EDICE ELEKTRONIKY SVAZARMU

Ing. Roman Kišš

OSOBNÍ MIKROPOČÍTAČ PMD 85

EXTENDED ROM BASIC

UŽIVATELSKÁ PŘÍRUČKA

ŘADA 5

výpočetní technika

EDICE
ELEKTRONIKY
SVAZARMU

Ing. Roman Kišš

OSOBNÍ MIKROPOČÍTAČ PMD 85

EXTENDED ROM BASIC

UŽIVATELSKÁ PŘÍRUČKA

ÚSTŘEDNÍ VÝBOR SVAZARMU

PRAHA 1987

Uživatelská příručka

Osobní mikropočítač PMD 85

EXTENDED ROM BASIC

Část A : Rozšiřující příkazy

Část B : Diskové příkazy

Ing. Roman Kišš

Edice elektroniky Svazarmu
Řada 5 - Výpočetní technika

OBSAH - Část A Standardní příkazy BASIC

OBSAH - Část B Diskové příkazy

A.I. Všeobecná část

Úkolem EXTENDED ROM BASIC, dále jen ERB, je rozšířit stávající BASIC-G o další příkazy a funkce, hlavně o příkazy sloužící k obsluze minifloppy diskové jednotky MFD-85. ERB je možno používat pouze v součinnosti se základním interpreterem BASIC-G, t.j., že uživatel nejprve musí zavést BASIC-G do operační paměti a až potom příkazem –

ROM 0

přesune ERB do operační paměti.

Poznámka: EXTENDED ROM BASIC se umísťuje do nulté pozice a zabírá dvě uživatelská místa.

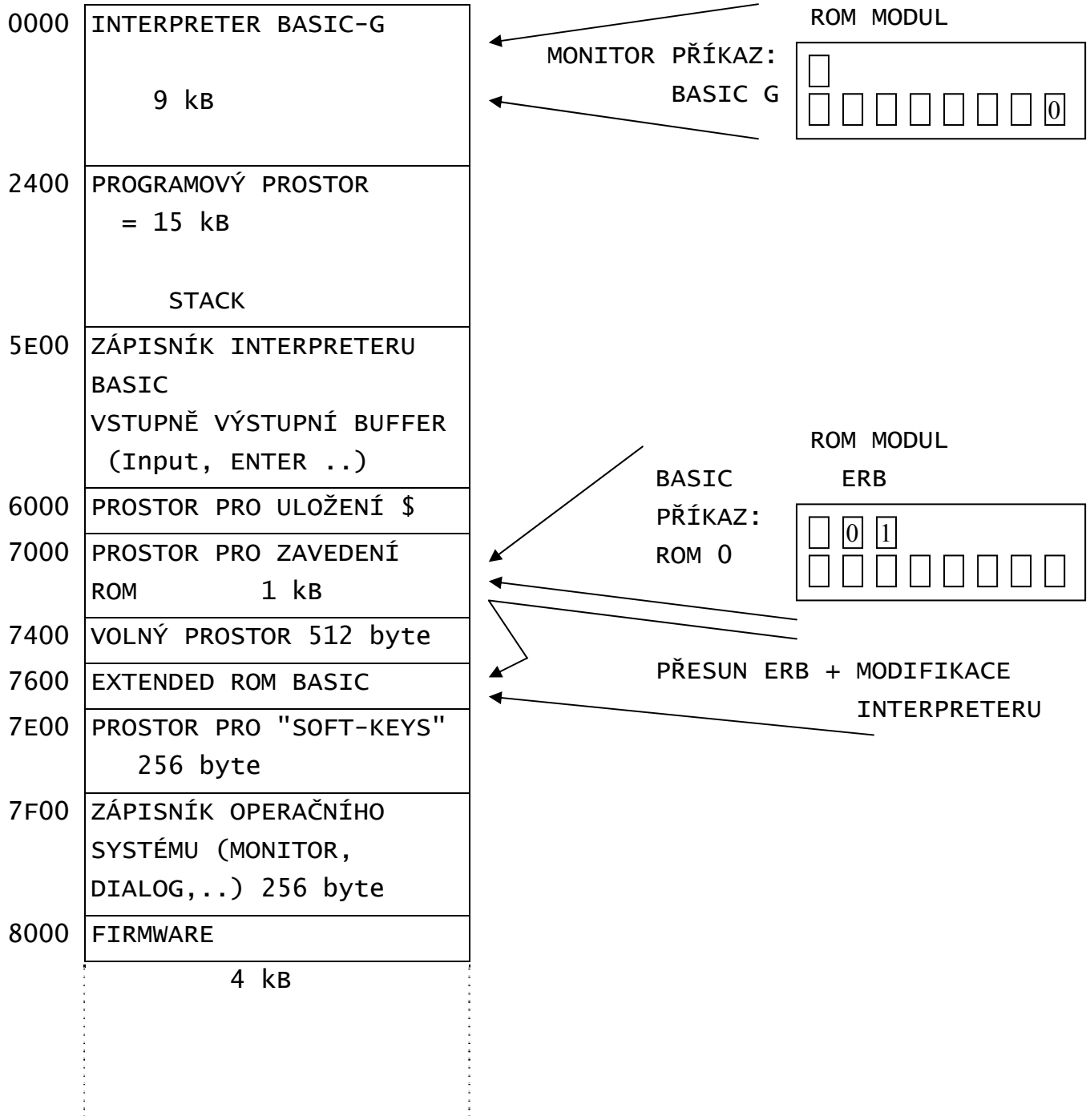
Po zavedení ERB se uskuteční přemodifikování standardního (základního) interpreteru BASIC-G v těch místech, která jsou potřebná pro rozšíření o další příkazy a funkce. Na obr. č. 1 je znázorněna mapa paměti, ve které jsou lokace pro jednotlivé programové moduly a jimi zabraná paměťová místa. Je třeba si uvědomit, že povel ROM uskutečňuje přesun jedné zvolené ROM-ky do pracovní oblasti 7000 - 73FF a jeho interpretování od adresy 7000. Takže tento prostor musí být k dispozici pro uživatelské ROM-ky. Z tohoto důvodu ERB se umísťuje až od adresy 7600 až 7DFF. Za tímto prostorem má uživatel volných 256 byte pro vkládání tzv. soft-keys (klíčů K0 - K11).

Programový prostor BASIC obsahuje jednotlivé "zkompilované" programové řádky a za nimi se nacházejí hodnoty proměnných. V případě, že je dimenzováno pole, vymezí se patřičný prostor pro danou proměnnou. Toto přiřazování je dynamické a závisí na velikosti vlastního programu, charakteristické pro každou dimenzi je, že kromě vlastního záhlaví je pro každý prvek rezervováno 4 byte. Tyto informace jsou nutné z důvodu speciálních příkazů, které budu vysvětlovat.

Aby bylo možné vytvářet moduly ERB i uživatelům, bude v následujících kapitolách uvedena filosofie rozšiřování základního BASICu.

A.I.1 Popis složení programového řádku BASIC

Samotný základní interpreter zabírá v operační paměti prostor od 0000 až 23FF. Hned za ním začíná prostor pro ukládání programových řádků. Obsah těchto řádků však není takový, jaký ho uloží uživatel. Před jeho archivací v tomto tzv. programovém bufferu interpreter provede jeho kompilaci do tzv. mezikódu, aby měl během jeho interpretování ulehčenu práci a samozřejmě, aby se zvýšila rychlost interpretace programu. Kompilace spočívá v nahrazení všech vyhrazených (známých) klíčových slov kódem. Pro tento kód je vyhrazena oblast 80 - FF (HEXA), tedy v podstatě by bylo možné použít maximálně 128 klíčových slov. Tento počet je však nepostačující, když vznikají nároky na speciální příkazy a funkce BASIC. Filosofie programového modulu ERB vychází z toho, že do základního BASIC-u se vloží tzv. semafor (v našem případě je to znak @), který upozorňuje interpreter, že se bude pracovat s druhou větou klíčových slov. Tento semafor se vkládá jen pro příkazy, u funkcí není nutný. V následujícím příkladu na obr. č. 2 je znázorněn převod vstupního uživatelského programového řádku do mezikódu a jeho pozice v programovém bufferu.



Obr. č. 1 - Organizace paměti pro BASIC

Každý programový řádek se skládá z identifikačních údajů, z vlastního těla řádku (příkazy, funkce) a z ukončovacího znaku 00. Identifikační údaje jsou dva, a to 16 bitové slovo udávající adresu následujícího řádku a 16 bitové slovo programového řádku. V případě, že obsah prvního identifikačního údaje je roven 0, považuje interpreter předcházející řádek za poslední a ukončuje svou činnost. Za tímto údajem interpreter ukládá obsahy proměnných a vymezuje si prostor pro dimenzi polí.

Každé pole obsahuje záhlaví, ve kterém jsou umístěny všeobecné údaje. Název pole může být maximálně dvouznakový. Rozdíl mezi řetězcovým polem a polem numerickým je dán nejvyšším bitem prvního byte názvu pole. V případě, že jde o pole řetězcové, jeho hodnota je jednička. Za názvem pole následuje OFFSET, který udává počet byte vyhrazených danou dimenzí, tedy velikost obsazovaného prostoru v programovém bufferu. Potom následuje údaj charakterizující rozměr pole (00 bez rozměru, 01 jednorozměrná atd.). Pro každý rozměr následuje dále údaj o jeho velikosti. V našem případě je to hodnota 0003, která udává, že pole A má tři prvky.

Za záhlavím se nacházejí informace o jednotlivých prvcích. Každý prvek je popsán čtyřmi byte. První byte udává jeho velikost (max. 255 znaků) a poslední dva byte udávají začáteční adresu jeho umístění.

Opačná funkce než při vkládání programových řádků je jejich výpis (LIST, LLIST).

Interpreter musí zabezpečit rekompilaci řádku, t.j. převod klíčového slova z mezikódu do jeho původního tvaru. Tu se opět uplatňuje činnost semaforu (znak @).

Třetí místo v interpreteru, kde se rozhoduje o typu klíčového slova, zaujímá procedura volání příslušné exekutivy na interpretování.

Všechna tato místa, která se modifikují v základním BASICu, jsou znázorněna ve vývojovém diagramu na obr. č. 3. Zdrojový program této procedury, která tvoří záhlaví modulu ERB je uveden

na obr. č. 4, 5. Pro úplnost je uvedena i tabulka klíčových slov druhé interpretační věty a vektory příslušných exekutiv - obr. č. 6.

A.I.2 Stavba vlastních příkazů

Vstupním bodem pro interpretování programového řádku je procedura začínající na adr. 04C6. Jejím úkolem je vyhledat podle mezikódu příslušný vektor exekutivy příkazu a začít s jeho interpretováním. Programový ukazatel BASIC, který ukazuje na znak v programovém řádku, je umístěn v registru HL. Je to důležitá hodnota, protože umožňuje správný návrat do dalšího interpretování programového řádku, popřípadě příkazu v daném řádku. Pokud uživatel bude vytvářet svoje exekutivy bez návaznosti na parametry BASIC, je struktura příkazové exekutivy velmi jednoduchá:

```
PUSH H    ; uchování progr. čítače BASIC
CALL USER ; uživatelská rutina
POP  H    ; obnovení programového čítače
RET       ; návrat do BASIC
```

```
10 DIM A$(2)
20 A$(0) = "TESLA"
30 PRINT A$(0)
40 GOTO 30
```

uživatelský program BASIC

2401:

adresa dal. řádku		číslo řádku		DIM	A	$	(	2	)	END MARKA
0C	24	0A	00	85	41	24	28	32	29	00

240C:

Adresa dal. řádku		číslo řádku		A	$	(	0	)	=	"	T	E	S	L	A	"	END MARKA
1E	24	14	00	41	24	28	30	29	AC	22	54	45	53	4C	41	22	00

241E:

adresa dal. řádku		číslo řádku		PRINT	A	$	(	0	)	END MARKA
29	24	1E	00	96	41	24	28	30	29	00

2429:

adresa dal. řádku		číslo řádku		GOTO	3	0	END MARKA
31	24	28	00	88	33	30	00

2431:

00	00

ukončení programu

zkompilovaný uživatelský program v programovém bufferu

záhlaví pole A$(2)

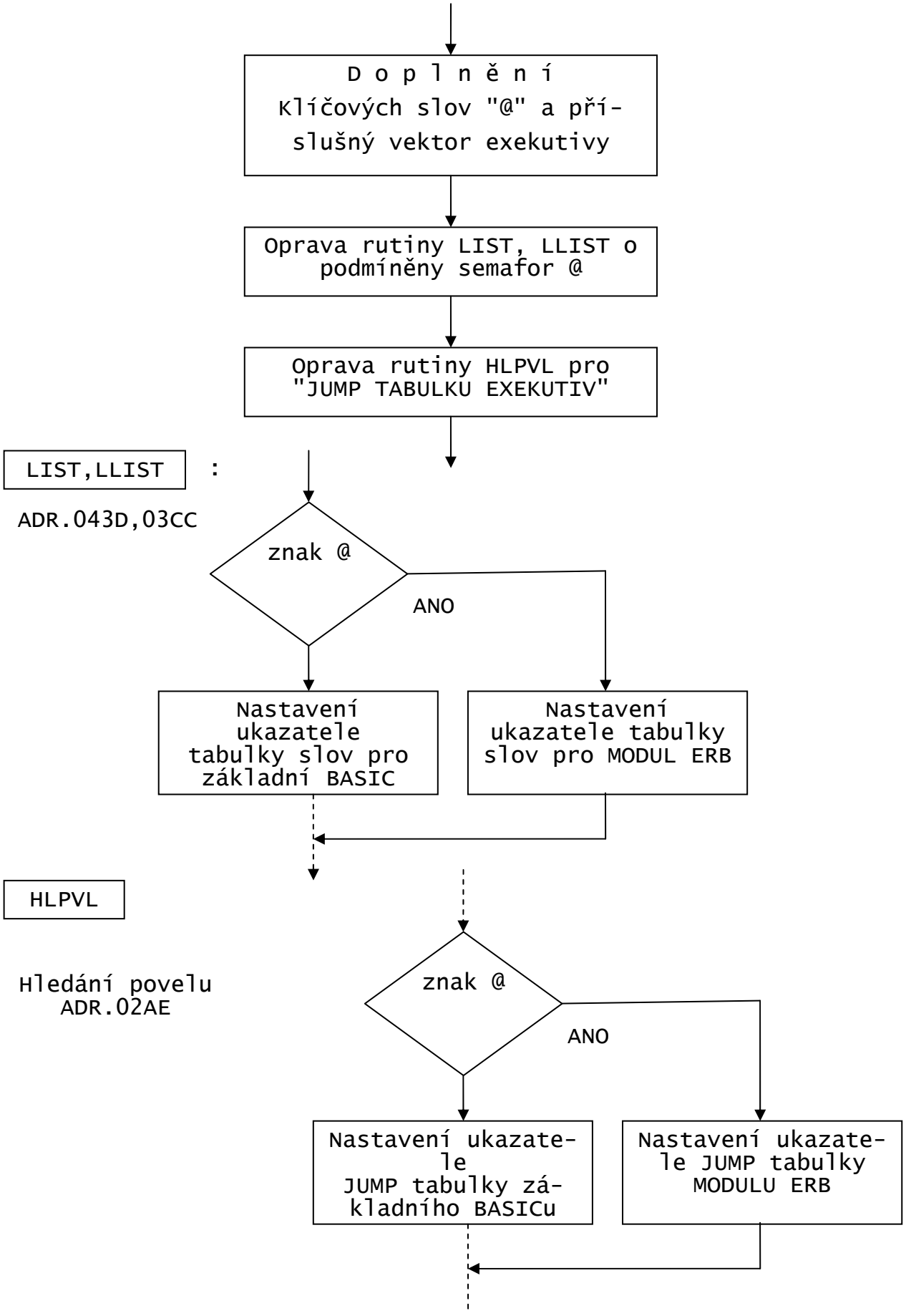
2433:

název pole		OFSET		rozměr	počet prvků	
80	41	0F	00	01	03	00

prvky pole

	délka řetězce	\	adresa řetězce		
			nižší	vyšší	
243A:	05	00	17	24	nultý prvek pole A$
243E:	00	00	00	00	první prvek pole A$
2442:	00	00	00	00	druhý prvek pole A$

Obr. č. 2 - Organizace dimenze pole A$



Obr. č. 3 - Modifikované místo pro rozšíření BASIC

procedura záhlaví ROM0	komentář
PUSH H	uchování prog. čítače BASIC
LXI H,80C0	vložení znaku @ do tab. slov
SHLD 1B60	
LXI H,EXEKUTIVA@ [7667]	vložení exekutivy @
SHLD 19F8	
MVI A,CD	modifikace rutiny HLPVL
STA 02AD	
LXI H,OPR_HLPVL [767A]	
SHLD 02AE	
LXI H,OPR_LIST [7691]	modifikace rutiny LIST, LLIST
SHLD 043D	
SHLD 03CC	
CALL 8C00	transfer modulu ERB
DW 2400	
DW 0900	
DW 7600	
POP H	restaurování program. čítače BASIC
RET	

Exekutiva@ [7667]	komentář
POP D	vyrovnání STACK
ORA A	nastavením FLAGS
JP 00FF	když není příkaz (0), SYNTAX ERROR
PUSH D	
RLC	výpočet OFSETU
ANI FE	
MOV C,A	
MVI B,0	
XCHG	
LXI H,TAB_ERB [77C0]	bázová adresa, tabulka ERB
JMP 04E0	pokračování podle standard BASIC

Obr. č. 4 - Procedury modifikace BASIC

OPR_HLPVL [767A]	komentář
STAX D	vložení původního interpreteru
INX D	
INR C	
CPI E2	porovnej kód E2 ...(@)
PUSH H	uchovej progr. čítač
LXI H,19F9	nastavení původní tab. Ex. BASIC
SHLD 0259	
POP H	restaurování progr. čítače
RNZ	pokud není kód @ - návrat
PUSH H	nastavení tabulky exekutiv pro ERB
LXI H,TAB_EX-1 [76FF]	
SHLD 0259	
POP H	
RET	návrat do původní procedury HLPVL

OPR_LIST [7691]	Komentář
CPI C0	ASCII znak @ + 128
PUSH H	
LXI H,19FA	nastavení původní tabulky výpisu
SHLD 0430	
SHLD 03BF	
POP H	
JNZ 03D9	rutina CONSOLA OUTPUT
PUSH H	nastavení tabulky výpisu slov
LXI H,7700	pro modul ERB
SHLD 0430	
SHLD 03BF	
POP H	
JMP 03D9	rutina CONSOLA OUTPUT

Obr. č. 5 - Procedury modifikace BASIC

tabulka klíčových slov: 7700	tabulka EXEC ERB: 77C0
slovo	vektor exekutivy
CAT	D678
DISK=	AF76
NAME	BA78
PURGE	2A79
INIT	2779
CREATE	6D79
XXXXX	FFFF
CLOSE#	4A7A
CLOSE	F676
PRINT#	BA79
READ#	047A
SECURE	2D79
WRITE (nepoužito)	FFFF
DOKE	4D7D
XXXXXX	FFFF
GSTORE	937B
GLOAD	C17A
BSTORE	AE7B
BLOAD	ED7A
STORE	5D7B
LOAD	F47A
COPY	247C
CMD	9B77
GET (nepoužito)	FFFF
BUF=	607D
OPEN	2E7A
ON ERROR	9B7C
OFF ERROR	AD7C
EXEC	7F7D
SEARCH#	A97D
XX	FFFF
rezerva 2 znaky	rezerva 6

poznámka: xxxx ... neobsazené slovo
vektor exekutivy daný v pořadí LH adr.

Obr. č. 6 - Tabulka klíčových slov a jejich adresy exekutiv

Pokud by však uživatelská rutina vyžadovala z BASIC-u parametry, které jsou parametry daného příkazu, je třeba znát podpůrné rutiny interpreteru BASIC. Jsou to hlavně tyto:

- rutina, která převezme celočíselný parametr v rozsahu 0 až 255. Vektor rutiny je 16D7 (HEXA).
  Vstupním argumentem je ukazatel programového čítače (registr HL) a výstupním argumentem je akumulátor, ve kterém je hodnota parametru příkazu, který může být uveden explicitně nebo implicitně.
- rutina, která převezme celočíselný parametr v rozsahu 16 bitového slova. Pro potřebu uživatele je nutno volat posloupnosti rutin:

```
CALL 0933
CALL 05C3
```

Přičemž opět vstupním argumentem je programový čítač BASIC (reg. HL) a výstupní argument se předává do registrů DE.

Následující příklad objasní blíže použití přecházejících rutin. V příkladu bude použita testovací rutina KZNK s adresou 006D, jejímž úkolem je zjistit, zda požadovaný znak, na který ukazuje programový čítač BASIC je ten, který je uveden za touto rutinou (vstupní argument rutiny KZNK).

V případě, že tento znak je stejný, pokračuje se v interpretování, jinak je výstup přes rutinu SYNTAX ERROR.

Zadání 1: Vytvořte exekutivu k příkazu TRANSFER, který by umožňoval přesun dat mezi definovanými oblastmi paměti. Příkaz by měl mít tři parametry, a to:

- stará adresa bloku dat; X
- délka bloku dat; Y
- nová adresa bloku dat; Z

Požadovaná syntaxe příkazu:
TRANSFER X, Y TO Z

Řešení:

1. krok: je třeba vložit do tabulky klíčových slov nové slovo TRANSFER tak, že první znak v slově (znak T) bude mít nejvyšší bit hodnotu 1, protože tento bit je současně oddělovačem mezi jednotlivými klíčovými slovy. Tedy:

T	R	A	N	S	F	E	R
D4	52	41	4E	53	46	45	52

Pokud by toto slovo bylo poslední v dané větě klíčových slov, je třeba za ním vložit byte 80H, který je identifikačním znakem konce věty.

2. krok: do tabulky exekutiv příkazů je nutno vložit adresu exekutivy TRANSFER. Jako první byte je byte nižší hodnoty.

3. krok: vytvořit exekutivu, která bude realizovat potřebný úkon a bude umístěna v paměti na adrese, která byla zapsána ve druhém kroku do tabulky exekutiv.

```
CALL 0933           Převzetí parametru "X" do
CALL 05C3           registru DE
XCHG
SHLD ODADR          Uložení - stará adresa
XCHG
CALL KZNK           kontrola znaku ","
2C                  znak ASCII ","
CALL 0933           převzetí parametru "Y" do
CALL 05C3           registru DE
XCHG
SHLD DELKA          uložení - délka bloku
XCHG
CALL KZNK           kontrola znaku kódu TO
9E                  kód TO
CALL 0933           převzetí parametru Z do
CALL 05C3           registru DE
XCHG
SHLD KAMADR         uložení - nová adresa
```

```
XCHG
PUSH H              uchování programového čítače BASIC
CALL MOVE           přesun dat
POP  H              návrat programového čítače do HL
RET                 návrat do BASICu.
```

V předcházející části bylo hovořeno o zkompilovaném programovém řádku BASIC. Této struktury může uživatel využít pod řízením svého nového příkazu. Princip spočívá v tom, že uživatelský příkaz si vytvoří pod sebou jednořádkový, jednopříkazový už zkompilovaný řádek BASIC, který bude vnucen proceduře LET. Tato procedura bude vykonávat tento vnitřní "mini BASIC" řádek jako kdyby interpretovala programový řádek z programového bufferu BASIC.

Tento vnitřní mini BASIC řádek umožňuje vykonávat všechny matematické operace a výsledek přiřazuje proměnné, kterou uživatel může zahrnout mezi parametry daného příkazu, nebo pevně vnutit zvolené vnitřní proměnné BASIC.

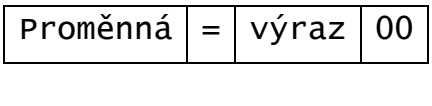
Uživatelský vnitřní mini BASIC řádek může být definován jako pevný (neměnný) řetězec znaků, nebo se může dynamicky sestavovat - doplňovat parametry z daného uživatelského příkazu.

