disk
- 2) při ukládání došlo k chybě (otevřená dvířka, plná disketa...)
- 3) snap byl nahrán z disku

```

00842
SNPRET   call  9526,DSKSTP ;zastav motory
          di                    ;zakaž přerušení
          ld    sp,16360     ;nastav zásobník
          xor   a            ;povol vypisování chyb při disk. Operacích
          ld    (16110),a
          pop  af            ;čti stav a mód přerušení
          jp   pe,886,SNPRET ;příznak PE = přerušení povoleno, skoč
          ld    i,a         ;do I dej vektor přerušení
          cp   63           ;je to 63?
          jr   z,865,N0IM2  ;ano, odskoč
          im   2            ;nastavení IM 2

00865
N0IM2    pop  iy            ;obnov všechny registry
          pop  ix
          pop  hl
          pop  de
          pop  bc
          pop  af
          ex   af,af'       ;i druhou (resp. první) banku
          exx
          pop  hl
          pop  de
          pop  bc
          pop  af
          ld   sp,(16382)   ;nastav vrchol zásobníku
          jp   5888        ;přestránkuj a rozběhni snap

```

Následující pasáž je úplně stejná, jen s tím rozdílem, že před rozeběhnutím snapu je povoleno přerušení (EI). Přijde mi, že kvůli jedné instrukci popisovat takový kus programu, je jako jít s lopatou na komára -jde to vyřešit daleko elegantněji, zvláště, když je k dispozici vlastní RAM.

```

00886
SNPRT1      ld      i, a
            cp      63
            jr      z, 899, NOIM21
            im      2

00899
NOIM21      pop     iy
            pop     ix
            pop     hl
            pop     de
            pop     bc
            pop     af
            ex     af, af'
            exx
            pop     hl
            pop     de
            pop     bc
            pop     af
            ld     sp, (16382)
            ei
            jp     5888

```

Pokročilejším programátorům určité neušlo, že "snap" není dokonalý (ani nemůže být) a že je možné se proti němu bránit (resp. lze zajistit zhroucení po uložení i po načtení):

- 1) snap neukládá hodnotu registru R (refresh); dokážete-li v programu udržet kontrolu nad jeho obsahem, po uložení (i po nahrání) to rozpoznáte a můžete se podle toho zachovat.
- 2) nastavíte-li: IM 1; EI; LD A,n (pozn.: $n < 63$); LD I,A, dosáhnete toho, že rutina SNPRET, špatně identifikuje přerušení a ke zhroucení dochází v 90% případů, můžete si dokonce vytvořit vlastní přerušení, které začne pracovat až po přepnutí do módu IM 2, které zajistí právě provedení SNAPu.

Popis nahrání snapu do paměti bude jednoduchý a stručný, protože všechny využívané rutiny jsme si již vysvětlili (začátek SNPLOA by se dal napsat trochu lépe: LD IX,16256; LD SP,IX):

```

00916
SNPLOA      ld      sp,16256      ;zásobník ukazuje do DRAM
            ld      ix,16256      ;kam
            ld      de,99280      ;kolik
            call   6574,LOABLK    ;nahraj blok
            jp     SNPRET        ;spust' snap

00932
SNAPNM      defm "SNAPSHOT00S"   ;předloha pro jméno snapu

```

Do rutiny SNPLOA se dostaneme z podprogramu na adrese 6549, TSTSNP (viz. kap. 2.4). Po nahrání snapu do paměti (rutinou LOABLK), dojde k jeho spuštění (vstupní bod SNPRET).

2.7 ERASE, LET ATTR

Vymazání a změna atributů jsou příkazy, jejichž programová struktura je obdobná. Při zadávání jména souboru, se kterým má být akce provedena, můžete použít hvězdičkovou nebo otazníkovou konvenci, což činí tyto operace skupinovými. Hvězdičkovou konvencí můžete dosáhnout např. toho, že se bude pracovat se všemi

soubory se stejnou příponou, se stejným počátečním znakem v názvu, atd. Z toho také vyplývá algoritmus pro vykonání příkazů:

- 1) je provedena analýza jména a přípony a na disku je hledán první vhodný soubor, který splňuje zadané požadavky
- 2) pokud je soubor nenalezen, je vypsáno odpovídající chybové hlášení
- 3) se souborem je provedena jedna z výše uvedených operací (umožňuje-li to nastavení atributů)
- 4) je hledán další vhodný soubor, jehož jméno nebo přípona by splňovaly zadané požadavky
- 5) není-li žádný další soubor nalezen, nevádí to a je vypsáno "0 OK..."
- 6) jinak vše opakuj znovu od bodu 3)

Příkaz LET ATTR začíná na stejné adrese jako LET FN (1776); interpret basicu si je sám rozliší, ošetří syntaxi a pak i vykoná...

Příkaz ERASE leží od adresy 4819.

2.8 LIST*, LLIST*

Příkaz LIST* i LLIST* začínají na adr. 2159, rozdíl je pouze v otevřeném kanálu, na který znaky pro tisk posílají (LIST* na obrazovku (kanál 2) a LLIST* na tiskárnu (kanál 3)).

```

02159
RINFO      rst   #18           ;podej znak
           cp    42           ;musí to být hvězdička,
           jp    nz,1739,SNERR ;jinak synt. chyba - "Nonsens in..."
           rst   #20           ;další znak; neboť příkaz je kompletní,
           call  9183,ENDCH1    ;prověř, jestli nebylo zadáno něco navíc
           ld    a,13          ;odřádkuj
           rst   #10
           xor   a
           ld    de,2917,INFMES ;tabulka hlášení pro LIST*
           call  956,PRTMES     ;tiskni: "MDOS release..."

