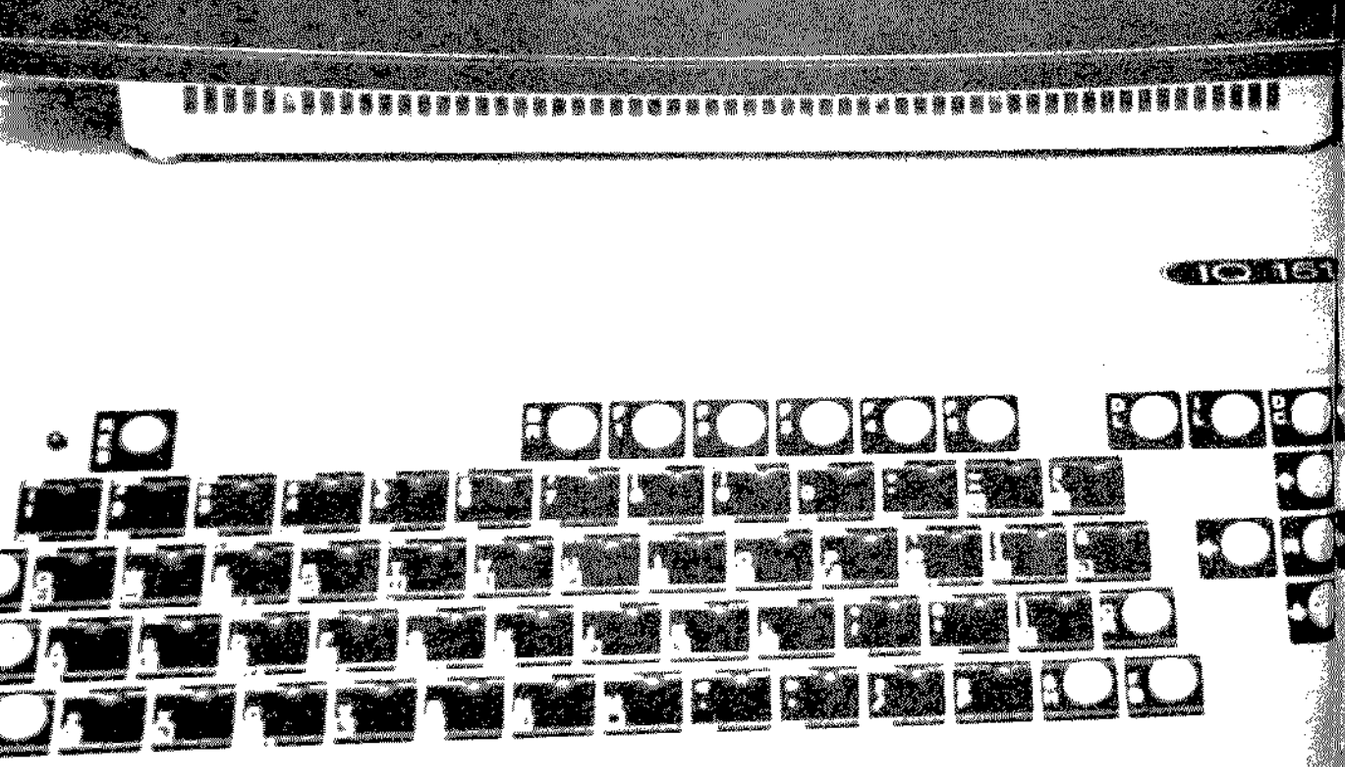
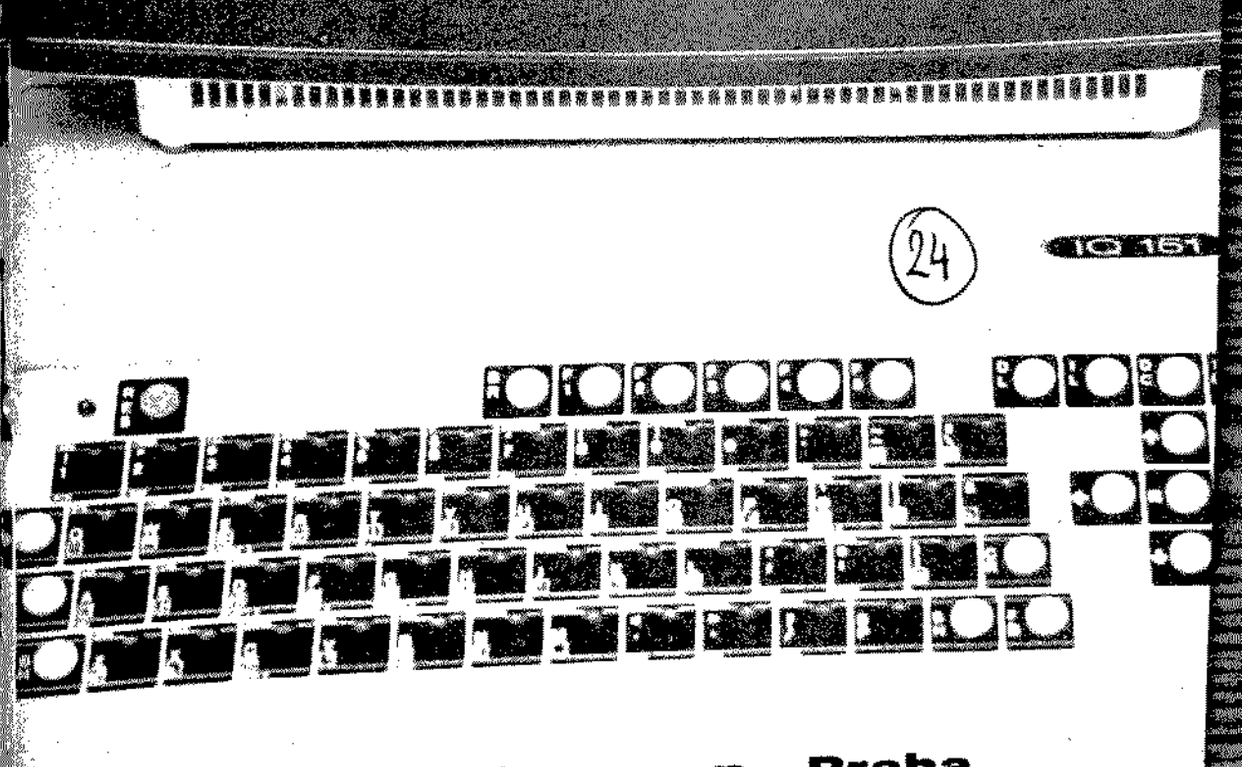
MONITOR IQ 151