Na obr. č. 7 je znázorněno schéma struktury mini BASIC řádku.

Mini řádek se vytváří stejně jako se sestavuje příkaz LET, pouze místo známých klíčových slov se vkládají kódy. Seznam všech klíčových slov a jejich kódů je uveden v tabulce na obr. č. 8.

Ukončovacím znakem řádku je byte 00. K danému řešení můžeme využít tyto podpůrné rutiny:

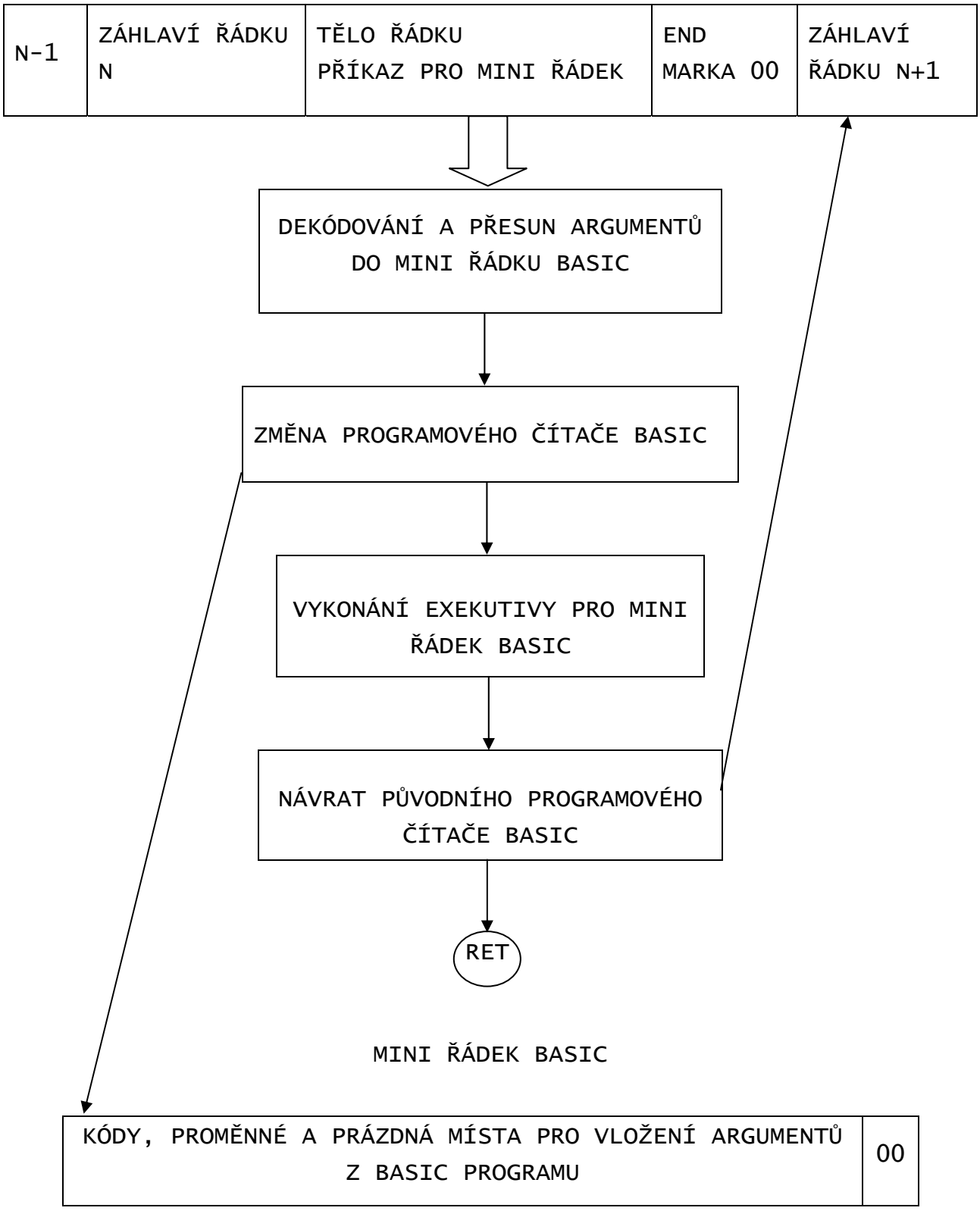
- rutina LET, která uskutečňuje přiřazení výrazu proměnné.
  Vektor rutiny je 06A6.
  Schéma použití rutiny - vzor A.




```
REG HL
PUSH H              uchování progr. čítače BASIC
LXI  H,ŘÁDEK        nastavení nového čítače řádku
CALL LET            vykonání příkazu LET
POP  H              návrat původního progr. čítače
RET                 návrat do BASIC programu.
```

- rutina ANP (analýzy proměnné), jejímž úkolem je najít (popř. vytvořit) pozici proměnné v programovém bufferu BASIC. Vektor rutiny je 0B07. Vstupním argumentem je ukazatel programového čítače BASIC reg. HL a výstupním argumentem je registr DE.

- rutina LETV, která provede přiřazení výsledku výrazu proměnné, která je definována vstupním argumentem v reg. DE.
  Vektor rutiny je 06AD.



TVAR MINI ŘÁDKU NA STRUKTURU PODLE VYKONÁVANÉ EXEKUTIVY.

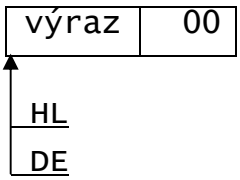
Obr. č. 7 - Schéma struktury začleněné v miniřádku BASIC

kód	příkaz/funkce		kód	příkaz/funkce
80	END		B0	ABS
1	FOR		1	USR
2	NEXT		2	FRE
3	DATA		3	INP
4	INPUT		4	POS
5	DIM		5	SQR
6	READ		6	RND
7	LET		7	LOG
8	GOTO		8	EXP
9	RUN		9	COS
A	IF		A	SIN
B	RESTORE		B	TAN
C	GOSUB		C	ATN
D	RETURN		D	PEEK
E	REM		E	LEN
F	STOP		F	STR$
90	BIT		C0	VAL
1	ON		1	ASC
2	NULL		2	CHR$
3	WAIT		3	LEFT$
4	DEF		4	RIGHT$
5	POKE		5	MID$
6	PRINT		6	SCALE
7	DEEK		7	PLOT
8	LIST		8	MOVE
9	CLEAR		9	BEEP
A	LLIST		A	AXES
B	MONIT		B	GCLEAR
C	NEW		C	PAUSE
D	TAB(		D	DISP
E	TO		E	?
F	FNC		F	BMOVE

kód	příkaz/funkce		kód	příkaz/funkce
A0	SPC(		D0	BPLOT
1	THEN		1	LOAD
2	NOT		2	SAVE
3	STEP		3	DLOAD
4	+		4	DSAVE
5	-		5	LABEL
6	x		6	FILL
7	/		7	AUTO
8	^		8	OUTPUT
9	AND		9	STATUS
A	OR		A	ENTER
B	>		B	CONTROL
C	=		C	CHECK
D	<		D	CONT
E	SGN		E	OUT
F	INT		F	INKEY
			E0	CODE
			1	ROM
			2	@

Obr. č. 8 - Tabulka klíčových slov s příslušným kódem

Schéma použití:




```
CALL ANP      nastavení ukazatele proměnné
PUSH H        uchování progr. čítače BASIC
LXI  H,ŘÁDEK  nastavení nového progr. čítače
CALL LETV     přiřazení výsledku výrazu proměnné
POP  H        návrat původního progr. čítače
RET           návrat do programu BASIC
```

Uvedené parametry si ozřejmíme na následujícím příkladu.

Zadání 2: Vytvořte příkaz CURSOR, který má vložit do svých parametrů X, Y hodnoty pozice "kreslícího pera".

Syntax příkazu: CURSOR X, Y

Matematický vzorec:

X = (PEEK(-16016) - X0)/X3

Y = (PEEK(-16014) - Y0)/Y3

přičemž funkci PEEK(-16016) vybere z paměťové lokace C170 hodnotu horizontální pozice fyzického hodu a funkci PEEK(-16014) vybere z paměťové lokace C172 hodnotu vertikální pozice fyzického hodu, na kterém zůstalo "kreslící pero".

- X0, X3, Y0 a Y3 jsou transformační konstanty z příkazu SCALE.

Řešení: Použijeme schéma vzoru B.

1.krok: Vytvoříme vnitřní dva (pro X a Y) mini BASIC řádky.

řádek X:

(	PEEK	(	-	1	6	0	1	6	)	-	X	0	)	/	X	3	END MARKA
28	BD	28	A5	31	36	30	31	36	29	A5	58	30	29	A7	58	33	00

řádek Y:

(	PEEK	(	-	1	6	0	1	4	)	-	Y	0	)	/	Y	3	END MARKA
28	BD	28	A5	31	36	30	31	34	29	A5	59	30	29	A7	59	33	00

2.krok: Provedeme tytéž úkony s textem CURSOR jako u zadání číslo 1. v prvním a druhém kroku.

3. krok: Sestavíme exekutivu podle vzoru B.

```
CALL ANP            nastavení proměnné X
PUSH H              uchování programového čítače BASIC
LXI  H,ŘÁDEK_X      nastavení nového progr. čítače
CALL LETV           přiřazení výrazu proměnné X
POP  H              návrat původního progr. čítače
CALL KZNK
2C                  test syntaxe (čárka)
```

```
CALL ANP              jako u souřadnice X
PUSH H
LXI  H,ŘÁDEK_Y
CALL LETV
POP  H                návrat původního progr. čítače
RET                   návrat do progr. BASIC
```

Tato filosofie vykonávání vnitřního BASIC řádku se dá výhodně použít i při vstupní funkci, když je požadován vstup údajů do proměnné. Programová struktura je obdobná. Předpokládejme, že potřebujeme přiřadit hodnotu měřené veličiny přístroje, který nemá standardní sběrnicí (např. voltmetr MT 100).

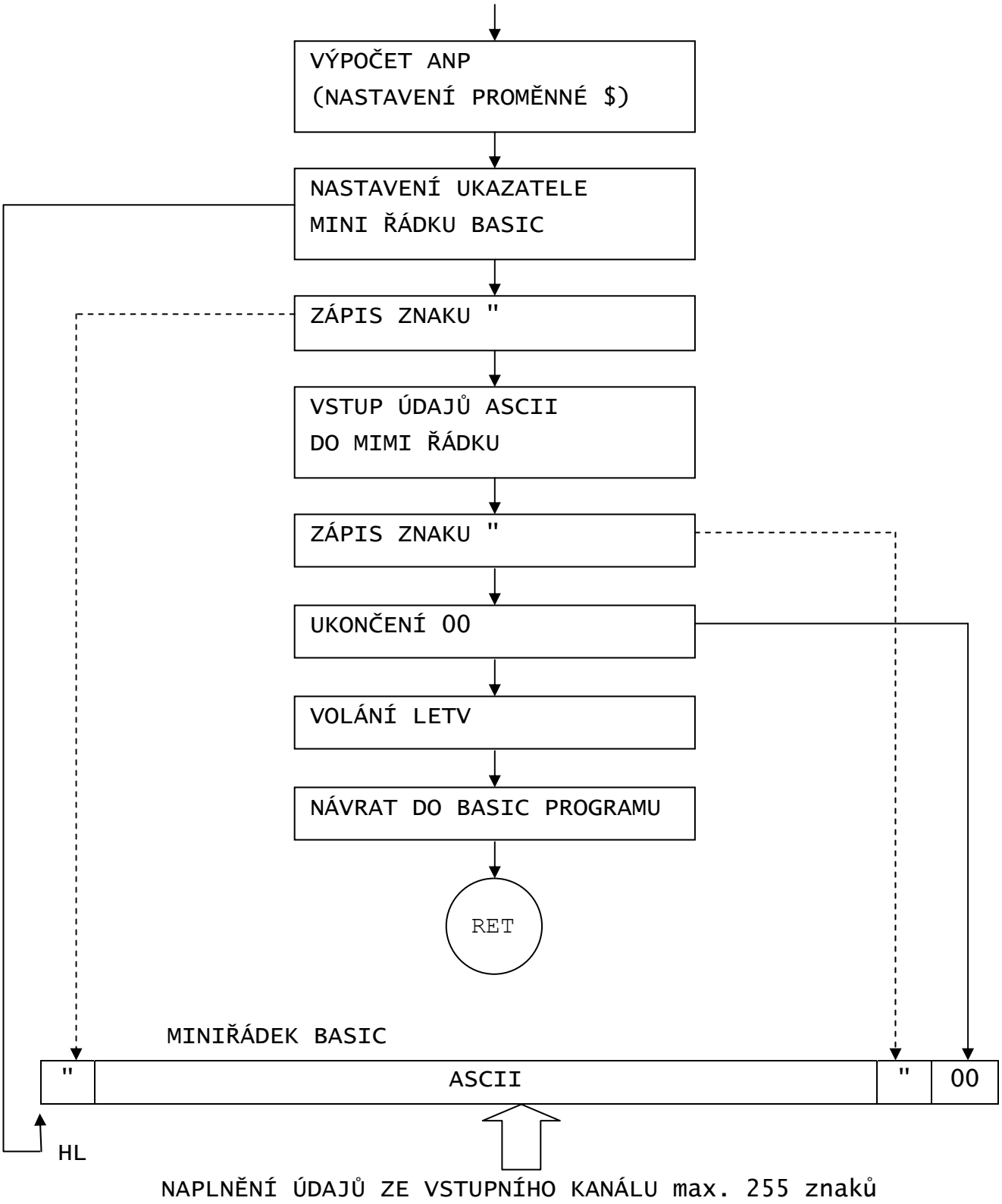
Údaje z přístroje jsou ve formě BCD. Naším úkolem bude vytvořit exekutivu, která tyto údaje z přístroje převezme, převede do ASCII tvaru a uloží na vyhrazené paměťové místo, které bude představovat vnitřní mini řádek BASIC. Řádek uzavře byte 00. Potom stačí vytvořit podle předcházejícího konkrétní uživatelský příkaz pro snímání údajů z měřeného přístroje.

Je-li požadavek přenosu ASCII dat do řetězcové proměnné, je třeba tento blok dat uzavřít do uvozovek. Na obr. č. 9 je vývojový diagram činnosti využívání mini řádku BASIC jako vstupního bufferu.

Doposud bylo vysvětlováno, jak lze tvořit nové uživatelské příkazy. Protože vlastní interpreter i modul ERB se nacházejí v operační paměti RAM, je možné je modifikovat standardními příkazy POKE, @DOKE, CODE. Princip spočívá v tom, že konstanta, která je v začátku inicializována, se během interpretování programu modifikuje. Používání této metody bude uvedeno v samostatné části uživatelské příručky. Zde vysvětlíme pouze jeden příklad, ze kterého bude zjevný způsob modifikace.

Zadání 3: Modifikujte výstupní rutinu VOC, která zabezpečuje ošetření výstupního znaku.

```
03D9 F5        PUSH PSW
03DA FE20      CPI  20
03DC DAEB03    JC   03EB
03DF 3A015E    LDA  5E01
03E2 FE3E      CPI  3E
```



Obr. č. 9 - Vývojový diagram miniřádku jako vstupního bufferu ASCII znaků

```
03E4 CCA807     CZ   07A8
03E7 3C         INR  A
03E8 32015E     STA  5E01
03EB F1         POP  PSW
03EC F5         PUSH PSW
03ED C5         PUSH B
03EE E67F       ANI  7F
03F0 4E         MOV  C,A
03F1 CD1400     CALL 0014
03F4 C1         POP  B
03F5 F1         POP  PSW
03F6 C9         RET
```

Seznam důležitých adres:

- 03E0 ukazatel místa, kde je umístěn aktuální počet znaků
  03E1 (kursor)
- 03E3 (decimálně 995) konstanta udávající logickou zarážku konce řádku. Po ztotožnění s aktuální hodnotou na adrese 5E01 dojde k automatickému generování CRLF (nový řádek)
- 03E4 (decimálně 996) - kód instrukce CZ pro volání podprogramu v závislosti na stanoveném bitu ZERO - kód CC (decimálně 204). Když tento kód nahradíme nulou, nedojde k vygenerování CRLF při dosažení logického konce řádku.
- 03EE (decimálně 1006) - kód instrukce ANI, která maskuje obsah akumulátoru (výstupní znak) s konstantou 7F umístěnou na následující paměťové buňce. Změnou tohoto kódu E6 (decimálně 230) na hodnotu 00 nedojde k vymaskování nejvyššího bitu výstupního znaku.
- 03F2 (decimálně 1010) - ukazatel výstupní rutiny pro výstup znaku.
  03F3 Změnou obsahu na těchto buňkách můžeme měnit typ výstupní rutiny.

Praktické ukázky:

```
POKE 995,10       ;nastavení logické zarážky na deset znaků na řádek
POKE 996,0        ;zrušení kontroly logické zarážky řádku
@DOKE 1010,A      ;změna výstupního zařízení na zařízení určené vekto-
                   rem A
```

Záleží jen na uživateli, jak požaduje změnit danou výstupní rutinu. Tento způsob je velmi efektivní, protože nepožaduje rozšiřovat a vytvářet nové příkazy, avšak podmínkou je znalost vlastního interpretem BASICu.

II. Popis příkazů a funkcí EXTENDED ROM BASIC

V této kapitole budou jednotlivě probrány nové příkazy a funkce BASICu, které uživatel získá zavedením modulu ERB. V podstatě můžeme tyto nové příkazy a funkce rozdělit do dvou základních skupin. Do první skupiny zahrnujeme ty, které rozšiřují původní grafický BASIC o výkonnější příkazy a funkce a v druhé skupině jsou příkazy pracující s inteligentní minifloppy diskovou jednotkou MFD-85.

V tabulce na obr. č. 10 jsou uvedeny ve stručnosti obě skupiny.

Příkaz/funkce		Význam
SK.I	TAB(	Tabulátor X,Y
	@DOKE	Zápis do paměti - 16 bitové číslo
	DEEK	Čtení z paměti - 16 bitové číslo
	@BUF=	Definováni bufferu
	@	Převod HEXA/DECIMAL
	POS	Pozice proměnné v programovém bufferu BASIC
	@EXEC	Vykonávání řetězců jako BASIC příkazy
	CODE	Vykonávání řetězců jako binární podprogram
	POKE	Zápis do paměti - 8 bitové číslo
SK.II	@CAT	Výpis katalogu diskety
	@DISK=	Přiřazení zařízení
	@NAME	Výpis názvu diskety
	@PURGE	Vymazání souboru z diskety
	@INIT	Inicializace
	@CREATE	Vytvoření logického souboru katalogu
	@CLOSE#	Uzavření souboru
	@PRINT#	Zápis do zvoleného logického záznamu
	@READ#	Čtení ze zvoleného logického záznamu
	@SECURE	Utajení souboru v katalogu
	@GSTORE	Zápis video RAM na disketu
	@GLOAD	Zápis do video RAM
	@STORE	Uložení programu BASIC na disketu
	@LOAD	Uložení programu z diskety do RAM
	@BSTORE	Binární program zápis
	@BLOAD	Binární program čtení
	@OPEN	Otevření datového souboru
	@ON ERROR	Přepínání zpracování chyby
	@OFF ERROR	- || -
	@CLOSE	Uzavření všech souborů
	@SEARCH#	Vyhledávání v logickém souboru

Obr. č. 10 - Tabulka příkazů a funkcí v modulu ERB

II.1 Popis a použití příkazů skupiny I

Úvodem je třeba upozornit, že v této skupině se nacházejí i některé už stávající příkazy a funkce a proto je není třeba označovat semaforem @.

POKE

Upravený původní příkaz o možnost vícenásobného ukládání celočíselné hodnoty výrazu v rozsahu 0 - 255 od zvolené adresy, přičemž oddělovač mezi argumenty inkrementuje aktuální adresu.

Syntax:

```
POKE ADR, BYTE 1 [,BYTE 2, BYTE 3 ...]
```

kde ADR je počáteční adresa pro první BYTE 1, který se má zapisovat na toto paměťové místo. V případě, že požadujeme zapsat na následující adresu ADR + 1, použijeme oddělovač čárku.

Příklad: POKE 10000, 255, 0, 128, 10
umožní uložit do paměťové oblasti následující data:

adresa	data
10000	255
10001	0
10002	128
10003	10

Výhoda tohoto příkazu je hlavně při vkládání binárního programu do paměťové oblasti, popřípadě dat do video-RAM.

@DOKE

Nový příkaz umožňující zápis 16bitového čísla na zvolenou paměťovou buňku.

Syntax:

```
@DOKE ADR, WORD
```

kde ADR určuje paměťové místo, WORD je celočíselná hodnota výrazu v rozsahu <-32 768, +32767> (16 bitů)

Příklad: @DOKE 10000, 1023
umožní vložit do paměťové buňky 10000 číslo 1023, tedy na adresu 10000 se zapíše hodnota 255 a na adresu 10001 hodnota 3.

@

Nová funkce, která umožňuje vkládat explicitně číslo v hexadecimálním tvaru. Tvar hexadecimálního čísla musí být čtyřznakový. Mezi znakem @ a vlastním číslem nesmí být mezera. Doporučuji používat pouze v přímém módu (ničí text v příkaze LIST).

Příklad: POKE @7000, 255 ... do buňky 7000 (hexa) se zapíše byte FF (hexa)
POKE @7000, @00C0 ... do buňky 7000 (hexa) se zapíše byte C0 (hexa)

DEEK

Nová funkce, která je svým posláním podobná jako funkce PEEK. Rozdíl je pouze v tom, že funkce DEEK pracuje se 16bitovou hodnotou.

Syntax:

```
A = DEEK  ADR
```

kde ADR představuje konkrétně paměťové místo a tedy může mít jen celočíselnou hodnotu v rozsahu 16 bitů.

Příklad: PRINT DEEK @7000
Ve vykonávání tohoto příkazu se vytiskne obsah 16 bitové hodnoty umístěné na adresách 7000 a 7001 (hexa).

@BUF=

Nový příkaz umožňující nastavit dvě důležité paměťové místa (77BE, 77BC) pro všeobecné použití (hlavně u diskových služeb). První hodnota udává počáteční adresu bufferu a druhá určuje počet, popřípadě konečnou adresu bufferu.

Syntax:

```
@BUF= ADR1, ADR2
```

kde ADR1, ADR2 mají obdobný rozsah jako u předcházejících příkazů.

Příklad: @BUF= @7000, @0100
provede nastavení parametrů takto:

Požadovaná	77BE	00
adresa	77BF	70
délka	77BC	00
bufferu	77BD	01

Poznámka: Při použití diskových služeb se obsahy těchto paměťových buněk mění.

CODE

Příkaz umožňující zahnízdit vykonávání binárních (ve strojovém kódu) programů pod programem BASIC. Tento příkaz má dvě možné syntaxi. První je obsažena v základním BASICu. Pro lepší pochopení činnosti zopakujeme její podstatu.

Syntax 1: CODE A$ [, A$ ...]

kde A$ představuje řetězcovou proměnnou, popřípadě prvek z řetězcového pole. Binární program je uložen ve formě hexa-znaků v příslušném prvku, nebo řetězcové proměnné. Binární program má charakter podprogramu, proto jeho návrat zpět do programu BASIC je vykonáním instrukce RET (C9), popřípadě jeho podmínkových návratových instrukcí (RZ atd.). Při interpretování příkazu CODE probíhají tyto fáze:

a) výčet adresy příslušné řetězcové proměnné
b) přesun dat při současném kontrolování hexa-znaku a převodu z místa řetězce do bufferu na jejich interpretování ve strojovém kódu. Počáteční adresa bufferu je 7F00 hexa.
c) vykonání podprogramu nad obsahem bufferu
d) testování pokračování v příkazu CODE

V případě, že binární program vyžaduje podmíněné

skoky, je možno tyto adresovat na stránce 7F00. Původní příkaz CODE v bodech B, C se dost omezuje v daném výkonu a proto tento nedostatek odstraňuje modul ERB svým doplněním.