```

Teď bude zjištěno, které drivy jsou definovány; tj. s kolika umožňuje MDOS pracovat. Vzhledem i k inicializační tabulce na adr. 3832 je jasné, že to jsou pouze A, B - ovšem v datových oblastech je vždy místo i pro C a D.

```

           ld    a,1
           ld    de,2417,INFMES
           call  456,PRTMES     ;tiskni: "Drives defined..."
           ld    a,65          ;první drive je vždy "A"
           ld    ix,15872      ;a jeho datová oblast začíná na 15872

02193
RINF02     push  ix           ;schovej začátek obl. a ASCII kód drivu
           push  af
           ld    a,(ix+2)      ;není-li drive definován, je třetí bajt
           and   a             ;oblasti roven nule,
           jr    z,2213,RINF01 ;a proto skoč dál
           pop   af           ;obnov ASCII kód drivu
           push  af
           rst   #10          ;a vytiskni ho
           ld    a,2
           ld    de,2417,INFMES
           call  956,PRTMES     ;tiskni dvojtečku, čárku a mezeru

02213

```

```

RINF01    pop    af            ;obnov drive a datovou oblast
          pop    ix
          ld     de,12        ;délka datové oblasti je 12 bajtů
          add   ix,de        ;posuň se na další oblast
          inc   a            ;posuň se i na další ASCII kód drive
          cp    69          ;když je to "E", pokračuj dál,
          jr    c,2193,RINF02 ;jinak opakuj smyčku RINF02

```

Následuje zjištění, které z definovaných drivů jsou skutečně připojeny. Smyčka je dosti podobná té minulé.

```

          ld     a, 3
          ld     de,2417,INFMES
          call   956,PRTMES   ;tiskni: "Drives installed..."
          ld     a,65        ;opět první možný drive je "A",
          ld     ix,15872    ;jeho datová oblast je od 15872

02290
RINF09    push   ix            ;obě hodnoty jsou uschovány
          push   af
          bit   0,(ix+0)     ;a je zjištěno, je-li drive připojen (NZ)
          jr    z,2260,RINF03 ;když ne, vezmi další
          pop    af         ;obnov ASCII kód drivu
          push   af
          rst   #10         ;a vytiskni ho
          ld     a,2
          ld     de,2917,INFMES
          call   456,PRTMES  ;následuje tisk ":", "

02260
RINF03    pop    af            ;drive
          pop    ix            ;datová oblast
          ld     de,12        ;posuň se do další datové oblasti
          add   ix,de
          inc   a            ;a zvětši ASCII kód drivu
          cp    69          ;když jsi došel k "E", skonči,
          jr    c,2290,RINF04 ;jinak opakuj smyčku
          ld     a,4
          ld     de,2417,INFMES
          call   456,PRTMES  ;tiskni: "Current device..."
          ld     hl,16092
          call   4749,NAMPRT ;a k tomu jméno aktuálního disku
          ld     a,5
          ld     de,2917,INFMES
          call   956,PRTMES  ;tiskni: "Volumes Available..."
          call   R009,SCNDRV ;projed' připojené mechaniky

```

V této smyčce budou vytisknuta jména všech dostupných disků. Pokud v mechanice disk není nebo jsou otevřená dvířka, nevytiskne se nic.

```

          ld     b,4          ;maximálně 4 přístupné disky
          ld     hl,15920    ;HL ukazuje do dat na jména disků

02303
RINF05    push   bc            ;uschovej hodnoty
          push   hl
          ld     a,(hl)      ;pokud drive není definován

```

```

and a ;skoč na další
jr z,2323,RINF06
ld a,32 ;tiskni 2 mezery (odsazení od kraje)
rst #10
ld a,32
rst #10
pop hl ;obnov HL
push hl
call 9749,NAMPRT ;tiskni jméno diskety
ld a,13
rst #10 ;odřádkuj

02323
RINF06 pop hl ;obnov hodnoty
pop bc
ld a,12 ;do A dej 12
call 4013,ADDHLA ;a přičti ho k HL (další oblast)
djnz 2303,RINF05 ;opakuj 4x výpis jmen
ld a,6
ld de,2417,INFMES
call 956,PRTMES ;tiskni: "Length.."
ld hl,(23627) ;začátek proměnných (VARS)
push hl
ld de,(23635) ;minus začátek programu (PROG)
and a
sbc hl,de ;dává jeho délku
ld c,l , - přesuň ji do 13C
ld b,h
call 4006,BCPRT ;a vytiskni
ld a,7
ld de,2417,INFMES
call 456,PRTMES ;tiskni: "Variables..."
ld hl,(23641) ;začátek editační oblasti (E-LINE)
pop de ;minus začátek proměnných
and a
sbc hl,de ;dává jejich délku
ld c,l , - přesuň ji do HC
ld b,h
call 4006,BCPRT ;a vytiskni
ld a,8
ld de,2417,INFMES
call 456,PRTMES ;tiskni: "Top..."
ld bc,(23730) ;čti hodnotu systémové proměnné RAM-TOP
call 4006,BCPRT ;a tiskni ji
ld a,9
ld de,2417,INFMES
call 456,PRTMES ;tiskni: "Free..."
rst #24 ;volej FREE-MEM (#1F1A) v NROM
defw 7962,FREE-MEM ; - vrací velikost obsazené paměti v BC
ld hl,65535 ;konec paměti (neuvažuje GAMU, ZXS 128)
and a
sbc hl,bc ;minus počet obsazených bajtů
ld c,l ;je zbývající volné místo
ld b,h ;přesuň do HC
call 4006,BCPRT ;a vytiskni

```

```

ld    a,13
rst   #10           ;odřádkuj
ret                   ;vrat se

```

Všimněte si, že texty většiny hlášení v tabulce INFMES začínají řídicími kódy, zpravidla CHR\$(13) - odřákování. Sekvence kódů 8, 8, 32, 13 znamená 2x kurzor vlevo, vytisknout mezeru a přejít na nový řádek, což má za praktický následek smazání poslední (resp. nadbytečné) čárky ve výčtu definovaných nebo připojených drivů.

