Floppy kurz

maximální využití disketové jednotky VC 1541

Floppy kurz

Maximální využití disketové jednotky VC 1541

Dříve než zahájíme náš kurs, najděte ve svém archivu nebo nahrajte od kolegy program z rodu diskmonitorů. Typickými představiteli jsou programy DISK DEMON, DISK DOCTOR, FAST HACK' EM, SUPER KIT nebo EDDI. V dalším textu se budeme odvolávat na program EDDI.

A nyní se již podívejme na disketu. Následující popis se bude vztahovat na již zformátovanou disketu.

Disketa je rozdělena na 35 koncentrických stop (angl. TRACKS). Každá stopa je dále rozdělena na určitý počet sektorů, jejichž počet klesá směrem ke středu. Přesné počty jsou následující:

stopa	1 - 17	21 sektor
	18 - 24	19 sektorů
	25 - 30	18 sektorů
	31 - 35	17 sektorů

Stopy jsou číslovány od vnější č. 1 k vnitřní č. 35. Sektory na stopě jsou číslovány ve směru proti hodinovým ručičkám. Každý blok obsahuje 1 BLOCK, tj. 256 bytů informací. Zadáním příslušného údaje stopa-sektor je možné vyvolat kterýkoliv z 683 bloků diskety. Uživateli však slouží k volnému použití pouze 664 bloků, protože celá stopa 18 je využívána operačním systémem disku.

Pro následující pokusy bude užitečné si naformátovat a vyhradit zvláštní disketu. Podívejme se nejdříve na Direktory (adresář). Po povelech

```
LOAD"$",8: LIST
```

bude na prvním řádku číslo disku (Drive Number: zde vždy 0) a jméno diskety s identifikačním číslem (ID) a formátovací značkou (viz dále). Pokud na disketě není zapsán nějaký soubor, má být na druhém řádku napsáno 664 BLOCKS FREE.

Protože tyto (a další) informace se nacházejí na již zmíněné stopě 18, podívejme se na ni pomocí EDDI poněkud podrobněji. Natáhněte editor do počítače, vložte disketu a nastartujte editor pomocí RUN. Klávesou F3 zadejte povel čtení bloku (BLOCK LESEN). Pak zadejte číslo stopy a sektoru (oddělené čárkou). V

našem případě tedy "18,0". Po stisknutí RETURN se zobrazí první stránka bloku (řádek s prvními 16ti byty). Je nutné zdůraznit, že počítání bloků a bytů začíná zásadně od nuly. V našem případě (stopa 18, sektor 0) je zobrazený tzv. BAM (Block Availability Map), což česky znamená "Mapa volných bloků". Tato mapa udává, které bloky na disketě jsou ještě volné a které jsou již obsazené. Dále obsahuje jméno diskety, ID, formátovací značku a začátek direktory (adresáře).

První 2 byty (0, 1) tohoto bloku obsahují stopu a sektor prvního bloku direktory (normálně vždy "18, 1"). Přesný význam jednotlivých bytů:

byte	význam
000	stopa prvního bloku direktory, zde 18 ($12)
001	sektor prvního bloku direktory, zde 1 ($01)
002	typ formátu, zde 65 ($41), což v ASCII odpovídá "A"
003	obsahuje 0 pro jednostranný disk (1 pro dvoustranný)
004	počet volných bloků (sektorů) pro stopu č. 1
005	bitový vzorek obsazení sektorů 0-7 stopy č. 1 (1= sektor volný, 0= sektor obsazen)
006	bitový vzorek obsazení sektorů 8-16 stopy č. 1
007	bitový vzorek obsazení sektorů 17-21 stopy č. 1
008 - 011	totéž jako byty 004 až 007, ale pro stopu č. 2
140 - 143	totéž jako byty 004 až 007, ale pro stopu č. 35
144 - 161	jméno diskety doplněno 160 ($A0) = (SPC v REV)
162 - 163	ID diskety v ASCII kódu (zkoumáno při výměně diskety)
164	160 ($A0) - SPC v REV
165 - 166	verze DOS a typ formátu, zde#$32 a#$41 (v ASCII = "2A")
167 - 170	160 ($A0)
171 - 179	$00 pro mód 1541, $A0 pro mód 1570/71
180 - 220	$00 nepoužitý rozsah

221 - 225 pro 1541 a 1570 nepoužito

Poznámka: DOS je zkratka z anglického výrazu Disk Operating System (diskový operační systém). DOS sídlí přímo v diskové jednotce a ne v počítači. Počítač tak má zjednodušenou práci a COMMODORE se tím liší od řešení většiny jiných výrobců. Na druhé straně je to důvodem pro vyšší cenu a nekompatibilitu COMMODORE diskových jednotek.

Commódore vyrábí různé disky (např. 1441, 4040, 8050). Skoro vždy se odlišují počtem stop a sektorů, a proto i označením formátu v bytu 2 (pro 1541 to je "A"). Nesouhlasí-li přečtené označení s předpokládaným (pro náš "A"), vypíše se chybové hlášení. Vyjímkou z tohoto pravidla je čtecí kompatibilita, což znamená, že cizí disketu lze číst (např. disketa ze 4040 na 1541), ale v žádném případě nelze při zjištění rozdílu zapisovat.

Pro zkoumání následující stránky (řádek s dalšími 16ti byty) použijeme F1 (listování dopředu). Listovat zpět lze pomocí F2. Takto si lze postupně prohlédnout celý blok přečtený z diskety.

Podle ID (byty 162 a 163) rozeznává disk, jestli byla disketa vyměněna. Proto by každá disketa měla mít svoje vlastní ID číslo.

Jak bylo uvedeno, je normální hodnota bytu 0 a 1 (ukazatel na první blok direktory - normálně hned za BAM) 18 a 1. Proto pomocí F3 a vložením 18,1 přečteme první blok direktory. Význam jednotlivých bytů je následující:

byte	význam
000 - 001	stopa a sektor následujícího direktory-bloku
002 - 031	popis (ENTRY) souboru č. 1
032 - 033	nepoužito
034 - 063	popis (ENTRY) souboru č. 2
064 - 065	nepoužito
066 - 225	popisy (ENTRY) souborů č. 3 - 7, popř. nepoužito
226 - 255	popis (ENTRY) souboru č. 8

Každý direktory-blok tedy obsahuje vektor (ukazatel) na další direktory-blok (byty 0 a 1). Je-li číslo následující stopy (byte 0) roven 0, pak se jedná o poslední použitý direktory-blok. V následujícím bytu je počet zde užitých bytů (v našem případě 255). V bloku je dále 8 skupin popisu (ENTRY) jednotlivých souborů, oddělených dvěma nepoužitými. Popisy sestávají celkem ze 30 bytů, jejichž význam je následující:

byte	význam
000	typ souboru (FILE) - viz dále
001 - 002	stopa a sektor prvního datového bloku
003 - 018	jméno souboru, doplněné $A0
019 - 020	REL: stopa a sektor prvního vedlejšího bloku (SIDE-SEKTOR)
021	REL: délka datové věty
022 - 025	nepoužito
026 - 027	stopa a sektor náhradního souboru při přepisu s náhradou (jen jako mezipaměť)
028 - 029	počet bloků v souboru (spodní byte, horní byte)

Byte 0 určuje typ souboru. Význam jeho bitů a jejich kombinací je tento:

bit	bitový obsah (v závorce) a význam				
0	(0)	(1)	(0)	(1)	(0)
1	(0) = DEL	(0) = SEQ	(1) = PRG	(1) = USR	(0) = REL
2	(0)	(0)	(0)	(0)	(1)
3 - 5	nepoužito				
6	(0) = normál (1) = FILE nelze smazat přes SCRATCH				
7	(0) = FILE neuzavřen (1) = FILE uzavřen				

DEL (z angl. "Delete" neboli zrušené (smazané) soubory

Tyto soubory nejsou při výpisu direktory (adresáře) normálně zobrazovány. Příslušný byte pro TYP FILE je tedy = 0. Přepíše-li se tento byte na 120 ($80), což je nastavení bitu 7 do 1, bude se i tento typ souboru zobrazovat.

SEQ (Sekvenční, datové) soubory

Slouží k uložení dat na disketu (nezaměňovat s uložením programu). Výstavba tohoto typu souboru je relativně jednoduchá. Byty 0 a 1 datového bloku ukazují na následující blok souboru a zabezpečují tak tzv. řetězení, jehož délka je omezena pouze počtem volných bloků. Ukončení souboru (řetězce) se provádí vložením 0 do čísla stopy (byte 0). Byte 1 pak neobsahuje sektor následujícího bloku, ale počet bytů použitých v tomto posledním bloku. Zbývajících 154 bytů datových bloků (byty 2 až 255) obsahují vlastní data, přičemž jednotlivé rekordy (záznamy) jsou odděleny pomocí znaku CR (Carridge Return).

USR (User, uživatelské) soubory

Svojí výstavbou odpovídají souborům SEQ, mají však v DOS ještě jednu funkci, která bude vysvětlena později.

PRG (Programové) soubory

Představují nejčastější typ souboru. Slouží k ukládání programu na disketu a mají stejnou výstavbu jako SEQ soubory. Jediný rozdíl je v bytech 2 a 3 prvního bloku, které obsahují startovací adresu programu v počítači. Je-li tato adresa rovna adrese začátku BASICU (u počítače C-64 to je 2049, neboli $0801, pak je možné zavádět (LOAD) programy povelem

```
LOAD "jméno",8
```

Tento povel ignoruje obsah těchto bytů a zavádí program vždy od začátku BASICU. Tomuto zavádění se také říká "relativní". Má-li být program natažen do pozice v RAM (udané těmito byty), odkud byl na disketu napsán (např. různé zaváděče ve strojovém jazyce), je nutné příslušné byty využít s pomocí povelu LOAD "jméno",8,1.

REL (Relativní) soubory

Bývají také označovány jako soubory s náhodným (volitelným, přímým atd.) přístupem. Tento typ souboru je svou stavbou nejsložitější. K objasnění nejprve poznámka k SEQ souborům. SEQ soubory jsou sestaveny ze souvislého řetězce dat a jsou výhodné jen pro určitý typ úloh. Máme-li však dlouhý seznam adres a chceme najít jednu určitou adresu, je u SEQ souboru nutné postupně prohledávat celý seznam. U relativních souborů se prohledávání provádí jinak a rychleji. Mějme určitý počet datových záznamů (např. 100), přičemž všechny záznamy musí mít stejnou délku (do 254 znaků, což je délka datového bloku). DOS nyní vytváří při práci s REL souborem tzv. vedlejší bloky (SIDE-SEKTOR). Těchto bloků může být až 6 a obsahují (v jednom SIDE bloku) ukazatele (vektory) na celkem až 120 datových bloků. Datové i SIDE bloky jsou i u těchto souborů řetězeny již známým způsobem s využitím bytu 0 a 1 pro stopu a sektor dalšího bloku. V bytech 2 až 256 datových bloků jsou podobně jako u předchozích typů souborů uložena data. Prázdné záznamy začínají bytem $FF a pokračují byty $00. Také částečně naplněné záznamy se doplňují byty $00. Výstavba vedlejších bloků je následující:

byte	význam
000 - 001	stopa a sektor následujícího vedlejšího bloku
002	číslo vedlejšího bloku (v rozsahu 0 až 5)
003	délka použitého záznamu
004 - 005	stopa a sektor prvního vedlejšího bloku (0)

006 - 007 stopa a sektor druhého vedlejšího bloku (1)
008 - 009 stopa a sektor třetího vedlejšího bloku (2)
010 - 011 stopa a sektor čtvrtého vedlejšího bloku (3)
012 - 013 stopa a sektor pátého vedlejšího bloku (4)
014 - 015 stopa a sektor šestého vedlejšího bloku (5)
016 - 255 ukazatele stopy a sektoru na 120 datových bloků

K lepšímu porozumění jeden příklad. Máme relativní soubor s 250-ti datovými záznamy, každý záznam má 127 znaků. Soubor tedy obsahuje 125 datových bloků (250/2) a 2 vedlejší bloky (v prvním ukazatele na 120 datových bloků a ve druhém na zbývajících 5). Nyní např. požadujeme vyvolat záznam č. 248. Jeden záznam obsahuje 127 bytů dat a jsou proto vždy dva záznamy v jednom datovém bloku. DOS vypočte číslo příslušného datového bloku ze vztahu:

(248-1/2) = 123,5 (mínus jedna proto, že bloky jsou číslovány od 0). Protože v prvním vedlejším bloku jsou ukazatele na bloky 1 až 120, bude ukazatel na příslušný záznam ve druhém vedlejším bloku. Tento bude s pomocí údajů v prvním vedlejším bloku přečten (byty 0 a 1, nebo 6 a 7) a v něm je v bytech 22 a 23 nalezen vektor na datový blok č. 3. Pozice prvního bytu příslušného záznamu se vypočte z necelé části shora uvedeného dělení: 0,5 x 254 = 127

Začátek našeho je potom na bytu 127 + 2 = 129.