Syntax 2:

```
CODE (B, C) A$ [, A$ ...]
```

kde - "B" určuje počáteční 16bitovou adresu, kam se mají data z řetězce přenášet, tedy určuje počáteční adresu bufferu (fáze B)
"C" znamená 16bitový vektor podprogramu (adresu prvého interpretovaného binárního kódu), který má vykonávat (fáze C)
A$ - řetězcová proměnná (prvek), ve kterém se nacházejí binární data v tvaru hexa.

Z předcházejícího je zřejmé, že uživatel může použít příkaz ve dvou formách, v původním tedy s konstantními argumenty ve fázích B, C (adresa 7F00), nebo si je může modifikovat.

V zásadě lze předepsat, že první způsob syntaxi (CODE A$) je výhodný pro interpretování podprogramů ve strojovém kódu a uživateli nezáleží, kde se bude tento binární program interpretovat. Druhý způsob poskytuje velké služby v tom, že se může použít:

1. - jako původní příkaz, kde se vykonává binární program na zvoleném místě, které je určeno adresami B,C
2. - pro přesun obsahu řetězce do zvolené oblasti určené adresou B, přičemž obsah řetězce nemusí být binárním programem, ale data hexa. Přičemž mohou nastat následující dvě varianty:

2a) vykoná se jiný uživatelský podprogram na adrese C

2b) nevykoná se žádný podprogram. V tomto případě uživatel musí vnutit uvedené fázi C skok na adresu s kódem RET. Může se použít adresa 144 decimálně.

Varianta 2b se výhodně používá pro zkomponování většího bloku dat, které v podstatě mohou představovat binární podprogram. Tehdy je aktivujeme příkazem USR.

Další využití najde uživatel v použití této varianty při vytváření obrazců (přesun dat do video RAM).

Příklad:
```
10 A$ = "3E41C30085"
20 CODE A$
30 CODE (@7000, @7000) A$, A$
40 CODE (@D300, 144) A$
50 CODE (@D500, @88A3) A$
```

Komentář:

10 naplnění řetězce A$ podprogramem v hexatvaru, který má za úkol vytisknout znak A do pracovní části obrazovky

20 vykonání binárního podprogramu, který je obsažen v řetězci A$

30 vykonání dvou po sobě jdoucích binárních podprogramů A$. Každý podprogram se interpretuje v oblasti 7000 HEXA.

40 Přesun obsahu řetězce A$ do video RAM na adresu D300 HEXA, přičemž se nevykoná žádný úkon s daty (okamžitý návrat z adresy 144 decimálně)

50 obdobně jako u řádku 40, ale po přenosu dat z řetězce A$ do oblasti D500 HEXA se vykoná podprogram na adrese 88A3 HEXA (BELL)

TAB

Příkaz nahrazuje původní příkaz TAB(I), který sloužil jako horizontální tabulátor pod příkazem PRINT, DISP. Tento původní příkaz je nutné nahradit příkazem SPC (I), který provede tutéž funkci. V modulu ERB příkaz TAB (X,Y) nabývá novou formu. Umožňuje horizontální i vertikální pohyb zapisovacího místa v příkazu PRINT. Výpočet místa zápisu je s podporou rutiny BMOVE a proto je nutno při vzájemném použití s příkazem BPLOT brát toto v úvahu.

Syntax: `TAB(X,Y)`

kde X, Y mají tentýž význam jako u příkazu BMOVE

Mohou být v tomto rozsahu:
X ... 0 až 47
Y ... 0 až 242

Je třeba ještě upozornit na tu skutečnost, že při každém standardním vykonaném příkazu PRINT dochází ke generování CRLF, a tedy k posunutí obrazu. Toto generování je možno odstavit středníkem na konci příkazového řádku PRINT. Dále je nutno zablokovat kontrolu logické zarážky konce řádku, protože by mohlo dojít ke generování znaků CRLF. Toto se provede příkazem POKE 996,0. Pozice kurzoru je totožná s fyzickým kurzorem video RAM. Následující obrázek znázorňuje jeho parametry vzhledem k pracovní oblasti video RAM.

Poznámka: Návrat původní funkce TAB(I) je umožněn
@DOKE 1880, 2007 (decimální hodnoty)
Změna pro TAB(X,Y):
@DOKE 1880, 31956 (decimální hodnoty)

Použití: Tento nový způsob tabulátoru umožňuje velmi efektivně pracovat nad celým pracovním prostorem obrazovky. Je zvlášť výhodný pro sestavování tabulek, dále pro rychlý výpis textů a taktéž pro rychlý výpis textů a pro psaní znaků jako indexů.

POS

Toto klíčové slovo se dekóduje jako funkce, přičemž původní význam v základním BASICu byl zrušen a proto počet znaků, které byly vydány přes PRINT, lze

získat funkcí:

PEEK (@5E01)

Aby bylo možné pochopit význam této nové funkce, která se nenachází na standardním interpreteru BASIC, je nutno vysvětlit, proč vlastně vznikla. Uživatel má k dispozici přibližně 15 kB programový buffer. Povětšině tento prostor vystačuje a zůstává přibližně 5-6 kB volného prostoru. Přístup k tomuto prostoru je dost problematický, protože je pod neustálým pohybem ze strany BASICu. Vymezit prostor v programovém bufferu BASIC je možno výhodně s podporou příkazu DIM.

Velikost DIM byla vysvětlena v kapitole I. Jednoduché pravidlo pro určení rozsahu vymezeného prostoru je, že každý prvek proměnné zabírá 4 byte. Tedy např.:

DIM A (5, 2)

nám vymezí prostor za programem BASIC v počtu 6 x 3 x 4 = 72 byte. (Nepočítáme záhlaví pole!)

nebo DIM A$(1000)

dimenzuje prostor v rozsahu 1001 x 4 = 4004 byte. Aby bylo možno přistoupit k jednotlivým prvkům pole (ne k jeho obsahu!), je třeba znát jeho absolutní adresu, která byla vytvořena příkazem DIM. Tuto adresu nelze získat běžnými funkcemi nebo příkazy interpreteru a proto bylo nutné vytvořit novou funkci POS (pozice).

POS A$

kde A$ představuje proměnnou, řetězec proměnné nebo prvek pole. Většinou se funkce POS bude užívat pro pole, i když ji lze rovněž určit pro přístup k proměnné nebo řetězcové proměnné.
Jak již bylo uvedeno, funkce POS umožňuje pracovat nad parametry dimenze u řetězcových proměnných a nad obsahem proměnných u polí numerických proměnných. Podpůrnými příkazy a funkcemi pro tuto činnost jsou

POKE @DOKE
PEEK DEEK

Ze strany BASIC je obsah dimenze modifikován pouze přiřazovacím příkazem LET (=) a destruktivním příkazem DIM. Ostatní příkazy, funkce mají pouze pasivní vztah (využívají argumenty) k obsahu dimenze.

Poznámka: Dimenzováním polí, proměnných dochází ke změnám stávajících dimenzí. Každá proměnná se automaticky umísťuje do prostoru za programem BASIC. Doporučujeme pro kritickou fázi programu provést dimenze všech proměnných, které by mohly poškodit pracovní prostor vymezený dimenzováním pole.

S podporou funkce POS lze vytvořit i jinou programovou strukturu příkazů, které budou modifikovat obsah dané dimenze pro další použití standardními příkazy pro práci s obsahy polí.

Například:

- numerické pole naplníme jako blok dat, které však reprezentují u jednotlivých prvků podle konkrétní hodnoty, se kterou později pracujeme jako kdyby byly vytvořeny standardními prvky (jednotlivě každý prvek!)
- odevzdávání obsahů numerických proměnných popřípadě obsahů řetězcových proměnných mezi interpreterem BASIC a binárním programem a naopak. Programu BASIC lze vnutit obsahy řetězcových proměnných, které mohou být uloženy mimo prostor BASIC (i z paměti ROM).

Specifickým případem může být využití dimenze jako bufferu pro uložení konstant, tabulek, ale i binárních programů. S podporou příkazu BUF= lze vymezit prostor, kam se budou zapisovat binární data z diskové jednotky (@BLOAD).

K výše uvedeným úvahám několik praktických příkladů, ze kterých bude zřejmé, jakou podporu lze očekávat od funkce POS.

Příklady:

```
DIM A$(100)                     ;vymezení prostoru 101 x 4 byte
PRINT POS A$(0)                 ;tisk adresy prvního prvku pole A$
PRINT POS A$(100) - POS A$(0)   ;výpočet velikosti dimenze
@BUF=POS A$(0), 400             ;nastavení parametrů BUFFERU pro
                                 práci s binárními daty z disku
PRINT DEEK POS A$(0) + 2        ;tisk ukazatele adresy pro první
                                 prvek z pole A$
PRINT PEEK (POS A$(0))          ;tisk velikosti prvého prvku pole A$
@DOKE POS A$(100) + 2, @7000    ;nastavení ukazatele adresy na 7000
                                 Hexa u posledního prvku pole A$
A=USR (POS A$(0))               ;vyvolání binárního podprogramu,
                                 který je uložen v dimenzi pole A$.
                                 Startovací adresa začíná na adrese
                                 prvního prvku pole A$.
```

Příklady:

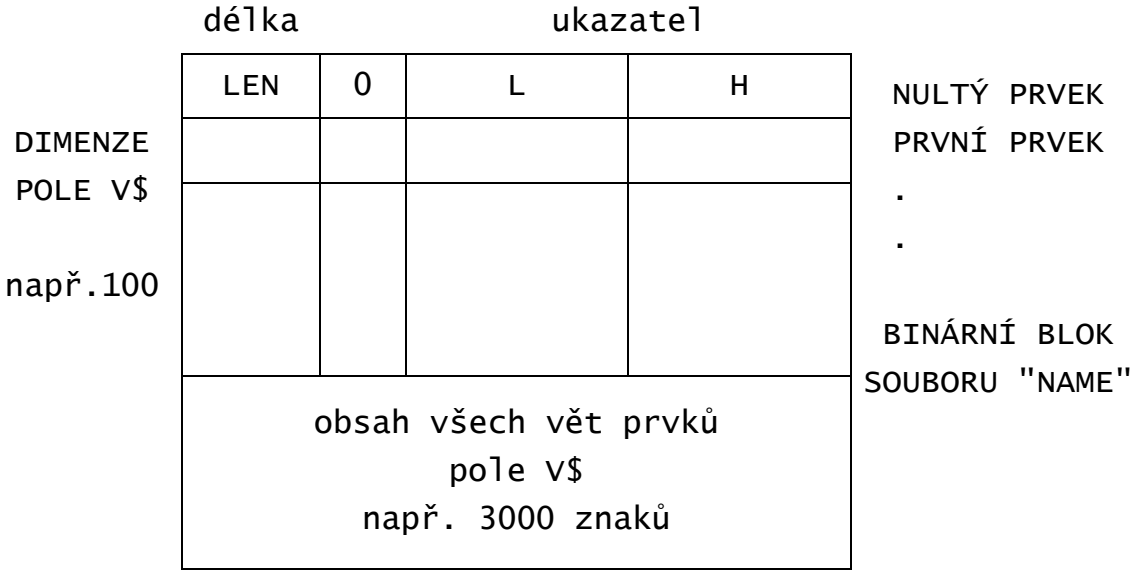
```
          A=DEEK POS A$(0) + 2
          @DOKE POS A$(0) + 2, DEEK POS A$(100) + 2
          @DOKE POS A$(100) + 2, A
        - přehození dvou prvků 0 a 100 v poli A$

          @BUF=POS A$(0), 200
          @BLOAD "NAME"
          A=USR (POS A$(2))
        - zavedení binárního programu z disku a jeho vykonání

          DIM A(500)
          @BUF=POS A(0), 2004
          @BLOAD "MATICE"
          PRINT A(0)
        - naplnění numerického pole A z disku
```

Zadání úkolu č.4: Vytvořte programový segment pro prohledávání a práci s konstantními větami řetězcového pole V$ ve spolupráci s diskovou jednotkou.

Předpoklad: Dané řetězcové pole bylo archivováno na disketu ve formě binárního bloku dat. Tento blok dat byl vytvořen jako vlastní dimenze. Obsah jednotlivých vět prvků viz následující obrázek:



celkový počet byte v bloku = 101 x 4 + 3000 = 3404

Řešení:

```
  5 POKE 996,0
 10 DIM V$(3404/4)
 20 @BUF=POS V$(0), 3404
 30 @BLOAD "NAME"
 40 FOR I = 0 to 100 STEP 24
 50 FOR J = 1 TO 24
 60 PRINT TAB(2, J * 10); V$(J + I - 1)
 70 NEXT J
 80 ?
 90 GCLEAR
100 NEXT I
110 POKE 996, 204
120 PRINT
```

```
Komentář:    5 zrušení logické zarážky pro výpis znaků
            10 dimenzování pole V$
            20 nastavení parametrů BUFFERU
            30 naplnění BUFFERU dimenzí a obsahy prvků pole V$
            40 až 100 výpis obsahů prvků pole V$
            60 použití tabulátoru
           110 nastavení logické zarážky pro výpis znaků
           120 rolování video RAM (obrázek)
```

Předložené řešení zadání má nevýhodu v tom, že daná dimenze je spojena s obsahem vět jednotlivých prvků pole. Tato koncepce byla zvolena pouze z cvičných důvodů. V praxi je výhodné oddělit dimenzi od obsahu prvků pole. Potom programový segment bude obsahovat v inicializované části zavedení pouze samotné dimenze pole s pevnými ukazateli adres u jednotlivých prvků pole, které budou definovány do pracovní oblasti řetězců (oblast 6000 - 6E00 Hexa).

V této oblasti se budou nacházet všechny věty daného souboru. Soubory lze měnit s podporou diskové služby, přičemž dimenze pole V$ bude konstantní pro všechny soubory.

Obdobnou filosofií jako pro řetězcová pole můžeme přenášet z disku do zvolené dimenze celé numerické matice prvků.

Částečnou náhradou diskové jednotky může být magnetofon nebo, pokud se jedná o konstantní soubory, uživatelské ROMky 2 až 6. V následující kapitole vysvětlím, jak může uživatel modifikovat příkaz ROM pro různé použití.

Modifikace příkazu ROM

Úlohou základního příkazu ROM je přesunout obsah definované uživatelské ROMky do oblasti 7000 Hexa a jeho interpretace od adresy 7000 Hexa. V příkazu ROM se nacházejí tři důležitá místa, která jsou inicializována pro uvedenou oblast, ale uživatel ji může modifikovat. Jsou to:

pol.	adresa		inicializováno hodnotou		význam
	Hexa	Dec.	Hexa	Dec.	
1.	2334	9012	0500	1280	udává počet byte, který se má přenášet
2.	2336	9014	7000	28672	parametr udávající paměťovou oblast kam se budou data ukládat
3.	233A	9018	7000	28672	adresa buňky, na které se bude pokračovat v interpretování příkazu ROM

Poznámky: U položky 1. je nutné brát v úvahu skutečnost, že testování počtu přenesených byte je na vyšším byte počítadla, a proto je třeba ho zvýšit o 256 byte. Např. pro rozsah 1 kB je parametr počtu nastaven na hodnotu 1280 (0500) Hexa.
Položka 3. udává pokračování i interpretaci příkazu ROM, proto mohou nastat tyto varianty:

a) ukončení příkazu ROM, tehdy je třeba vložit vektor adresy návratu RET např. 144 decimálně
b) bude se pokračovat podprogramem, který byl přenesen danou ROMkou do specifikované oblasti (položka 2)
c) bude se pokračovat jiným podprogramem ze systému

Pro případy b, c platí společná zásada, že je nutno volat pouze podprogramy, které jsou zakončeny RET, nebo jeho podmíněnými kódy (RZ atd.) a nesmí být zničen registr HL, protože jeho prostřednictvím se binární podprogram vrací do interpretování programu BASIC.

Modifikaci příkazu ROM si prakticky ukážeme na následujícím příkladě, který navazuje na aplikaci funkce POS:

Zadání č. 5: Vytvořte programový segment, který by umožňoval používat jinou vstupní tabulku znaků (např. malá písmena, semigrafické znaky). Symboly znaků jsou umístěny v uživatelské ROMce č. 2.

Řešení: Přeneseme obsah tabulky znaků z ROM 2 do vyhrazeného paměťového prostoru a přemodifikujeme systémový znakový ukazatel.

```
10 DIM T$(255)
20 @DOKE @2336, POS T$(0)
30 @DOKE @233A, 144
40 ROM 2
50 @DOKE @C03C, POS T$(0)
60 PRINT "malá písmena"
```

Komentář:
10 dimenzování prostoru v rozsahu 1 kB
20 modifikace ukládací adresy v příkazu ROM
30 modifikace ukončení příkazu ROM
40 transfer dat z ROM 2 do oblasti dimenze pole T$
50 modifikace systémového ukazatele výstupních znaků ASCII
60 tisk znaků z uživatelského obsahu ROM 2

Použití funkce POS umožňuje vnikat do vnitřního řízení interpreteru BASIC a ovlivnění výsledků funkcí a příkazu u řetězcových polí.

Například, pokud byla vytvořena dimenze řetězcového pole a naplněn jeho obsah, lze přistoupit s podporou funkce POS k parametru délky řetězcového prvku (funkce LEN) a jeho modifikací zpřístupňovat obsah řetězcového prvku.

Příklad:

```
10 DIM A$(10)
20 A$(0)="TESLA PIEŠŤANY"
30 PRINT LEN(A$(0))
40 PRINT A$(0)
50 POKE POS A$(0),0
60 PRINT A$(0)
70 POKE POS A$(0),5
80 PRINT A$(0)
```

Komentář:
10 dimenzování řetězcového pole A$
20 naplnění prvku pole A$
30 tisk velikosti prvku A$(0)
40 tisk obsahu prvku A$(0)
50 modifikace délky prvku A$(0) na hodnotu 0

```
60 prázdný tisk obsahu prvku A$(0)
70 modifikace délky prvku A$(0) na hodnotu 5
80 tisk obsahu prvku A$(0) v rozsahu 5 znaků t.j. TESLA
```

@EXEC

Tento příkaz přináší uživateli kvalitativně nové BASIC programové struktury. Kdybychom chtěli vystihnout jednou větou jeho definici, měla by text: "BASIC v BASICu". Jestliže uživatel pracoval s příkazem CODE, tak s jeho podporou zahnízďoval do programu BASIC binární podprogramy. Příkaz @EXEC umožňuje se stejnou filosofií zahnízďovat BASIC příkazy a funkce. Nosiči těchto příkazů (funkcí) jsou opět jako u příkazu CODE řetězcové proměnné. Vytvoření řetězcové proměnné může být explicitní (přímo) nebo implicitní s podporou řetězcových funkcí (MID$, LEFT$, RIGHT$). Na obr. č. 11 je uveden vztah příkazu k BASICu.

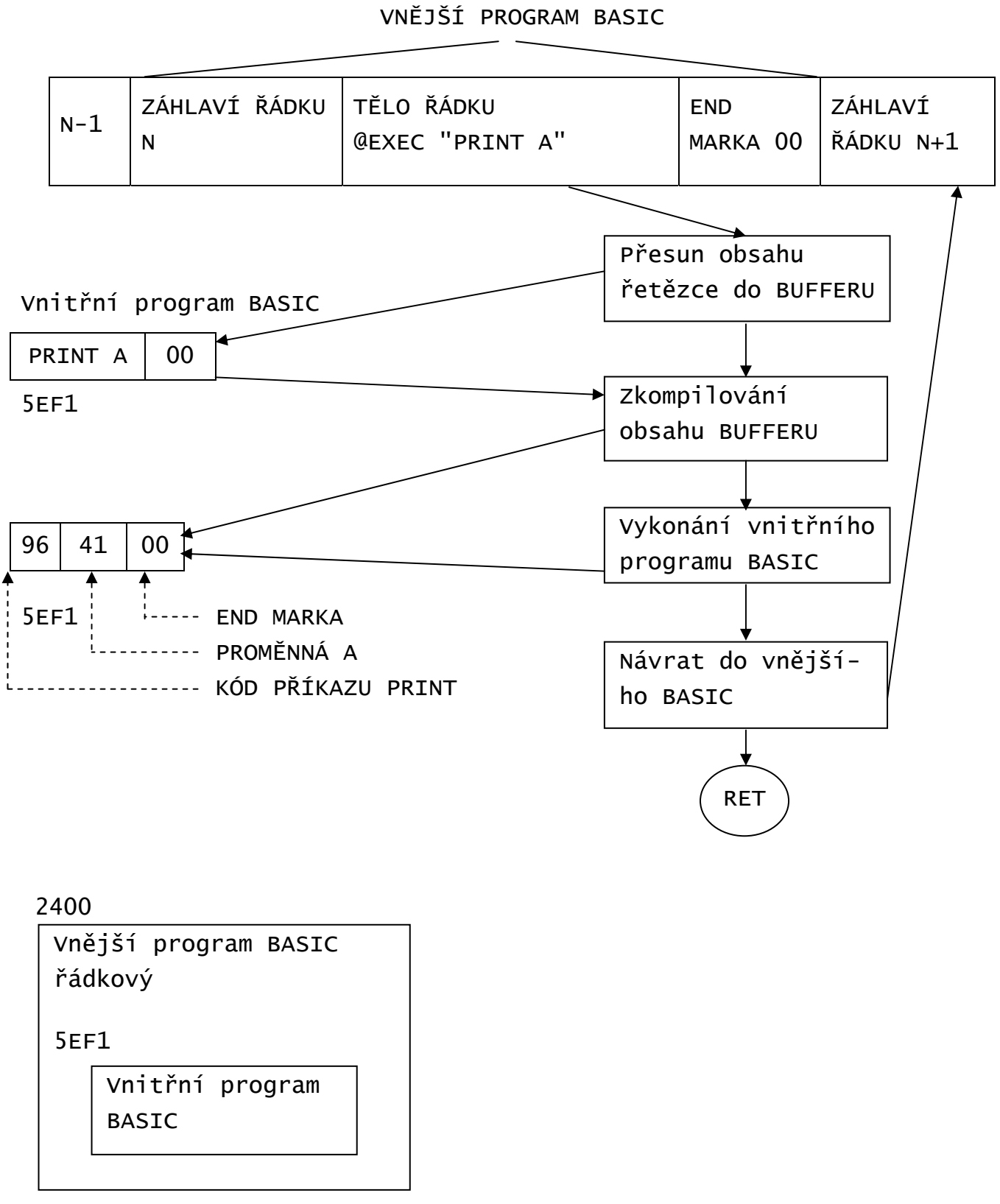
Syntax:

```
@EXEC A$ [, A$ ...]
```

přičemž A$ představuje řetězcovou proměnnou, prvek z řetězcového pole nebo řetězcovou konstantu. Její obsah musí představovat BASIC příkaz nebo funkci. Oddělovačem čárka můžeme vykonávat vícenásobné "BASIC" řetězce pod příkazem @EXEC (obdoba oddělovače dvojtečka u řádkového BASIC programu).

Příkaz @EXEC má tato omezení:

- nelze pod ním vykonávat příkazy skoků a rozhodování (GOTO, GOSUB, IF, ON)
- pod příkazem musí být uvedena řetězcová proměnná a ne výraz
- není možné další zahnízďování BASIC programu, t.j. použít příkaz @EXEC do vnitřního BASICu.