Všechny texty jsou ukončeny invertovaným znakem (text je uzavřen v apostrofech namísto v normálních uvozovkách):

```

02417
INFMES  defb 128
        defm "MDOS Release: 1.0 (17-May-91)." ;hlášení č. 0
        defm '(c) Didaktik Skalica 1991.'
        defm "Drives defined : "           ;hlášení č. 1
        defb 160
        defb ".","",",",160           ;hlášení č. 2
        defb 8,8,32,13               ;hlášení č. 3
        defm "Drives installed:"
        defb 160
        defb 8,8," ",13               ;hlášení č. 4
        defm "Current Device : "
        defb 160
        defb ".","13",13               ;hlášení č. 5
        defm "Volumes Available:"
        defb 141
        defb 13                       ;hlášení č. 6
        defm "Length of Program : "
        defb 160
        defb 13                       ;hlášení č. 7
        defm "Length of Variables:"
        defb 160
        defb 13,13                    ;hlášení č. 8
        defm "Top of RAM : "
        defb 160
        defb 13                       ;hlášení č. 9
        defm "Free memory:"
        defb 160

```

2.9 Ostatní příkazy MDOSu

Zbývajících příkazů MDOSu se již nebudeme zabývat podrobně a pouze si některé z nich stručně popíšeme.

CAT (adr 4575); CAT* (adr. 4442)

S poznatky uvedenými v první kapitole byste měli být schopni stvořit svůj vlastní katalog disku (alespoň jednoduchý). MDOSovský CAT však není tak úplně jednoduchou operací - pracuje totiž na stejném principu, jako příkazy LET a ERASE: vypsání informací bude provedeno pro všechny soubory, které vyhovují zadanému příkazu.

Kompletní syntaxe příkazu CAT vypadá asi takto: **CAT [-|["jmeno.p"]]**. Parametry v hranatých závorkách můžete vynechat; jméno a přípona mohou obsahovat hvězdičkovou a otazníkovou konvenci.

POKE# (1729)

Ukládá do posledního sektoru DRAM (oblast systémových proměnných MDOSu). Relativní adresa je v rozmezí od 0-512, což odpovídá absolutním adresám 15872-16383. Většina systémových proměnných je pro basic nepoužitelná a má význam pouze při práci ve strojovém kódu.

MOVE (6740)

Kopírování pracuje stejně jako v ostatních DOSech: buď kopírujete na dvou mechanikách nebo na jedné. Ve druhém případě můžete navíc kopírovat na jedné (pak ale musíte změnit jméno kopírovaného souboru) nebo na dvou disketách. Syntaxe vypadá takto: **MOVE "dev1:jmeno.p", "dev2:*"** nebo **MOVE "dev1:jmeno1.p", "dev1:jmeno2.p"**. Příkaz pracuje i s hvězdičkovou a otazníkovou konvencí.

To je vše, co o kopírování řeknu, protože na jeho samotném řešení není nic zajímavého (kromě zákeřné chyby). Nikomu nedoporučuji příkaz MOVE používat, protože je až neskutečně pomalý a práci jen zdržuje. Pro kopírování na jedné mechanice se nejvíce hodí program **SINGLEcopy**, který pojme až 41kB "najednou" (na délce souborů mu nijak nezáleží); pro dvě mechaniky stačí používat Tools 80.

Příkaz MOVE neslouží pouze ke kopírování; můžete jím také určit aktuální disk.

RUN (4409)

Bootovací příkaz RUN hledá na disketě spustitelný soubor s názvem "run" - může to tedy být jak program, tak Snapshot. K hledání dochází pouze v případě, že v paměti není žádný basicový program.

LET FN (1776)

Přejmenování souboru. MDOS neumožní vytvořit na disku dva totožně označené soubory (rozlišuje se nejen podle jména, ale i podle přípony).

READ* (2640), RESTORE* (2635)

příkazy pro čtení a zápis sektoru jsou využívány i rutinami SAVE a LOAD (MERGE). Práci se sektory se budeme podrobně věnovat ve třetí kapitole,

III. Rutiny pro čtení a zápis sektoru a jejich použití

V poslední kapitole se konečně dostáváme k popisu elementárních diskových operací - čtení a zápisu sektoru (a formátování stopy). Tohle všechno dělá jedna a ta samá rutina, která má pro každou z operací jeden vstupní bod.

3.1 LOAD, SAVE sektoru, FORMAT stopy

Rutina je tvořena smyčkou, uprostřed níž je zavolání požadované operace. Do rutiny vstupujete s takto nastavenými registry:

HL=odkud (kam) číst (zapisovat) data B = číslo stopy
 C = číslo sektoru ve stopě
 D = kolik sektorů číst/zapisovat
 E = počet opakování, než dojde k výpisu chybového hlášení
 A = číslo aktuálního disku (zbytečné, ty nejspodnější rutiny se o to starají samy)

V programové nadstavbě se běžně nepracuje s číslem stopy a sektoru v ní, ale s logickými sektory (viz. kap. 1.1), proto je nutno provést přepočítání. Postačí k tomu jednoduchá dělicí rutina (na vstupu je v HL číslo logického sektoru 0 - 1704), na výstupu je v registru B číslo stopy a v registru C číslo sektoru na ní.

```

DIVIDE    ld    a, (16979)    ;čti číslo aktuálního disku
          call  8620         ;vypočti do IX začátek datové oblasti disku
          ld    e, (ix+3)    ;do E dej počet sektorů ve stopě
          ld    d, 0         ;D = 0
          ld    b, -1        ;B = 255
          or    a            ;znič případné CARRY
Dv0       inc    b           ;zvyš číslo stopy
          sbc   hl, de       ;odečti od sektoru počet sektorů na stopě
          jr    nc, DV0      ;zbytek větší než 0 - pokračuj v dělení
          add   hl, de       ;zbytek po dělení zpátky do DE
          ld    c, 1         ;přendej do C
          ret                ;vrat! se

```

Rutina přepočítá podle formátu diskety číslo logického sektoru na číslo fyzické stopy a fyzického sektoru ve stopě. Výsledek je uložen v BC.