Návod k obsluze programu EDDI (Disk-Monitor/Editor)

EDDI umí bloky nejen číst a zobrazit, ale umí také měnit jejich obsah a umí je zpětně zapisovat. K tomu je nejprve třeba (již uvedeným způsobem) natáhnout požadovaný blok a nalézt v něm příslušnou stránku. Editovací mód se pak odstartuje pomocí F5. Nyní můžeme byty přepisovat vložením nové decimální hodnoty a stisknutím RETURN. Z editovacího módu se dá vystoupit do listování stránek pomocí samotného RETURN nebo pomocí (šipka nahoru)

RETURN se vrátíme do povelového módu. Zpětný zápis bloku se provádí pomocí F4, následované číslem stopy a sektorem.

POZOR: Při výměně diskety je nutné stisknout F6 a po vložení nové diskety stisknout libovolnou klávesu. Jinak disk reaguje chybovým hlášením (obvykle se znakem @ "zavináč").

Náměty na pokusy:

a) zkuste změnit na zkušební (!) disketě značku formátu (stopa 18, sektor 0, byte 2) ze 65 na 66 a uložte blok zpět na disketu. Pak zkuste na disketu uložit nějaký krátký program.
b) změňte byte direktory, udávající typ souboru, a soubor pak zkuste natáhnout.

Experimentujte v klidu a zkoušejte i další možnosti. To vám usnadní pochopení dalších částí kurzu.

DISK kontra DATASSETTE

Jistě každý, kdo chtěl rychlejší periferii než je datassette, zjistil, že cena 1541/70/71 je větší (nebo srovnatelná) s cenou samotného počítače. Commódore disky jsou však "inteligentní". Mají vlastní operační systém nazývaný DOS (Disk Operating System) a vlastní mikroprocesor. Pracují téměř nezávisle na počítači a jeho paměti. Kromě přímé výměny dat a povelů nevyžadují čas počítače. Jako příklad lze uvést povel "N" (formátování). Během formátování je počítač téměř plně k dispozici a formátování probíhá samo v disku. Počítač již hlásí READY a 1541/70/71 přitom ještě pracuje.

Povely pro 1541

Nyní se budeme věnovat povelům pro přímý přístup a povelům pro ukládání dat.

Pro uklidnění: nemusíte se obávat poškození 1541/70/71 přímým vstupem do DOS, pokud se něco stane, stačí disk vypnout a znovu zapnout a disk je opět v normálním stavu.

V předchozím jste si zkusili změnit na disketě formátovací značku. Jistě jste si všimli, že pak již není možné na disketu nic napsat. Pomocí tohoto triku, jenž má stejný (téměř) důsledek jako přelepení výřezu umožňujícího zápis, si můžete disketu jednoduše pojistit proti neúmyslnému smazání.

Disk zná celou řadu dalších povelů jako sestavení vlastního formátu, ochrana diskety proti čtení, módifikace natahovacích a ukládacích rutin atd. Je ovšem nutné se tyto povely učit krok po kroku a je nutné ovládat i strojové programování C-64. Vyplatí se vedle BASICU zvládnout i programování v assembleru. K tomu existuje vhodná literatura. Nejdříve se ale zaměříme na BASIC a základní diskové povely.

Jak již bylo řečeno, je 1541/70/71 plnohodnotný počítač, který obsahuje mikroprocesor, RAM a operační systém (DOS), uložený v ROM paměti. Nejen na disketě, ale i v RAM 1541/70/71 jsou paměťové rozsahy rozděleny do úseku po 256 bytech. Ty se však již nejmenují bloky (sektory), ale stránky (pages). RAM 1541/70/71 obsahuje 8 stránek, číslovaných od 0 do 7, celkem tedy 2kB. Stránka 0 (nultá stránka - Zero Page) je stejně jako u C-64 používána operačním systémem jako pracovní paměť, a proto není k volnému použití. Podobně je to se stránkami 1 a 2. Stránky 3 až 7 se používají jako tzv. oddělovací či vyrovnávací paměť (BUFFERS). Zde se dočasně ukládají všechna data, která se z diskety čtou nebo se na disketu zapisují. Má-li být např. na disketě změněn obsah jediného bytu, je napřed úplný blok přečten do některé volné stránky (bufferu), zde je provedena změna a nakonec se celá stránka znova napíše zpět na disketu. Z toho plyne, že je užitečné si před přístupem na disketu nějaký buffer rezervovat a s ním pak dále pracovat. Syntaxe příslušného povelu je následující:

OPEN fn,dn,kn,"#" kde:

fn = číslo souboru (FILE Number)

dn = číslo zařízení (Device Number)

kn = číslo kanálu

Číslo souboru může být v rozsahu 1 až 255. Používá se v průběhu celého programu pro identifikaci souboru, se kterým je komunikováno. Z důvodu tisku je lépe užívat čísla do 127, neboť větší čísla generují signál pro nový řádek.

Číslo zařízení je obvykle 8. Při větším počtu diskových jednotek se dále užívají hodnoty 9, 10 a 11.

Číslo kanálu je normálně v rozsahu 2 až 15. Vztahuje se na kanál, používaný pro komunikaci s diskem. Kanály 0 má DOS rezervován pro povel LOAD a kanál 1 pro povel SAVE. Kanál 15 je povelový kanál, po němž disk dostává povely z počítače a zpět posílá chybová hlášení. Pro datové soubory tedy zůstávají kanály 2 až 14.

Např. povel OPEN 1,8,2."#" otevírá v počítači jeden soubor s číslem 1, číslo diskové jednotky je 8 a je v ní rezervován jeden kanál č. 2, jemuž je přiřazen jeden buffer.

Využitím odpovědi na kanálu 15 a s pomocí povelu INPUT lze po každém povelu kontrolovat úspěšnost jeho provedení:

```
10 OPEN 15,8,15
20 INPUT#15, A$,B$,C$,D$
30 PRINT A$,B$,C$,D$
```

První, třetí a čtvrtá proměnná přicházejí jako čísla a můžeme je proto vložit i do numerických proměnných. Význam jednotlivých proměnných je tento:

A$= číslo chyby (0 = bez chyby)
B$= popis chyby
C$= číslo stopy, na které došlo k chybě
D$= číslo sektoru, na kterém došlo k chybě

Chceme-li používat některý určitý buffer (třeba pro uložení strojového programu), pak je disku nutno sdělit, který buffer je požadován. Např. buffer č. 1 se přidělí povelem:

```
OPEN 1,8,2,"#1"
```

Zde je třeba dát pozor, aby nebyl zvolen již obsazený buffer, jinak disk vyšle chybové hlášení. Všeobecně je buffer č. 3 užíván pro Direktory a buffer č. 4 pro BAM. Přenecháte-li volbu bufferu disku (použitím znaku "#" bez čísla), pak můžete dále uvedeným způsobem zjistit, který buffer byl vybrán:

```
10 OPEN 1,8,2,"#"
20 GET#1,D$
30 D=ASC(D$+CHR$(0))
40 REM ČÍSLO BUFFERU JE V "D"
```

Blokové povely

a) Čtení bloku: B-R (Block-Read)

Povelem B-R je možno přečíst libovolný blok diskety do dříve rezervovaného bufferu. Syntaxe je následující:

PRINT#fn,"B-R" kn,un,t,s kde nově použité jsou symboly:

un = číslo jednotky (Unit), vždy 0
t = číslo stopy (Track)
s = číslo sektoru (Sector)

Například povel PRINT#15,"B-R" 2,0,18,0 přečte do bufferu blok ze stopy 18 a sektoru 0. Povelem B-R nelze přečíst první byte bloku. K tomu je pak nutno použít (uživatelsky - User) povel U1. Ten má stejnou syntaxi a může být použit v každém případě: PRINT#15,"U1" 2,0,18,0. K User povelům se vrátíme ještě později. Smyčkou s pomocí GET# lze po přečtení bloku číst do počítače jednotlivé byty. Následuje příklad programu na přečtení bloku 2 stopy 18 a převedení dat do proměnné B$:

```
10 OPEN 15,8,15
20 OPEN 5,8,5,"#"
30 PRINT#15,"B-R" 5,0,18,2
```

```
40 B$=""
50 FOR L=0 TO 255
60 GET#5,A$
70 IF ST=0 THEN B$=B$ + A$: NEXT L
80 PRINT "KONEC"
90 CLOSE 5: CLOSE 15
```

Poznámky:

1) Kanál povelu (15) je vhodné otevírat jako první a uzavírat jako poslední.
2) Při vkládání dat do bufferu sleduje ukazatel (pointer), kolik je znaků. Při zápisu je ukládán i ukazatel. Při čtení je pak možno kontrolovat, zda není pokus o čtení za konec dat (tedy za značku EOF - End Of File). Systémová proměnná ST se pak stává nenulovou (viz test na řádku 70).

b) Psaní bloku: B-W (Block-Write)

Povelem B-W umožňuje zapsat na disketu data z bufferu. Syntaxe je stejná jako u povelu B-R. Příklad: PRINT#15,"B-W" 2,0,18,0. Přirozeně existuje i příslušný User povel U2. Příklad: PRINT#15,"U2 2 0 18 0".Následuje příklad zapsání dat (50 z "TEST") na stopu 1, sektor 1:

```
10 OPEN 15,8,15
20 OPEN 5,8,5,"#"
30 FOR L=1 TO 50
40 PRINT#5,"TEST"
50 NEXT
60 PRINT#15,"B-W" 5,0,1,1
70 PRINT "KONEC"
80 CLOSE 5: CLOSE 15
```

c) Povel BUFFER-POINTER: B-P

Pro každý buffer existuje jeden vektor (ukazatel = Buffer Pointer), ukazující na aktuální byte v bufferu, a tento ukazatel je při každém přístupu o 1 zvyšován, a tím je možno přečíst postupně všech 256 bytů. Je také možné vsadit do ukazatele určitou hodnotu a zajistit tak přímý přístup k požadovanému bytu. Syntaxe:

PRINT#fn,"B-P" kn,pozice s hodnotou 0 až 255

Příklad načtení hodnoty 123. bytu ze stopy 1, bloku 16 do proměnné A:

```
10 OPEN 15,8,15
20 OPEN 1,8,2,"#"
30 PRINT#15,"U1" 2,0,1,16
40 PRINT#15,"B-P" 2,122
50 GET#1, A$
60 A=ASC(A$+CHR$(0))
```

d) Přidělení (alokace) bloku: B-A (Block-Allocate)

Při práci se soubory s přímým přístupem je před zápisem nutné předem v BAM nalézt volné (neobsazené) bloky a po zapsání v BAM označit, že bloky byly naplněny (obsazeny, přiřazeny) a nelze je dále používat. Povel B-A pro toto přidělování má syntaxi:

PRINT#fn,"B-A" un,t,s

Příklad: povel PRINT#15,"B-A" 0,1,16 vyznačí blok 16 na stopě 1 za obsazený. Byl-li blok již dříve obsazen, dá disk chybové hlášení č. 65: 65, NO BLOCK, XX, YY, kde v XX a YY je adresa (stopa a sektor) prvního volného bloku.

Před zápisem bloku je tedy nutné najít a přidělit volný blok. Není-li přidělovaný blok volný, lze z odpovědi na povelovém kanálu najít první volný blok. Příklad procedury alokace bloku:

```
10 OPEN 15,8,15
20 OPEN 5,8,15,"#"
```

```
30 PRINT#5,"DATA"
40 T=1: S=1
50 PRINT#15,"B-A",0,T,S
60 INPUT#15,A,B$,C,D
70 IF A=65 THEN T=C: S=D: GOTO 50
80 PRINT#15,"B-W" 5,0,T,S
```

e) Uvolnění bloku: B-F (Block-Free)

Tento povel je opakem k povelu B-A. Deklaruje obsazené bloky jako volné pro další přístup. Syntaxe je stejná jako u povelu B-A.