MONITOR IQ 151



Komenium, n. p., Praha

RNDr. Miloslav Feil, CSc.

MONITOR IQ 151

příručka pro začátečníky

Komenium, n. p., Praha

Úvod

Tato příručka je určena všem, kteří již zvládli základy programování počítače IQ 151 alespoň v rozsahu příručky Jedlička, Z. - Feil, M. : " Basic pro začátečníky " , vydané n. p. Komenium v roce 1985 a chtějí se dále něco dozvědět o paměti počítače, o způsobu ukládání programu v paměti, o ukládání různých proměnných do paměti jmenovaného počítače a chtějí se naučit používat základní příkazy v režimu MONITOR. Příručka je určena nejen vyučujícím na školách různých stupňů a zaměření, ale je psána tak, aby ji mohl úspěšně studovat i žák střední školy bez pomoci vyučujícího.

Příručka tvoří logické pokračování publikace uvedené v předchozím odstavci, proto se ve své úvodní části zaměřuje na některé doplňky k programování počítače IQ 151 v režimu BASIC, doplňuje některé dřívější poznatky s perspektivou jejich využití ve druhé části příručky, která je již výhradně zaměřená k provozu počítače v režimu MONITOR.

Kromě znalosti základů programování počítače IQ 151 v jazyce BASIC je jedinou podmínkou pro studium této příručky znalost těchto číselných soustav:

a/ desítkové /decimální/, jejíž základem je číslo 10

b/ šestnáctkové /hexadecimální/, jejíž základem je číslo 16

c/ dvojkové, jejímž základem je číslo 2 .

Text příručky je doplněn řadou jednoduchých příkladů, které mají za úkol ukázat v konkrétním případě funkci vykládaného jevu, mnohdy je některý jev vykládán přímo prostřednictvím příkladu. Čtenáři se doporučuje, aby si sám sestavil některé analogické příklady a tím si utvrzoval průběžně získané znalosti. Zvládnutí obsahu této příručky je podkladem pro pozdější úspěšné studium programování počítače IQ 151 v instrukčním kódu mikroprocesoru 8080. V tomto smyslu jsou uváděny i některé poznámky ve výkladu funkce příkazů MONITORU.

Zmíněné příklady v textu jsou jednoduché a minimálního rozsahu, aby demonstrovaly pouze vykládaný jev, jejich smyslem tedy není efekt, ale pouze efektivnost výkladu a jeho názornost.

Autor

1. Doplňky k programování počítače IQ 151 v jazyce BASIC

V této části příručky si jenom doplníme a ucelíme naše dosavadní znalosti z programování počítače IQ 151 v jazyce BASIC. Zkušenější programátor tuto stať možná přeskočí, ale doporučujeme i v tomto případě, aby si ji alespoň přečetl a tak si uvědomil některé logické souvislosti, které bude potřebovat dále.