Ještě než přejdu k rutině čtení a zápisu sektoru, vrátím se k parametrům v D a v E. Rutina dokáže pracovat nejen s jedním sektorem, ale i s řadou po sobě jdoucích sektorů. To můžete využít třeba při čtení FAT, která obsazuje pět sektorů (č. 1-5). Načíst ji můžete najednou a to takto:

```

. . . .
ld    hl, DIR+512    ;místo v RAM pro boot, FAT, adresář
ld    d, 5           ;čti pět sektorů
ld    e, 1           ;počet opakování při chybě
ld    bc, 1          ;začni na stopě 0, od sektoru 1
call  8866           ;volej načtení sektoru
. . .

```

parametr v registru E udává, kolikrát se má program pokusit opakovat diskovou operaci (zápis, čtení, najetí na stopu, atd.), než ohlásí chybu (CRC error, SEEK error, etc.). MDOS nastavuje E = 1, tj. dva pokusy.

A teď již samotná rutina...

První vstup je pro zápis sektoru:

```

08854
BWRITE    push  hl          ;ulož datový ukazatel
          ld    hl,9150,DWRITE ;adresa rutiny pro zápis sektoru
          jr    8873,BRWR0   ;skoč dopředu

```

Druhý vstup je pro naformátování stopy:

```

08860
BFORMA    push  hl          ;ulož datový ukazatel
          ld    hl,9176,DFORMA ;adr. rutiny pro zformátování stopy
          jr    8873,BRWR0   ;skoč dopředu

```

Třetí vstup je pro čtení sektoru:

```

08866
BREADA    ld    a,(15979)    ;číslo aktuálního disku
08869
BREAD     push  hl          ;ulož datový ukazatel
          ld    hl,9066,DREAD ;adresa rutiny pro čtení sektoru

```

Zbývající část je již společná:

```

08873
BRWR0     ld    (15972),hl    ;ulož adresu rutiny na 15972
          ld    hl,15971     ;nastav se na 15971
          ld    (hl),195     ;a dej tam kód instrukce JP

```

Teď je na adrese 15971 buď JP 9150, JP 9176 nebo JP 9066. Instrukcí CALL 15971 může společná část rutiny vyvolat nastavenou diskovou operaci.

```

          pop  hl          ;obnov v HL datový ukazatel
08882
BRWLO     push  bc          ;ulož číslo stopy a sektoru
          push af          ;ulož číslo disku
          push de          ;ulož počet sektorů a opakování
          push hl          ;ulož datový ukazatel
          call 15971       ;volej čtení/zápis/zformátování
          inc  c           ;je-li C = 0
          dec  c
          jr  z,8935,BREAD1 ;skoč dopředu
          dec  b           ;pokud B <> 1,
          jr  nz,8960,BREAD2 ;tak skoč dopředu
          bit  7,c         ;testuj 7. bit registru C
          jr  z,8915,BREAD3 ;je-li nulový, skoč dál
08900
BREA71    ld    a,54        ;hlášení "Drive is not ready"
08902
BREA72    ld    de,993,DOSMES
08905
          call 8639,SYSRQ   ;tiskni a čekej na stisk klávesy
          jr  c,8929,BREAD4 ;při CARRY skoč dál (stisknuto P, R)
          ld  a,91         ;hlášení "I/O device error"
          jp  516,ERRR     ;skokem do výpisu chyby předčasně skonči

```

```

08915
BREAD3    ld    a,c          ;C dej do A
          and   24          ;zachovej 3. a 4. bit
          jr    nz,8925,BREA31 ;výsledek nenulový, skoč

08920
ERR45     ld    a,59        ;hlášení "Internal error"
          jp    516,ERRR     ;skonči tiskem hlášení

08925
BREA31    ld    a,55        ;hlášení "SEEK error"
          jr    8902,BREA72   ;skoč na možnost "R = Retry"

08929
BREAD4    pop   hl          ;obnov registry, abys mohl
          pop   de          ;provést "Retry"
          pop   af
          pop   bc
          jr    8882,BRWLO    ;zkus to znovu

08935
BREAD1    pop   hl          ;obnov datový ukazatel
          pop   de          ;obnov počet sektorů a opakování
          pop   af          ;obnov číslo disku
          ld    bc,512       ;délka sektoru
          add   hl,bc        ;přičti k HL (HL ukazuje další sektor)

```

(pozn.: přitom by stačilo pouhé INC H, INC H - navíc je to kratší, rychlejší a hezčí)

```

          pop   bc          ;obnov číslo stopy a sektoru
          push  af          ;schovej číslo disku
          inc   c           ;posuň se na další sektor
          ld    a,(ix+3)    ;A = počet sektorů na stopu (formát disku)
                          ;porovnej
          jr    nz,8959,BREA11 ;při nerovnosti skoč dál
          ld    c,0         ;jinak C = 0 (první sektor ve stopě)
          inc   b           ;další stopa

08954
BREA11    pop   af          ;číslo disku
          dec   d           ;sniž počet čtených sektorů
          jp    nz,8882,BRWLO ; D > 0 - opakuj smyčku
          ret              ;vrať se, čtení sektorů skončeno

08960
BREAD2    dec   b           ;testuj B = 2
          jr    nz,8988,BREAD7 ;ne, skoč dál
          ld    a,c          ;čti C do A
          and   %10011101    ;zachovej některé bity
          jr    z,8935,BREAD1 ;výsledek nulový, skoč - operace je OK

08968
BREA81    bit   7,c         ;testuj 7. bit registru C
          jr    nz,8900,BREA71 ;je-li nenulový skoč na "Not ready"
          bit   4,c         ;testuj 4. bit registru C
          jr    z,8980,BREADB ;je-li nulový, skoč dál
          ld    a,56        ;hlášení "Sector not found"
          jr    8902,BREA72   ;skoč na možnost "R = Retry"

08980

```



```

BREAD8      bit    3,c          ;testuj 3. bit registru C
            jr     z,8920,ERR45 ;skoč na "Internal error"
            ld     a,57         ;hlášení "CRC error"
            jr     8902,BREA72  ;skoč na možnost "R = Retry"