Obr. č. 11 - Struktura vnitřního programu BASIC

Interpretování příkazu @EXEC probíhá v těchto fázích:
- přesun obsahu řetězce nebo řetězcové proměnné do vyhrazené paměťové oblasti - bufferu
- zkompilování obsahu bufferu do mezikódu
- interpretování obsahu bufferu
- test na delimitr - čárku
- návrat do původního programu BASIC

Je už na první pohled zřejmé, že s podporou příkazu @EXEC můžeme vykonávat celé programy BASIC, které se mohou vytvářet během externího (vnějšího = řádkového) BASIC programu. S využitím předcházejících příkazů a funkcí (ROM, POS, @DOKE atd.) můžeme archivované řetězcové pole zavádět do hlavního programu, který tyto obsahy prvků bude vykonávat jako BASIC příkazy.

Následující příklady blíže objasní význam příkazu @EXEC.

Příklady:

```
@EXEC "PRINT A"              ;vytištění obsahu proměnné A
@EXEC "A=A+1"                ;přiřazení jedničky proměnné A
@EXEC "PRINT A$": PRINT A$   ;vykonání výpisu obsahu A$ pod příkazem
                              @EXEC a pod externím BASICem
A$="PRINT SIN(A)":@EXEC A$   ;vykonání příkazu PRINT SIN(A), který je
                              obsahem řetězce a$.

A$="PRINT"                    příklad demonstrace modifikace
B$="SIN"                      obsahu řetězců, které budou
C$="A + 2"                    interpretovány jako BASIC
D$=B$+"("+C$+")"              příkazy
E$="A="+D$
F$=A$+D$
@EXEC E$,F$,"PRINT A":PRINT A
```

Poznámka: V obsahu vnitřního BASIC programu není dovoleno používat oddělovač (dvojtečka) vícenásobného příkazu na řádek, ve kterém se interpretuje vnitřní BASIC program.

Zadání č.6: Vytvořte segment, který by umožňoval realizaci libovolných BASIC příkazů a funkcí.

Řešení:
```
10 DISP "VLOZ PRIKAZ/FUNKCI:"
20 INPUT A$
30 @EXEC A$
40 IF A$<>"REM" THEN 10
50 END
```

Komentář:
10 návěstí vstupního příkazu
20 naplnění řetězce A$ z klávesnice
30 vykonání vnitřního BASIC programu, tedy obsahu řetězce A$
40 rozhodování ukončení cyklu na příkaz REM
50 ukončení vnějšího BASIC programu

Z řešení zadání je hned na první pohled zřejmé, že bez podpory příkazu @EXEC by nebylo tak jednoduché realizovat danou úlohu.

Příklad:
```
 10 FOR I = 0 to 100
 20 GOSUB 100
 30 NEXT I
 40 END
100 @EXEC "PRINT SQR(I + 1)"
105 RETURN
```

V příkladu je ukázáno, jak lze používat příkaz @EXEC při podprogramech GOSUB. Doporučujeme neumísťovat návratový příkaz RETURN společně na řádek s příkazem @EXEC.

Předpokládáme, že těmito ukázkami bude uživateli význam tohoto příkazu jasný, a že může s jeho podporou vytvářet nové kvalitativně vyšší programy BASIC. Tak jako není možné vysvětlit všechny možnosti a kombinace použití příkazů a funkcí při vnějším BASIC programu, tak toto platí dvojnásobně pro příkaz @EXEC. Je třeba ještě upozornit, že nasazování příkazu @EXEC je nutné po zvážení, zda nebude na škodu jeho asi 5-8 % snížená rychlost interpretace, protože kompilace vnitřního BASIC programu probíhá při interpretaci vnějšího BASIC programu.

@COPY

Příkaz umožňující výstup videostránky na tiskárnu.

Syntax:

@COPY

Poznámka: Pro tento příkaz je nutno použít tiskárnu, která umožňuje grafický tisk s protokolem podle fy HP. Tomuto vyhovuje tiskárna PRT 80 (MLR). V jiném případě je třeba upravit servisní rutinu pro daný typ tiskárny. Výstupní data pro tiskárnu jsou vyvedena na GPIO skupina 0 (konektor K3).

Tímto příkazem jsme uzavřeli kapitolu doplňování základního interpreteru BASIC o nové výkonnější příkazy a funkce. Další část příručky se bude věnovat příkazům diskových služeb.

B.I. Všeobecná část

I.1. ÚVOD

Floppy disková jednotka MFD-85 představuje inteligentní vnější paměť pro libovolný typ počítače, který má implementovanou sběrnici IMS-2 (HP-IB) podporovanou řídícími příkazy (ve strojovém kódu popřípadě pod vyšším jazykem). Je určena hlavně pro práci se soubory dat nebo pro jejich koncentraci a sběr. Komunikace s nadřízeným systémem je prostřednictvím sběrnice IMS-2. Zpracovávání diskových služeb v jednotce MFD-85 zabezpečuje operační systém MFS (Management FILE SYSTEM). Do operačního systému se vstupuje příkazem COMMAND se svými parametry, který lze implementovat do uživatelských programů, které nejsou uzpůsobené na diskové operační systémy a mají sběrnici IMS-2. V případě počítače PMD 85 tuto implementaci zabezpečuje programový přídavný modul EXTENDED ROM BASIC (ERB) se syntaxí příkazů podle fy HP (kalkulátor HP-85).

Protože se jedná o nestandardní architekturu a postavení floppy diskové jednotky v počítačové hierarchii, vysvětlíme v následujících kapitolách základní charakteristiky jednotky MFD-85, které budou pokračovat popisem uživatelských diskových služeb a speciálních funkcí.

B.I.2 Postavení MFD-85 v architektuře počítačového systému

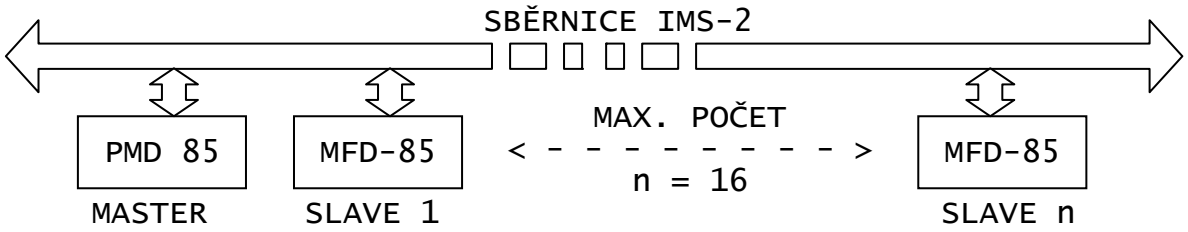
Disková jednotka MFD-85 může pracovat ve dvou režimech:

- MASTER když nepožaduje žádné příkazy od vyššího systému a její činnost je autonomní na základě vykonávání programu ze systémové diskety.

- SLAVE když očekáváme výzvu od nadřízeného systému (může to být i jednotka MFD-85 jako MASTER) na vykonání diskové služby.

V následujících obrázcích jsou znázorněny některé vazby jednotky MFD-85.

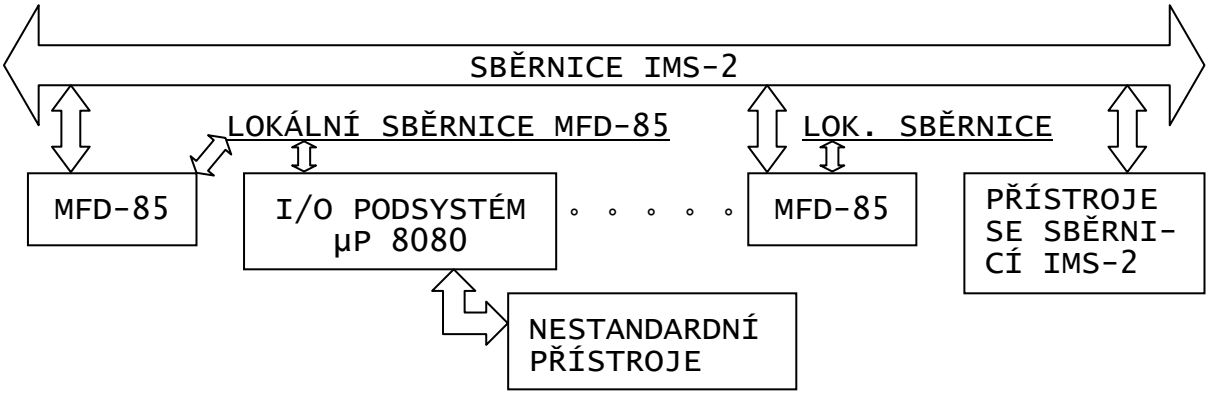
a) Jednoduché propojení počítač – MFD-85



Floppy disková jednotka je plně pod řízením počítače PMD-85. Zvyšování vnější paměti se uskutečňuje pouze připojením ke sběrnici IMS-2, přičemž každá jednotka MFD-85 spotřebuje dvě adresy, takže celkový počet jednotek může být 16.

Tato konfigurace se využívá v aplikacích, kde se zpracovávají rozsáhlé soubory dat.

b) Propojení master MFD-85

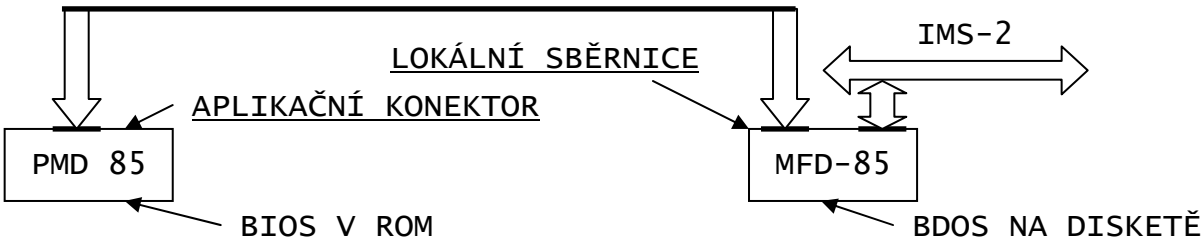


V této konfiguraci zastupuje jednotka MFD-85 - MASTER řídící jednotku na sběrnici IMS-2. Řídící program této jednotky je umístěn na systémové disketě nebo je-li v rozsahu do 1 kB, může být přímo v její operační paměti. MFD-85 dává k dispozici svou lokální sběrnici určenou pouze pro rozšíření I/O podsystému. Na tuto sběrnici mohou být připojeny stavební prvky mikropočítačových rodin 8080, 8048 v počtu 8 prvků, pro komplikovanější měřící systémy sběru dat můžeme připojit do sběrnice IMS-2 taktéž další jednotku MFD-85/SLAVE. Tuto jednotku

můžeme vyhradit pro shromažďování dat, přičemž jednotka MASTER bude interpretovat jen daný systémový program.

c) Speciální propojení s jednotkou MFD-85

Do této skupiny můžeme zahrnout konfigurace orientované na připojení s lokální sběrnicí MFD-85. Tato sběrnice je obousměrná pro adresní linky A0 - 7 a linky I/OR, I/OW a INTR. Po technické stránce to umožňuje samostatný přístup k jednotlivým I/O podpůrným obvodům řídícího mikropočítače umístěného v jednotce MFD-85.



V tomto případě je floppy disková jednotka řízená přímo programovými moduly BDOS + BIOS, které jsou umístěny v operační paměti PMD-85. Velkým přínosem je zvýšená (maximální) přenosová rychlost dat mezi operační pamětí a diskem. Na druhé straně je obsazení velikosti přibližně 8 kB, takže uživatel má k dispozici operační diskový systém CPM verzi 2.2 s pracovní pamětí v rozsahu 20 kB. Tato konfigurace je výhodná pro případy, kdy uživatel chce použít osobní mikropočítač PMD-85 jako vývojový prostředek, pro tvorbu programů ve strojovém jazyce 8080 nebo na interpretaci účelových programů pod CPM.

Další využití této konfigurace se naskýtá u těch počítačů, které nemají diskový operační systém a ani sběrnici IMS-2. Doplněním základního systému BIOS o tuto vstupně/výstupní stranu můžeme pod BDOS vykonávat sběr dat po sběrnici IMS-2.

B.II. Technická data jednotky MFD-85

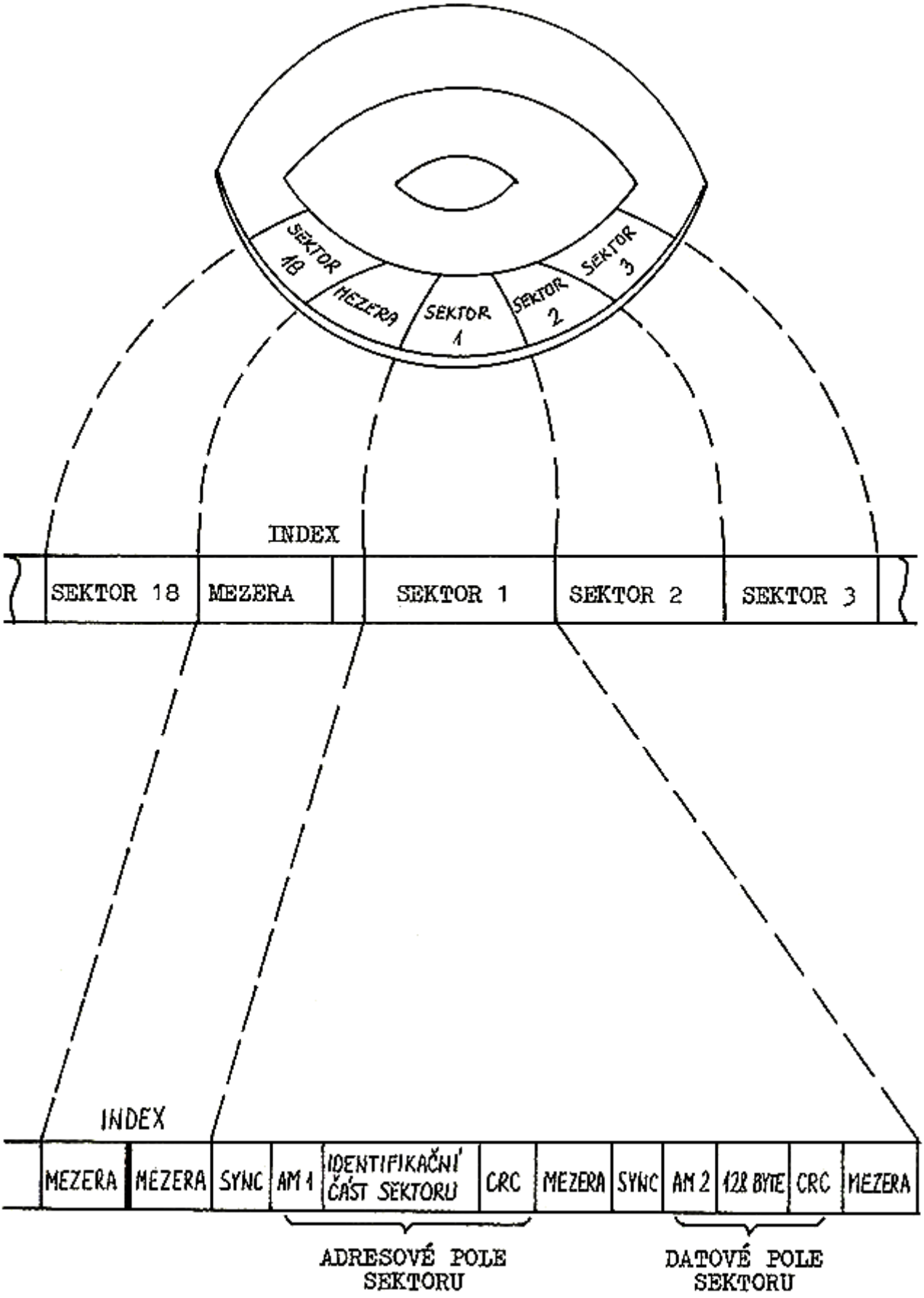
- velikost diskety 5,25"
- kapacita naformátované diskety 87 kB, jednostranná
- záznam IBM 3740
- způsob záznamu FM
- počet stop 40, sektorů 18, velikost sektoru 128 byte
- vlastní řídící logika s mikropočítačem 8080 a s řadičem FDC typ 8271
- vstupní napětí jednotky 220V/50Hz/0,2A
- komunikační kanály: sběrnice IMS-2, lokální sběrnice řídícího mikropočítače
- rozsah operačního systému MFS: 4 kB
- rozsah uživatelské operační paměti: 1,25 kB
- počet adresovaných I/O registrů (obvody 8271, 8255): 5

B.III. Organizace záznamu IBM 3740

V této kapitole si vysvětlíme způsob záznamu dat na disketu podle normy IBM 3740. Pro jednoduchost je uvedeno formátování týkající se dané inicializace v jednotce MFD-85.

Disketa - magnetické médium je rozděleno do 40 stop. Každá stopa je dělena do sektorů, kterých je 18. Sektor má identifikační pole, které udává o jaký sektor jde a na jaké stopě. Za tímto identifikačním polem po krátké mezeře se nachází vlastní datové pole začínající datovou značkou a ukončené dvoubajtovým kontrolním údajem CRC. Vlastní délka datového pole je 128 byte. Jednotlivé sektory na dané stopě jsou odděleny mezisektorovými mezerami se synchronizačními značkami (6x byte 00). Délky mezer jsou závislé na délce vlastního datového pole. Jejich úlohou je kompenzovat mechanické nepřesnosti disku, odlišnosti mezi jednotlivými médii a synchronizace zpracovávání získaných údajů v reálném čase.

Formátování každé stopy je zabezpečeno činností vlastního řadiče FDC 8271, který obdrží pouze potřebné údaje pro jejich inicializaci (počet byte v sektoru, délka mezer, číslo stopy, atd.). Tato problematika překračuje rámce kapitoly, protože se týká vlastního operačního systému MFS.
Na následujícím obrázku je znázorněn formát jedné stopy.


SEKTOR 18
MEZERA
SEKTOR 1
SEKTOR 2
SEKTOR 3
INDEX
SEKTOR 18
MEZERA
SEKTOR 1
SEKTOR 2
SEKTOR 3
INDEX
MEZERA
MEZERA
SYNC
AM 1
IDENTIFIKAČNÍ ČÁST SEKTORU
CRC
MEZERA
SYNC
AM 2
128 BYTE
CRC
MEZERA
ADRESOVÉ POLE SEKTORU
DATOVÉ POLE SEKTORU

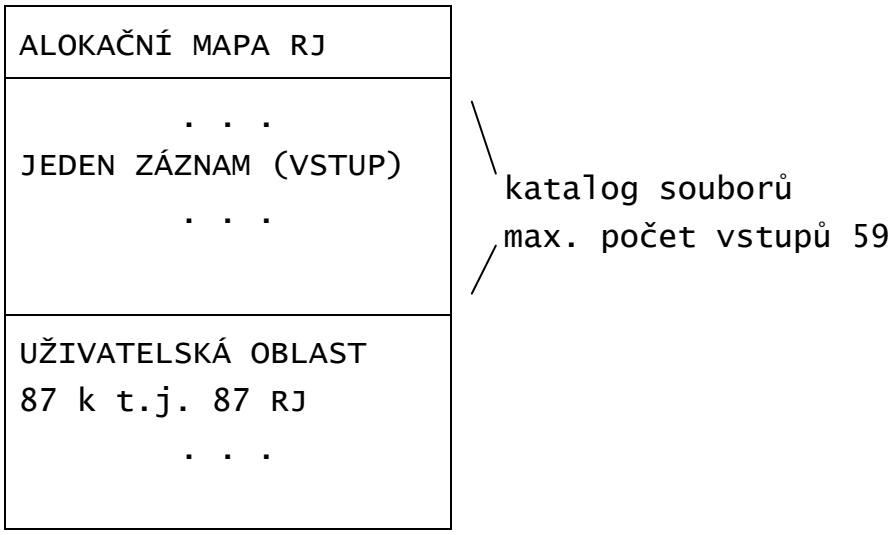
B.IV. Organizace diskové paměti

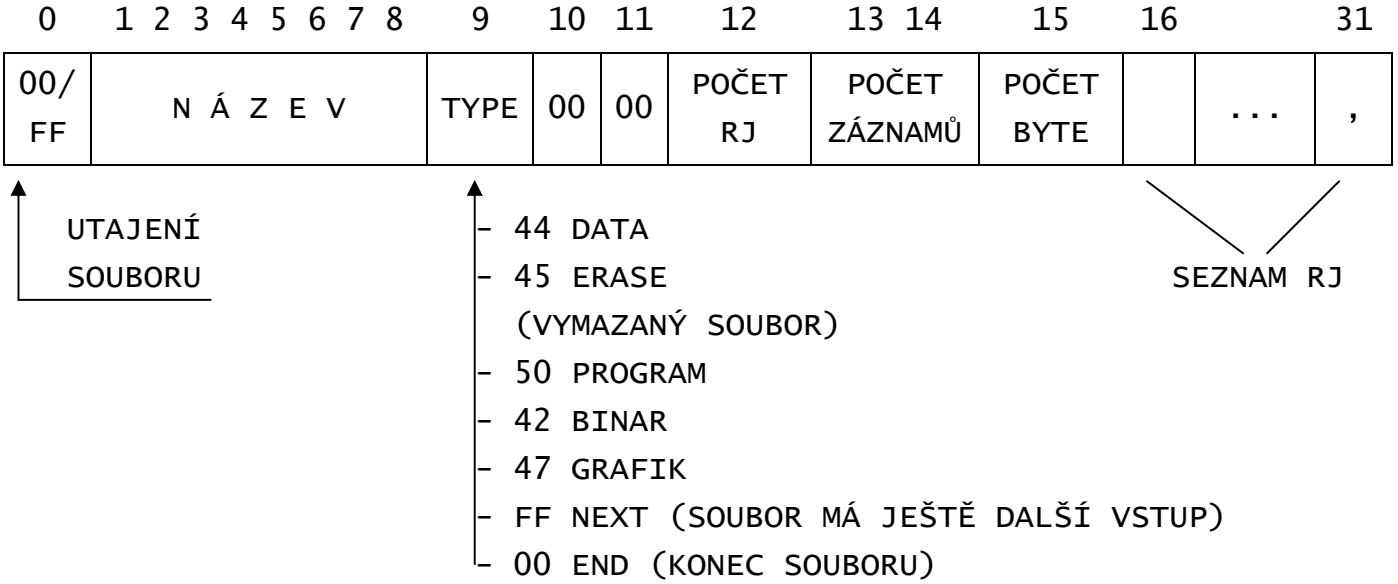
Naformátovaná disketa jak byla uvedena v předcházející kapitole je dána k dispozici operačnímu systému MFS, který je jediným správcem. Aby bylo možné se lépe orientovat v paměťovém rozsahu, má MFS vytvořeny tzv. rozmísťovací jednotky (RJ), které jsou rozsahu 1 kB, t.j. 8 sektorů. Všechny sektory jsou chápány jako posloupnost těchto RJ. Celkově tedy na disketě se nachází 89 RJ. Adresace těchto RJ i v případě dvojnásobné hustoty nebo při 8" disketě nepřesáhne 1 byte. Obsazenost RJ eviduje MFS na každé disketě a to na stopě 0, sektor 1.

Přidělování RJ souboru dat popřípadě binárním programům je evidováno v katalogu diskety, který zabírá celkem necelé dva RJ od sektoru 2 po sektor 16 na stopě 0. Katalog diskety patří mezi nejdůležitější část diskety, protože operační systém MFS přistupuje k požadovaným datům prostřednictvím jeho informací.

Kapacita katalogu je 59 vstupů (záznamy). Délka jednoho záznamu v katalogu je konstantní v rozsahu 32 byte. Po provedení vlastní inicializace diskety zaznamená MFS na stopě 0 do dvou RJ dva bloky; rozmísťovací (a lokační) mapu RJ a katalog diskety v závěru provede verifikaci celé diskety.