1.1. Příkazy DATA, READ a RESTORE

Pomocí příkazu DATA a READ se zavádí do programu řada proměnných, ať již číselných nebo řetězcových, přičemž se jednotlivým proměnným přiřazují automaticky příslušné identifikátory.

Za příkazem DATA následuje seznam číselných hodnot nebo řetězců, s nimiž se bude v programu pracovat, jednotlivé hodnoty nebo řetězce jsou odděleny navzájem čárkou. Řetězcové proměnné za příkazem DATA není nutno uvádět v uvozovkách, pokud nezačínají mezerou nebo neobsahují čárku jakožto vlastní součást řetězcové proměnné. Všemu, co je uvedeno za příkazem DATA, se říká někdy seznam - seznam proměnných apod.

Za příkazem READ jsou uvedeny jednotlivé identifikátory, jimiž budou postupně označovány po řadě jednotlivé položky ze seznamu za příkazem DATA. Jednotlivé identifikátory za příkazem READ jsou opět navzájem odděleny čárkou. Jestliže je na určitém místě seznamu řetězcová proměnná, pak na odpovídajícím místě v řadě identifikátorů za příkazem READ musí být identifikátor pro řetězcovou proměnnou. Přiřazování proměnných a identifikátorů pomocí příkazu READ začíná od první hodnoty v seznamu uvedeném za DATA, při dalším příkazu READ

se pokračuje ve čtení seznamu dat tam, kde předchozí příkaz READ skončil - tedy od první dosud nepoužité hodnoty v seznamu. Pokud je třeba číst blok dat od jeho první hodnoty, nutno před příkazem READ použít příkaz RESTORE. To dělá automaticky i příkaz RUN při následném dalším startu programu. Činnost příkazů DATA, READ a RESTORE nám názorně ukáže následující příklad.

Příklad:

Vložme do počítače tento jednoduchý program:

```
1Ø DATA 1, A, 2, B, 3, C, 4, D, 5, E, 6, F
2Ø READ A, A$, B, B$
3Ø PRINT A; A$; B; B$
4Ø GOTO 2Ø
```

Po startu programu příkazem RUN se na obrazovce objeví:

```
 1  A  2  B
 3  C  4  D
 5  E  6  F
* Ø3  ERROR IN 2Ø
```

To znamená, že část programu mezi řádky 2Ø až 4Ø proběhla celkem třikrát, pokaždé se ze seznamu přečetly čtyři hodnoty a vytiskly se na obrazovku v řádku. Počtvrté již příkaz READ nenašel v seznamu hodnot na řádku 1Ø žádné další dosud nepoužité hodnoty a proto se objevilo hlášení Ø3 ERROR IN 2Ø.

Doplníme-li do programu řádek

```
35 RESTORE
```

a spustíme-li takto upravený program, bude příkaz READ číst vždy jen první čtyři hodnoty ze seznamu, příkaz RESTORE vrací čtení hodnot ze seznamu na první hodnotu - tedy číslo 1. Na obrazovce se nyní proto objeví stále se opakující

čtveřice

```
 1  A  2  B
 1  A  2  B
 1  A  2  B
 .....
```

1.2. Příkaz POKE a funkce PEEK

Dané paměťové místo počítače /byte - čti bajt/ je možno obsadit celým číslem z intervalu <Ø; 255> . Každé paměťové místo má své pořadové číslo - tzv. adresu. Chceme-li tedy obsadit paměťové místo o adrese n celým číslem m z uvedeného intervalu, provedeme to takto:

```
POKE n, m
```

Obě čísla m a n jsou celá a decimální.

Chceme-li naopak zjistit, jakým celým číslem z uvedeného intervalu je obsazeno paměťové místo o dané adrese, použijeme funkce PEEK. Nutno si uvědomit, že PEEK je funkce, tedy před ní musí být nějaký příkaz, nejčastěji PRINT a za ní musí být v závorce argument, kterým je adresa paměťového místa, jehož obsazení zjišťujeme. Adresa je opět decimální číslo.

Příklad:

Část paměti, která je zobrazována na televizní obrazovce /tzv. VIDEORAM/ má adresy v intervalu <6Ø 416; 6! 439> . Stiskneme-li např. tlačítko RES, je na obrazovce pouze nápis BASIC a READY s blikajícím kurzorem - tedy na adrese 6Ø 416 je číslo, které charakterizuje písmeno B ve slově BASIC. Toto číslo nazýváme dekadickým kódem znaku /písmena/ B. Zjistíme ho snadno pomocí příkazu

```
PRINT PEEK (6Ø 416)
```

Odešleme-li tento příkaz tlačítkem CR, objeví se na obrazovce číslo 66, tedy dekadický kód písmena B je číslo 66. Nyní tuto adresu obsadíme dekadickým číslem o 1 menším, tedy 65 pomocí příkazu

POKE 6Ø416,65 .