```

```

08988
BREAD7      dec    b           ;testuj B = 3
            jr     nz,9002,BREAD9 ;ne, skoč dál
            ld     a,c         ;do A dej C
            and   %11011101    ;zachovej nastavené bity

```

(pozn.: na tomto místě pravděpodobně chybí JR Z,8393, BREADA)

```

            bit    6,c          ;testuj 6.bit registru C
            jr     z,8968,BREA81 ;je-li nulový, skoč zpátky
            ld     a,58         ;hlášení "Disk is write protected"
            jr     8902         ;skoč na možnost "R = Retry"

```

```

09002
BREAD9      bit    0,c          ;testuj 0. bit registru C
            jr     z,9011,BREA91 ;je-li nulový, skoč dopředu
            ld     a,34         ;hlášení "Device unavailable"
            jp     516,ERRR     ;skonči výpisem chyby

```

```

09011
BREA91      bit    1,c          ;testuj 1. bit registru C
            jr     z,8920,ERR45 ;je-li nulový, skoč na "Internall error"

```

```

09015
BADTYP      ld     a,32         ;hlášení "Bad device type"
            jp     516,ERRR     ;skonči výpisem chyby

```

Uvedená rutina ještě stále není tou "nejspodnější" úrovní MDOSu, ale pro programovou obsluhu je to ta pravá, pokud nemáte (zbytečnou) chuť zatěžovat se přímým ovládáním řadiče. Jak je vidět, budete-li si chtít sami ošetřit chybová hlášení typu "R = Retry", nezbyde Vám nic jiného, než celou rutinu opsat z DROM do RAM - potom si v ní sami změníte JP 516 na JP MESSAG2 a pro SYSRQ vytvoříte vlastní podprogram.

Rutiny, které přímo čtou a zapisují sektor se jmenují DREAD a DWRITE. Zavoláte je s podobným nastavením registrů jako jejich nadstavbu:

HL = ukazatel na data (nebo na místo pro data)
 B = stopa
 C = sektor ve stopě
 E = počet opakování, než se bude neúspěšná operace považovat za chybu

Rozborem podprogramů se zabývat nebudeme (zabralo by to pár dalších stránek) a pouze si řekneme, jak probíhá výměna dat s řadičem:

DREAD a DWRITE ukládají do registru IX adresy následujících dvou podprogramků:

```

09706
REANMI      ini          ;pro čtení
            ret

```

```

09709
WRINMI      outi         ;pro zápis
            ret

```

Komunikaci pak zajišťuje nemaskovatelné přerušení NMI:

```
00102
NMI      jp      (ix)      ;skoč do rutiny
```

Při čtení/zápisu sektoru je hardwarově 512x vyvoláno nemaskovatelné přerušení, které skáče v DROM na adresu 102. Zde pak pokračuje skokem na adresu určenou registrem IX (buď REANMI nebo WRINMI). Instrukce INI a OUTI jsou tím nejrychlejším způsobem, jak s řadičem komunikovat (trvají nejmenší počet taktů). (pozn.: Rutina pro formátování stopy je ještě složitější a neřeknu o ní ani Ň.)

Ostatní operace s řadičem se provádějí normálně pomocí instrukcí IN, OUT; kód chyby, k jaké došlo, je uložen MDOSem do registrového páru BC a následně vyhodnocen ve vyšší programové nadstavbě (viz. BREAD, BWRITE). Popis některých podprogramů, řídicích řadič, najdete v příloze C.

3.2 Čtení a zápis bootu, FAT a adresáře

Operace s datovými oblastmi na disku jsem popisoval už v první kapitole. Neuvedl jsem ale to nejdůležitější - a sice, jak si tuto část disku do paměti načíst, eventuelně, jak ji zpět na disk uložit (ten READ* / RESTORE* v basicu se nedá brát vážně...).

Podprogram je podobný jako pro načtení FAT (viz. 3.1), ale složitější. Protože budeme načítat 14 sektorů, a ty už všechny nemohou ležet v jedné stopě, bude zapotřebí nastavit proměnné MDOSu. Normálně to dělá podprogram SETDSK (adr. 7841), který řeší i různé krizové situace (nesmyslný formát, DS disk v SS mechanice, atd.), ale pokud to chcete udělat sami a nebrat v potaz, že by informace na disketě mohly být poškozeny, použijte klidně moji rutinu:

```
READALL  call  SETPRMS      ;"očmuchej" disketu
          ld   hl,BREAD     ;rutina pro čtení sektoru (Vaše nebo MDOS)
RWALL    ld   (RWAL1+1),hl  ;ulož adresu rutiny
          ld   hl,RWTAB     ;tabulka pořadí sektorů
          ld   de,DIR       ;místo v paměti
          ld   b,14        ;délka = 14 sektorů
RWAL0    push bc           ;všechno ulož
          push hl
          push de
          ld   l,(hl)      ;čti číslo logického sektoru do HL
          ld   h,0
          call DIVIDE      ;vypočti stopu a sektor do BC
          pop  hl         ;adresa dat (DIR)
          ld   de,257     ;jeden sektor, jedno opakování při chybě
RWALL    call 0           ;volej rutinu čtení
          ex  de,hl       ;adresa dat do DE
          pop  hl         ;tabulka pořadí sektorů
          inc  hl         ;posuň se na další
          pop  bc         ;počet sektorů
          djnz RWAL0     ;opakuj B krát
          jp   9526       ;ukonči vypnutím motorů
SETPRMS  ld   hl,14336    ;místo v DRAM
          ld   de,257     ;jeden sektor, jedno opakování
          ld   bc,0       ;stopa nula, sektor nula (= boot)
          call BREAD      ;čti sektor
          call 8609       ;vypočti do IX datovou oblast
          ld   a,(19336+177) ;čti údaje o formátu
          ld   hl,(14336+178)
          set  4,(ix+1)   ;nastav oboustranný
          and  16        ;testuj oboustranný
          jr  nz,RA0     ;ano, skoč
```

```

RA0      res    9,(ix+1)      ;nastav jednostranný
         ld     (ix+2),1     ;nastav počet stop a sektorů
         ld     (ix+3),h
         ret                    ;vrať se

RWTAB    defb   0,1,2,3,9,5  ;boot a FAT v normálním pořadí
         defb   6,8,10,12    ;adresář je uložen "ob jeden"
         defb   7,9,11,13