Poznámka: tímto povelem jsou bloky pouze uvolněny, ale do jejich přepsání v nich zůstávají původní data.

f) Povel vykonání bloku B-E (Block-Execute)

Povel je určen pro zkušené programátory. Je v principu shodný s povelem B-R s tím rozdílem, že blok přečtený z diskety do bufferu se sám odstartuje od adresy 0 bufferu jako program ve strojovém kódu a běží do instrukce RTS (Return from Subroutine - návrat ze subrutiny).

Povely pro paměť

a) Čtení paměti M-R (Memory-Read)

Povel Memory-Read lze srovnat s basic-povelem PEEK. V diskové jednotce je paměť 16kB ROM a 2kB RAM. Pomocí povelu M-R lze přečíst libovolné místo těchto pamětí. Syntaxe:

PRINT#fn,"M-R",CHR$(adL),CHR$(adH),CHR$(n) kde:

adL = spodní (Low) byte adresy paměti
adH = horní (High) byte paměti
n = počet čtených bytů (v rozsahu 0 až 255)

Pomocí povelu GET# mohou být přečtená data vypsána přes povelový kanál.

Příklad: vypsat obě ID značky v ASCII formátu inicializované diskety:

```
10 OPEN 15,8,15
20 PRINT#15,"M-R" CHR$(18),CHR$(0),CHR$(2)
30 GET#15,A$,B$
40 PRINTA$,
```

Tato rutina přečte v nulté stránce adresy 18 a 19, kde jsou obsaženy požadované hodnoty.

b) Psaní do paměti M-W (Memory-Write)

Povel Memory-Write lze srovnat s basic-povelem POKE. Jeho pomocí lze zapsat do libovolného místa RAM paměti najednou až 34 bytů. Následujícími povely "MEMORY-EXECUTE" a "USER" lze pak takto zapsaná data spustit jako program.
Syntaxe povelu M-W":

```
PRINT#fn, "MW", CHR$(adL), CHR$(adH), CHR$(n), CHR$
                (data1), CHR$(data2)...
```

c) Odstartování programu v RAM M-E (MEMORY-EXECUTE)

Také tento povel má ekvivalent v basicu a to povel SYS. Povelem M-E můžeme tedy spustit strojový program z libovolného místa RAM paměti. Syntaxe:

```
PRINT#fn,"M-E" CHR$(adL),CHR$(adH)
```

Následuje příklad, užívající povelu M-W a M-E pro vykonání instrukce RTS (kód 96 - návrat ze subrutiny):

```
10 OPEN 15,8,15
20 PRINT#15,"M-W",CHR$(0),CHR$(5),1,CHR$(96)
30 PRINT#15,"M-E",CHR$(0),CHR$(5)
40 CLOSE 15
```

Uživatelské povely U (USER Commands)

Kromě již uvedených povelů USER 1 a USER 2 (zkráceně U1 a U2, případně UA a UB) existují i další USER (uživatelské povely). S diskem 1541 může pracovat rovněž počítač VIC 20. Potřebuje však přenos nižší rychlostí přenosu. Pro toto přepínání slouží povel U9- a U9+. Po zapnutí se disk automaticky přepíná pro C-64. Pro přepnutí pro VIC 20 užij povely:

OPEN 15,8,15 "UI-":CLOSE 15

a pro návrat do C-64 užij:

OPEN 15,8,15:CLOSE 15

Povely U3 až U8 slouží k odstartování programu (dříve zapsaných do RAM paměti disku) od příslušných adres. Přehled všech USER povelů je v následující tabulce:

povel	význam
U1 nebo UA	čtení bloku beze změny buffer ukazatele
U2 nebo UB	psaní bloku beze změny buffer ukazatele
U3 nebo UC	skok na#$0500
U4 nebo UD	skok na#$0503
U5 nebo UE	skok na#$0506
U6 nebo UF	skok na#$0509
U7 nebo UG	skok na#$050C
U8 nebo UH	skok na#$050F
U9 nebo UI	skok na#$FFFA (NMI - Non Masc. Interrupt)
U9-nebo UI-	přepnutí rychlosti na VIC 20
U9+nebo UI+	přepnutí (návrat) rychlosti na C-64 (normál)
U nebo UJ	Pover-Up Vektor (= Reset, podobně jako SYS 64738 u C-64)

1541/70/71 a ASSEMBLER

V následující tabulce jsou uvedeny zkratky a startovací adresy rutin DOS, které budeme dále používat a které lze s výhodou používat i v jiných editovacích programech:

Zkratka	Adresa	Zkratka	Adresa
FILPAR	$FFBA	SECLST	$FF93
FILNAM	$FFBD	IECOUT	$FFA8
OPEN	$FFC0	IECIN	$FFA5
CLOSE	$FFC3	FILTAB	$F30F
LISTEN	$FFB1	FILSET	$F31F
UNLIST	$FFAE	LOAD	$FFD5
TALK	$FFB4	SAVE	$FFD8
UNTALK	$FFAB	BASOUT	$FFD2
SECTLK	$FF96	CLALL	$FFE7

Následuje popis těchto rutin:

FILPAR a FILNAM

Při povelech OPEN, LOAD a podobných musíme sdělit odpovídajícím rutinám (podprogramům), které soubory a kde chceme otevřít. Abychom nemuseli ručně vkládat parametry do nulté stránky, budeme využívat odpovídající rutiny. Číslo souboru, zařízení a sekundární adresu vkládá rutina FILPAR. Vyžaduje následující umístění parametrů v registrech procesoru:

Číslo souboru	-> Akumulátor
Číslo zařízení	-> X - registr
Sekundární adresa (+ $60)	-> Y - registr

Příklad: Potřebujeme pracovat se souborem číslo 1, adresou zařízení 8 a sekundární adresou 15 (povelový kanál disku). Příslušná činnost je:

```
LDA #$01,  soubor č. 1
LDX #$08,  zařízení č. 8
LDY #$6F,  sekundární adresa 15 (= $0F) + $60
JSR #FILPAR, skok do subrutiny FILPAR, která vsadí parametry
```

Jak je v příkladu vidět, musí být k sekundární adrese přičtena hodnota $60. V mnoha případech se musí také zadat název souboru. K tomu slouží rutina FILNAM. Vyžaduje tyto parametry:

```
Délka názvu souboru        -> Akumulátor
LO Adresa jména souboru    -> X - registr
HI Adresa jména souboru    -> Y - registr
```

Musíme tedy předem vědět, kde je jméno souboru v paměti uloženo a jak je dlouhé.

Příklad: Předání parametrů k otevření direktory (soubor se jménem "$"):

```
LDA  #$24 , kód "$" do akumulátoru (1 znak)
STA  #$FF , a jeho uložení na adresu $FF
LDA  #$01 , délka souboru = 1
LDX  #$FF , LO byte adresy do X
LDY  #$00 , HI byte adresy do Y (adresa =#$00FF)
JSR  FILNAM
```

Přes povelový kanál lze disku poslat povely sekvencí v basicu

```
OPEN x,8,15 "povel"
```

a přirozeně je také možno všechny parametry vsadit ručně na odpovídající místa.

OPEN a CLOSE

Jakmile jsou všechny parametry a název souboru uloženy, je možné zavolat rutinu OPEN (JSR OPEN), a tím je soubor otevřen. Je však nutno dát pozor na to, aby v počítači nebylo současně otevřeno více než 10 souborů! Rutina CLOSE pracuje analogicky, ale zadává se pouze číslo souboru. Číslo zařízení a sekundární adresu si C64 vyhledá sám z příslušné tabulky (budeme o ní mluvit později).

Příklad: LDA #$01
JSR CLOSE

LISTEN a UNLISTEN, TALK a UNTALK

Po otevření souboru ještě nemůže začít přenos dat. Je třeba odpovídajícímu zařízení nejprve sdělit, jestli má přijímat nebo vysílat. Nejlepší příklad je povelový kanál. Přes něj může disk přijímat povely a stejně tak posílat do počítače chybová hlášení. Abychom zařízení nastavili na příjem, použijeme rutinu LISTEN (z anglického "slyšet" - nezaměňovat proto s basic povelem LIST). Ze zařízení se po zavolání LISTEN stane přijímač a z počítače je vysílač.

Před zavoláním kterékoliv z těchto 4 rutin je nutno vložit do akumulátoru číslo zařízení.

Po skončení vysílání dat z počítače je nutno poslat přijímacímu zařízení UNLISTEN (užitím rutiny s tímto jménem), a tím se zařízení uvolní.

Analogicky se zachází s rutinami TALK a UNTALK. Slouží k přípravě zařízení k vysílání a k opětnému uvolnění.

SECTLK a SECLST

Obě rutiny jsou velmi důležité pro přenos dat. Jestliže byla při povelu OPEN zadána určitá sekundární adresa, je ji nutno posílat i při každém dalším přenosu na příslušné zařízení. Toto má dva významy. Na jedné straně je možné současně otevřít více kanálů s

tím, že disk vždy ví, pro který kanál je určen následující přenos dat, případně, který kanál má vysílat, a pak se musí po TALK provést SECTLK. Kromě toho si počítač zadané sekundární adresy pamatuje, ale nevysílá je.

IECOUT a IECIN

Jsou-li konečně provedeny všechny přípravy, můžeme přenášet byty mezi diskem a počítačem. IECOUT přenáší byte z AKKU na aktuální zařízení a IECIN naopak byte ze zařízení přijímá a ukládá ho do AKKU.

Při přenosu může někdy dojít k chybě. K signalizaci chyby některé subrutiny testuje operační systém stav CARRY-FLAG (návěští, vlajka). Je-li tento flag nastaven (stav "1"), je něco špatného. Podrobnosti o chybě jsou uloženy ve Status-bytu (stavový byt). Ten je uložen v paměti na adrese $90. V následující tabulce je význam jednotlivých bitů Status-bytu:

Bit	Význam (je-li bit v logické "1")
1	Chyba při IEC vstupu (časové překročení)
2	Chyba při IEC výstupu (časové překročení)
3-5	Pouze pro kazetovou jednotku (magnetofon)
6	Přenos je ukončen
7	Zařízení se nehlásí

Je-li např. bit 7 v "1", pak buďto volané zařízení neexistuje, je vypnuto nebo má nějakou poruchu. V basicu je to převedeno do zprávy "DEVICE NOT PRESENT ERROR". Bit 6 slouží k testování ukončení přenosu a lze jej užít např. takto:

```
10 OPEN 1,8,15
20 GET#1,A$: PRINTA$;: IF ST<>64 then 20
30 CLOSE 1
```

Jak je z příkladu vidět, je obsah Status-flagu (buňky #$60) ukládán do již zmíněné systémové proměnné ST.

Před každým novým přenosem je nutno dbát na to, aby byl Status-byte vynulován (smazán)!!!

PRÁCE S VÍCE SOUBORY (FILTAB a FILSET)

Umíme již pracovat s jedním souborem. Co se však stane, když musíme otevřít dva soubory současně, např. když potřebujeme přečíst z diskety nějaký blok. K tomu potřebujeme povelový kanál a zároveň přenosový kanál. Je možné vždy pomocí CLOSE jeden kanál zavřít a pomocí OPEN jiný otevřít, ale jde to také jednodušeji. Nejprve musí být požadované soubory otevřeny. Pak je možno libovolně přepínat mezi soubory 1 až 10. K tomu slouží rutiny FILTAB a FILSET. Rutina FILTAB vyžaduje, aby v AKKU bylo číslo souboru, na který má být přepnuto. Rutina sama pak najde v tabulce souboru odpovídající další parametry požadovaného souboru. Dojde-li k chybě (např. soubor není otevřen), pak je "nulový-flag" smazán a je možné pomocí instrukce BNE provést přezkoušení. Rutina FILSET zapisuje nalezené parametry do příslušných míst nulté stránky. Příklad přepnutí na jiný soubor:

```
LDA  #$XX , XX = číslo souboru
JSR  FILTAB , vyhledání parametrů v tabulce
BNE  ERROR , chyba ?
JSR  FILSET , zapsání parametru
```

Předtím je pochopitelně ještě nutno zapsat rutinu ERROR. Po přepnutí na požadovaný (aktuální) soubor se pak všechny následující rutiny (LISTEN, TALK atd.) týkají tohoto aktuálního souboru. Před každým přepínáním na jiný soubor je nutné nejprve zavolat UNLIST a UNTALK. Naproti tomu se CLOSE volá teprve při úplném ukončení činnosti se souborem.