Odešleme-li příkaz tlačítkem CR, pak na místě písmena B se objeví písmeno A, což mimo jiné znamená, že dekadickým kódem znaku /písmena/ A je decimální číslo 65.

O kódech jednotlivých znaků se podrobněji dozvíme později.

1.3. Příkazy DIM, FREE, CLEAR

Příkaz DIM rezervuje určitou část paměti pro určité množství indexovaných proměnných - ať již číselných nebo řetězcových. Provádí tzv. deklaraci pole. Za příkazem DIM musí být ještě uveden identifikátor příslušného pole, který je buď bez znaku $ pro číselné pole, nebo se znakem $ pro řetězcové pole. Za identifikatorem pole jsou dále v oblé závorce uvedeny maximální hodnoty indexů prvků pole. Je-li například pole trojrozměrné, tyto indexy jsou tři navzájem oddělené čárkami. Deklarované pole se ruší pomocí příkazů FREE nebo CLEAR - blíže nám to osvětlí příklad.

Příklad:

Provedeme-li příkaz

DIM A(9)

a odešleme-li jej tlačítkem CR, je v paměti rezervováno místo pro jednorozměrné číselné pole, jehož libovolný prvek je

A(I) ,

přičemž I může nabývat hodnot od Ø do 9.

Zadání konkrétních hodnot jednotlivých prvků pole musíme ovšem provádět zvlášť, nejčastěji pomocí příkazů LET obvyklou cestou. Pokud bychom nyní zadali příkaz

DIM A(2Ø)

a odeslali ho tlačítkem CR, objeví se na obrazovce hlášení chyby ve tvaru

Ø9 ERROR

což znamená, že pole A je již deklarováno a nelze deklarovat jiné číselné pole označené stejným identifikátorem. Pokud bychom chtěli deklarovat pole se stejným identifikátorem, je nutno předchozí deklaraci nejdříve zrušit, tedy prohlásit za neplatné DIM A(9) . To se dá provést celkem dvojím způsobem.

První spočívá v tom, že před změnou deklarace pole o stejném identifikátoru zadáme příkaz

CLEAR

a odešleme ho tlačítkem CR. Tento postup má však jednu nevýhodu - ruší totiž deklarace všech polí, které byly provedeny a ruší současně v paměti počítače všechny zavedené proměnné - jak číselné, tak i řetězcové.

Druhý spočívá v použití příkazu FREE místo CLEAR. Za příkazem FREE musí následovat identifikátor pole, jehož deklarace má být zrušena. Tedy v našem případě bychom pomocí tlačítka CR odeslali příkaz

FREE A .

Použijeme-li tento příkaz, ruší se deklarace pouze pole A, nikoliv dalších polí s jinými identifikátory a neruší se jiné proměnné v paměti počítače dříve zavedené.

Nyní po zrušení původní deklarace pole A jedním nebo druhým způsobem lze pole A opět deklarovat znovu odesláním

příkazu

DIM A (2Ø)

tlačítkem CR.

Poznámky:

1/ RUN ruší automaticky všechna zavedená pole.

2/ Pomocí příkazu

LET A7 = 6.5

zavádíme do paměti počítače proměnnou A7, která není prvkem pole deklarovaného příkazem DIM A (2Ø) . Pomocí příkazu

LET A (7) = 6.5

zavádíme do paměti počítače hodnotu prvku s indexem 7 číselného pole deklarovaného příkazem DIM A (2Ø) .

1.4. Příkaz MEM

Tento příkaz zjišťuje velikost paměti počítače, která dosud není obsazená nějakými programy nebo proměnnými. Provedeme-li bezprostředně po zapnutí počítače příkaz MEM a odešleme-li jej tlačítkem CR, objeví se na obrazovce hlášení

32211 bytes free .

Je-li v počítači vložen nějaký program, počet neobsazených paměťových míst se příslušně sníží - některá paměťová místa jsou programem obsazena. Provedeme-li příkaz MEM bezprostředně po nahrání nějakého programu do počítače, můžeme z rozdílu obou hlášení určit, jaký je rozsah programu - tedy kolik paměťových míst obsazuje. Rovněž zavedením libovolné proměnné do paměti počítače se počet volných paměťových míst snižuje.

Příklad:

Po zapnutí počítače vložme do jeho paměti tento jedno-

duchý program:

1Ø AŽ = "ABCDEFG" .

Aniž bychom tento program startovali, zjistíme nyní pomocí MEM rozsah zbývající volné paměti. Dostaneme hlášení

32 194 bytes free ,

což znamená, že náš program obsadil 17 paměťových míst.

Spustíme-li nyní náš program příkazem RUN a provedeme-li opět příkaz MEM, dostaneme nyní hlášení

32 188 bytes free .