```

RWALL načítá do paměti prvních čtrnáct sektorů a bere při tom ohled na uložení adresáře. Stejný podprogram použijeme i pro uložení oněch 14 sektorů, jenom změníme jeho volání:

```

WRITALL      call    SETPRMS ;nastav parametry
         ld     hl,BWRITE    ;rutina zápisu sektoru
         jr     RWALL       ;skoč do společné části

```

READALL a WRITALL budou pracovat jak s rutinami MDOSu (s rutinami pro čtení/zápis sektoru - pozn.), tak s Vašimi vlastními (resp. opsanými z DROM).

3.3 Vlastní rutina pro LOAD souboru

Využijeme teď spousty věcí, které jsme si až doposud řekli, a stvoříme vlastní rutinu pro načtení souboru do paměti. Nebudeme uvažovat žádné "mezí situace" - např. že se soubor do paměti nevejde, atd. -jde mi pouze o princip a takové stavy si můžete ošetřit později sami.

```

LDFILE    ld     hl,BREAD    ;rutina na čtení sektoru
         ld     (LDSVR+1),hl ;ulož do společné části pro LORD/SAVE
         call  FINDFL      ;hledej soubor (viz. 1.9)
         ld     a,27       ;hlášení "File not found"
         jp     c,MESSAG2  ;skoč při chybě
         push  hl          ;hlavička souboru do IX
         pop   ix
         ld     l,(ix+17)   ;první sektor
         ld     h,(ix+18)
LDF1      push  hl          ;ulož
         call  FDBLOCK     ;vzvedni číslo následujícího sektoru
         ld     hl,3072    ;prázdný soubor?
         sbc   hl,de
         jr     z,LDF3     ;ano, skoč dopředu
         ld     a,d        ;poslední sektor?
         cp   19
         jr     nc,LDF2    ;ano, skoč dopředu
         pop   hl          ;číslo sektoru
         call  LDSV        ;nahraj sektor
         call  FDBLOCK     ;najdi číslo dalšího (nebyl poslední)
         ex   de,hl        ;dej ho do HL
         jr   LDF1        ;opakuj

LDF2      and   1          ;zjistí, kolik bajtů z posledního sektoru
         ld   d,a          ;do souboru skutečně patří
         pop  hl           ;číslo sektoru
         push de           ;ulož délku
         call DIVIDE       ;přepočti číslo sektoru na stopy a sektory

```

```

        ld    hl,14898      ;místo v DRAM
        push hl             ;ulož
        ld    de,257       ;jeden sektor, jedno opakování
        call BREAD         ;čti sektor
        pop  hl            :odkud
        pop  bc            ;kolik
        ld    de,(LDADR+1) ;kam
        ldir                ;přenes
        defb 62            ;ignoruj příští instrukci
LDF3   pop  hl            ;čistí zásobník
        jp   9526         ;vypnutím motorů konči

LDSV   push hl           ;ulož číslo logického sektoru
        call DIVIDE       ;vypočti stopy a sektory
LDADR  ld    hl,0         ;kam/odkud nahrávat
        ld    de,257       ;jeden sektor, jedno opakování
LDSVR  call  0            ;volej LOAD/SAVE sektoru
        ld    hl,(LDADR+1) ;posuň adresu pro nahrávání
        inc  h             ;o jeden sektor dál (512 bajtů)
        inc  h
        ld    (LDADR+1),hl ;a ulož ji
        pop  hl            ;čti číslo sektoru
        ret                ;vrať se

```

Volání rutiny bude vypadat takto (program samozřejmě běží ve stránce):

```

        call READALL      ;boot, FAT, adresář
        ld    ix,HEAD     ;ukazuj na jméno souboru
        ld    hl,ADR      ;na tuto adresu nahrávej
        ld    (LDADR+1),hl
        call LDFILE      ;nahraj soubor

HEAD   defm "PTELEFONY"
        defb 0,0

```

Program na uložení souboru (SVFILE) je o trochu jednodušší, zkuste ho napsat sami. Pro zápis sektoru využijte LDSV (nastavte na LDSVR CALL BWRITE) a nezapomeňte u posledního sektoru vytvořit koncovou značku!

Příloha A: Popis DRAM a vybraných syst. prom. MDOSu

DRAM začíná na adrese 14336 a její velikost je 2048 bajtů, čili 4 sektory po 512 bajtech. První tři sektory jsou využívány pro načtení bootu, FATu, adresáře, "zbytku" souboru z posledního sektoru při nahrávání. Ve čtvrtém sektoru jsou umístěny systémové proměnné MDOSu (viz. dále). Posledních 128 bajtů sektoru používá MDOS pro ukládání hodnot registrů při snapu; místo mezi je v tomto případě využíváno jako zásobník.