V následující tabulce jsou uvedeny důležité adresy nulté stránky, do kterých se přenášejí automaticky parametry aktuálního souboru z celkové tabulky souboru. Celková ("velká") tabulka je v paměti na adresách $0259 až $0276. Vyjímkou je název souboru, který je potřebný jen při otevření souboru (přenáší se jen vektor názvu).

Adresa	Význam
$90	Status-flag
$93	Flag pro LOAD/VERIFY
$98	Počet otevřených souborů
$99	Vstupní zařízení pro BASIN
$9A	Výstupní zařízení pro BASOUT
$B7	Délka názvu souboru
$B8	Číslo aktivního souboru
$B9	Sekundární adresa
$BA	Číslo zařízení
$BB/$BC	Vektor na adresu názvu souboru

LOAD a SAVE

V principu je možno stejným způsobem natahovat a ukládat programy. Mnohem jednodušší je však používat specializované rutiny LOAD a SAVE. Všimněme si nejprve rutiny LOAD. Také zde je třeba nejprve zadat několik číselných parametrů. Pomocí FILPAR se vsadí číslo zařízení a sekundární adresa. Číslo souboru není třeba zadávat. Pro sekundární adresu platí následující: je-li rovna 0 (nebo nezadána), je program natažen od standardní adresy pro basic programy #$0800). Je-li rovna 1, je program zaveden od adresy, ze které byl pomocí SAVE převzat (tato adresa je na začátku programu zapsána). To se užívá zejména pro různé pomocné programy ve strojáku (Loadery atd.). Název programu se vsazuje pomocí FILNAM.

TAB a FV principu je možno program zavést od libovolné adresy. Pak je před zavoláním LOAD nutno vložit další dosud nepoužívané parametry:

```
LOAD/VERIFY-FLAG -> AKKU (0 = LOAD, 1 = VERIFY)
LOAD adresa LO   -> X-registru
LOAD adresa HI   -> Y-registru
```

Poslání adresy (do X a Y) má význam jen pro LOAD. Ještě poznámka k sekundární adrese. Je rezervována pro povel LOAD. Bez ohledu na její hodnotu je do disku posílána 0. Sekundární adresa se také nemůže bez uvážení dávat při povelu OPEN.

Po skončení rutiny LOAD je v X a Y uložena koncová adresa zaváděného programu. Příklad programu využívajícího LOAD:

```
LDX  #$08 , číslo zařízení
LDY  #$00 , sekundární adresa pro relativní zavedení
JSR  FILPAR
LDX  # (LO byte jména souboru)
LDY  # (HI byte jména souboru)
LDA  # (délka jména souboru)
JSR  FILNAM
LDA  #$00 , flag nastaven na LOAD
LDX  # (LO byte startovací adresy)
LDY  # (HI byte startovací adresy)
JSR  LOAD
RTS
```

Rutina SAVE vyžaduje jeden poněkud složitější parametr. FILPAR je nutno volat jen s číslem zařízení v X-registru a není třeba sekundární adresa ani číslo souboru. Vsazení souboru probíhá normálně přes FILNAM. Je nutno zadat ještě počáteční (start) adresu a koncovou (end) adresu + 1. Počáteční adresa se udává někam do nulové stránky v pořadí LO, HI. Doporučuje se použít buňky $FB

a $FC, protože nejsou používány operačním systémem ani basicem. V AKKU pak musí být adresa LO bytu (např. pro byty $FB a $FC to bude $FB). LO byte konečné adresy se ukládá do X a HI byte (+1) se ukládá do Y-registru.
Příklad programu užívajícího SAVE:

```
LDX  #$08 , číslo zařízení
JSR  FILPAR
LDX  # (LO byte jména souboru)
LDY  # (HI byte jména souboru)
LDA  # (délka jména souboru)
JSR  FILNAM
LDA  #$00 , flag nastaven na LOAD
LDX  # (LO byte startovací adresy)
LDY  # (HI byte startovací adresy)
STX  $FB , uložení do stránky 0
STY  $FC
LDA  #$FB , pointer na startovací adresu
LDX  ? (LO byte koncové adresy+1)
LDY  ? (HI byte koncové adresy+1)
JSR  SAVE
RTS
```

CO JE TO SPOOLING?

Pod pojmem SPOOLING rozumíme tisk (na tiskárnu) přímo z diskety, aniž by se zaměstnával počítač. Po odstartování SPOOLING programu zahájí tiskárna příslušný výpis a počítač se již hlásí READY pro další příkaz. "Zázrak" této možnosti je dán využitím sériového busu, spojujícího počítač s diskem a tiskárnou. Víme, že bus lze obsluhovat strojovým programem. Trik SPOOLINGU je v tom, že pomocí povelu CMD v basicu můžeme listing na disketu "obejít" a podobně to jde i s tiskárnou. Otevřeme i soubor a povelem CMD odešleme všechny následující výstupy místo na obrazovku na bus. Nyní už není adresátem tiskárna, ale disk.

Příklad: v paměti počítače je program "TEST" a jeho listing chceme uložit na disk (a pak vytisknout):

```
OPEN 1,8,2,"TEST,U,W"
CMD 1
LIST
```

Po těchto povelech je listing uložen na disku jako soubor typu USR. Nyní lze vyslat listing z disku přes bus na tiskárnu a počítač zůstane volný pro jinou činnost, která ovšem nemůže využívat bus. Po dokončení výpisu zůstávají obě zařízení přepnuta pro přenos a musí se tedy uvést do klidového stavu, dříve než budou znovu volána počítačem. Listing programu SPOOLING je v Sonderheft 9 na str. 43 jako listing 13. Program se spouští SYS 828, "název souboru". Potom se počítač ohlásí:

```
SPOOLING název souboru
READY
```

a disk a tiskárna zahájí svou práci. Po ukončení tisku se znovu napíše SYS 828, ale bez názvu souboru (LED na disku zhasne) a na obrazovce se objeví hlášení:

```
END OF SPOOLING
READY
```

Uvedený program bohužel nefunguje se všemi typy tiskáren. Je třeba jej vyzkoušet případ od případu.

FORMÁTY NA DISKETÁCH

Pro využívání diskety pro čtení a zápis dat je nutné, aby na ní byly předem zapsány formátovací (identifikační) značky, umožňující rychlé nalezení požadovaného sektoru. V podstatě jsou dva možné způsoby, a to tzv. "hardsektor" a "softsektor". Při "hardsektoru" se používají diskety, které mají na vnitřním kruhu pro každý sektor vlastní dírku. Ty jsou snímány fotobuňkou a určují tedy začátek každého sektoru. Tímto způsobem lze lépe využít kapacitu diskety, ovšem za cenu zvýšení ceny hardware. Běžnější je proto "softsektor". Při tomto způsobu jsou začátky sektoru označeny identifikační-

mi značkami, zapisovanými při tzv. formátování. Mezi jednotlivé sektory jsou navíc vkládány určité mezery a celková kapacita diskety je při tomto způsobu o trochu méně využita. Mezery slouží k rovnoměrnému rozdělení sektoru na stopě a mezera mezi hlavičkou bloku a vlastním blokem poskytuje Diskcontrolleru čas při přepínání z módu čtení na mód psaní. Většinou bývá na disketě jediná dírka, od které se odvozuje začátek stopy. Existují i vyjímky, mezi nimi je i Commódore 1541/70/71, které určují i začátek stopy z identifikační značky. Počty stop, sektorů, organizace identifikátorů, direktory a data i vlastní způsob zápisu se u jednotlivých výrobců i typů liší a formátovací programy, operační systémy a další programy jsou proto těmto rozdílům přizpůsobeny. Vzhledem k uvedeným rozdílům není často možné převádět data z disket zapsaných na jednom počítači na počítač jiného výrobce. Disk 1541 (nebo 1570/71 v módu C-64) používá 35 stop a počet sektorů na stopě se pohybuje od 21 (na vnějších stopách) do 17 (na vnitřních). V souladu s tím se k vnitřním drahám snižuje zapisovací frekvence a částečně se tak koriguje zkracování dráhy na menším vnitřním průměru. Uvedené řešení, spolu s GCR módulací (viz dále) a umístěním a organizací direktory je důvodem pro nezáměnnost (nekompatibilitu) disket s disketami ostatních výrobců.

Vraťme se k formátu na 1541. Jak již bylo řečeno, je v každém bloku (sektoru) kromě vlastního datového bloku ještě identifikátor (nebo též bloková hlavička, header atd.), kontrolní součty a mezery, které jsou někdy počítány za součást hlavičky a dat. Na začátku blokové hlavičky i bloku dat je tzv. SYNC byte (zkráceně SYNC). Sestává ze vzorku $FF (samé jedničky) a slouží k synchronizaci rozpoznávací logiky (pro nalezení začátku). Po SYNC je u hlavičky byte s hodnotou $08 (bitově 00001000 a u dat s hodnotou $07 (bitově 00000111). Tím diskový kontrolor (DC) obě části odlišuje.

Organizace celého bloku je následující:

	SYNC
	$08
	Kontrolní součet
Bloková	Sektor (tento Sektor)
hlavička	Stopa (tato Stopa)
	ID2 (2. byte identifikační značky diskety)
	ID1 (1. byte identifikační značky diskety)
	Mezera

	SYNC
	$07
Datový	Stopa (následující Stopa)
blok	Sektor (následující Sektor)
	254 bytů dat (vlastní datový blok)
	Kontrolní součet vlastního datového bloku
	Mezera

Všimněte se údajů pro Stopu a Sektor v Datovém bloku. Někdy se těmto údajům také říká Link-adresa (z anglického Link = spojené). Je to adresa následujícího bloku aktuálního souboru. Pokud při čtení nesouhlasí přečtený a vypočítaný kontrolní součet dat, je čtení několikrát opakováno. Maximální počet neúspěšných pokusů o čtení je dán konstantou v operačním systému. Po kontrolním součtu je vyrovnávací mezera a po ní je SYNC hlavičky následujícího sektoru. Z uvedeného popisu sektoru je názorně vidět, proč není kapacita diskety plně využita jen pro data.

ZPŮSOB ČINNOSTI DOS (Diskového Operačního Systému)

Podobně jako u počítače jde po zapnutí signál RESET na nulu (do LOW) a mikroprocesor 6502 obdrží systémovou startovací adresu. Pak se rozběhne RESET PROGRAM, který provede základní test disku. Poznáme to podle toho, že se na krátkou dobu rozeběhne motor a rozsvítí se červená LED dioda. Není-li při testu registrována chyba, LED zhasne a motor se zastaví. Pak je incializo-

vána RAM paměť disku a nastaví se všechny důležité vektory adres. Tím je disk připraven k normální činnosti. Od této chvíle v něm běží tři "quasi" programy:

1) hlavní program, běžící ve smyčce, z které vychází jen pro provedení povelu
2) program Diskcontrolleru (DC), jenž je řízen přes IRQ a pomocí časovače v DC je volán každých 10 msec.
3) rutiny Buscontrolleru (BC), které jsou volány podle potřeby pomocí signálu ATM (když jde do LOW)

Funkce jednotlivých programů si nyní popíšeme podrobněji:

HLAVNÍ PROGRAM

Normálně běhá v čekací smyčce, dokud nepřijde povel od počítače. Povel nejprve aktivuje Busrutiny, které uloží do paměti vyslané byty (parametry povelu). Pak hlavní program obdrží hlášení, že přišel povel (hlavní program stále hlídá stav obou řídících rutin DC i BC). Vyhodnotí jej podobně jako Basic interpreter a pokud nedošlo k syntaktické chybě, tak jej provede. Při chybě posílá počítači chybové hlášení. Po správném provedení jsou povelové parametry opět smazány a hlavní program se vrátí do čekací smyčky.