To znamená, že se řetězec AŽ pomocí našeho programu uložil do paměti počítače jako řetězcová proměnná a proto se ještě dále snížil rozsah volné paměti o 6 paměťových míst. Odešleme-li nyní tlačítkem CR příkaz CLEAR, počítač zavedenou řetězcovou proměnnou "zapomene" a tato zůstává pouze jako součást programového řádku. Provedeme-li ještě nyní znovu příkaz MEM, dostaneme hlášení

32 194 bytes free ,

tedy se zvýšil počet volných paměťových míst o 6 - zavedená řetězcová proměnná AŽ byla z paměti pro proměnné vymazána - počítač ji "zapomněl".

K analogickým poznatkům bychom dospěli, kdybychom pomocí jednoduchého programu zaváděli do paměti i číselnou proměnnou. Tedy "zapamatování" libovolné proměnné - ať již číselné nebo řetězcové - znamená snížení počtu volných paměťových míst o 6.

1.5. Funkce ASC a CHRŽ

Již v článku 1.2 jsme narazili na skutečnost, že každému znaku - písmenu, interpunkčnímu znaménku apod. - odpovídá jednoznačně určité dekadické číslo, které se nazývá jeho kódem.

Toto číslo lze převést z desítkové číselné soustavy do soustavy, jejíž základ tvoří číslo 16 /tzv. hexadecimální/. Důvody těchto převodů si ujasníme později, zatím si jen pamatujme, že pracujeme-li s příkazem jazyka BASIC, doplňujeme k němu příslušné kódy znaků v soustavě decimální /její základ je 10/. Na následující stránce je tabulka jednotlivých znaků a jejich kódů. Ve sloupci označeném písmenem D jsou kódy decimální, ve sloupci označeném H jsou převedeny do soustavy hexadecimální.

Z tabulky vidíme, že znaky jsou přiřazovány jednotlivým dekadickým kódům až od čísla 32. Funkce ASC přiřazuje znaku jeho dekadický kód v souladu s tabulkou, funkce CHR$ pracuje opačně, tedy danému dekadickému celému číslu z vhodného intervalu ⟨32; 127⟩ přiřadí odpovídající znak. Uvědomme si současně, že CHR$ má za argument číslo a výsledek je znak, tedy řetězec. ASC má za argument řetězec - proto je ho nutno psát v úvozovkách - a výsledkem je celé dekadické číslo.

Příklad:

Odešleme-li tlačítkem CR příkaz

```
PRINT CHR$(77)
```

dostaneme jako výsledek písmeno M.

Odešleme-li tlačítkem CR příkaz

```
PRINT ASC(" M ")
```

dostaneme jako výsledek dekadické číslo 77.

Použijeme-li za argument funkce CHR$ celé číslo z intervalu ⟨0; 31⟩, pak takové příkazy budou mít jiný význam - jsou to tzv. řídící znaky.

Tabulka znaků a jejich kódů

D	H	znak	D	H	znak	D	H	znak	D	H	znak	D	H	znak
0	0		26	1A		52	34	4	78	4E	N	104	68	h
1	1		27	1B		53	35	5	79	4F	O	105	69	i
2	2		28	1C		54	36	6	80	50	P	106	6A	j
3	3		29	1D		55	37	7	81	51	Q	107	6B	k
4	4		30	1E		56	38	8	82	52	R	108	6C	l
5	5		31	1F		57	39	9	83	53	S	109	6D	m
6	6		32	20	␣	58	3A	:	84	54	T	110	6E	n
7	7		33	21	!	59	3B	;	85	55	U	111	6F	o
8	8		34	22	"	60	3C	<	86	56	V	112	70	p
9	9		35	23	#	61	3D	=	87	57	W	113	71	q
10	A		36	24	$	62	3E	>	88	58	X	114	72	r
11	B		37	25	%	63	3F	?	89	59	Y	115	73	s
12	C		38	26	&	64	40	@	90	5A	Z	116	74	t
13	D		39	27	'	65	41	A	91	5B	[	117	75	u
14	E		40	28	(	66	42	B	92	5C	\	118	76	v
15	F		41	29	)	67	43	C	93	5D	]	119	77	w
16	10		42	2A	*	68	44	D	94	5E	↑	120	78	x
17	11		43	2B	+	69	45	E	95	5F	-	121	79	y
18	12		44	2C	,	70	46	F	96	60	`	122	7A	z
19	13		45	2D	-	71	47	G	97	61	a	123	7B	{
20	14		46	2E	.	72	48	H	98	62	b	124	7C	|
21	15		47	2F	/	73	49	I	99	63	c	125	7D	}
22	16		48	30	0	74	4A	J	100	64	d	126	7E	~
23	17		49	31	1	75	4B	K	101	65	e	127	7F	DEL
24	18		50	32	2	76	4C	L	102	66	f			
25	19		51	33	3	77	4D	M	103	67	g			

Příklad:

Odešleme-li tlačítkem CR příkaz

PRINT CHR$(31) ,

zjistíme, že působí stejně, jako příkaz CLS - tedy maže obrazovku. Tabulka řídících znaků následuje za poznámkami.