Při použití vlastních rutin pro základní diskové operace, smíte první tři sektory DRAM použít k čemu uznáte za vhodné (Váš program tam klidně může ležet - pokud se vejde) - nelze ale zavolat žádnou "vyšší" diskovou operaci MDOSu (tj. všechno, co sahá na disk, krom čtení, zápisu a formátování sektoru).

| adresa | název | velikost | význam |
|--------|--------|----------|--|
| 14336 | DIRBUF | 512 | Sektor pro práci s adresářem (při LOAD, SAVE...) |
| 14848 | AUXBUF | 512 | Místo pro nahrání posledního sektoru souboru (odsud je přenesen jen potřebný počet bajtů) |
| 15360 | FATBUF | 512 | Sektor pro práci s FATkou. |
| 15672 | DRZONE | 4*12 | Obsahuje údaje o připojených mechanikách a o jejich parametrech (blíže viz. kapitola 2.2). |
| 15920 | VNZONE | 4*12 | Místo pro názvy disket v mechanikách (10 bajtů) + pro náhodný identifikační dvojbajt (2 bajty). |
| 15558 | DEBUG | 1 | Bajt, určující, mají-li být při práci s MDOSem vypisovány ladící tisky (několik čísel vytisknutých vždy před provedením diskové operace). |
| 15969 | SNPCNT | 1 | počítadlo SNAPSHOTŮ. |
| 15970 | CONFRM | 1 | Nenulová hodnota znamená, že při ukládání souboru se MDOS nebude ptát na přepsání starého (existuje-li). |
| 15971 | MODJP | 1 | Sem bývá uložena instrukce JP (kód 195) při volání čtení, zápisu, zformátování sektoru. |
| 15972 | MODJPA | 2 | Adresa, následující za instrukcí skoku. |
| 15974 | DESTOR | 2 | V případě potřeby sem bývá ukládán obsah reg. páru DE. |
| 15979 | ACTDR | 1 | číslo aktuálního drivu (A = 0, B = 1). |
| 15980 | FATACT | 1 | Nulová hodnota znamená, že sektor FAT, který je načten v paměti, není třeba zapisovat na disk (aktualizovat). |
| 15951 | FATEL | 1 | Číslo posledně čteného sektoru FAT (255 = žádný). |
| 15982 | FATDR | 1 | číslo drivu, ze kterého (na který) jsou čteny (zapisovány) sektory FAT. |
| 15983 | DIRTS | 2 | Stopa a sektor ve stopě, udávající na disku polohu sektoru adresáře, se kterým se právě pracuje. |
| 15985 | DIRDR | 1 | Číslo drivu, ze kterého (na který) jsou čteny (zapisovány) sektory adresáře. |
| 15986 | ENTADR | 2 | Adresa nalezené hlavičky souboru v DIRBUF. |
| 15988 | LOADIX | 2 | Místo pro uschování začátku bloku dat při čtení/ukládání souboru z/na disk(u). |
| 15990 | LOALEN | 2 | Místo pro uschování délky čteného/ukládaného bloku dat. |
| 15992 | VALX | 2 | Pomocná systémová proměnná pro uschování startovní adresy souboru (té, která je uvedena v hlavičce souboru, což se nemusí shodovat se skutečnou adresou v paměti, od které se soubor |

ukládá).

15999 VALY 2 Pomocná systémová proměnná pro uschování délky Basicu (opět té z hlavičky) při ukládání programu.

...

16000 DNZONE 10 Jméno disku, se kterým se pracuje.

16010 FNZONE 10 Jméno souboru, se kterým se pracuje.

16020 FILTYP 1 Přípona tohoto souboru.

16021 DNZON2 10 Jméno druhého disku, se kterým se pracuje (např. při kopírování).

16031 FNZON2 10 Jméno druhého souboru, se kterým se pracuje.

16041 FILTY2 1 Přípona tohoto souboru.

...

16084 LDBUF 6 Místo pro výpočet kapacity disku z počtu stop a sektorů.

16090 LDNUM 8 Místo pro ASCII kódové vyjádření 24-bitového čísla.

16098 INTCNT 1 Počítadlo průchodů přerušení IM 1 při nastaveném EI a stínové ROM.

16099 TERADR 2 Nová adresa návratu z přerušení.

16101 HERRSP 2 Místo pro uschování SP.

16103 DOSIX 2 Místo pro uschování IX při čtení/zápisu.

16105 SELSTA 1 Posledně posílaný příkaz řadiči.

...

16108 IREG 2 Místo pro uschování registru I a stav přerušení.

16110 SNAPO 1 Je-li zde nenulová hodnota, znamená to, že se provádí SNAPSHOT (ukládání) a na místo výpisu chybových hlášení se skáče na konec ukládání snapu (842, SNPRET).

16111 SYSMRK 9 Osm bajtů pro kontrolní tabulku (vytvořena při inicializaci a testována při každém přestránkování) a jeden pro uložení příčiny přestránkování (adr. 16119)

Příloha B: Chyby v MDOSu

MDOS, stejně jako každý jiný program, obsahuje chyby, kterých se dopustil jeho autor. Tuto přílohu jsem nenapsal proto, abych vyžíval v popisu toho, jaký byl autor břídil - podle mne je (až na výjimky) MDOS celkem kvalitní operační systém a jeho autor v žádném případě břídil není. Zdá se, že na výsledné podobě MDOSu se nejvíce podepsala pravděpodobně co nejrychlejší adaptace rutin ze SINDOSu, a to je škoda.

- Za největší chybu považuji zablokování možnosti připojit si až tři nebo čtyři mechaniky. Použitý řadič dokáže obsloužit až čtyři disketové jednotky - softwarové zablokování disků C a D (především C, protože pro něj je na portu 137 místo) vidím jako docela dost demotivující nápad (kdyby se ale MDOS opravila vypálil znovu do EPROMky... hm...).

- Některé mazací LDIRy mažou o bajt víc; jediné štěstí, že to dělají ve chvílích, kdy to náhodou nevadí.

- Nejdrastičtější chyba, na kterou jsem zatím narazil a před kterou Vás musím varovat, je na adrese 6827, v podprogramu pro kopírování souborů. Nepřišel jsem na to, co zde mělo být původně (asi nic), ale určité ne tohle:

```
06827      1d      (49152),bc
```

Můžeme mluvit o štěstí, že tento zápis nebyl v programu proveden o kousek dál, nejlépe ve chvíli, když už je v paměti načten soubor ke zkopírování - to by pak všechny soubory delší než cca 25kB byly poškozeny (brrr!).