PROGRAM DISKCONTROLLERU

DC obsahuje obvod VIA 6522 (Virtual Interface Adapter), pomocí kterého je stále v kontaktu s mikroprocesorem. Tento obvod obsahuje také časovač, vyvolávající v nastaveném rytmu (zhruba 10 msec.) IRQ a jeho prostřednictvím je zavolán Diskcontroller -Program. V tomto místě je třeba vysvětlit rozdíl mezi Diskcontrollerem (DC) a Diskcontroller-Programem. Vlastní Diskcontroller (DC) je hardware uvnitř disku. Pod Diskcontroller-Programem se rozumí část programu v DOS, řízená přes IRQ a ovládající vlastní DC. Úplné rozdělení těchto dvou pojmů není pro naše potřeby nutné ani účel-

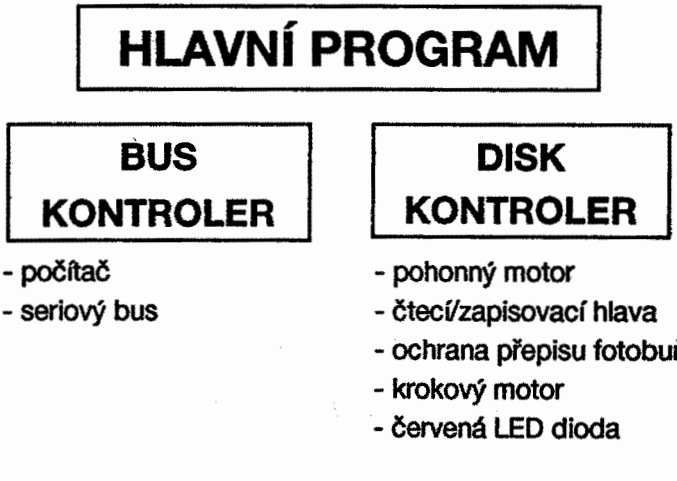
né. Výrazem DC proto budeme rozumět společné Hardware i příslušné Software (program).

Nyní opět k DC programu. Také zde je čekací smyčka, čekající na povel. Je-li přes bus přijat povel od počítače, aktivizuje se hlavní program, a ten předává povel dále do DC, jenž pak sám řídí aktivity disku. Jde zejména o řízení pohonné jednotky, krokovacího motoru, čtení a zápis dat. Veškeré procesy v disku jsou tedy řízeny přes interrupt (IRQ).

BUSRUTINY

Rutiny Buscontrolleru jsou rovněž řízeny přes IRQ vodič. Také BC obsahuje obvod VIA 6522. Zde ovšem nejsou rutiny volány přes časovač, ale pouze přes ATN signál sériového busu. Jde-li tento signál do LOW, pak bude v disku (stejně jako v ostatních periferiích) vyvolán IRQ. Tím jsou vyvolány příslušné BC rutiny, které pak přebírají další provoz busu. Zpracovává-li disk právě povel a přijde požadavek na další, pak BC čeká s přijetím až do uvolnění disku (vyřízení zpracovávaného povelu).

DOS je poměrně složitý. Základní schéma propojení tří uvedených programů je následující:



Chceme-li do tohoto systému vstoupit a nechat zde běžet svoje vlastní programy, pak musíme přesně znát "pravidla hry" celého systému. Jinak můžeme velmi snadno vyvolat malou katastrofu.

Poznámky:

- při pokusech s diskem je užitečné si jej otevřít a provozovat jej bez krytu. Umožní vám to sledovat pohyb hlav a zjistíte, co se děje při chybě.
- zjistíte také, že čtecí/psací hlavička je na spodní straně (odvrácené od etikety). Proto je také především nutno chránit spodní stranu diskety. Pochopitelně to neplatí u disket užívaných z obou stran (obrácených) a u disku 1571. Ten má hlavy z obou stran.
- při hlubším zájmu o práci DOS si opatřete listing (např. kniha vydaná Markt+Technik pro 1541 nebo 1570/1571).

PROGRAMOVÁNÍ DISKU

V listingu č. 14 je uveden miniprogram, napsaný pro práci s bufferem 0 (od adresy $0300). Po odstartování programu je zkoumán bit, odpovídající v DC za stav přelepky, chránící disketu proti zápisu. Je-li výřez přelepen (neprůsvitnou přelepkou!), je tím přerušen paprsek světelné závory. Pak je disketa chráněna proti zápisu. Náš program posílá bit světelné závory do bitu pro červenou LED diodu. Po novém nastartování LED zhasne (je-li závora přerušena). Když disketu vyjmete nebo vložíte jinou bez přelepky, LED se znovu rozsvítí. S tímto programem můžete tedy testovat umístění a neprůhlednost ochranné přelepky. Protože program pracuje v nekonečné smyčce, lze z něj vystoupit pouze přes RESET. Má také jednu chybu: neovlivňuje totiž pouze bity pro LED diodu v buňce paměti $1C00 (Buscontroller), ale maže při každém průchodu i ostatní bity. Pro naše účely to zatím nevadí.

Význam bitů obou řídících portů (Diskcontrolleru a Buscontrolleru) je v následujících tabulkách:

Diskcontroller (DC), adresa $1800, VIA 6522, PORT B

Bit	Význam
0	DATA IN
1	DATA OUT
2	CLOCK IN
3	CLOCK OUT
4	ATN OUT
5	Číslo zařízení
6	
7	ATN IN (CB 2)

Buscontroller (BC): adresa $1C00, VIA 6522, PORT B

Bit	Význam
0	Krokovací motor - cívka 1
1	Krokovací motor - cívka 2
2	Pohonný motor
3	Červená LED dioda v disku
4	Světelná závora (Write Protect)
5	Bit synchronizace pro DC
6	(Spurbereichen) rozsahy stop
7	SYNC-signál

POVEL "&"

Nyní se seznámíme s povelem &. Je také znám pod jménem "Autostart". Je nepochopitelné, proč není v žádném manuálu řádně popsán. Povel & odpovídá povelu BLOCK-EXECUTE, kde je natažen z diskety blok a ihned spuštěn jako program. U povelu & nebude natažen jeden blok, ale je vykonán celý soubor, načítaný do vybraného bufferu. Soubor, který má být spuštěn povelem &, musí mít specielní označení, a to znak & na prvním místě v názvu souboru. Má-li být tedy např. autostartem spuštěn soubor s názvem "TEST", pak je třeba zadat jméno "&TEST":

```
OPEN 1,8,15,"&TEST"
```

Máte-li na disketě jen jeden soubor s autostartem, pak jej lze volat jen znakem &:

OPEN 1,8,15,"&"

POZOR !!! Soubor určený pro autostart musí mít speciální syntaxi:

Byte	Význam
1-2	Startovací adresa u 1541 ve formátu HI/LO
3	Počet následujících programových bytů
4-N	Vlastní program
N+1	Kontrolní součet
N+2	Zde může být u dalších programů další díl (část) ve formátu opět začínajícím od bytu 1

U & souborů nejsou spojovací (linkovací - LINKER) popřípadě koncové znaky, které bývají u datových bloků v prvních dvou bytech. Při otevření nebo zápisu & souboru se vytvářejí automaticky.

V listingu č. 15 máme opět náš LED testovací program, tentokrát v basic podobě. Program můžete uložit na disketu v & podobě a potom již uvedeným povelem natáhnout do bufferu, a tím automaticky odstartovat. Před uložením na disketu však napřed musíte spočítat délku programu a kontrolní součet (zkušební sumu). Ta se počítá podle postupu v další kapitole.

Výpočet zkušební sumy

Je třeba sečíst všechny byty programu a k výsledku ještě přičíst dva byty startovací adresy a počet bytů v programu. Výsledkem je celé číslo, sestávající z dolního (LO) a horního (HI) bytu. Dolní byte je vlastní zkušební suma, zapisovaná na N+1. místo & programu. K tomuto bytu se přičítá přenos v horním bytu. Výpočet probíhá podle následujícího vzorce, kde Suma je úplný součet programových bytů:

HI = INT(Suma/256)

LO = Suma - HBx256

POZOR: Výpočet přenosu musí následovat po každém přičtení nebo změně hodnoty, protože konečný výsledek musí být menší než 256.

Jak vidíte, je použití &-souboru poměrně nesnadné. Dříve byl tento druh používán profesionály k ochraně souboru proti kopírování, a to z důvodu malé známosti povelu & a výstavby příslušných souborů.

Je třeba se ještě zmínit o dvou chybových hlášeních, s nimiž se můžete setkat:

- OVERFLOW IN RECORD se objeví, jestliže nesouhlasí skutečný počet bytů se zapsaným údajem.
- RECORD NOT PRESENT se objeví, jestliže nesouhlasí kontrolní suma.

Práci s &-soubory vám může usnadnit program v listingu č. 16. Program sám vypočítává kontrolní sumu a určuje ukončovací adresu.

Důležité adresy v DOS

Protože v našem kursu není uveden listing DOS, jsou v následující tabulce uvedeny adresy některých důležitých rutin, které lze užívat v našich programech:

$FD9E Návrat do JOB smyčky

$F556 Očekávání SYNC signálu z diskety

$FE00 Přepnutí na čtení

$FE03 Zápis stopy se vzorkem $55 (bin 0101 0101)

$FDA3 Zápis stopy na SYNC

$F510 Čtení blokové hlavičky
- disketa musí být incializována
- adr. $32/33 musí obsahovat adresu čísla stopy a sektoru (L/H), např. $00/03 pro čísla v buňkách $0300 a $0301.
- návrat jen při bezchybném čtení

$F527 Čtení blokové hlavičky

- disketa musí být inicializována
- nejdříve změnit $12 n $16 a $13 na $17
- číslo stopy a sektoru na adresy $18 a $19
- návrat jen při bezchybném čtení

$F50A Vyhledání začátku datového bloku
- parametry viz adr. $F510

POVELY DC - JOB kódy

V následující tabulce jsou uvedeny adresy všech "JOB-kódů" disku 1541. Jedná se o povely, provádějící určité samostatné činnosti (Joby):

$80 Přečtení bloku z diskety do určitého bufferu
$90 Zapsání bloku z určitého bufferu na disketu
$A0 Verify (ověření) jednoho bloku (srovnání s blokem v bufferu)
$B0 Test na existenci určitého sektoru
$C0 "BUMP" (rekalibrace, návrat na 0 a zpět, RTZ-Return to Zero)
$D0 Spuštění strojového programu v bufferu
$E0 Spuštění strojového programu v bufferu a potom

Vemte si nyní k ruce obsazení nulté stránky. Když pozorujete místa $0000 až $0005, pak si všimněte, že tyto adresy mají co dělat s JOB-kódy. Je zde totiž tzv. "JOB-paměť", jejíž úlohou je obstarávat dialog mezi hlavním programem a DC. Tato paměť neslouží jen k přenosu povelu z hlavního programu do DC, ale přicházejí do ní zpětná hlášení z DC a z nich si může hlavní program zkontrolovat bezchybnost vykonání povelu. Význam zpětných hlášení je v následující tabulce:

Hodnota	Význam	Chybová hlášení
$01	Bezchybné provedení	00, OK
$02	Nenalezena hlavička bloku	20, READ ERROR
$03	Nenalezena SYNC značka	21, READ ERROR

$04	Nebyl nalezen datový blok	22, READ ERROR
$05	Nevyšla zkušební suma	23, READ ERROR
$07	Chyba při VERIFY	25, WRITE ERROR
$08	Disketa chráněna proti zápisu	26, WRITE PROTECT ON
$09	Chybná zkušební suma	27, READ ERROR
$0A	Datový blok na disketě dlouhý	28, WRITE ERROR
$0B	Špatný ID v hlavičce bloku	29, DISK ID MISMATCH
$0F	V disku není disketa (atd.)	74, DRIVE NOT READY
$10	Chyba v dekódování	24, READ ERROR

Porovnáte-li bitový vzorek JOB-kódu a jejich zpětných hlášení, ihned zjistíte určité rozdíly. JOB-kódy jsou bez vyjímky větší než $80 (128). Mají tedy tzv. znaménkový bit 7=1 a jsou to negativní čísla. Při jejich provádění ve strojovém kódu se proto ve statusovém registru procesoru nastaví N-flag (příznak negativních čísel).

Zpětná hlášení jsou skoro vždy čísla do $10 (16). Tato velikost nehraje přímo žádnou roli, ale důležité je, že nesmí být větší než $7F (127). K objasnění tohoto rozdělení dospějeme dále.

Jak jsme viděli z obsazení nulté stránky, existuje pro každý buffer vlastní JOB-paměť. To umožňuje velmi dynamický provoz disku, kdy je možno například pracovat s více buffery.

Zapamatujte si důležité pravidlo. Vyšlete-li do DC nějaký JOB-kód, obsadí DC většinou jeden buffer. Přitom při podávání KOB-kódu do odpovídající adresy lze zvolit, který buffer bude obsazen.