Poznámky:

1/ Zvýšíme-li argument funkce o decimální číslo 128, obdržíme na obrazovce stejný znak - tedy příkazy

PRINT CHR$(77) a

PRINT CHR$(205)

napíší oba na obrazovku stejný znak - písmeno M.

2/ Na předchozí tabulce si všimněte, jak vypadají tvary daného čísla v soustavě decimální a hexadecimální - tabulka vám může sloužit také jako převodní tabulka mezi oběma číselnými soustavami pro čísla menší než 128 /decimálně/.

Tabulka řídících znaků volitelných pomocí CHR$

Příkaz PRINT CHR$(X) provede pro dané X:

X:	činnost:
7	pípnutí
8	posun kurzoru vlevo
9	posun kurzoru o 8 pozic vpravo
12	přesun kurzoru do levého horního rohu obrazovky
13	ukončení řádku, zrušení grafického a inverzního režimu
14	přepnutí z grafického do normálního režimu
15	přepnutí do grafického režimu
18	přepnutí z inverzního do normálního režimu
19	přepnutí do inverzního režimu
24	posun kurzoru vpravo
25	posun kurzoru nahoru
26	posun kurzoru dolů
28	vsunutí znaku do řádku
29	zrušení znaku v řádku
31	mazání obrazovky

Poznámka:

V programech v jazyce BASIC se nejčastěji mohou uplatnit tyto příkazy s čísly 7, 9, 14, 15, 18, 19 a 31. Můžeme pomocí nich vyvolávat akustické signály nebo programově pře-

pínat jednotlivé režimy počítače - grafický, inverzní a opačně.

1.6. Funkce INKEY$

Tato funkce v průběhu chodu programu testuje, zda je nějaké černé tlačítko na klávesnici počítače v daném okamžiku stisknuté, nebo nikoliv. Je-li nějaké tlačítko stisknuté, pak funkční hodnotou funkce INKEY$ je znak - tedy řetězec, který je základním významem stisknutého tlačítka.

S výhodou je možno této funkce používat, má-li docházet ke změně některých parametrů a proměnných během chodu programu.

Příklad:

Vložme do počítače tento jednoduchý program:

```
10 C = 0
20 C$ = INKEY$
30 IF C$ = " A " THEN C = C + 1
40 IF C$ = " B " THEN C = C - 1
50 PRINT C
60 GOTO 20
```

Spustíme-li jej pomocí příkazu RUN, pak se na obrazovce objevují nuly, pokud nestiskneme buď A nebo B na klávesnici. Pomocí tlačítek A a B měníme tedy hodnotu proměnné C za chodu programu.

Poznámka:

Není-li žádné tlačítko stisknuto, je hodnota funkce INKEY$ prázdný řetězec - tedy řetězec, který neobsahuje žádný znak.

1.7. Logické operátory

Jsou to operátory AND, OR, NOT. Umíme je již používat jako operátorů mezi jednoduchými výroky /nejčastěji v podmínkách za příkazem IF/. Takové použití logických operátorů nám připomene následující příklad.

Příklad:

Sestavte program, který rozhodne o tom, zda do počítače vložené číslo leží v intervalu $\langle 1; 5 \rangle$, nebo nikoliv. Pokud číslo leží v tomto intervalu, počítač napíše "UVNITR", leží-li číslo vně intervalu, počítač napíše "MIMO". Program je následující:

```
10 INPUT A
20 IF ( A < 1  OR  A > 5 )  THEN PRINT "MIMO" : GOTO 40
30 PRINT "UVNITR"
40 GOTO 10
```

Dále je však možno pomocí těchto logických operátorů pracovat i s čísly z intervalu $\langle -32768; 32767 \rangle$ - čísla jsou opět decimální, krajní hodnoty intervalu se dají napsat také -2^{15} a $2^{15} - 1$. Pokud bychom použili čísel mimo tento interval, dostaneme hlášení #4 ERROR na počítači. Dále je nutno pracovat pouze s celými čísly uvedeného intervalu.