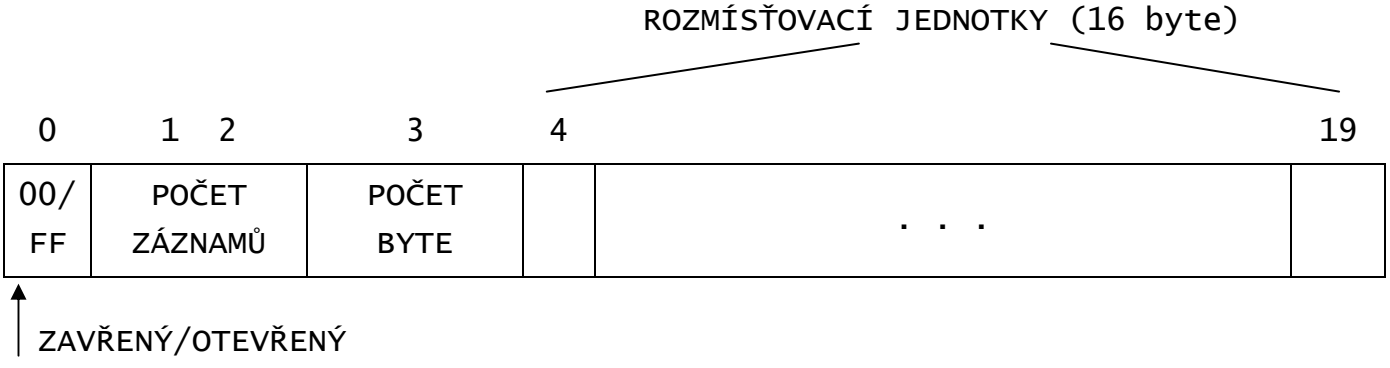
Tím je ukončena činnost INIT diskety a je připravena se svou kapacitou 87 kB pro vykonávání diskových služeb. Pro lepší představu, následující obrázek znázorňuje plošné rozložení diskové paměti a záznam jednoho vstupu do katalogu diskety.





Poznámka: V případě, že binární program má délku více než 16 kB, bude mu přidělen další vstup do katalogu, přičemž přijme v atributu TYPE hodnotu FF (NEXT soubor).

Operační systém MFS si neuchovává katalog v operační paměti. Při práci s datovými soubory je třeba dát příkaz (@OPEN) na konkrétní datový soubor a ten se vyhledá v katalogu diskety. Údaje od 13. pozice z daného vstupu se uloží do zvláštní části jeho zápisníku (OPEN BUFFER). Celkově lze otevřít až 8 souborů (0 až 7). Potom při práci s daným záznamem v souboru se operační systém MFS obrací na OPEN BUFFER. Následující obrázek znázorňuje jeden vstup v OPEN BUFFER.



Pozice vstupu v OPEN BUFFER udává parametr (#0 - #7) bufferu.

B.IV.1 Popis typů záznamů na disketě

Jak už bylo vysvětleno v předcházejícím odstavci, k uživatelské části diskové paměti se přistupuje přes katalog s alokační mapou obsazení. (RJ) Každý vstup do katalogu musí mít přidělen i o jaký typ dat jde. V podstatě lze tyto typy rozdělit do těchto skupin:

a) DATOVÝ SOUBOR, v katalogu má označení D a pod jedním názvem může být přidělen maximálně jen 16 RJ.

Tedy platí:

Počet záznamů x počet prvků v záznamu <= 16384

přičemž počet prvků (byte) v každém záznamu je konstantní max. 255.

Z těchto podmínek vyplývají následující omezující dimenze daného souboru:

64 x 255	a	9999 x 1

V těchto dimenzích lze vytvářet datové soubory pod jedním názvem. Údaje se ukládají ve tvaru ASCII v rozsahu, v jakém byla vytvořena dimenze souboru. Je-li zapisovaný záznam kratší než dimenzovaný, je zaznamenávaná věta ukončena znakem CR (0D HEXA).

Poznámka: nedovoleným znakem (nebo vědomě povoleným) je znak čárka (2C HEXA), který může při čtení způsobit problémy, pokud to není programem ošetřené.

b) BINÁRNÍ SOUBOR, v této skupině se nacházejí všechny ostatní typy záznamů na disketě. Jde o záznamy, které mají údaje ve tvaru HEXA (00 až FF). Podle zdroje jak vznikly (nebo odkud jsou získány) lze jejich příslušnost rozdělit na záznamy:

G - grafické, které nesou údaje video obrazu
P - programové, které jsou produktem interpreteru BASIC
B - binární, všeobecné datové údaje z operační paměti počítače (PMD 85).

Záznamy G, P nepožadují určování místa transferu. U typu B (binárního) je třeba předem příkazem @BUF= určit adresové místo a jeho rozsah.

Poznámka: V případě, že nebude definován paměťový prostor a použije se transfer záznamu typu B, může dojít k zničení nedefinované části operační paměti počítače! Doporučujeme používat paměťový prostor programového bufferu interpreteru BASIC (2400 - 5E00).

Vstup do katalogu je rozdělen podle uvedených skupin a platí pro:
- datový soubor příkaz @CREATE
- binární soubor příkaz @STORE

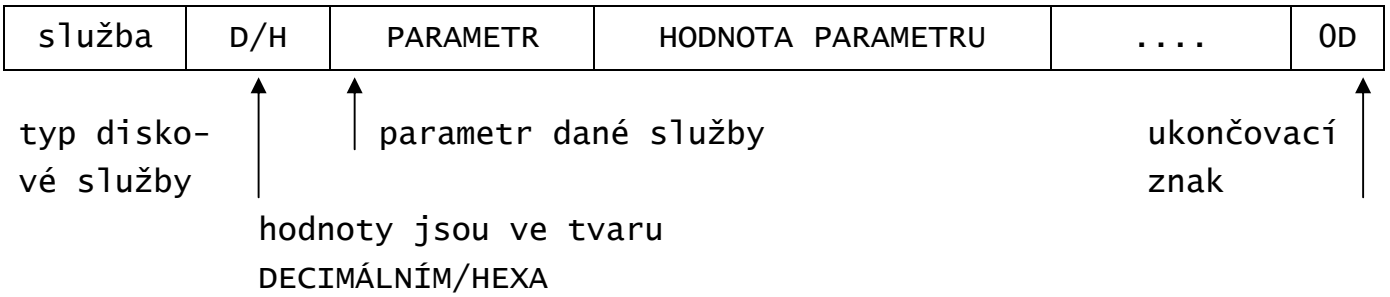
Bližší vysvětlení těchto příkazů bude v samostatné části.

B.V. Charakteristika operačního systému MFS

Organizaci diskové jednotky včetně komunikace s okolím zabezpečuje 4 kB firmware označovaný jako organizátor souborů diskety Management file system (MFS). Pro svou práci si vymezuje v paměti RWM 0,75 kB, které mají význam ukazatelů konstant, vyrovnávacích registrů (BUFFER) různých délek. Hrubé rozdělení zápisníkové paměti RWM je na následujícím obrázku:

183F	STATUS	8 byte	stavový registr - odezva na daný úkon
1870	COMMAND	80 byte	registr pro příjem výzvy na diskovou službu
18E0	OPEN	160 byte	registr otevřených souborů z katalogu
1980	DISK	128 byte	registr pro fyzický sektor diskety
1A00	HPIB	256 byte	vstupně/výstupní registr pro práci s logickými sektory
1B00	USER	1,25 kB	uživatelská část RWM, prostor pro interpretování programů získaných přenosem externích dat, nebo z vlastní diskety

Proveďme si analýzu těchto registrů. Místem pro příjem výzvy na vykonávání dané diskové služby je registr COMMAND. Formát tohoto registru je následující:



První znak ve slově COMMAND určuje, o jaký typ diskové služby půjde. V současnosti jsou zpracovány a přiřazeny tyto znaky k diskovým službám:

ZNAK:
- W zápis na fyzický sektor
- R čtení z fyzického sektoru
- P zápis na logický sektor
- I čtení z logického sektoru
- C vytvoření datového souboru
- U uzavření souboru
- F formátování diskety
- O otevření datového souboru
- G čtení z operační paměti
- A uzavření všech souborů
- J skok na interpretování podprogramu
- E vymazání z katalogu
- M zápis do operační paměti
- D čtení katalogu
- N čtení názvu diskety
- Z zápis binárních dat
- L čtení binárních dat
- B utajení souboru v katalogu
- H vyhledání slova na souboru (SEARCH)

Za tímto kódem následuje znak určující, zda hodnoty v parametrech mají tvar hexa nebo decimální. V případě decimální hodnoty musí se nacházet za kódem parametr hodnota nula nebo mezera.

Seznam všech parametrů udává následující tabulka:

znak	hodnota	význam
T	xx	nastavení parametru stopa /TRACK/
S	xx	nastavení parametru sektor
#	xx	přesun údajů z katalogu do OPEN BUFFER
$	xxxx	nastavení čísla záznamu
@	xx	nastavení délky záznamu
'	XXXXXXXX	název souboru /max. 8 znaků/
&	Xxxx	nastavení adresového souboru
!	žádná	provedení kontroly všech par. v COMMAND slově
*	D	typ souboru ASCII
	B	binární
	G	grafický
	P	program BASIC

Poznámka: xx znamená decimální, popř. hexadecimální hodnotu parametru kromě znaku apostrof ('), kde jde o znaky ASCII.

Vyslání COMMAND slova do jednotky MFD-85 může být i prostřednictvím příkazu OUTPUT, jako řetězec posloupnosti znaků, které byly popisovány. Tvar všech COMMAND slov uvádí následující tabulka:

COM-MAND	TVAR	
1.	W D/H T X S X @ X * X	zápis na fyzický sektor
2.	R D/H T X S X @ X * X	čtení z fyz. sektoru
3.	P D/H # X $ X !	zápis na logický sektor
4.	I D/H # X $ X !	čtení z log. sektoru
5.	C D/H ' X ' $ X @ X * X	vytvoření souboru
6.	U D/H # X	uzavření souboru
7.	F D/H ' X '	formátování diskety
8.	O D/H ' X ' # X	otevření souboru
9.	G D/H & X @ X	čtení dat z RAM
10.	A D/H	uzavření souborů
11.	J D/H & X	skok na podprogram
12.	E D/H ' X '	vymazání z katalogu
13.	M D/H & X @ X	zápis do RWM
14.	D D/h	čtení katalogu
15.	N D/H	čtení názvu diskety
16.	Z D/H ' X ' $ X * X	zápis binární
17.	L D/H ' X ' * X	čtení binární
18.	B D/H ' X '	utajení souboru
19.	H D/H # X $ X @ X	hledání v souboru

Poznámka: Znak X v tabulce COMMAND slov reprezentuje hodnotu parametru, popřípadě znaky ASCII, pokud se jedná o název parametru nebo typ souboru.

Pro lepší pochopení si uvedeme příklad odeslání COMMAND slova pro vytvoření souboru v katalogu.

```
CD'NAME'$0100@050*D
```

Obsah tohoto řetězce znaků požaduje od MFS následující:

C	vytvořit datový soubor
D	hodnoty v parametrech jsou udány decimálně
'NAME'	název souboru bude NAME
$	počet záznamů má být 100
@	každý záznam má mít délku 50 znaků
*	jde o typ souboru datový

Na výkon přijetí COMMAND slova reaguje operační systém MFS vyslá-ním STATUS slova, které může mít hodnotu ze dvou skupin, a to:
- úspěšné ukončení
- transfer dat
- chybové hlášení.

Interpretování MFS se uskutečňuje v podstatě přes dva vstupní bo-dy:

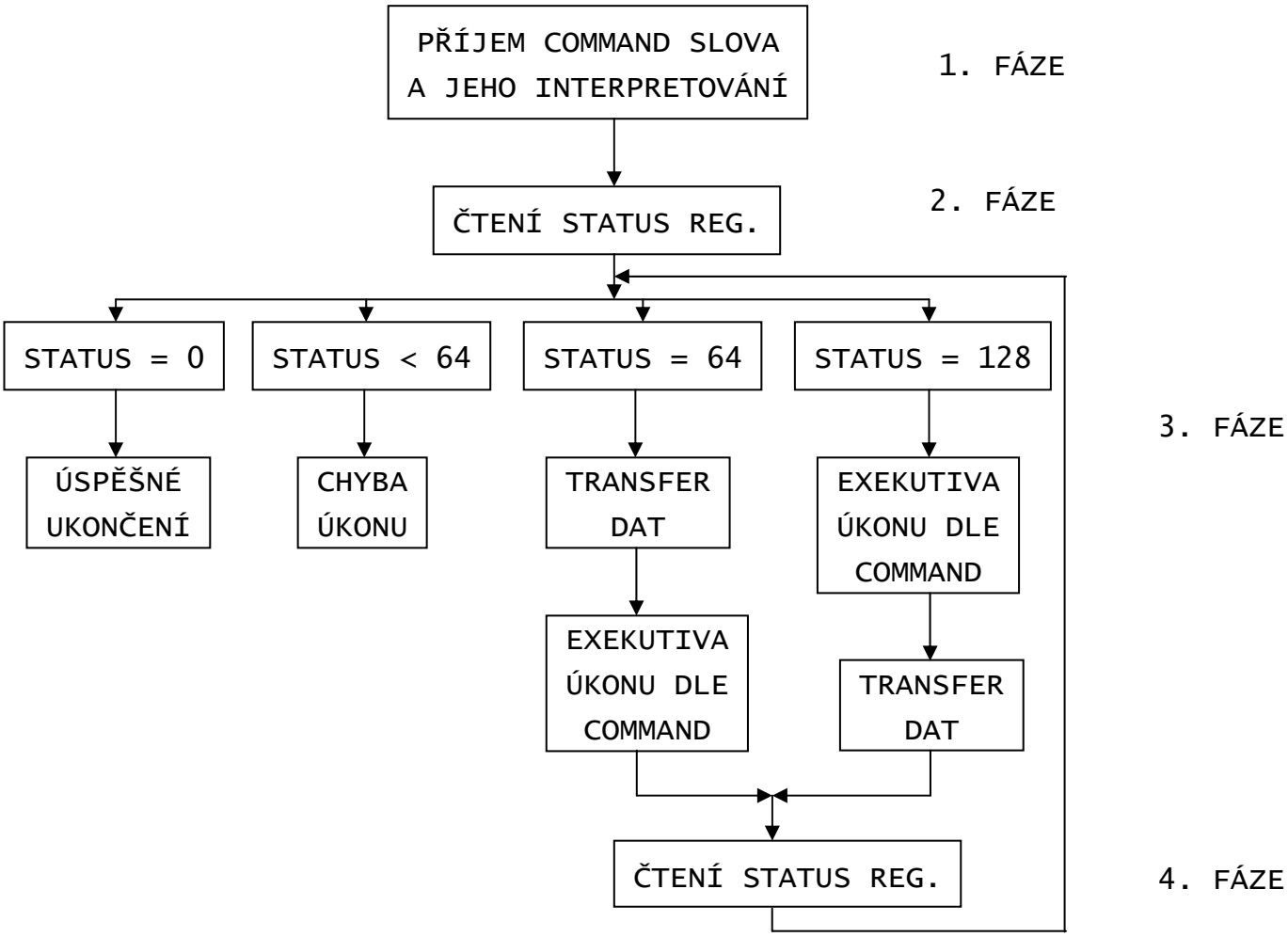
- BOD PRVNÍ: zabezpečuje nastavení všech požadovaných parametrů zada-ných v COMMAND slovu. V této fázi nedochází k přenosu dat. Je to pro většinu služeb pouze přípravná fáze. Ode-zvou na odeslaný příkaz se volající dozví z obsahu STATUS registru. Tento registr je třeba sledovat, proto-že on určuje, zda se přistoupí k další fázi. Pokud se jedná o konkrétní příkaz, mohou vzniknout pouze tři hod-noty STATUS registru.

  0 ..... úspěšné ukončení služby
 64 ..... správně přijatý COMMAND, operační systém MFS očekává, že volající bude posílat data pro zápis
128 ..... správně přijatý COMMAND, operační systém oznamuje, že volající může číst data.

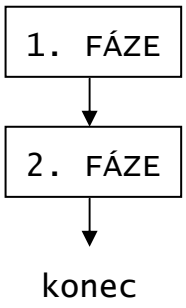
Z dosud uvedených informací je možno konstatovat, že pro vykonání diskové služby jsou nutné následující úkony:
- volající musí vyslat daný COMMAND
- volající musí žádat přečtení STATUS, podle kterého se rozhodne, zda daná služba úspěšně skončila nebo dojde k transferu dat
- volající musí po transferu dat přečíst STATUS registr, aby zjistil úspěšnost přenosu, popřípadě, zda se v přenosu nebude pokračovat.

Tedy je nutné vykonat minimálně dvě fáze COMMAND - STATUS a dále pokud je to nutné DATA - STATUS.

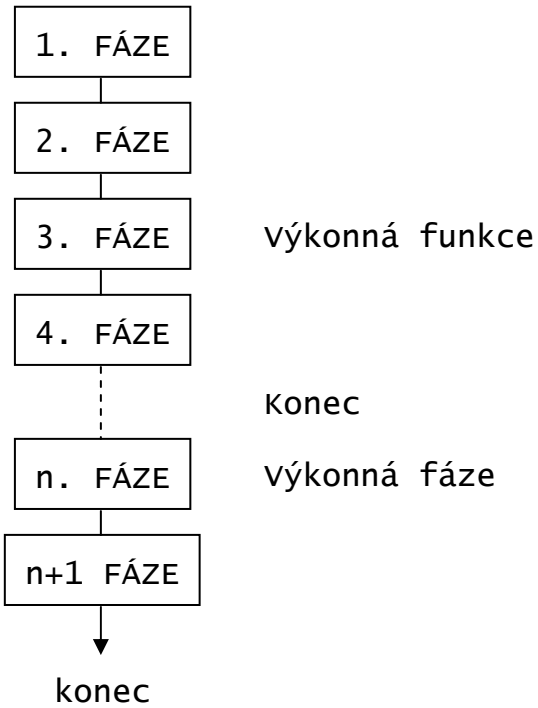
Pro vykonání fáze DATA se volá:
- BOD DRUHÝ: operačního systému. Pro lepší názornost činnosti MFS uvádíme vývojový diagram popisující jednotlivé fáze od přijetí slova.



Typ služby bez transferu dat



Typ služby s transferem dat



Z hlediska komunikačního protokolu operační systém MFS spotřebuje dvě adresy IMS-2. První adresy IMS-2 platí pro COMMADD a STATUS a je z množiny adres od 1 do 16. Pro přenos dat DATA READ a DATA WRITE je adresa vypočítávána s ofsetem + 16. Tedy například:

```
OUTPUT 701; ... COMMAND
ENTER  701; ... STATUS
```

```
OUTPUT 717; ... DATA WRITE
ENTER  717; ... DATA READ
```

B.VI. Chybové hlášení

Operační systém MFS oznamuje volajícímu o diskovou službu vzniklou chybu na danou službu decimálním číslem, určujícím o jaký typ jde. Pokud uživatel užívá (dále bude popsáno) EXTENDED ROM BASIC, lze tuto chybu programově ošetřit v reálném čase, (příkazy @ON ERROR, @OFF ERROR).

Následující tabulka uvádí všechny možné typy chybových hlášení.

číslo chyby	význam
1	chyba při snímání dat, CRC ...
2	chyba na mechanice disku (otevřená dvířka)
3	nenalezený sektor, stopa
5	chybná syntaxe v COMMAND slovu
6	chyba v parametru
7	chybná hodnota v parametru
8	nenašel typ služby
9	neukončený COMMAND
10	disk zaplněn
11	není otevřený soubor
12	chyba v názvu souboru nebo duplicita
13	parametr mimo rozsah
14	soubor utajen
15	chyba v typu souboru
17	katalog zaplněn

Poznámka: V případě, že vznikne chyba č. 1 nebo 3, není možno ji žádným systémovým přístupem odstranit.

B.VII. Programový přístup k diskové jednotce MFD-85

K diskové jednotce můžeme přistupovat prostřednictvím jednoduchých příkazů OUTPUT/ENTER nebo s programovou podporou u modulu ERB (jen u počítače PMD-85) prostřednictvím agregovaných diskových příkazů. V této kapitole si podrobně probereme oba způsoby. Na základě uvedených informací o činnosti systému MFS provedeme nejprve nejjednodušší programový přístup, který lze realizovat libovolnými počítači, které mají implementovánu sběrnici IMS-2 (HPIB), např. HP 85, HP 87, TEKTRONIX, DIDAKTIK ALFA.

Vysvětlíme si pouze takové příkazy, které patří do standardních diskových služeb potřebných pro DATA BASE.

1. OUTPUT/ENTER programový přístup

Předpokládejme, že daná jednotka MFD-85 má zvolenu adresu 1. Potom vyslání COMMAND slova bude

```
OUTPUT 701; řetězec COMMAND slova
```

čtení STATUS registru do proměnné A

```
ENTER 701; A
```

Pro datový přenos platí:

```
OUTPUT 717; A$
ENTER  717; A$
```

Tuto činnost si nejlépe vysvětlíme na několika příkladech:

Příklad: Ukázka příkazů pro formátování diskety.

```
10  OUTPUT 701; "FD'TESLA'"
20  ENTER 701; A
30  IF A<>0 THEN PRINT "ERROR DISK 1"; A
40  END
```

Příklad: Ukázka příkazů pro vytvoření souboru a zápis dat do jeho zvoleného záznamu

```
 10  OUTPUT 701; "CD'GAGA'$0300@050*D"
 20  ENTER 701; A
 30  IF A <> 0 THEN PRINT "ERROR DISK 1"; A:STOP
 40  OUTPUT 701; "OD'GAGA'#00"
 50  ENTER 701; A
 60  IF A <> 0 THEN PRINT "ERROR DISK 1"; A:STOP
 70  DISP "ZADEJ CISLO ZAZNAMU A OBSAH"
 80  INPUT Z,Z$
 90  A$="PD#00$"+STR$(Z)+"!"
100  OUTPUT 701; A$
105  ENTER 701; A
110  IF A <> 64 THEN PRINT "ERROR ..."; A:STOP
120  OUTPUT 717; Z$
130  ENTER 701; A
140  IF A <> 0 THEN PRINT "ERROR ..."; A:STOP
150  GOTO 70
```

Poznámka: daný příklad lze zkompresovat, avšak záměrně jsou uvedeny jednotlivé kroky, jak lze sestavit program pro volání diskových služeb.

Příklad: Ukázka příkazů pro výpis katalogu diskety

```
 10  OUTPUT 701; "DD"
 20  ENTER 701; A
 30  IF A <> 128 THEN PRINT "ERROR"; A:STOP
 40  ENTER 717; A$
 50  ENTER 701; A
 60  IF A = 0 THEN PRINT A$ : END
 70  IF A = 128 THEN PRINT A$
 80  GOTO 30
```

Pod programovým přístupem OUTPUT/ENTER není možno řešit diskovou službu čtení programu BASIC, protože by došlo ke zničení vlastního programu čtení.