POZOR!!! Nikdy při tom nepoužívejte buffer, ve kterém máte uložen svůj vlastní program, jinak bude program smazán. Máte-li např. program uložen od $0300 (Buffer 0), pak se musíte postarat o použití adresy $000 (nulté stránky) jako JOB-paměti. Jako mezipaměť jsou doporučeny adresy $0000 až $0005.

Nelze opomenout význam paměťových míst $0006 až $0011. K JOB-kódu je zpravidla nutné přidat určité parametry. Jde zejména o číslo stopy a sektoru. V předchozí tabulce je např. uveden JOB

-kód $80 pro čtení bloku. Je přirozeně nutné mu sdělit požadovanou stopu a sektor. Chcete-li tedy předat do DC bufferu 1 povel $80, pak se do adresy $0008 zapíše číslo stopy a do adresy $0009 číslo sektoru požadovaného bloku. Nakonec do adresy $0001 zapište povel $80 (JOB-kód).

Zbývá ještě jeden problém. DC potřebuje na provedení povelu určitý čas. Odkud se dozvíme, že je již např. přečten celý blok a že jej lze z bufferu převzít? K tomu lze využít již uvedených rozdílů v hodnotách povelových bytů a zpětných hlášení z DC. Protože zpětná hlášení z DC jsou posílána do stejných paměťových míst jako jsou povely, tak stačí u těchto buněk testovat přechod z hodnoty nad $80 (JOB-kódy) na hodnotu menší (zpětná hlášení). Tento přechod je příznakem ukončení povelu. Toto bude dále podrobněji objasněno v popisu jednotlivých JOB-kódů.

1) ČTENÍ SEKTORU DO BUFFERU

Zadání tohoto povelu je z hlediska JOB-smyčky stejně jednoduché jako v basicu u povelu "B-R" (případně "U1"). K přečtení sektoru zadáme jeho číslo stopy a sektoru do příslušných paměťových buněk a pošleme do DC kód $80. Disk se rozeběhne a přečte se požadovaný blok. Tento proces můžete realizovat i v basicu s povely MEMORY- a BLOCK-.

POZOR: Disketa musí být pro práci v JOB smyčce inicializována. Pamatujte si, že pokud nesouhlasí obsah bufferu s disketou, pak je to s velkou pravděpodobností způsobeno chybějící inicializací. O tom si povíme později více.

Dále si ukážeme činnost JOB kódu na malých příkladech, včetně zpětného hlášení z DC, abychom rozeznali chybu. Postupně pomocí JOB kódu přeneseme blok 18,1 do bufferu 0 a přečteme jej včetně zpětného hlášení. S pomocí povelu POKE (v programu) přepíšeme obsah bufferu přímo na obrazovku, abychom měli kontrolu.

```
1  OPEN 1,8,15,"I"
2  PRINT#1,"M-W"CHR$(6)CHR$(0)CHR$(2)CHR$(18)CHR$(1)
3  PRINT#1,"M-W"CHR$(0)CHR$(0)CHR$(1)CHR$(128)
4  FOR X=0 TO 2000: NEXT X
5  PRINT#1,"M-T"CHR$(0)CHR$(0)CHR$(1)
6  GET#1,A$:PRINTASC(A$+CHR$(0))
7  FOR X=0 TO 255
8  PRINT#1,"M-R"CHR$(X)CHR$(3)CHR$(1)
9  GET#1,A$:POKE 1024+X,ASC(A$+CHR$(0))
10 NEXT X
11 CLOSE 1
```

Tento malý program nejprve inicializuje disketu, pak přečte stopu 18, sektor 1 a zapíše JOB kód do adresy $0000. Ten se postará o to, aby náš blok byl natažen do bufferu 0. Po malé čekací smyčce (řádek 4), ve které se disku poskytne čas na provedení povelu, bude opět přečtena JOB paměť. Kontrolou zpětného hlášení z DC můžeme zjistit, jestli všechno proběhlo v pořádku (odpověď $01). Na obrazovce se pak objeví obsah bufferu, což je v našem případě část direktory.

2) ZÁPIS BLOKU NA DISKETU

Analogicky ke čtení jednoho bloku probíhá také zápis. Předávají se stejné parametry, přičemž musí být požadovaný blok již předem uložen do bufferu. Volbou JOB-paměti můžeme vybrat libovolný buffer disku (tedy 0 až 4) k přepsání bloku diskety.

3) VERIFIKACE BLOKU S DISKETOU

Tato činnost probíhá normálně automaticky vždy při povelu SAVE. Z tohoto důvodu také trvá záznam na disketu déle než čtení z diskety do počítače. Pomocí odpovídajícího JOB-kódu ($A0) lze verifikaci vyvolat i jindy pro porovnání obsahu buffer-paměti s blokem na disketě. Zjistí-li se při porovnávání rozdíl, obdržíme jako

zpětné hlášení číslo 7, čemuž odpovídá hlášení "VERIFY ERROR". Pokud jsme zapomněli disketu iniciovat, dostaneme na obrazovku chybové hlášení č. 11.

4) HLEDÁNÍ SEKTORU

Tento povel neslouží ke čtení bloku z diskety. Zde je pouze zjišťováno, zda-li se udaný blok na disketě vůbec nachází. Jestliže tam není, dostanete chybovou odpověď č. 2 ("20, READ ERROR"). Jistě vás již napadla další vlastnost JOB smyčky. Neprovádí při udání ilegálního bloku již žádnou další kontrolu. Když např. pošlete do DC povel pro přečtení bloku 2 na stopě 38, tak DC to provede. Zkuste si to s povelem "U1". Dostanete odpověď "ILLEGAL TRACK OR SEKTOR", protože stopa 38 u našeho formátu není vůbec použita (ale je v normálním rozsahu pohybu hlavičky). Když však budete chtít třeba číst ze stopy 100, dojde k nárazu vozíku hlavičky na mechanický doraz. Navenek se to projeví slyšitelnou ránou a při opakování může dojít i k poškození disku nebo ke změně nastavení polohy hlavičky. Změna nastavení se pak může projevit zhoršenou nebo úplně ztracenou kompatibilitou s dříve zapsanými disketami nebo disketami z jiných strojů. Tedy pozor na takovéto chyby, zvlášť opakované v programové smyčce.

5) BUMP - NOVÉ NASTAVENÍ HLAVY

Tento povel je znám i pod jinými jmény (rekalibrace, RTZ - návrat na nulu a zpět, znovunastavení). Nemůže-li DC identifikovat stopu, pak je možné, že hlava stojí na ilegální stopě. DC nemůže získat informaci o pozici z blokové hlavičky a je proto nutno provést povel BUMP. Ten nejprve vede hlavu zpět až na náraz ("bump") na mechanický doraz. Pak se hlava posune o jeden krok dopředu, a tím se dostanete na stopu 1. Tím je určena přesná poloha a pak je snadné vypočítat diferenci mezi první a požadovanou stopou a

dalším povelem hlavu vrátit na tuto stopu. Povel BUMP má hodnotu $C0.

6) START STROJOVÉHO PROGRAMU V BUFFERU

Pomocí JOB-kódu $D0 uděláte téměř to, co externě povelem "M-E". Rozdíl je v tom, že program spuštěný JOB-kódem bude pracovat jako interruptový, bude tedy začleněn do JOB-smyčky, a proto nesmí končit příkazem RTS, ale vždy návratem do JOB -smyčky pomocí JMP. Jak se z takovéhoto programu skočí zpět, bude vysvětleno později.

7) START PROGRAMU V BUFFERU PO SPUŠTĚNÍ POHONU DISKU

Většina programů vytvořených pro práci v JOB-smyčce vyžaduje čtení nebo zápis dat na disketu. Pak však nelze použít předchozí povel ($D0), protože disk normálně stojí. Proto použijeme povel s $E0. Ten pracuje následovně: jestliže DC pozná JOB-kód $E0, rozběhne disk a hardware se připraví na přístup na disketu a mimo jiné najede hlava na požadovanou stopu. Také zde musí být program ukončen instrukcí JMP do JOB-smyčky. Důležité také je, aby program odstartovaný s $E0 i $D0 vždy stál na začátku odpovídajícího bufferu. Má-li být spuštěn program, nalézající se na adrese vyšší než $XX00 (kde XX=00 až 07), pak jej spouštíme skokem na tuto adresu ze začátku bufferu.

JAK ZAPISUJE DOS NA DISKETU

S povelem $E0 se budeme v našem kursu často setkávat, protože tvoří základ přístupu na disketu. DOS jej používá např. pro formátování. Při použití JOB-kódu s $E0 nikdy nesmíme zapomenout na zadání čísla stopy a sektoru.

Nyní již víme, jak uložit strojový program do bufferu a jak jej tam odstartovat. JOB-kódy nám navíc umožňují přímo v průběhu

JOB-smyčky přistupovat na disketu a "ručně" s ní manipulovat. Jako poslední nám chybí znalosti, jak přímo přistupovat na čtecí/zapisovací hlavu a manipulovat s jednotlivými bity na magnetickém povrchu diskety, a to přímo, bez nějaké okliky a bez omezení, způsobeného blokovou strukturou zápisu na disketě. Tím se budeme zabývat nyní. Nejprve však několik poznámek ke čtecí a zapisovací elektronice disku 1541.

Byty jsou na disketu zapisovány sériově, bit po bitu. Obvod VIA 6522 může pro nás pracovat jako přechodná paměť. Pošleme-li do něj byte, pak sám automaticky provede převod do sériové podoby a zapsání na disketu a my se tímto problémem nemusíme dále zabývat. Problémem celé záležitosti je otázka časování. Zapisovací nebo čtecí proces vždy vyžaduje určitý čas. To znamená, že když např. chceme číst data, musí nám DC nejprve sdělit, kdy je na výstupu připraven k přečtení (odebrání) další byte. K řízení tohoto časování je u 1541 použit V-flag (přetečení) procesorového stavového (status) registru. Procesor 6502 má tu výhodu, že tento flag lze ovládat externě. Zpravidla se to děje následujícím způsobem:

Má-li elektronika přečtený celý byte, změnil se V-flag na "1". Podobně při zápisu, kdy je V-flag změněn na "1" při úplném zapsání bytu na disketu. Jediné, co programátor nesmí zapomenout, je vrácení V-flagu zpět na "0" poté, co zjistil jeho změnu na "1" a zahájil příslušnou činnost. Paměťová buňka, určená pro čtecí a zapisovací provoz, je port A Diskcontrolleru s adresou $1C01.

KONEČNĚ PRAXE

Nyní již máme dostatek znalostí k praktickému ověření na konkrétním programu. Budeme však potřebovat Monitor s miniassemblerem. Protože naše programy (např. k úmyslnému vytvoření chyby na disketě) jsou poměrně krátké, bude vhodné užívat monitor sedící v paměti od adresy $C000 a pro naše programy užívat paměť od $8000. Zajištění proti basicu:

POKE 56,127: POKE 52,127: NEW (nebo CLR)

Naše krátké strojové programy tedy uložíme od adresy $8000 a pošleme je basicovským programem do 1541, kde je odstartujeme.

POZOR! Při RESETU bude paměť disku smazána (vynulována). Proto doporučujeme program před každým novým startem opět zapsat do bufferu paměti disku.

CHYBA č. 21 NA DISKETĚ

V minulosti byla velice oblíbena ochrana programu proti kopírování pomocí úmyslně zapsané chyby na disketě. Příslušný program po startu hledal chybu na určitém místě, a pokud ji nenalezl, tak se zhoutil a zabránil kopírování. Naproti tomu se normální kopírovací programy zhroutily, když narazily na chybu. V dříve uvedené tabulce nalezneme rozdílná hlášení pro různé druhy chyb. Chyba č. 21 je hlášena v případě nenalezení SYNC značky. To se stane např. u neformátované nebo poškozené diskety. Stejnou chybu vyvoláme i programem v následujícím listingu:

```
0500                    ... mazání stopy č. 1, start programu
                            na #$0506
0500 JSR $FEOE          ... smazat stopu
0503 JMP $FD9E          ... skok do do JOB-smyčky
0506 LDA #$01           ... číslo stopy
0508 STA $0A            ... do JOB-paměti
050A LDA #$E0           ... JOB-kód
050C STA $02            ... zapsat
050E WAIT LDA $02       ... zpětná hlášení
0510 BMI WAIT           ... konec čekání (povel proveden)
0512 RTS                ... konec programu
```

Program popisuje určitou stopu diskety vzorkem $55 (binárně 0101 0101). To má za následek přepsání všech dat včetně hlaviček a SYNC značek. Když program spustíte a budete se snažit o ná-

sledné čtení ze stopy 1, dostanete stejně jako u neformátované nebo poškozené diskety chybové hlášení "21, READ ERROR". Jak vidíte, je tato chyba snadno vytvořitelná přepsáním celé stopy.