Abychom mohli vyložit, jak jednotlivé logické operátory s uvedenými čísly pracují, je nutno celá čísla zobrazit v tzv. dvojkovém doplňkovém kódu. Tento dvojkový doplňkový kód je totožný se zobrazením kladných čísel ve dvojkové číselné soustavě, navíc však umožňuje pouze pomocí nul a jedniček zapsat i čísla záporná. Ukážeme, jak se zobrazují ve dvojkovém do-

plňkovém kódu celá čísla z intervalu ⟨- 1Ø; 1Ø⟩ , z toho již pozorný čtenář sám odvodí tvar ostatních čísel v tomto kódu.

decimální číslo	dvojkový doplňkový kód
- 1Ø	1111 1111 1111 Ø11Ø
- 9	1111 1111 1111 Ø111
- 8	1111 1111 1111 1ØØØ
- 7	1111 1111 1111 1ØØ1
- 6	1111 1111 1111 1Ø1Ø
- 5	1111 1111 1111 1Ø11
- 4	1111 1111 1111 11ØØ
- 3	1111 1111 1111 11Ø1
- 2	1111 1111 1111 111Ø
- 1	1111 1111 1111 1111
Ø	ØØØØ ØØØØ ØØØØ ØØØØ
1	ØØØØ ØØØØ ØØØØ ØØØ1
2	ØØØØ ØØØØ ØØØØ ØØ1Ø
3	ØØØØ ØØØØ ØØØØ ØØ11
4	ØØØØ ØØØØ ØØØØ Ø1ØØ
5	ØØØØ ØØØØ ØØØØ Ø1Ø1
6	ØØØØ ØØØØ ØØØØ Ø11Ø
7	ØØØØ ØØØØ ØØØØ Ø111
8	ØØØØ ØØØØ ØØØØ 1ØØØ
9	ØØØØ ØØØØ ØØØØ 1ØØ1
1Ø	ØØØØ ØØØØ ØØØØ 1Ø1Ø

Všimněme si, že opačné číslo dostaneme tak, že zaměníme nuly a jedničky v zápisu daného čísla ve dvojkové soustavě a k takto upravenému číslu přičteme na posledním řádu jedničku -

algoritmus je však nutno začít od kladného celého čísla, jehož vyjádření ve dvojkové číselné soustavě dovedeme určit.

Všimněme si ještě, že vezmeme-li vyjádření kladného celého čísla n ve dvojkovém doplňkovém kódu a vyjádření čísla - n - 1 v tomtéž kódu, liší se oba zápisy vzájemnou záměnou nul a jedniček na všech řádech. Všechna kladná čísla mají na nejvyšším řádu nulu, všechna záporná jedničku.

Logická spojka AND provede součin cifer příslušných řádů dvou zadaných čísel v jejich vyjádření v dvojkovém doplňkovém kódu. Výsledek se zobrazí jako číslo dekadické.

Příklad:

Odešleme-li tlačítkem CR příkaz

PRINT 1Ø AND -1Ø ,

dostaneme výsledek 2.

Rozbor:

Vyjadření dekadického čísla 1Ø ve dvojkovém doplňkovém kódu je ØØØØ ØØØØ ØØØØ 1Ø1Ø .

Vyjadření dekadického čísla -1Ø ve dvojkovém doplňkovém kódu je 1111 1111 1111 Ø11Ø .

Provedeme-li součiny cifer obou čísel ve všech řádech, dostaneme číslo ØØØØ ØØØØ ØØØØ ØØ1Ø .

Převedeme-li toto číslo do deciméiní soustavy - viz tabulka - dostaneme číslo 2.

Logická spojka OR provede logický součet cifer příslušných řádů dvou zadaných čísel v jejich vyjádření ve dvojkovém doplňkovém kódu. Výsledek se zobrazí jako číslo dekadické.

Příklad:

Odešleme-li tlačítkem CR příkaz

PRINT 3 OR 4 ,

dostaneme výsledek 7.

Rozbor:

Vyjádření dekadického čísla 3 ve dvojkovém doplňkovém kódu je ØØØØ ØØØØ ØØØØ ØØ11 .

Vyjádření dekadického čísla 4 ve dvojkovém doplňkovém kódu je ØØØØ ØØØØ ØØØØ Ø1ØØ .

Provedeme-li logický součet cifer obou čísel ve všech řádech, dostaneme číslo ØØØØ ØØØØ ØØØØ Ø111 .

Převedeme-li jej do dekadické soustavy, dostaneme číslo 7.

Logická spojka NOT provede logickou negaci - výměnu nul za jedničky a naopak - ve všech řádech zadaného čísla v jeho vyjádření ve dvojkovém doplňkovém kódu- výsledek se opět zobrazí jako dekadické.

Příklad:

Odešleme-li tlačítkem CR příkaz

PRINT NOT 6 ,

dostaneme výsledek -7 .

Rozbor:

Vyjádření dekadického čísla 6 ve dvojkovém doplňkovém kódu je ØØØØ ØØØØ ØØØØ Ø11Ø .

Logická negace je 1111 1111 1111 1ØØ1 , což je dekadické číslo -7.