Zásadně můžeme uvést, že tento programový přístup je vhodný všude tam, kde se bude MFD-85 využívat ve funkci koncentrátoru dat, t.j. pro zpracování datových souborů. V případě, že uživatel si zmodifikuje svoje programové vybavení (operační systém

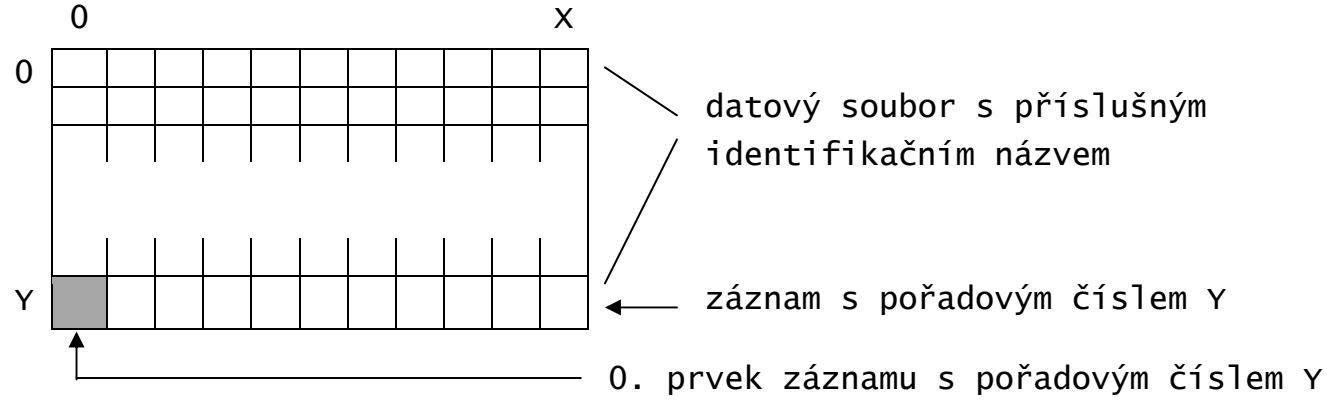
nebo programovací jazyk), může vykonávat diskové služby agregované do jednoho povelu. Takovouto úlohu zastává i rozšiřující modul ERB (EXTENDED BASIC) pro počítač PMD 85. V následující části si vysvětlíme jednotlivé diskové příkazy modulu ERB.

2. Programový přístup pod ERB

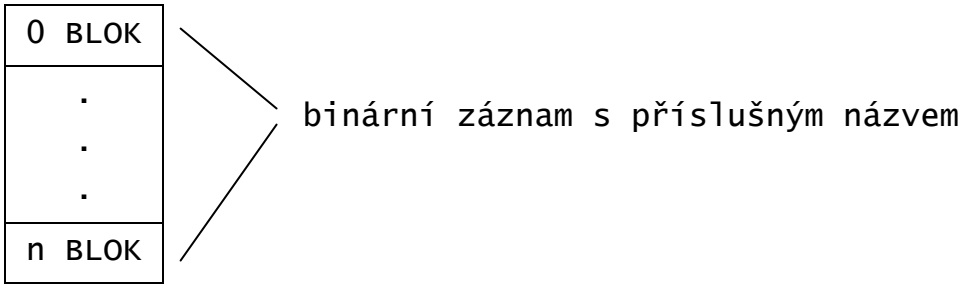
Vzhledem k tomu, že většinou jednotky MFD-85 budou připojeny k osobnímu počítači PMD 85, tento způsob komunikace bude nejčastěji používán, vysvětlíme si nejprve některé základní pojmy pro práci se soubory dat.

Na následujícím obrázku jsou znázorněny části jednotlivého souboru a binárního záznamu.

datový soubor:



binární záznam:



Každý blok má konstantní délku - délku fyzického sektoru (128 byte). Při přenosu dat se přenáší z posledního bloku "n" pouze potřebná část.

Programátor si musí uvědomit při používání diskových příkazů, že výstupní příkazy se interpretují podobně jako příkaz PRINT včetně výrazů apod.

Při příkazech vstupních (přiřazovacích) se uskutečňuje interpretace obdobně jako u příkazu INPUT, tj. platí tytéž zásady a samozřejmě i omezení (čárka, dvojtečka). Syntax příkazů je až na prefix "@" stejná jako u kalkulátoru HP-85 (fy HEWLET PACKARD).

a) Příkaz pro přiřazení diskové jednotky

Disková jednotka MFD-85 má v zadní části umístěn přepínač DIL, pomocí kterého se navolí adresa dané jednotky. Tuto hodnotu je třeba programově nastavit v počítači PMD 85. Uskutečňuje se to příkazem @DISK=.

Syntax: `@DISK= A`

kde A je proměnná nebo konstanta v rozsahu 1-16.

Příklad: `@DISK= 2`

Vykonáním tohoto příkazu budou všechny diskové služby směrovány na diskovou jednotku s adresou 2.

Poznámka: V případě, že uživatel má jen jednu MFD-85, doporučujeme ji přiřadit adresní hodnotu 1, protože při inicializaci modulu ERB je tato adresa přiřazena implicitně.

b) Příkaz pro inicializaci diskety

Syntax: `@INIT A$`

- kde A$ představuje řetězcovou konstantu, proměnnou nebo prvek řetězcového pole. Max. počet znaků v řetězci je omezen na 8 a udává název diskety.

Příklad: `@INIT "TESLA"`

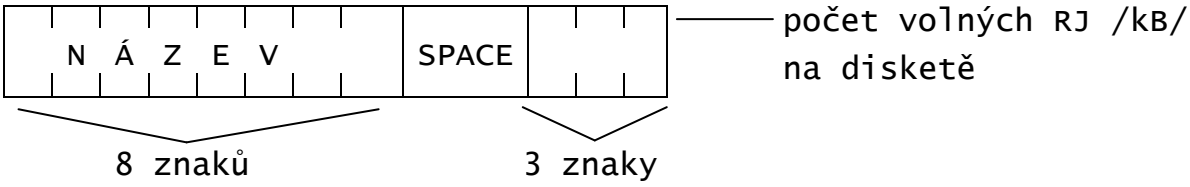
Poznámka: Používání tohoto příkazu je třeba zvážit, protože způsobí úplné "vyčistění" diskety a vytvoření nového katalogu.

c) Příkazy pro práci s katalogem

- Pro zjištění názvu diskety použijeme příkaz @NAME

Syntaxe: `@NAME [; A$]`

Název diskety zabírá 12 znaků a má následující strukturu



V případě, že chceme s tímto obsahem pracovat v programu, doplníme syntaxi příkazu o obsah v hranatých závorkách.

Příklad: Ukázka kontroly správné vložené diskety

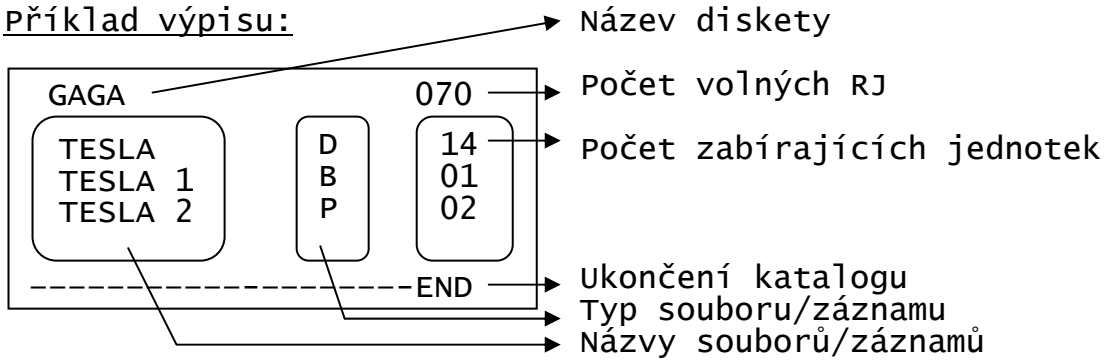
```
10 @NAME ; A$
20 IF A$<>"AGAG" THEN PRINT:STOP
30 .
   .
   .
```

Výpis obsahu katalogu uskutečníme příkazem @CAT.

Syntax: `@CAT`

Po vykonání příkazu se zobrazí seznam všech evidovaných souborů a záznamu v katalogu včetně jejich počtu RJ. Katalog začíná uvedeným jménem diskety s počtem volných RJ tj. kB.

Příklad výpisu:



- zrušení souboru/záznamu v katalogu

Syntaxe: `@PURGE A$`

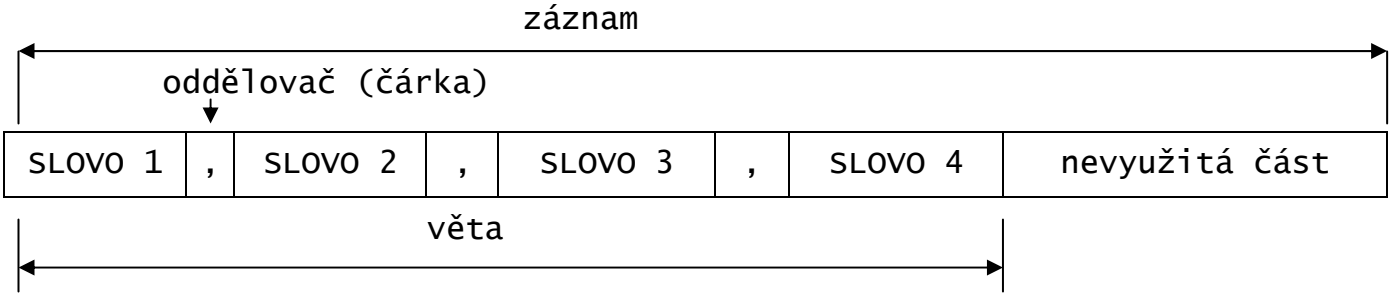
- kde A$ má tentýž význam jako u příkazu @INIT

Po vykonání příkazu @PURGE dojde k vymazání daného názvu v katalogu a zpětnému přiřazení jeho rozmísťovacích jednotek do alokační zóny. V případě, že se jedná o datový soubor, lze bezprostředně po této činnosti vytvořit nový datový soubor, ale musí mít délku záznamu stejnou jako předcházející, který byl vymazán. Počet záznamů může být zvolen jiný. Operační systém může přidělit tytéž RJ za předpokladu, že je to první v pořadí vstup do katalogu, který nese atribut ERASE (vymazaný). Doporučujeme uživateli, aby tuto informaci důkladně promyslel, protože mu umožní některé užitečné operace (dále bude popisováno).

d) Příkazy pro práci s datovým souborem

Operační systém MFS zná pouze jeden způsob přístupu k datovým záznamům. Je to přístup přímý (RAF - RANDOM ACCESS FILE) ke konkrétnímu datovému záznamu.

V případě, že v daném záznamu použijeme oddělovač "čárku", může v daném záznamu pracovat s jednotlivými částmi samostatně - viz obrázek.



Jak ukazuje obrázek, do zvoleného záznamu s danou délkou je umístěna VĚTA (v našem případě menší délky než dovoluje dimenzovaný záznam).

Věta má 4 slova, která jsou vzájemně oddělena čárkami.

Zápis do záznamu se musí uskutečnit celou větou včetně oddělovačů - čárka. Čtení takového zápisu je uskutečněno postupným čtením každého slova do přiřazené proměnné - obdobně jako při funkci INPUT. V případě, že v přiřazovacím příkaze se nachází jen jedna proměnná, bude do ní přiřazeno jen počáteční první slovo - SLOVO 1.

Pro práci s datovými soubory jsou implementovány v MFS následující diskové služby:

- Vytvoření souboru

Syntax: `@CREATE A$; X, Y`

kde A$ představuje název datového souboru a může být zadáno jako řetězcová proměnná, prvek nebo konstanta
X znamená počet záznamů v daném souboru
Y udává délku každého záznamu
Hodnoty X, Y mohou být i výrazy, přičemž jak už bylo uvedeno, musí být v rozsahu

`64, 255` a `9999, 1`

tak, aby jejich součin nepřesáhl 16 kB.

Při otevírání datových souborů je třeba mít na zřeteli, že operační systém MFS přiděluje RJ, které jsou velikosti 1 kB. Takže je třeba zvážit, který parametr (X nebo Y) může být v dané aplikaci zvětšen, aby se využila celá přidělená paměťová plocha RJ.

Vykonáváním tohoto příkazu zabezpečí MFS vstup do katalogu, kde se pro dané jméno souboru vyhradí paměťový prostor diskety.

Příklad: `@CREATE "TESLA", 100, 18`

Význam: vytvoření datového souboru s názvem TESLA s počtem 100 záznamů (číslování je od nuly) a každý záznam má dimenzi 18 znaků.

Poznámka: 1) Operační systém přidělil tomuto záznamu dvě RJ, přičemž aktivně se využívá pouze 1 800 byte. Zbylých 248 byte nejsou adresně přístupné. Výhodné z hlediska perspektivy je otevřít soubor s počtem záznamů 113.
2) Příkazem @CREATE se nevykoná žádná manipulace s

obsahem přidělených RJ, takže obsah záznamů není definován. Jejich obsah je možno inicializovat zvláštním příkazem viz dále.

- otevření souboru

Syntax: `@OPEN A$# A`

kde: A$ představuje název souboru, platí tu tytéž zásady jako u příkazu @CREATE
A udává pořadové číslo tzv. OPEN BUFFER a může být v rozsahu 0 až 7 celočíselně.

Význam tohoto příkazu je otevřít datový soubor pro další práci. Operační systém MFS si převezme všechny potřebné údaje od tohoto souboru z katalogu a uloží do svého pracovního zápisníku (OPEN BUFFER). Takto může být otevřených až 8 souborů.

Příklad: `@OPEN "TESLA"# 0`

Po vykonání tohoto příkazu se k danému souboru přistupuje ne podle názvu, ale pod pořadovým číslem 0.

Poznámka: Přiřazení názvu k pořadovému číslu může být vícenásobné, platí poslední přiřazení.

- zavření souboru

Syntax: `@CLOSE# A`

kde: A udává pořadové číslo daného souboru, který se má uzavřít, Parametr A může být i výraz, jehož hodnota se pohybuje celočíselně od 0 do 7.

Význam tohoto příkazu spočívá v tom, že se uzavírá přístup k záznamu daného souboru. Pro opětovný přístup je třeba opakovat příkaz @OPEN.

Pokud se požaduje uzavření všech souborů, které byly otevřeny, stačí vydat příkaz z diskové služby.

```
@CLOSE
```

- zápis do záznamu

Syntax:
```
@PRINT# A,X; výraz
```

kde: A určuje pořadové číslo otevřeného souboru
X udává číslo záznamu, do kterého se má zapsat obsah výrazu

Pro vykonání tohoto příkazu MFS vyhledá na základě údajů OPEN BUFFER příslušný logický záznam a provede do jeho prostoru zápis obsahu výrazu takové délky, jak mu to dovoluje vytvořená dimenze.

Příklad: `@PRINT# 0, 50; "TESLA"; SIN (123)`

Úkon: do otevřeného souboru #0 na záznam s pořadovým číslem 50 se zapíše řetězec znaků TESLA a hodnota výrazu SIN(123).

Pokud bychom požadovali oddělit jednotlivá slova v záznamové větě, můžeme to udělat následujícím způsobem:

Příklad: `@PRINT# 0, 50; "SLOVO 1,SLOVO 2"`
nebo `@PRINT# 0, 50, A$+","+B$`

Poznámky: Příkaz @PRINT# uskutečňuje zápis údajů ve formě ASCII znaků, tak jako kdyby údaje se měly zobrazovat. Je třeba si uvědomit toto hlavně při zápisech numerických hodnot.
Pro daný "výraz" v příkazu @PRINT# platí tatáž pravidla jako pro příkaz PRINT, tj. pozor na logickou zarážku, kterou třeba zrušit pro větší počet znaků než 64.

- čtení ze záznamu

Syntax:
```
@READ# A, X; proměnná  [, proměnná ...]
```

kde: parametry A, X mají tentýž význam, tj. pořadové číslo otevřeného souboru a číslo žádaného záznamu. Obsah záznamu X se přiřadí zadané proměnné nebo proměnným.

Příklad: @READ# 0, 50; Z$

Úkon: Proměnná Z$ nabude obsah záznamu s pořadovým číslem 50. V případě, že záznam má obsah kratší než je jeho dimenze, proměnná Z$ bude mít přiřazeny všechny znaky až po ukončovací znak CR (0D Hexa).

Pro přiřazení záznamu, který má delimitr čárku je třeba použít tytéž zásady jako při běžném příkaze INPUT, tedy:

```
@READ# 0, 50; Z1$, Z2$ ...
```

- vyhledávání v souboru (SEARCH)

Příkaz na tuto diskovou službu představuje velmi výkonná činnost, protože jde o agregovanou službu, která umožňuje pracovat na určeném zvoleném prostoru souboru.

Předpokládejme, že máme v daném datovém souboru zaznamenánu určitou větu. Pro zjištění, na kterém záznamu je zapsána, bychom museli použít příkazy pro postupné načítání každého záznamu do paměti počítače a tam provádět srovnávání.

Takováto činnost by byla velmi zdlouhavá a monotónní pro operační systém MFS. Z tohoto důvodu obsahuje MFS inteligentní diskovou službu SEARCH, která obdrží parametr pro svou činnost a bude to vykonávat ve své kompetenci. Po ukončení nebo při úspěšném nalezení žádaného obsahu podává zprávu s číslem záznamu.

Syntax:

```
@SEARCH# A, X, Y, Z; výraz
```

kde: A udává pořadové číslo otevřeného souboru
X je pořadové číslo záznamu
Y pozice prvku v záznamu
Z proměnná, které se přiřadí číslo záznamu při úspěšném nalezení hledaného výrazu. V opačném případě k hodnotě

posledního pořadového čísla záznamu se připočte 900 000.

Parametry X, Y jsou z oblasti dimenze daného souboru a určují prostor v daném souboru, na kterém se bude uskutečňovat SEARCH.

Hodnota Y je kontrolována v součinnosti s délkou zadaného výrazu, zda je korektní.

V případě, že by bylo žádoucí hledat v daných záznamech několik výrazů, je třeba tyto oddělit čárkami v jejich obsahu (obdobné jako u PRINT#).

Příklad: @SEARCH# 0, 1, 1, I;"ESLA"

Úkon: Příkaz dává pokyn na vyhledání výrazu ESLA na datovém souboru s pořadovým číslem # 0. Hledání započne od záznamu 1 a od prvku s pořadovým číslem 1 (v každém dalším záznamu).
Pokud by se tento výraz nacházel v záznamu s číslem 50, potom proměnná I obdrží hodnotu 50. V opačném případě pokud má soubor např. 100 záznamů, nabude proměnná I hodnotu 900 100.

Při hledání vícero výrazů v záznamech, by byl tvar příkazu následující:

```
@SEARCH# 0, 0, 0, I; A$+","+B$
```

Poznámka: hledané výrazy A$, B$ se musí nacházet současně v daném pořadí na daném záznamu.
Doporučujeme při použití příkazu @SEARCH# použít za příkazem @OPEN příkaz @READ#, který restauruje parametry souboru.

Použití příkazu @SEARCH# může být vhodné pro přístup k záznamům, které jsou označeny identifikačními řetězci a ne pořadovými čísly.

e) Příkazy pro binární záznamy

Do této třídy patří volání všech diskových služeb, které uchovávají údaje na fyzických sektorech v binárním tvaru (HEXA).
Jsou to:

- zápis a čtení programů BASIC
  Syntax:

```
@STORE "NAME"
@LOAD "NAME"
```

- zápis a čtení videostránky počítače
  Syntax:

```
@GSTORE "NAME"
@GLOAD "NAME"
```

- zápis a čtení binárních dat z operační paměti počítače
  Syntax:

```
@BSTORE "NAME"
@BLOAD "NAME"
```

Při zápisu/čtení binárních dat z RWM je třeba nastavit paměťový prostor na daný transfer dat. Při zápisu se udává i délka tohoto prostoru. Výkonným příkazem je příkaz:

```
@BUF=adr,kolik
```

přičemž "adr" uvádí konkrétní paměťové místo (decimálně) přenosu a parametr "kolik" počet dat, která se budou zapisovat. Tyto parametry mohou být zadány explicitně i implicitně.

Pro čtení binárních dat z diskety postačuje uvést pouze počáteční adresu paměťového prostoru a druhý parametr musí být nula.

Tedy:

```
@BUF=adr,0
```

Poznámka: Doporučujeme používat pro tuto diskovou službu volný paměťový prostor BASICu.
Příkaz @LOAD není možno použít v samotném programu BASIC.

f) Příkazy pro zpracování chybových hlášení

Aby bylo možné v reálném čase zpracovat diskové služby pod programovým řízením a reagovat na jejich odezvy pro každou vzniklou situaci, obsahuje modul ERB dva příkazy pro jejich obsluhu.

Příkaz @ON ERROR uvolní v případě vzniklé diskové chyby programový čítač BASIC na interpretování výrazů, umístěných za tímto příkazem. Stavový registr operačního systému MFS naplní vnitřní proměnnou "ER", která je na tyto účely vyhrazenou hodnotou diskové chyby, takže uživatel má možnost programovými prostředky identifikovat vzniklou chybu a na ni reagovat.

Pozice příkazu @ON ERROR v programovém řádku je následující:

@ON ERROR	standardní příkazy BASIC

Zrušení programového zpracování diskových chyb se uskuteční příkazem:

@OFF ERROR

Příklad: Ukázka zpracovávání diskové chyby pro otevření datového souboru.

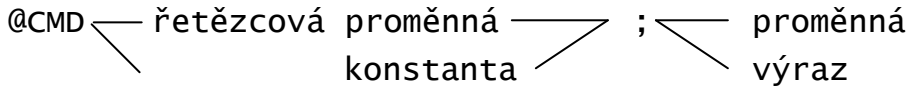
```
  5  @ON ERROR IF ER=12 THEN PRINT "CHYBNY NAZEV"
 10  DISP "ZADEJ NAZEV"
 15  INPUT A$
 20  @OPEN A$ # 0
 30  @OFF ERROR
 40  END
```

Poznámka: Program neřeší jiné chyby, které by při dané diskové službě mohly vzniknout.
V případě, že se uskuteční jiný východ z programu (STOP atd.), nadále se ponechává ukazatel zpracování diskové služby.

B.VIII. Speciální diskové služby

Předcházející kapitola popisovala diskové služby, které byly zabezpečovány systémově od MFS. Vedle těchto příkazů existuje ještě jeden všeobecný příkaz pro práci s diskovou pamětí i přímá komunikace s operačním systémem MFS.
Používání tohoto příkazu vyžaduje znát komunikační protokoly (kapitola V) a organizaci operačního systému MFS.

Syntax tohoto příkazu je:



kde: řetězcová proměnná nebo konstanta představuje danou diskovou službu ve formátu jak bylo uvedeno v kapitole V. Za oddělovačem, čárka se může vyskytnout proměnná, jde-li o službu čtení dat, nebo výraz, jedná-li se o zápisovou funkci a v případě, když nejde o službu, při které nebude transfer dat, je příkaz ukončen tímto oddělovačem (středník).

Volání diskové služby příkazem @CMD se uskutečňuje většinou v takových případech, když chceme obejít systémový přístup od MFS. V podstatě jsou to služby:

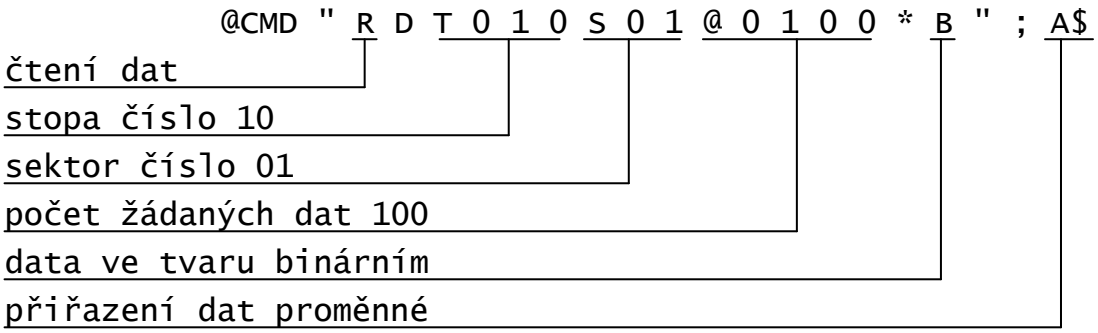
- 1. skupina příkazů, která manipuluje přímo s fyzickými sektory po celé diskové paměti (rozsah 87 kB).
- 2. skupina příkazů pro modifikaci operační paměti RWM.
- 3. kombinované diskové služby.