Co přesně chyba způsobuje? Tím, že vždy zničí obsah adresy 49152, může se při kopírování stát navíc toto:

- 1) RAMTOP je vysoko nad 49152 a program v Basicu i jeho proměnné končí pod 49152 nebo RAMTOP je pod 49152 a nad ním nic není - nestane se nic (světlá výjimka).
- 2) Program v Basicu nebo jeho proměnné leží přes adresu 49152 - dojde k narušení (programu nebo dat), což může v nejhorším případě vést až ke zhroucení.
- 3) RAMTOP je kousek nad 49152 - program zničí zásobník Basicu; opět může vést až ke zhroucení.
- 4) RAMTOP je pod 49152 a nad ním je program ve strojovém kódu - bude poškozen (co se asi tak stane...).

- Inicializace interface v disketové jednotce je provedena špatně (resp. špatně je zjišťování jiného připojeného obvodu 8255).

- MDOS špatně pracuje s některými formáty. Zkuste na D40 v Basicu naformátovat disketu na 20 stop a 9 sektorů, něco na ni uložit a zase něco přečíst - všechno je OK. Potom vyresetujte počítač a zkuste opět z diskety něco přečíst - a - ne vždy to jde.

- Žádná další fatální chyba už mě nenapadá (tím netvrdím, že v MDOSu už žádná není), takže přílohu uzavřu a přejdeme k té nejposlednější části této příručky.

Příloha C: Seznam některých rutin MDOSu a jejich volání

Protože při komentování MDOSu jsem se odkazoval na spoustu nepopsaných rutin, uvádím zde alespoň stručný výčet a popis některých z nich (řazeno abecedně):

- **ADDHLA 4013**
vstup : číslo v HL (0-65535), číslo v A (0-255)
výstup : přičte A k HL (HL=HL+A)
- **BCPRT 4006**
vstup : číslo v registru BC (0-65535)
výstup : vytiskne číslo (používá rutiny kalkulátoru z NROM!)
- **BREADA 8866**
vstup : viz. kapitola 3.1
výstup : přečte sektor (řadu za sebou jdoucích sektorů) z disku
- **BWRITE 8854**
vstup : viz. kapitola 3.1
výstup : zapíše sektor (řadu za sebou jdoucích sektorů) na disk
- **DELAY 9671**
vstup : -
výstup : čeká 163 taktů
- **DSELECT 9547**
vstup : A = číslo drivu
výstup : zapne drive a nastaví stopu podle IX+4
- **DSKEX 9504**
vstup : IX na 16103 (DOSIX), C = číslo chyby
výstup : na (IX+4) uloží stopu a smaže 0. bit na (IX+O)
- **DSKSTP 9526**
vstup : -
výstup : zastaví motory, zhasne diody
- **DRVCMP 8620**
vstup : A = číslo drivu
výstup : IX = začátek datové oblasti drivu (IX = 15872 + A*12), ZERO = mechanika nepřipojena
- **DRWSU 9396**
vstup : B = stopa, C = sektor
výstup : najede na stopu a sektor, uloží IX na 16103 (DOSIX) CARRY - operace OK (B=1), NC - chyba (BC = číslo chyby)
- **FINTYP 6608**
vstup : IX ukazuje na první bajt "páskové" hlavičky (tj. typ souboru 0-3)
výstup : uloží na 16020 odpovídající příponu souboru (0=P, 3=B, atd.)
- **HOME 9035**
vstup : -
výstup : pošle hlavu "domů" (na začátek disku)
- **PRTMES 456**
vstup : DE = adresa tabulky hlášení (začínající invertovaným znakem), A = číslo hlášení v tabulce (číslováno od 1)

výstup : vytiskne hlášení

■ **SEAFIL 8491**

vstup : jméno a přípona souboru na 16010

výstup : Z -soubor nenalezen, NZ -soubor nalezen, HL - adresa začátku hlavičky (ukazuje do DIRBUF)

■ **SELSEL 9660**

vstup : A = příkaz pro řadič

výstup : pošle A na port 137 a zapíše ho na adresu 16105

■ **SETIX 8609**

vstup : číslo aktuálního drivu na 15979

výstup : nastaví IX jako DRVCMP, ale nenastavuje příznak

■ **SETACT 7311**

vstup : -

výstup : zapne aktuální mechaniku, přečte boot, nastaví systémové proměnné; pokud disk není MDOSový hlásí "X Bad device type"

■ **SYSRQ 8639**

vstup : DE = tabulka hlášení (zpravidla DOSMES), A = číslo hlášení 7, bit registru A: I = Retry, 0 = Proceed

výstup : vypíše do editační zóny hlášení a přidá k němu dotaz Retry nebo Proceed

OBSAH

| | |
|--|-----------|
| I. Organizace dat na disketě, rutiny pro obsluhu FAT a adresáře | 3 |
| 1.1 Formát | 3 |
| 1.2 BOOT | 3 |
| 1.3 FAT | 4 |
| 1.4 Adresář | 8 |
| 1.5 Závěrem... | 12 |
| II. Popis některých rutin MDOSu a jejich volání | 14 |
| 2.1 Komunikace MDOSové ROM a normální ROM | 14 |
| 2.2 Reset MDOSu a inicializace připojených mechanik | 17 |
| Mezikapitola o ošetření chybových hlášení | 20 |
| 2.3 Formátování | 23 |
| 2.4 LOAD souboru | 25 |
| 2.5 SAVE souboru | 28 |
| 2.6 SAVE, LOAD a spouštění SNAPSHOTŮ | 30 |
| 2.7 ERASE, LET FN, LET ATTR | 32 |
| 2.8 LIST*, LLIST* | 33 |
| 2.9 Ostatní příkazy MDOSu | 36 |
| III. Rutiny pro čtení a zápis sektoru a jejich použití | 38 |
| 3.1 LOAD, SAVE sektoru, FORMAT stopy | 38 |
| 3.2 Čtení a zápis bootu, FAT a adresáře | 42 |
| 3.3 Vlastní rutina pro LOAD souboru | 43 |
| IV. Přílohy | 45 |
| Příloha A: Popis DRAM a vybraných syst. prom. MDOSu | 45 |
| Příloha B: Chyby v MDOSu | 47 |
| Příloha C: Seznam některých rutin MDOSu a jejich volání | 48 |