Těžší je to s jinými chybami, které jsou třeba jen v jednom bloku, přičemž některé z nich (20, 22) mohou být zapsány na celou stopu. Nám teď půjde o chyby 23, 24, 27, 28 a 29. Abychom takové chyby vytvořili, musíme najít nejprve místo, kde má být sektor poškozen, a pak v něm vytvořit chybu. V tabulce důležitých podprogramů DOS byly uvedeny jejich počáteční adresy. Tyto podprogramy lze s příslušnými parametry volat i zevnitř JOB-smyčky. Např. chybu č. 22 můžeme vyvolat takto: Zavoláme rutinu, která vyhledává datový blok, po nalezení bloku se vrátíme hned zpět pomocí RTS, přepneme na zápis a zapíšeme několik libovolných bytů. Zkusí-li pak DC takovýto blok později číst, je výsledkem chyba č. 22, způsobená poškozením hlavičky bloku, nacházející se hned za SYNC značkami. Listing programu pro vyvolání chyby č. 22:

```
Přepnutí na zápis
LDA  $1C0C    ... PCR
AND  #$1F     ... na mód zápisu
ORA  #$D0
STA  $1C00    ... přepnout
LDA  #$FF
STA  #$1C03   ... Port A na výstup

Přepnutí na čtení (také JSR#$FE00)
LDA $1C0C     ... PCR
ORA #$E0      ... na mód čtení
STA $1C0C     ... přepnout
LDA #$00
LDA #$1C03    ... Port A na výstup
```

Chceme-li vytvořit chybu č. 23, pak je třeba uvnitř datového bloku provést změnu dat. Potom přirozeně nebude souhlasit kon-

trolní suma na konci bloku a dojde k ohlášení "23, READ ERROR". Listingy 18a a 19a ukazují programy, vytvářející chyby 22 a 23 (listingy 18a a 19a jsou příslušné zdrojové programy). Výhoda chyby č. 23 je v tom, že data jsou zpravidla v bufferu dříve, než je tato chyba rozpoznána. Můžeme tedy prohlížet datový blok přečtený z diskety i přes ohlášenou chybu. Popsané chyby se vynikajícím způsobem hodí pro ochranu diskety před kopírováním. Účinné a bezpečné jsou přitom chyby, obsahující současně data. Existuje totiž velké množství programů, které chyby převezmou a na kopii ji opět simulují.

A ještě jedna poznámka. Slyšeli jste někdy něco o "Killertracks"? Toto podivné slovo označuje manipulaci se stopou, kdy jsou využita všechna bezpečnostní opatření DOS. Možná, že jste měli po ruce disketu, vykazující následující příznak: když jste zkusili přečíst blok na určité stopě, pak byla čtecí hlava správně vystavena a DC pak zahájil čtení bloku - a nic se nepřečetlo. Červená LED zůstala svítit a disk četl a četl ... Disk se tzv. "zavěsil" při pokusu o čtení "killertracku".

Následuje listing programu pro zapsání takovéto stopy ("killertrack"). Vlastní start je na adrese $0506.

```
0500 JSR $FDA3     ... stopu smazat
0503 JMP $FD9E ... do JOB smyčky
0506 LDA #$01      ... číslo stopy
0508 STA $0A       ... do JOB paměti
050A LDA #$E0      ... JOB kód přerušení
050C WAIT STA $02
050E LDA $02       ... zpětná hlášení
0510 BMI WAIT      ... čekání na konec
0512 RTS
```

Celá stopa obsahuje v tomto případě pouze SYNC značky. Protože ty se čtecí elektronikou zpracovávají specielním způsobem, naruší se normální činnost DC. Protože disk se při chybě snaží opakovat čtení i více než 200krát, narůstá při tomto typu narušení

značně čas. Projevuje se to ve zpomalení blikání LED a ve zdánlivé nekonečnosti čtení.

Při další intenzivní práci s diskem brzy naleznete nové možnosti a jak dále poznáte, tvoří ochrana proti kopírování jen malou část možností 1541.

FORMÁTOVÁNÍ

Je všeobecně známo, že každá disketa musí být před prvním vkládáním dat předem naformátována. Jak vypadá disketa po naformátování na 1541 jsme si již řekli. Nyní nás zajímá, co se v disku při vlastním formátování děje a proč 1541 potřebuje na tak jednoduchý proces tolik času. Připomeneme si: při formátování jsou vytvořeny všechny značky a hlavičky sektoru a jsou umístěny včetně doplňujících dat na příslušná místa. Při formátování se používá nám již známý JOB kód $E0. Předtím, než DOS odstartuje vlastní formátování, bude na adresu $0600 (buffer č. 3) uložen příkaz skoku JMP $FAC7. Tento příkaz ke skoku je určitý druh vektoru, jenž leží v RAM a může být tím pádem měněn. Tím se nabízí uživateli možnost vytvořit si vlastní rutiny a ty využívat při každé změně stopy pro dosažení svých cílů. Obvykle ukazuje tento vektor přímo na jednu JOB rutinu, příslušnou pro formátování. Jde o rutinu volanou z hlavního programu s JOB kódem $E0, který je zapsán v paměťové buňce $03.

Formátování v JOB smyčce

Na začátku JOB rutiny se testuje, zda již byla formátována alespoň jedna stopa. V opačném případě se do smyčky nejprve vloží všechny parametry pro krokovací motor a pak následuje návrat do JOB smyčky. Zde bude čtecí/psací hlava vrácena 45 stop (!), což se projeví charakteristickým zahrčením a klapnutím. Počet 45 je zvolen s ohledem na bezpečné vrácení hlavy, umístěné např. po chybě programu za poslední stopou, zpět na stopu 1. Pak následuje

znovu odskok do formátovací rutiny a do příslušné buňky se zaznamenává umístění hlavy na první stopě a může začít vlastní formátování. V něm se nejprve otestuje, zda je stopa již naformátována. Když ano, odskočí se znovu do JOB smyčky pro přechod na další stopu atd., až do stopy 35, po které se formátování ukončí.

Měření stopy a mezery mezi sektory

Nyní jsou splněny všechny předpoklady k formátování stopy. Činnost, která se nyní rozbíhá, je rovněž náročná na čas. Než se totiž SYNC značky a sektory zapíší na stopu, je tato stopa z DOS "změřena". Operační systém disku 1541 totiž přesně ví, kolik bytů pro SYNC značky a sektory určité stopy je potřeba. Jednotlivé sektory jsou od sebe oddělovány mezerami, neobsahujícími žádná data. Vzhledem k tomu, že se délka stop směrem od okraje (stopa č. 1) ke středu (stopa č. 35) zmenšuje, zmenšují se i tyto mezery. DOS zajišťuje, aby mezery mezi sektory na jedné stopě byly vždy stejné. Pro dosažení symetrického formátování diskety zjišťuje DOS pomocí zapisovacích a čtecích průběhů poměr mezi potřebným a existujícím místem stopy. Z tohoto poměru pak může určit, kolik volného místa bude muset vynechat mezi jednotlivými sektory. Po tomto komplikovaném měření, při němž došlo k několika otáčkám diskety a s tím i k další ztrátě času, může začít vlastní formátování diskety, které za normálních okolností netrvá déle než 1/3 sekundy pro jednu stopu.

Uložení dat sektoru do bufferu

Před vlastním zápisem na disketu je nutno nejprve sestavit obsah dat v bufferu. Protože se jednotlivé sektory liší jen svojí hlavičkou, mění se v bufferu jen tyto hlavičky. Obsah datového bloku se vloží jen poprvé a dále se nemění. Je tvořen znakem $4B, následovaný 225ti byty $01. Hlavičky se ukládají na adresy $0300 až $03FF a obsah datových bloků je na adresách $0500 až $05FF.

Zápis stopy na disketu

Všechny přípravné práce jsou již ukončeny a můžeme začít se zápisem na disketu. Nejprve bude DC nastaven do zapisovacího módu a maže se stopa na disketě. Úplný obsah stopy je zapsán během jedné otáčky a sektory jsou tvořeny již známým vzorkem SYNC značky blokové hlavičky, vlastní bloková hlavička, mezera 9 bytů, SYNC datového bloku, datový blok a již zmíněná mezera, sestávající z předem vypočteného počtu bytů.

Pro jistotu následuje se zápisem rutina VERIFY, která srovnává čtená data s požadovanými. Při rozdílu se vydá chybové hlášení "24, READ ERROR". Po VERIFY je formátování jedné stopy skončeno. Následuje dotaz na dosažení 35. stopy. Po dosažení této poslední stopy se zpětně vsadí všechny flagy formátování, opustí se JOB smyčka a následuje návrat do hlavního programu. Hlavní program nejprve zajistí přesun hlavy na stopu 18. Sestaví se BAM diskety a uloží se do bloku 18,0. Nakonec se do prvního Direktory -bloku (18,1) zapíší samé 0 a formátování je skončeno.

Variace při formátování

U již používané diskety je možno použít i zkrácené formátování. Jestliže je formátovací povel zadán bez ID (N-povel), pak se neprovádí úvodní kroky formátování. Pouze se zkontroluje v BAM, je-li zde správný formátovací znak (tzn. $42/65/"A") a pak se provádí jen zápis na stopě 18.

Jak víme, je formátovací rutina volána pomocí skokového povelu na adresu $0600 v RAM. Tato adresa je volána při každé stopě a nabízí se možnost při vytváření stopy zde odbočit na vytváření vlastní stopy. K tomu je však nutno mít k dispozici podrobnou dokumentaci DOS (listing), a to není předmětem tohoto kursu. Neznamená to však, že nejsme schopni provést jinou cestou zásah do formátování. Pokud nechceme pracovat s pevně zabudovanými

rutinami, pak zapíšeme do RAM disku vlastní program a zde se nám nabízí mnoho možností.

Vytvoření vlastního formátu

Prohlédneme si listing č. 21. Je to formátovací systém, který pracuje jednodušeji a rychleji než DOS rutiny, a přesto nabízí několik nových možností. Přestože téměř souhlasí s DOS rutinou, můžete si pomocí zdrojového listingu ověřit jeho nové možnosti. Abychom vám zapisování programu ulehčili, nabízíme v listingu č. 22 příslušný DATA-loader. Strojový program je natahován pomocí Basicu. Program je třeba nejprve uložit (SAVE) na disketu a pak teprve odstartovat (RUN). Po krátkém zpoždění se objeví na obrazovce kurzor a hlášení READY. Formátovací program je nyní přesunut na adresu od $C000 (49152) a SAVE vektor je patřičně změněn.

Nyní jednoduše napište SAVE a název souboru a stiskněte RETURN. Objeví se startovací hlášení formátovacího programu. Nyní můžeme zadat jméno pro disketu (max. 16 znaků). Počítač pak očekává zadání dvoumístného ID znaku. Pak lze zadat první a poslední formátovanou stopu, a to je zvláštnost tohoto programu oproti DOS. Údaj musí být v hexadecimálním tvaru a teoreticky může být v rozsahu $01 až $FF. Prakticky však nemůže být větší než 41 ($29). Při větší hodnotě jsou již hlavy mimo rozsah a dochází k mechanickému nárazu na doraz. Ještě je třeba upozornit na to, že opětovné formátování některé stopy na plné disketě není s tímto programem (beze změny) možné, protože Direktory je v každém případě zapisováno znova. Nebude-li disketa formátována úplně, musí být při dalším formátování zadáno stejné ID, jinak dojde k chybě s hlášením "29, DISK ID MISMATCH ERROR".

Potřebujete-li přesto znovu formátovat jednu stopu, aniž by došlo k poškození Direktory, pak musíte provést změnu v programu. Najdete si v listingu č. 21 adresu $06B5. Povel JSR $EE40 a další RTS změňte na NOP, NOP. Pak se změna Direktory nebude provádět.

V každém případě ale platí: při formátování jednotlivých stop musí být zadáno stejné ID, jaké již disketa obdržela dříve.