Dále si tyto skupiny vysvětlíme podrobněji:

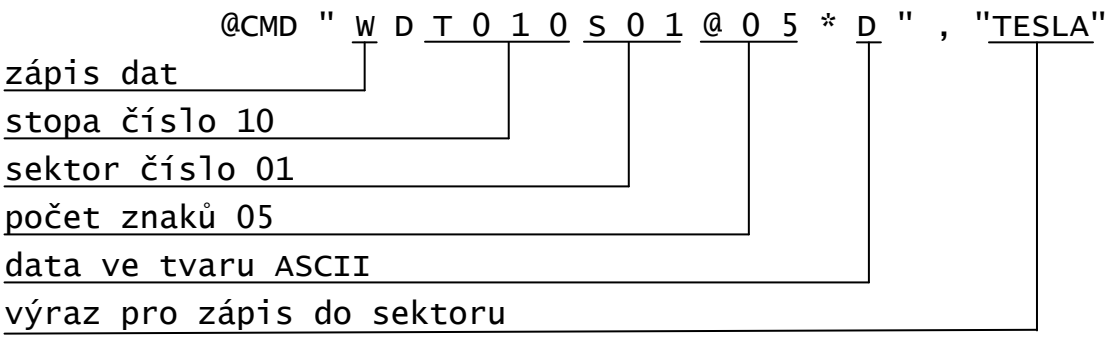
SKUPINA 1:

V této skupině můžeme využít každou agregovanou diskovou službu, avšak nejčastěji se používají následující dvě, které umožňují přečíst nebo zapsat data na libovolný sektor diskety. Je třeba mít na paměti, že na disketě se nachází 40 stop a každá stopa má 18 sektorů, přičemž délka sektoru je 128 byte.

- READ DATA



- WRITE DATA



Pokud si uživatel zkontroluje tvar příkazu @CMD, zjistí, že prostřednictvím něho by bylo možné vykonat libovolnou diskovou službu, avšak na úkor komfortu programového přístupu. Tyto dvě služby, které byly uvedeny, se většinou budou používat pokud požadujeme tzv. "DISK DOCTOR" - odstraňování "nemocných" míst na disketě, způsobených neodborným zásahem nebo když chceme manipulovat s neznámou disketou nebo katalogem, který má utajené názvy, popřípadě zničenu svou část.

Pro doplnění uvádíme, že maximální počet znaků, které lze přiřadit proměnné, je 255, to znamená, že žádáme-li binární přenos, parametr délka nabude hodnoty max. 127 a při datovém ASCII přenosu 255.

Příklad: Zobrazte obsah alokační mapy RJ.

Řešení:

```
POKE 995, 32

@CMD "RDT00S01@0127*B";A$
PRINT A$
```

nastavení displeje na formát 32 znaků na řádku

přečtení sektoru 01 ze stopy 00 do A$ a zobrazení obsahu A$

Poznámka: Předpokládáme-li, že obsah fyzického sektoru není v ASCII znacích, doporučujeme přijímat data jako binární.

SKUPINA 2:

V této skupině se nacházejí tři příkazy, které mohou komunikovat a ovlivňovat činnost operačního systému. Jde o příkazy umožňující na úrovni strojového kódu zahnízďovat vlastní procedury a ty aktivovat. Pro tuto činnost má uživatel vyhrazeny 1,25 kB operační paměti od adresy 1B00 (Hexa). Používání těchto příkazů si podmiňuje znalost programování ve strojovém kódu 8080 a operačního systému MFS.

Skladbu těchto příkazů si vysvětlíme na následujících příkladech:

- ČTENÍ PAMĚTI

```
@CMD " G H & 1 B 0 0 @ 0 0 0 F " ; A$
```

čtení dat z RWM
hodnoty zadané HEXA
adresa 1B00
počet byte 000F
údaje přiřazené proměnné A$

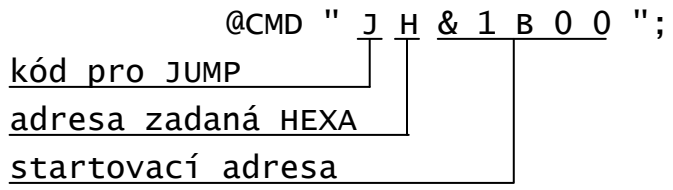
- ZÁPIS DAT DO PAMĚTI

```
@CMD " M H & 1 B 0 0 @ 0 0 0 3 " , "C30000"
```

zápis dat
hodnoty zadané HEXA
adresa 1B00
počet byte 0003
zapisovaná data C30000

Poznámka: Počet zadaných dat při zápisu musí odpovídat počtu vydaných dat při přenosu do jednotky MFD-85. V obsahu zapisovaných dat se nesmí nacházet znak, který nepatří do množiny HEXA znaků (0 - F).

- START PODPROGRAMU



Návrat do komunikačního procesu je vykonáním instrukce RETURN, tedy příkaz má charakter volání podprogramu.

Jak je již teď jasné, můžeme si vytvořit vlastní procedury a ty umístit do volné části paměti RWM a interpretovat. Taktéž to umožňuje vytvořit vlastní operační systém a ten provozovat. Naplnění uživatelské paměti je možné několika cestami, uvedenými příkazy, celý obsah nebo zavedením pouze z daných fyzických sektorů diskety nebo funkcí MASTER MFD-85.

- 3. skupina: Kombinované diskové služby

Pod pojmem kombinované diskové služby rozumíme vyvolání diskové služby, která není agregovaná do jednoho příkazu nebo není možné ji vykonat příkazem @CMD (není evidovaná v sestavě diskových služeb). Takový typ služeb si může uživatel vytvořit podle konkrétní vzniklé situace aplikace. Kombinovaná disková služba se skládá ze dvou příkazů.

První příkaz je obvykle standardní diskový příkaz, kterého úloha je nastavit v operačním systému MFS dané parametry. Druhým příkazem je příkaz @CMD, kterým se odstartuje příslušný podprogram pracující se zavedenými parametry v prvním příkazu. Takto je realizována disková služba, která vykonává naplnění záznamů v daném souboru konstantní větou od zvoleného záznamu. Vysvětlíme si to na následujícím příkladu:

Příklad:
```
@PRINT# 0, 50;"TESLA"
@CMD "JH&0B58“;
```

Vykonáním těchto dvou příkazů zabezpečíme, že datový soubor s pořadovým číslem #0 bude mít od 50 záznamu až do posledního zapsán řetězec znaků TESLA. Úkon se bude vykonávat max. rychlostí pod řízením MFS.

Nastavení parametrů jsme uskutečnili příkazem PRINT# a vlastní vykonání exekutivou s adresou 0B58. Tato exekutiva je součást operačního systému, avšak lze takovéto exekutivy vytvořit speciálně pro daný požadavek přímo uživatelem.

B.IX. Popis podprogramů operačního systéme MFS

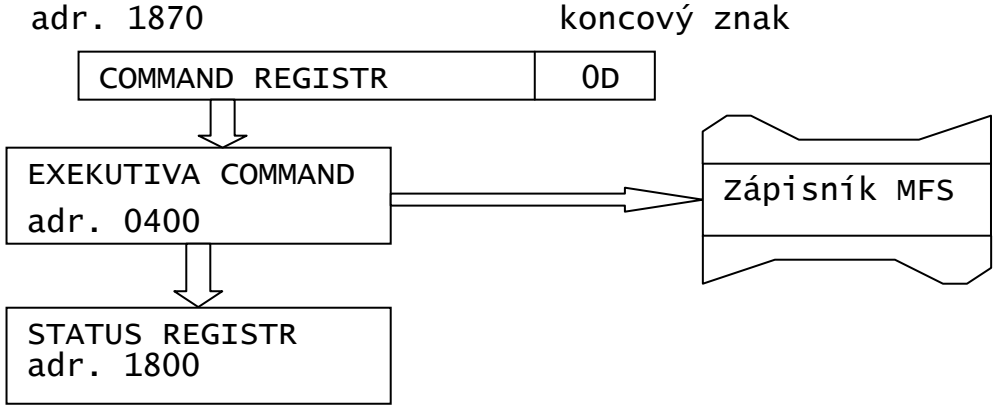
Cílem této kapitoly je seznámit s některými podprogramy operačního systému MFS, které by mohl uživatel využít pro stavbu vlastních procedur. Jsou to:

- Podprogram COMMAND

Nastavuje parametry v zápisníku MFS, které jsou potřebné k vykonání požadované diskové služby, jde o velmi agregovanou proceduru. Vstupními údaji je obsah COMMAND registru a výstupními jsou nastavení daných parametrů a registru STATUS.

- vstupní údaje - COMMAND REGISTR: adr. 1870 Hexa
- volání exekutivy COMMAND; adr. 0400 Hexa
- výstup a) nastavené parametry v zápisníku MFS
  b) STATUS REGISTR; adr. 1800 Hexa

Následující schéma znázorňuje vztah jednotlivých úkonů při podprogramu COMMAND.



Obsah COMMAND registru udává o jakou diskovou službu půjde. Jeho tvar je dán podle konkrétní služby a byl uveden v souhrnné tabulce v kapitole V.

Po vykonání exekutivy COMMAND je třeba testovat obsah registru STATUS, v případě, že je 0, 40 nebo 80 Hexa, můžeme přistoupit k realizaci dané služby, vykonáním následujícího podprogramu, jinak jeho obsah udává o jakou chybu jde (viz kapitola 5).

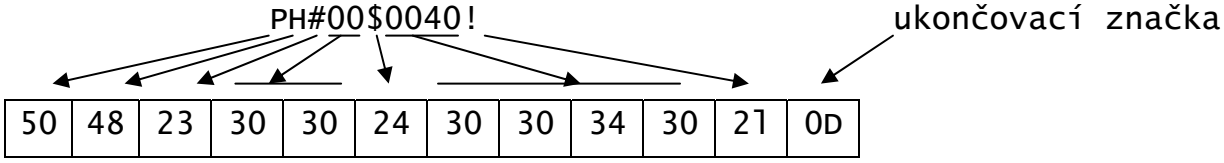
- Podprogram EXECUTE

Tento podprogram vykonává danou diskovou službu. Startuje se vždy po interpretaci podprogramu COMMAND, protože on zaručuje korektnost parametrů v dané službě. Volání podprogramu EXECUTE je na adrese 04B9 Hexa. Správnost vykonání procedury je oznamována prostřednictvím STATUS registru. Jeho testování je obdoba jako pro proceduru COMMAND.

Pro doplnění obrazu používání těchto dvou procedur uživatelem v jeho systémovém programu uvádíme následující příklad:

Příklad: Zapište obsah komunikačního bufferu HPIB (adr. 1A00 Hexa) do otevřeného souboru s pořadovým číslem 0 a záznamu s pořadovým číslem 64.

Řešení: 1. krok: Vytvoříme si pro zápis na logický sektor příslušný COMMAND s požadovanými parametry, tedy:



2. krok: Obsah tohoto COMMAND slova umístíme do jeho registru; adr. 1870 Hexa

3. krok: Vykonáme podprogram COMMAND

4. krok: Testujeme obsah STATUS registru, který v našem případě musí mít hodnotu 80 Hexa (128 dec.), což udává, že všechny parametry jsou správně nastaveny a může se přistoupit k výkonné fázi

5. krok: Vykonáme podprogram EXECUTE

6. krok: Testujeme opět registr STATUS. Jeho obsah musí mít v našem případě hodnotu nula, protože disková služba je ukončena. V opačném případě jeho hodnota udává konkrétní chybu při dané službě.

Poznámka: Doporučujeme zadávat parametry COMMAND slova v hodnotách HEXA. Všeobecně platí, že u těch parametrů, které nabývají hodnoty v rozsahu 0 - 255, je tvar v hexa formátu 00 - FF, tj. : zabírají dvě pozice ASCII znaků.

- Podprogram DISK - vykonává základní úkony s diskem. Daný úkon je uložen do zápisníku jako vstupní argument a představuje vlastně adresní vektor exekutivy. Parametry stopa/sektor se ukládají do zvláštních buněk zápisníku MFS. Ukončení úkonu prošetříme testováním STATUS registru, viz předcházející podprogramy.

vstup: a) úkon disku na adr. 1809 a 180A jako adresní vektor (adresa L, H). Přiřazení adresního vektoru exekutivám znázorňuje následující tabulka:

úkon	Read	Write	Track 0	Verify
1809	2B	14	0F	17
180A	06	07	06	07

b) Vstupním prostorem pro data je DISK buffer délky 128 byte začínající na adrese 1980 Hexa.

c) Nastavení stopy (TRACK) adr. buňka 1825 Hexa
nastavení sektoru adr. buňka 1826 Hexa

Volání podprogramu DISK: 074A Hexa

výstup: a) provedení úkonu na disku
b) naplnění výstupního DISK buffer na adrese 1980 Hexa

Příklad použití podprogramu DISK:

Zadání: Přečtěte obsah sektoru 02 stopy 08.

Řešení:
```
LXI  H,062B        nastavení úkonu READ
SHLD 1809
LXI  H,0208        nastavení stopa/sektor
SHLD 1825
CALL 074A          exekutiva DISK
LDA  1800
...
```

Poznámka: V případě, že uživatel chce měnit lokace DISK BUFFER, může to učinit modifikací jeho ukazatele na adrese 180B. Systém tento ukazatel nastavuje na hodnotu 1980 Hexa.

- podprogram MOVE - zabezpečuje přesun dat určených registrovými páry DE a HL.

vstup:
```
registr DE     ; odkud
registr HL     ; kam
registr B      ; počet
```
Volání: 05CC Hexa
výstup: data v adresním prostoru určeném registrem HL

- podprogram MASTER - tento podprogram zabezpečuje současně několik služeb. Umožňuje přečíst do zvolené uživatelské části operační paměti MFS zadaný libovolný sektor z diskety a uskutečnit jeho interpretaci ve strojovém kódu. Startovací vektor je dán první ukládací adresou.

vstup:
a) ukazatel ukládací adresy – 180B Hexa
b) nastavení stopy adr. 1825
c) nastavení sektoru adr. 1826

volání: 0956 Hexa

Poznámka: V případě, že dojde k chybnému načtení sektoru z diskety nebo jiné závadě během transferu dat, není možná další činnost interpretace podprogramu a systém je třeba inicializovat tlačítkem RESET. Jestliže má uživatel vytvořené okolí na identifikaci závady, je třeba podprogram MASTER vytvořit z následující sekvence instrukcí:

```
MASTER*:  SHLD 1825    - nastavení STOPA/SEKTOR
          CALL 032D    - načtení dat a diskety
          JNZ  ERROR   - testování chyby přenosu
          LHLD 180B    - adresa interpretace do HL
          PCHL         - skok na řízení podle načítaného sektoru
```

- NÁVRAT DO FUNKCE SLAVE

Návrat nebo přechod do funkce SLAVE jednotky MFD-85 je možno vyvoláním skoku na adresu 0C00. Od tohoto okamžiku se jednotka chová jako podřízená a komunikuje po sběrnici IMS-2 s řídícím počítačem. Pokud bychom požadovali z této činnosti návrat k MASTER, postačí vyvolat diskovou službu

```
@CMD "JH&adr.";
```

- Podprogram VERIFY DISK: úkolem podprogramu je provést verifikaci celé diskety. Podprogram nepožaduje žádné vstupní hodnoty.

volání: adr. 0A1C (Hexa)

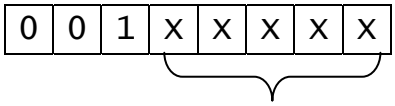
Poznámka: Podprogram můžeme aktivovat taktéž systémovým příkazem @CMD, tedy:
@CMD "JH&0A1C"; nám provede kontrolu celé diskety.

Dosud byly vysvětleny podprogramy, které nevyžadovaly styk s okolím. Uvažovaly vždy transfer dat ze stykového bufferu. Nyní si vysvětlíme, jak lze tento datový buffer naplnit. Existují dvě cesty přístupu do bufferu. Jde o transfer dat prostřednictvím lokální sběrnice, popřípadě s podporou obvodu 8255A, který se nevyužívá ve funkci styku pro sběrnici IMS-2 nebo transfer dat po sběrnici IMS-2. Jakou podporu uživateli poskytuje operační systém MFS si vysvětlíme následovně:

1. V prvním způsobu jde o běžný komunikační kanál realizovaný uživatelským programem a podporovaný standardními I/O prvky (8255A, 8251 atd.). Jejich obsluhu musí uživatel vytvořit pro konkrétní aplikaci.

2. Druhý způsob - řízení sběru dat po sběrnici IMS-2 poskytuje velmi všestranný styk s měřícími přístroji. Operační systém obsahuje podprogramy pro funkci jednotky MFD-85 jako CONTROLLER.

   Jsou to následující procedury:
   a) INIT - inicializace sběrnice IMS-2
      volání: 0E9F (Hexa)
   b) LISTENER - výstup dat na zvolené zařízení. Zabezpečuje vlastní naadresování příslušného zařízení na sběrnici IMS-2 hodnotou zadanou v akumulátoru. Tato hodnota se získá následovně:



vlastní adresa zařízení LISTENER

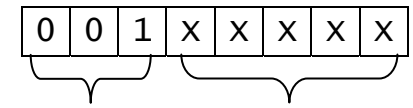
   volání: adresa 0EA8 (Hexa)

Poznámka: V případě, že požadujeme měnit komunikační buffer dat, změníme hodnotu na jeho ukazateli (adr. 180D). Délka tohoto bufferu je předpokládaná v rozsahu 256 byte od adresy 1A00, avšak uživatel může využít další prostor, protože už zasahuje do uživatelské části paměti.
Poznamenáváme, že data musí být ukončena znaky CRLF (0D 0A Hexa).

Příklad: Demonstrace příkazu LISTENER

Zadání: Vytvořte sekvenci volání podprogramů pro výstup dat na tiskárnu s adresou 02.

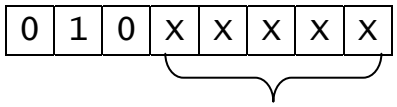
Řešení: adresa zařízení (tiskárna) je 02, jeho doplněním + 20 HEXA získáme adresní parametr 22 (HEXA) pro proceduru LISTENER, který uložíme do akumulátoru.




```
Potom:   CALL 0E9F    - inicializace IMS-2
         MOV  A,22    - adresa tiskárny
         CALL 0EA8    - exekutiva LISTENER
```

nám provede naadresování tiskárny a výstup dat z HPIB bufferu od adresy 1A00 po znak CRLF.

c) TALKER - vstup dat ze zvoleného zařízení. Jde o obdobnou proceduru jako LISTENER, přičemž adresa nabývá tvaru



volání: 0F40

Poznámka: dtto jako u procedury LISTENER

Na závěr této kapitoly uvádíme adresaci I/O podsystému řídícího mikropočítače pro možnosti externího rozšíření uživatelem.

Následující tabulka znázorňuje obsazenost příslušných adresních I/O linek.

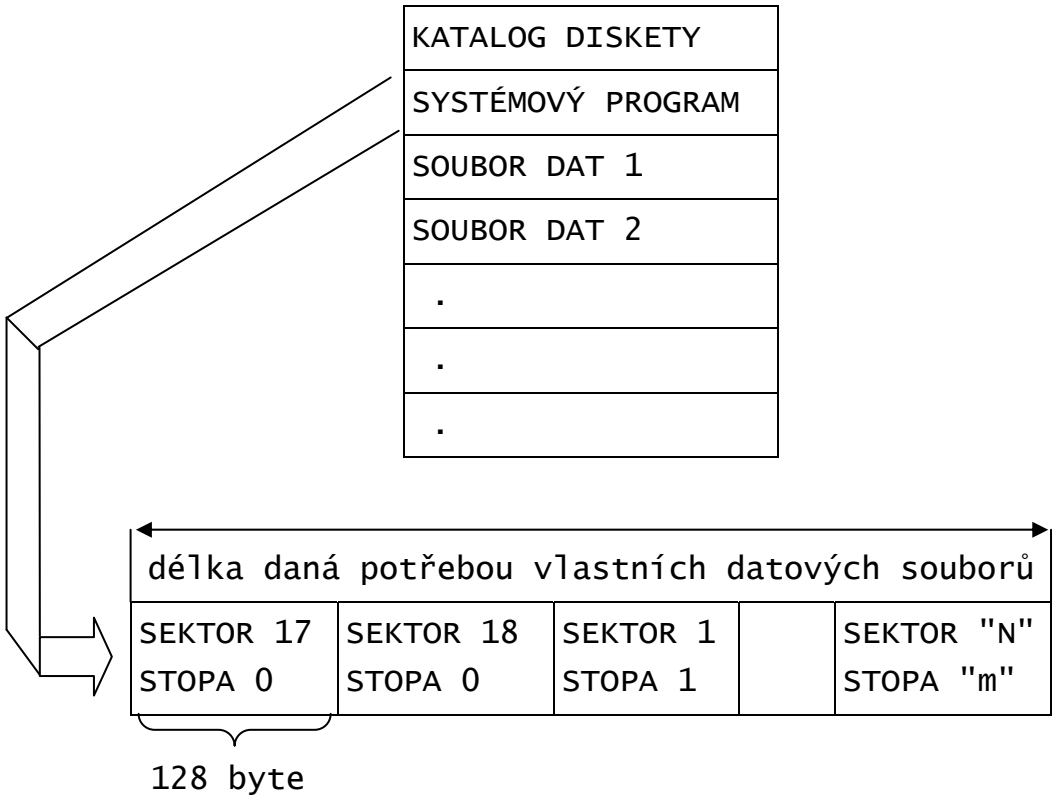
A7	A6	A5	A4	A3	A2	A1	A0	význam
X	1	0	1	X	X	X	X	8255A (sběrnice IMS-2)
X	1	1	0	X	X	X	X	registr 8271
X	0	1	1	X	X	X	X	DMA 8271
X	1	1	1	X	X	X	X	uživatel

Poznámka: Vzhledem k tomu, že lokální sběrnice MFD-85 je řešena oboustranně, je třeba provést požadované přepojky na konektoru lokální sběrnice. Potřebné informace najde uživatel v příloze této publikace, kde je popisován daný konektor.

B.X. Systémová disketa

Z dosud uvedených informací je jasné, že operační paměť MFD-85 můžeme modifikovat prostřednictvím dvou cest. Jedna cesta je klasická podporovaná kanálem POČÍTAČ - MFD-85 a druhá je prostřednictvím SYSTÉMOVÉ diskety ve funkci MASTER:
Systémovou disketu může uživatel získat i vlastní produkcí.

Pro její konstrukci je nutno ještě uvést některé údaje, následující obrázek znázorňuje rozložení souborů/záznamů, při jejím vytváření.



Obsah sektoru 17 na stopě 0 má zvláštní postavení v systémovém programu. Tento sektor se automaticky při zvolené funkci MASTER a stisknutí tlačítka (nebo generováním signálu) RESET natahuje do paměťového prostoru DISK buffer (adresa 1980) a interpretuje. Proto je třeba, aby uživatel na tento sektor umístil tzv. LOADER, který umožní "natahování" dělicích sektorů k jejich interpretování.

Je výhodné při sestavování systémového programu shromáždit nejvíc používané podprogramy uživatelského systémového programu do volné uživatelské operační pamětí (prostor od adr. 1C00 do 1FFF) a tyto potom vyvolávat ve svých fragmentech (umístěných na jednotlivých sektorech) délky 128 byte.

Takovýto přístup umožňuje potom přepnutí funkce MFD-85 do SLAVE a tyto podprogramy využívat jako speciální diskové služby volané počítačem.

Systémovou disketu vytváří ve spojení POČÍTAČ - MFD-85 uživatel. Je nutné, aby při této činnosti byla disketa "naformátována" pro dané datové soubory v tomto pořadí, jak uvádí předcházející obrázek.

Vytvoření systémového programu můžeme v podstatě dvěma způsoby:

- přímo v operační paměti počítače PMD-85 ve sledu 128 bytových fragmentů. Celý obsah těchto fragmentů se zapíše na disketu, která je čistá (po inicializaci) diskovým příkazem

```
@BSTORE "SYSTEM"
```

- vytvořením datového souboru v požadovaném rozsahu s počtem prvků v každém záznamu 128 byte, čímž vyhradíme prostor pro systémový program.