Další možnost je ovládání pohonu disku. Když si znovu prohlédnete program, pak naleznete na adrese $0696 povel pro DC, a to povel pro BUMP. Když jej zde změníte z $C0 na $00, pak zabráníte mechanickému nárazu hlavy na začátku formátování. Toto opatření je užitečné zejména při formátování více disket bezprostředně jedna za druhou.

Uvedený program naformátuje disketu asi za 30 sec. a pracuje tedy rychleji než vlastní program v DOS 1541. Proč tomu tak je, se dozvíme v dalším odstavci.

Rychlost, ale jak

V našem programu byl vynechán výpočet mezer mezi sektory. K tomu můžeme dojít konstatováním, že mezery jsou na každé disketě přibližně stejné. Proto můžeme použít určitou prakticky ověřenou hodnotu mezery, ke které přidáme ještě malou jistící hodnotu. Tato hodnota je u listingu č. 21 na adrese $05DF.

Na rozdíl od jiných rychlých formátovacích programů však nebyla vynechána rutina pro verifikaci, protože takto lze správně rozeznat vadné diskety a zabránit tak zbytečným pozdějším ztrátám dat. Navíc verifikace zabírá v celém procesu formátování jen malou část, a proto byla dána přednost jistotě před získánímskem několika sekund. Chcete-li přesto verifikaci vynechat, pak změňte konec formátovací rutiny u $05FD a místo JSR $FE00 vložte JMP $FD9E.

Další změny a vylepšení oproti DOS 2.6 jsou více méně kosmetického charakteru. Významnější je vyplnění datových bloků nulami. Normální rutina zapisuje $4B a 255x$01 byte. Tento obsah vlastně vede zpětně na chybu v DOS. Proto je obsah, tak jako u velkých Commódore disků, změněn na 265x byte $00.

Ještě několik poznámek k použití našeho formátovacího programu. Po RUN bude automaticky vsazen nový vektor pro start formátovací rutiny. Není-li zadáno žádné jméno souboru, pak násle-

duje skok do formátovacího programu. Po stisknutí (RUN/STOP + RESTORE) je možno vrátit SAVE vektor na původní hodnotu. Při chybějícím názvu souboru nebude žádný program odstartován. Jestliže jste museli stisknout (RESTORE), je možné formátovací systém restartovat pomocí SYS 49664 ($C200), po ukončení bude mimo jiné také zpětně nastaven SAVE-vektor. Chcete-li znát Disk Status, pak užijte SYS 49962. Poté následuje otázka na další formátovací průběh, kterou zodpovězte. Po zobrazení bude znovu vytvořen SAVE-vektor.

Jak víte, je v nulté stránce po resetu vsazeno několik konstant, které však uživatel může libovolně měnit. Konstantou je míněno např. $08 jako rozlišovací znak blokové hlavičky nebo $07 jako znak datového bloku. Jak můžeme zjistit z listingu nulté stránky, nachází se tyto hodnoty na adresách $39 a $47 a lze je zde změnit. Nové hodnoty musí být v rozsahu $00 až $0F, jinak by došlo k těžkostem při čtení. Pokud disketu vytvoříte s těmito hodnotami, důsledek čtecích pokusů s normálními hodnotami vyvolá chybové hlášení: "20, READ ERROR" nebo "22, READ ERROR". Protože ke změněným hodnotám v hlavičce můžete připsat normální datový blok, lze tento postup použít jako účinnou ochranu proti kopírování.

S pomocí našeho formátovacího programu si můžete vytvořit také ilegální stopy, zapsané za 35 stopou, tedy stopy 36 až 41.

GCR KÓDOVÁNÍ

V odstavci o formátování nebylo podáno vysvětlení ke skokovým povelům v listingu v S-formátu. Vzpomeňte si také na odstavec, kdy jsme se poprvé zabývali zápisem dat na disketu. Zde byla řeč o SYNC značkách, které slouží DC jako ukazatelé pozice. Zmiňovali jsme se o tom, že tyto SYNC značky se skládají z pěti bytů $FF zapsaných na disketě za sebou. Co se však stane, když datový blok obsahuje za sebou 5 či více bytů $FF. Vždyť tyto byty vlastně napodobují SYNC značky, a tím matou čtecí a vyhodnocovací elektroniku. Z praxe však víme, že ani u bloků se samými byty $FF

nedochází ke komplikacím. Při konstrukci disku se s tím totiž počítalo a jednoznačnost dat je dána jejich kódováním. Použitý kódovací systém se nazývá GCR (anglicky Group Code Recording) a znamená kódování po bitových skupinách. Pro vysvětlení si nejprve podrobněji popišme čtení a psaní disku.

Jak pracuje GCR kódování

Čtení bytu pomocí čtecí hlavy řídí časovač v DC. Na disketě samotné je každý jedničkový bit zaznamenán jako změna směru magnetizace a nulový bit jako zachování směru magnetizace. Změna směru vyvolá ve vinutí čtecí hlavičky elektrický puls a ten je dále zesílen a tvarován. Protože DC ví, kdy má který bit očekávat, lze pak tyto pulsy, (odpovídají změnám směru magnetizace) převést zpět na bitový vzorek. Otáčky diskety však nejsou zcela konstantní a je nutno počítat i s určitými rozdíly mezi jednotlivými diskovými jednotkami a vlastními disketami. Proto je nutné zajistit kompenzaci těchto chyb. Provádí se to pomocí znovunastavení (retriger) vyhodnocovacích obvodů (obvodů očekávajících puls, tedy změnu směru magnetizace) při každém jedničkovém bitu. Z toho plyne, že jedničkové bity mají nejen informační, ale i synchronizační úlohu a nelze proto připustit dlouhou řadu nulových bitů. Na druhé straně ale také nelze připustit dlouhou řadu jedniček, protože ty by byly vyhodnoceny jako SYNC značka.

Z těchto důvodů se SYNC značky na disketu zapisují normálně a data jsou před zápisem na disketu překódována pomocí GCR kódování. Toto kódování zajišťuje, že za sebou nemůže následovat více než 8 jedničkových a více než 2 nulové bity. Na disketě máme tedy dva druhy zápisu:

a) Zápis SYNC značek

SYNC značky jsou tvořeny pěti byty $FF. Je to tedy vzorek 40ti jedničkových bitů po sobě a jsou zapsány normálně bez kódování.

b) Zápis dat

Data jsou před zápisem na disk kódována podle následující tabulky:

hexa	binárně	GCR
$0	0 0 0 0	0 1 0 1 0
$1	0 0 0 1	0 1 0 1 1
$2	0 0 1 0	1 0 0 1 0
$3	0 0 1 1	1 0 0 1 1
$4	0 1 0 0	0 1 1 1 0
$5	0 1 0 1	0 1 1 1 1
$6	0 1 1 0	1 0 1 1 0
$7	0 1 1 1	1 0 1 1 1
$8	1 0 0 0	0 1 0 0 1
$9	1 0 0 1	1 1 0 0 1
$A	1 0 1 0	1 1 0 1 0
$B	1 0 1 1	1 1 0 1 1
$C	1 1 0 0	0 1 1 0 1
$D	1 1 0 1	1 1 1 0 1
$E	1 1 1 0	1 1 1 1 0
$F	1 1 1 1	1 0 1 0 1

Jak lehce zjistíme, jedná se u GCR o 5ti bitový kód. Každé čtyři bity binárního kódu (tzv. nibble) jsou převedeny na jeden pětibitový GCR-nibble. Jeden byte, sestávající před zakódováním z 8 bitů, bude po zakódování dlouhý 10 bitů. Délka kódovaných dat obecně vzroste s faktorem 5/4. GCR kódování tedy není jednoduchá záležitost. Při zakódování dvou bytů dostaneme pak dva a půl bytu a z toho plynou těžkosti. Ty však vyřešíme tak, že vždy najednou kódujeme 4 byty. Z nich vznikne 5 bytů a ty bez problému dále zpracujeme. Uveďme si ještě příklad. Dejme tomu, že máme kódovat 4 byty s obsahem $FF. Tedy vzorek, který by bez kódování byl zaměnitelný se SYNC. Z uvedené tabulky vidíme, že bytu $FF odpo-

vídá binární vzorek 10101 10101. Zařazením takovýchto 4 bytů za sebou, rozdělením na skupiny po čtyřech bitech s provedením na hex tvar dostaneme tento výsledek:

1 0 1 0 + 1 1 0 1 = $AD
0 1 1 0 + 1 0 1 1 = $6B
0 1 0 1 + 1 0 1 0 = $5A
1 1 0 1 + 0 1 1 0 = $D6
1 0 1 1 + 0 1 0 1 = $B5

Ze 4 bytů $FF jsme dostali výsledek $AD $6B $5A $D6 $B5. Můžete se přesvědčit, že tento vzorek již není pro DC nebezpečný pro záměnu se SYNC značkou.

Kódovací rutiny

V DOS existují následující rutiny pro konverzi:

$F6D0:

Vyzvedne čtyři HEXA byty z počítačových buněk $52 až $55, překóduje je na 5 GCR bytů a ty uloží do Buffer adresy $30/$31 (L/H) s Buffer vektorem v $34. Buffer adresa a vektor musí být zadány před zavoláním této rutiny.

$FD8F:

Tato rutina překóduje do GCR celý buffer, jehož adresa se musí nacházet v $30/$31 (L/H) a výsledek uloží zpět do doplňkové paměti ($01BB až $01FF) a do původního bufferu. Obsah bufferu (a doplňkové paměti) se při tomto kódování zvětší z 256 na 324 bytů.

$F7E6:

Tato rutina je inverzní k rutině na adrese $F6D0. Dekóduje pět GCR bytů (adresa v $30/$31 a vektor v $34) na čtyři HEXA byty a uloží je na adresy $52 až $55.

$F8E0:

Inverze k $FD8F. Rutina dekóduje celý obsah doplňkové paměti (adresy $01BB až $01FF) a bufferu (celkem 324 bytů) v GCR

kódu zpět na 256 bytů v HEXA. Výsledek uloží zpět do původního bufferu.

Použití těchto rutin je mnohostranné. Můžete je použít v Diskmonitoru k zobrazení GCR a normálních bytů. Jediné, co musíte provést, je přepočet adres pro rozsah paměti v počítači a zadání nových parametrů rozsahu pro buffer a nultou stránku. Kromě rozsahu paměti vašeho počítače není vaše fantazie ničím omezována.

Až 365 bytů v jednom bloku

Při kódování do GCR dochází k nárůstu bytů. Je obsazen celý buffer (256 bytů) a prvních 68 bytů je uloženo v doplňkové paměti na adresách $01BB až $01FF.

Celý datový blok s kontrolní sumou pak představuje 324 bytů. Přirozeně jsou před zápisem na disketu kódovány do GCR i parametry v hlavičce (tj. ID, stopa, sektor, kontrolní suma a rozlišovací značka) a délka hlavičky se zvětší včetně dvou prázdných bytů z 8 na 10 bytů. Zápis pro jeden sektor pak vypadá následovně:

5	bytů SYNC
10	bytů bloková hlavička
9	bytů $55 - odpovídají normě GCR a slouží jako oddělovací mezera, v níž DC získává čas pro přepínání mezi čtením a zápisem
5	bytů SYNC
324	bytů datový blok včetně kontrolní sumy
8-12	bytů mezera mezi sektory

Celkem má potom jeden sektor na disketě délku od 361 do 365 bytů.

Nyní jsou nám také jasné některé podivné dříve používané JSR-povely. U formátovacího systému byl použit povel JSR $FE30

a na jiném místě JSR $F78E. Tyto adresy jsou začátky kódovacích rutin.

Pamatujete si ještě, jak jsme vyráběli "Killertracks"? Zde se celá stopa plnila byty $FF a vytvořily se tak obrovité SYNC značky. Protože takovýto sled bitů není normálně možný, vypadne při pokusu o čtení vyhodnocovací elektronika z konceptu a Controller se zboří.

Pokud potřebujete získat další vědomosti o rychlých kopírovacích programech, metodách zabezpečování disket proti okopírování nebo dobrý listing DOSU, pak nahlédněte do knihy "Floppybuch 1541". Také uživatelům 1570 a 1571 doporučujeme prostudovat tuto knihu (nebo knihu pro 1570/71), protože oba tyto disky umí značně víc, než bylo vyloženo v našem kursu, zaměřeném na 1541.

COMMPAS
vydavatelství, služby a distribuce
Dr. Ivan Pavlíček, Pavel Škvrně
p. o. box 80
140 00 Praha 4