  Naplnění sektorů (záznamů) uskutečňujeme příkazem

```
@CMD "WDT00S017@0127*D";"DATA"
```

atd.

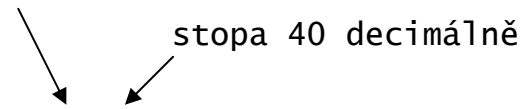
Pro praktické použití se lépe hodí první způsob, t.j. přímo v operační paměti a příkazem @BSTORE. Poznamenáváme, že je nutné předem vhodně zvolit délku programového bufferu (příkaz @BUF=), vzhledem k dalším změnám v systémovém programu. Po této činnosti si vytvoříme požadované datové soubory, se kterými bude pracovat systémový program.

Závěrem této kapitoly si uvedeme jednoduchý příklad pro sestavení systémové diskety. Příklad má řešit vytvoření tzv. testovací diskety, která má bez vnějšího vlivu zjistit správnost chodu jednotky MFD-85.

Zadání: Vytvořte programový fragment systémové diskety, která bude uskutečňovat pohyb hlavy disku ze stopy 0 na poslední stopu. Řídící program k poslední stopě a sektoru ať je uložen na sektoru 17/stopě 0 a řídící program návratu k stopě 0/sektoru 17 na sektoru 18/stopě 0.

Řešení: Je třeba sestavit dva 128 byte bloky. První blok bude zapsán na sektor 17/stopa 0 a bude realizovat skok na poslední stopu a druhý blok bude na následujícím sektoru a bude řešit návrat na stopu 0.

Takže jednotlivé bloky budou mít sekvenci instrukcí:




```
BLOK I:     LXI H,12 27         21 27 12    ; nastavení stopy
            JMP MASTER          C3 56 09    ; a sektoru

BLOK II:    LXI H,10 00         21 00 10
            JMP MASTER          C3 56 09
```

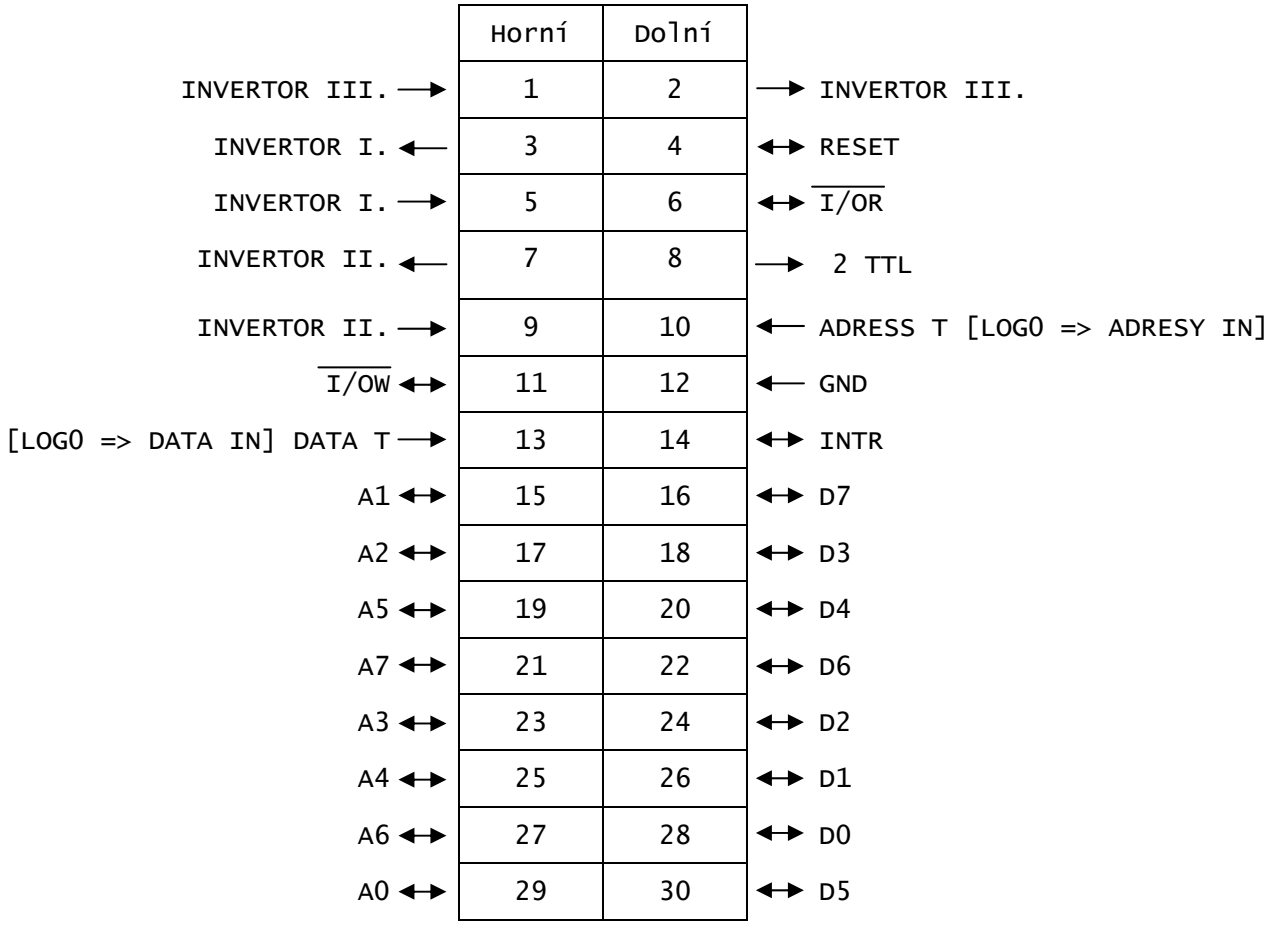
V dalším kroku zapíšeme jednotlivé bloky na dané sektory, tedy:

```
BLOK I:     @CMD "WDT00S017@06*B"; "212712C35609"
BLOK II:    @CMD "WDT00S018@06*B"; "210010C35609"
BLOK II*:   @CMD "WDT040S017@06*B"; "210010C35609"
```

Potom přepneme do funkce MASTER a stiskneme tlačítko RESET. Od tohoto okamžiku se bude pohybovat hlava mezi zadanými stopami.

Poznámka: Blok II* jsme zapsali i na poslední stopu/sektor, aby bylo možné nepřetržitě pohybovat hlavou mezi krajními stopami. Tento úkol může provést i samotná systémová disketa.

B.XI. Příloha č. 1 - Konektor lokální sběrnice



	Horní	Dolní	
INVERTOR III. →	1	2	→ INVERTOR III.
INVERTOR I. ←	3	4	↔ RESET
INVERTOR I. →	5	6	↔ $\overline{\text{I/OR}}$
INVERTOR II. ←	7	8	→ 2 TTL
INVERTOR II. →	9	10	← ADRESS T [LOG0 => ADRESY IN]
$\overline{\text{I/OW}}$ ↔	11	12	← GND
[LOG0 => DATA IN] DATA T →	13	14	↔ INTR
A1 ↔	15	16	↔ D7
A2 ↔	17	18	↔ D3
A5 ↔	19	20	↔ D4
A7 ↔	21	22	↔ D6
A3 ↔	23	24	↔ D2
A4 ↔	25	26	↔ D1
A6 ↔	27	28	↔ D0
A0 ↔	29	30	↔ D5

Pro využití obvodů 8271 a 8255A jako stykových obvodů externího počítače je třeba provést přepojky:

10 – 12 ... nastavení směru adresy
6 - 1; 2 – 13 ... ovládání směru dat podle signálu I/OR

Poznámka: Invertory I. až III. jsou vyvedeny na volné použití pro případ, že některé řídící signály je třeba negovat.

Příloha č. 2 - Zásady při práci s jednotkou MFD-85

Pro bezporuchovou činnost a ochranu informací na magnetickém disku je třeba dodržet následující pokyny:

1) Zapínat jednotku MFD-85 při vytažené disketě
2) Nerozpojovat komunikační kanál (kabel) IMS-2 během zapnuté jednotky
3) Nevypínat jednotku MFD-85 ve stavu vložené diskety.
4) V případě zablokování činnosti MFD-85 vynulovat počítač a samotnou jednotku.
5) Dodržovat zásady ochrany před poškozením vlastní diskety, které jsou uvedeny na obale diskety.

Příloha č. 3 - Hexa výpis EXTENDED ROM BASIC

```
0000  E5 21 F0 5E 22 12 22 23 22 5C 08 21 C0 80 22 60
0010  1B 21 67 76 22 F8 19 21 60 76 22 09 20 3E CD 32
0020  AD 02 21 7A 76 22 AE 02 21 91 76 22 3D 04 22 CC
0030  03 CD 00 8C 00 24 00 09 00 76 3E 2C 32 83 08 21
0040  DA 0F 22 19 15 21 D1 7B 22 E6 20 3E C3 32 03 0F
0050  00 00 00 CD DB 7D E1 C9 FF FF 3E 41 D3 7E C9 FF
0060  4F CD 46 20 C3 E1 1F D1 B7 F2 FF 00 D5 07 E6 FE
0070  4F 06 00 EB 21 C0 77 C3 E0 04 12 13 0C FE E2 E5
0080  21 F9 19 22 59 02 E1 C0 E5 21 FF 76 22 59 02 E1
0090  C9 FE C0 E5 21 FA 19 22 30 04 22 BF 03 E1 C2 D9
00A0  03 E5 21 00 77 22 30 04 22 BF 03 E1 C3 D9 03 CD
00B0  D7 16 B7 CA 3D 09 FE 11 D2 3D 09 47 F6 20 32 67
00C0  78 C6 10 32 6F 78 78 F6 40 32 6B 78 C6 10 32 73
00D0  78 C9 CD 75 00 E5 21 F1 5E 22 BE 77 21 E7 76 E5
00E0  E5 21 EB 20 C3 C9 78 21 F0 76 22 5F 08 C3 1B 08
00F0  2B C9 41 48 0D 0A E5 21 F2 76 CD 3C 78 E1 C9 C9

0100  C3 41 54 C4 49 53 4B 3D CE 41 4D 45 D0 55 52 47
0110  45 C9 4E 49 54 C3 52 45 41 54 45 FF 7F 7F 7F 7F
0120  C3 4C 4F 53 45 23 C3 4C 4F 53 45 D0 52 49 4E 54
0130  23 D2 45 41 44 23 D3 45 43 55 52 45 D7 52 49 54
0140  45 C4 4F 4B 45 D3 7F 7F 7F 7F 7F C7 53 54 4F 52
0150  45 C7 4C 4F 41 44 C2 53 54 4F 52 45 C2 4C 4F 41
0160  44 D3 54 4F 52 45 CC 4F 41 44 C3 4F 50 59 C3 4D
0170  44 C7 45 54 C2 55 46 3D CF 50 45 4E CF 4E 20 45
0180  52 52 4F 52 CF 46 46 20 45 52 52 4F 52 C5 58 45
0190  43 D3 45 41 52 43 48 23 FF FF 80 CD F4 78 3E 0D
01A0  12 13 3E 0A 12 CD 6D 00 3B E5 21 EC 79 22 EF 78
01B0  21 10 7A 22 97 78 21 F1 5E C3 08 78 5E C3 F1 5E
01C0  D6 78 AF 76 BA 78 2A 79 27 79 6D 79 FF FF 4A 7A
01D0  F6 76 BA 79 04 7A 2D 79 FF FF 4D 7D FF FF 93 7B
01E0  C1 7A AE 7B ED 7A 5D 7B F4 7A 24 7C 9B 77 FF FF
01F0  60 7D 2E 7A 9B 7C AD 7C 7F 7D A9 7D FF FF FF FF

0200  FF FF FF FF FF FF FF FF CD 3C 78 CD 4A 78 2A BE
0210  77 CA 28 78 FE 40 CA E0 78 FE 80 CA 88 78 21 75
0220  78 22 74 C0 CD 89 8A 3A E1 C9 C3 18 01 2A BE 77
0230  7E CD 60 76 23 79 FE 0A C2 30 78 C9 CD 42 20 E5
0240  21 65 78 CD B5 1F E1 C3 30 78 CD 42 20 21 69 78
0250  CD B5 1F CD 5A 76 21 83 78 E5 CD EB 20 E1 CD D7
0260  16 36 0D B7 C9 3F 55 21 00 3F 35 41 00 3F 55 31
0270  00 3F 35 51 00 2A 2A 20 44 49 53 4B 20 45 52 52
0280  20 3D 20 30 30 30 0D 0A E5 21 71 78 CD 42 20 CD
0290  B5 1F CD 5A 76 E1 CD A1 78 C3 0B 78 FF FF FF FF
02A0  FF CD EB 20 2A BE 77 7E B7 CA 56 7A CD 00 85 23
02B0  C3 A7 78 FF FF FF 4E 44 0D 0A FE 3B CA D2 76 E5
02C0  21 F1 5E 22 BE 77 21 A1 78 22 97 78 21 B6 78 C3
```

```
02D0  08 78 44 44 0D 0A CD BF 78 E5 21 D2 78 C3 08 78
02E0  E5 00 00 00 CD 42 20 21 6D 78 CD B5 1F E1 CD 2D
02F0  78 C3 0B 78 7E FE 22 CA 14 79 CD 07 0B E5 1A 47

0300  13 13 1A 6F 13 1A 67 11 F1 5E 7E 12 23 13 05 C2
0310  0A 79 E1 C9 11 F1 5E 23 7E B7 CA FF 00 FE 22 CA
0320  75 00 12 13 C3 17 79 3E 46 01 3E 45 01 3E 42 01
0330  4A 79 C5 32 EE 5E CD F4 78 C1 CD 4D 79 E5 21 F0
0340  5E 36 27 2B 36 48 2B C3 08 78 27 0D 0A 0A 12 FE
0350  0A 03 13 C2 4D 79 C9 3E 24 12 13 D5 CD 33 09 CD
0360  C3 05 D5 C1 D1 E5 C5 E1 CD 98 79 E1 C9 3E 43 32
0370  EE 5E CD F4 78 3E 27 12 13 CD 6D 00 3B CD 57 79
0380  CD 6D 00 2C CD A8 79 01 8D 79 C3 3A 79 2A 44 0D
0390  0A 2A 7A 5E 01 04 DC 09 EB 7A CD 3B 81 23 7B CD
03A0  3B 81 23 EB 22 BC 77 C9 3E 40 12 13 D5 CD D7 16
03B0  D1 E5 26 00 6F CD 98 79 E1 C9 3E 50 E5 21 EC 79
03C0  22 EF 78 21 EE 5E 77 23 36 48 23 36 23 EB E1 01
03D0  01 7A C5 CD AB 79 CD 6D 00 2C CD 57 79 CD 6D 00
03E0  3B C1 CD 4D 79 E5 21 EE 5E C3 08 78 C1 21 60 76
03F0  22 15 00 E1 CD 76 00 CD 44 07 E5 C3 0B 78 FF FF

0400  78 21 0D 0A 3E 49 E5 21 10 7A 22 97 78 C3 C3 79
0410  21 F1 5E CD EB 20 C1 21 E7 76 E5 E5 C3 0B 78 E5
0420  21 EE 5E 70 23 36 48 23 71 EB E1 C3 AB 79 3E 4F
0430  32 EE 5E CD F4 78 CD 6D 00 23 3E 27 12 13 3E 23
0440  12 CD AB 79 01 02 7A C3 3A 79 01 23 55 CD 1F 7A
0450  01 02 7A C3 E2 79 3E 0D C3 00 85 3E 49 D3 7E DB
0460  7D E6 02 CA 5F 7A DB 7C F5 3E 45 D3 7E DB 7D E6
0470  02 C2 6D 7A 3E 4D D3 7E F1 C9 2A BE 77 3E 92 D3
0480  7F 3E 4D D3 7E CD 5B 7A FE 0D CA 5B 7A CD E0 80
0490  07 07 07 07 77 CD 5B 7A CD E0 80 B6 77 23 7D E6
04A0  30 FE 30 CC B1 7A 22 BE 77 C3 85 7A 7D F6 0F 6F
04B0  23 C9 27 2A 42 0D 0A 27 2A 50 0D 0A 27 2A 47 0D
04C0  0A E5 21 00 C0 01 BC 7A 22 BE 77 21 AC 7A 22 A4
04D0  7A 3E 4C 21 7A 7A 22 97 78 E1 C3 32 79 E5 21 00
04E0  24 01 B7 7A 22 BE 77 21 B1 7A C3 CE 7A E5 01 B2
04F0  7A C3 E7 7A CD DD 7A E5 C3 04 1E 2A 7A 5E 01 04

0500  DC 09 EB 7A CD 3B 81 23 7B CD 3B 81 23 EB 22 BC
0510  77 C9 01 80 00 2A BC 77 11 80 FF CD A9 7B DA 25
0520  7B 4D 21 80 00 19 22 BC 77 41 2A BE 77 7E 0F 0F
0530  0F 0F E6 0F CD 4F 81 CD 60 76 7E E6 0F CD 4F 81
0540  CD 60 76 23 7D E6 30 FE 30 CC B1 7A 22 BE 77 05
0550  C2 2D 7B 3E 0D CD 60 76 3E 0A C3 60 76 E5 21 00
0560  24 22 BE 77 21 B1 7A 22 4A 7B 01 B8 7A 21 FB 7A
0570  22 8D 7B 21 12 7B 22 EF 78 E1 3E 5A 32 EE 5E C5
0580  CD F4 78 3E 27 12 13 E5 3E 24 12 13 CD FB 7A E1
0590  C3 39 79 E5 21 00 C0 22 BE 77 21 AC 7A 01 BD 7A
05A0  22 4A 7B 21 BD 7B C3 70 7B 79 95 78 9C C9 E5 01
```

```
05B0  B3 7A 21 B7 7B C3 70 7B 2A BC 77 C3 02 7B 21 00
05C0  2E C3 02 7B 21 F1 5E CD EB 20 23 CD AD 06 C3 0B
05D0  78 2F C3 CD 20 D7 5E C5 3E 0E 32 E2 7B E5 06 04
05E0  7E CD 0E 7C F5 79 17 4F F1 79 17 4F 19 05 C2 E0
05F0  7B 2A D5 7B 71 23 22 D5 7B 2A E2 7B 2B 22 E2 7B

0600  7E B7 E1 C2 DD 7B C1 C9 00 0F 0F 0F 0F 0F 0F C9
0610  DB 4E E6 08 CA 10 7C 7E 2F D3 4C C9 1B 26 6C 36
0620  44 0A FF FF E5 21 00 C0 3E A0 D3 4F 3E 0D D3 4F
0630  0E 3D 11 40 00 06 30 CD D7 7B 05 23 C2 37 7C 00
0640  00 E5 CD 7E 7C 21 D7 5E 22 D5 7B CD 10 7C CD 10
0650  7C 23 7C FE 5F C2 4B 7C 7D FE F7 C2 4B 7C 21 7D
0660  7C CD 10 7C E1 19 19 19 19 2E 00 CD 74 8C C3 8C
0670  7C 1B 26 6C 39 44 1B 2A 62 35 37 36 47 0A 21 71
0680  7C CD 10 7C 23 7E FE 0A C2 81 7C C9 CA 93 7C 0D
0690  C2 35 7C 21 1C 7C CD 81 7C E1 C9 E5 22 FF 79 21
06A0  B6 7C 22 25 78 E1 7E B7 C8 23 C3 A6 7C E5 21 89
06B0  8A 22 25 78 E1 C9 E1 E1 E1 21 7D 78 CD 07 0B 23
06C0  CD AD 06 2A FF 79 2B C3 C6 04 D1 12 CD 76 00 C8
06D0  13 C3 FB 0E 23 CD E3 1C E5 2A 7A C1 22 3E C0 E1
06E0  CD 6D 00 29 C3 4E 07 E5 21 00 7F 22 E7 22 22 F8
06F0  22 E1 CD 76 00 FE 28 C2 D5 22 23 CD 16 7D EB 22

0700  E7 22 EB CD 6D 00 2C CD 16 7D EB 22 F8 22 EB CD
0710  6D 00 29 C3 D5 22 CD 33 09 C3 C3 05 CD 75 00 CD
0720  07 0B 4B 7A 22 74 5E CD BB 0C C1 C3 55 09 CD 75
0730  00 CD 16 7D 1A 4F 13 1A C3 24 7D FE 97 CA 2E 7D
0740  FE B4 CA 1C 7D FE E2 CA 75 7D C3 F3 09 CD 16 7D
0750  D5 CD 6D 00 2C CD 16 7D 42 7B D1 12 13 78 12 C9
0760  CD 16 7D EB 22 BE 77 EB CD 6D 00 2C CD 16 7D EB
0770  22 BC 77 EB C9 23 CD 09 81 DA 3D 09 C3 22 7D F1
0780  CD 76 00 CD F4 78 E5 AF 12 11 F1 5E D5 E1 E5 CD
0790  2D 02 E1 7E CD CE 04 E1 CD 00 05 22 72 5E 7E FE
07A0  2C 23 CA 80 7D 2B C3 A9 04 E5 21 EC 79 22 EF 78
07B0  21 D3 7D 22 97 78 E1 01 23 48 CD 1F 7A 23 CD 57
07C0  79 23 CD A8 79 23 01 02 7A CD 4D 79 CD 07 0B D5
07D0  C3 E5 79 C1 E1 D1 E5 C3 C4 7B FF 21 CA 7C 22 04
07E0  0F 21 D4 7C 22 58 07 21 E7 7C 22 F4 19 21 C4 45
07F0  22 52 1A 21 45 4B 22 54 1A 21 3B 7D 22 B7 22 C9
```

OSOBNÍ MIKROPOČÍTAČ PMD 85. EXTENDED ROM BASIC.
Uživatelská příručka.
Autor Ing. Roman Kišš, k.p. TESLA Piešťany.
Lektor Ing. Josef Truxa.

Edice elektroniky Svazarmu, řada 5 - Výpočetní technika.
Obálka a grafická úprava Josef Svoboda.
Technická redakce Daniela Prokopová.
Odpovědný redaktor PhDr. Jindřich Jirka.
Vydal ÚV Svazarmu v 602. základní organizaci Svazarmu,
Dr. Z. Wintra 8, 160 41 Praha 6 pro potřeby vlastního aktivu.

Náklad 2050 výtisků.
Vytiskly Tiskařské závody Praha. Praha 1988.

TZ 66-13766-88-0

Závěrečné poznámky:

- při finální úpravě textu do elektronické podoby byly provedeny opravy překlepů nacházející se v původní publikaci
- byly opraveny ukázkové programy, chyby v tabulkách apod.
- byl doplněn HEXA výpis ERB, který v původní publikaci chyběl
- procedura záhlaví "ROM 0" je ve skutečnosti obšírnější a zahrnuje i modifikaci exekutivy některých stávajících příkazů BASIG-G (viz. HEXA výpis)
- u příkazu @DISK= se nesmí použít adresy 15 a 16, protože způsobují konflikt se stykovými povely IMS-2 sběrnice. Na jednotce MFD-85 se nastavuje adresa v rozsahu 0 až 15 a adresa 15 se taky nesmí z uvedeného důvodu nastavit. Platnou adresu 0, paradoxně, neumožňuje použít samotný příkaz @DISK=.
- v části „Příkazy pro práci s katalogem“ na stranách 64 a 65, chybí popis příkazu @SECURE A$ pro utajení souboru – soubor se nezobrazí v katalogu. A$ je jméno souboru, stejně jako u příkazu @PURGE. Příkaz @SECURE funguje jako přepínač a teda první vykonání soubor utají, druhé ho znovu „zviditelní“.

Převod do elektronické podoby :

Skenování, finální formátování textu, opravy:	Roman Bórik
OCR, formátování textu, úprava obrázků:	Lukáš Macura
Finální úprava do elektronické podoby:	Martin Bórik

2007