2. Základní struktura paměti počítače

2.1. Paměť ROM a RAM, pojem byte a jeho adresa

Rozlišujeme dva základní druhy paměti počítače IQ 151:

a/ paměť typu RAM - informace uložené v této paměti se uchovávají pouze po dobu, po níž je počítač zapnutý. Při vypnutí počítače se všechny informace z této paměti vymažou. Při práci s počítačem lze paměť tohoto typu libovolně obsazovat, určitou její část obsazuje vhodnými parametry sám počítač při zapnutí automaticky. Paměť RAM uchovává programy v jazyce BASIC nebo ve strojovém kodu, uchovávají se v ní hodnoty proměnných nebo řetězce.

b/ paměť typu ROM - informace vložené do této paměti jsou trvalé, nemažou se při vypnutí počítače ani při mazání informací z paměti RAM. Slouží k uchování překladače programovacího jazyka, jednotlivých znaků apod.

Paměť počítače si lze představit jako řadu za sebou jdoucích příhrádek, z nichž každá má své pořadové číslo - tzv. adresu. Adresa první příhrádky je Ø. Do každé příhrádky je možno umístit celé číslo z intervalu ⟨Ø; 255⟩ - čísla jsou decimální.

Místo slova "příhrádka" budeme používat termín paměťové místo nebo byte /čti bajt/.

Příklad:

Výraz "byte s adresou 2Ø je obsazen číslem 1"znamená, že v příhrádce - paměťovém místě - s pořadovým číslem 2Ø je uloženo číslo 1.

V této souvislosti je nutno připomenout příkaz POKE,

který přímo obsazuje byte s danou adresou daným číslem. Máme-li tedy v souvislosti s minulým příkladem obsadit byte s adresou 2Ø číslem 1, odešleme tlačítkem CR příkaz

POKE 2Ø, 1 ,

čímž je obsazení provedeno. Víme již, že příkaz POKE vyžaduje obě následující čísla dekadická.

2.2. Struktura paměťového místa, pojem bit

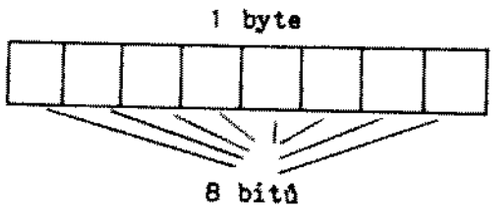
Otázka, proč dané paměťové místo může být obsazeno jen celým číslem počínaje Ø a 255 konče - tedy 256 čísly, nás přivádí k výkladu struktury paměťového místa.

Všimněme si nejdříve, že

$$256 = 2^8 = 16^2 \quad .$$

Vyjádříme-li libovolné decimální celé číslo z intervalu $\langle Ø;255\rangle$ ve dvojkové číselné soustavě, pak toto číslo má nejvýše 8 cifer, každá cifra je buď Ø nebo 1.

Každý byte je možno si představit jako celek, který obsahuje celkem 8 za sebou uspořádaných malých příhrádek - tzv. bitů. Tedy:



To, že 1 byte /paměťové místo/ obsahuje právě 8 bitů, se vyjadřuje termínem "délka slova" nebo "délka slabiky". Počítač IQ 151 má tedy délku slova 8 bitů. V současné době již existují počítače, které mají délku slova 16 bitů.

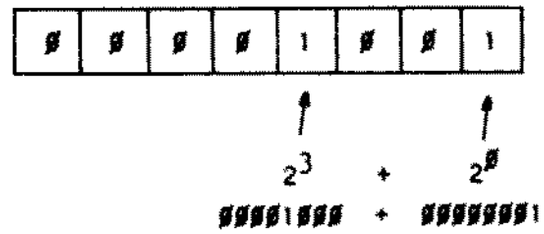
Jednotlivé bity každého paměťového místa jsou očíslovány přirozenými dekadickými čísly 7, 6, 5, 4, 3, 2, 1 a Ø . Bit s pořadovým číslem Ø je na pravém konci, bit s pořadovým číslem 7 je na levém konci paměťového místa.

Každý bit libovolného paměťového místa může být obsazen buď číslem Ø nebo číslem 1. Tedy do každého čtverečku na předchozím obrázku lze napsat Ø nebo 1. To znamená, že na každém paměťovém místě může být uloženo libovolné číslo vyjádřené ve dvojkové číselné soustavě počínaje od ØØØØ ØØØØ a konče 1111 1111 , což jsou právě všechna celá čísla z dekadického intervalu $\langle Ø; 255\rangle$.

Příklad:

Je-li paměťové místo s adresou 27 obsazeno číslem 9 /obě čísla decimální/, vypadá obsazení jednotlivých bitů takto:

byte 27



Poznámka:

Adresy a obsahy jednotlivých paměťových míst se někdy vyjadřují pomocí hexadecimální číselné soustavy. Výhody takového vyjádření jsou následující:

a/ Každá adresa paměti je maximálně čtyřciferné hexadecimální číslo.

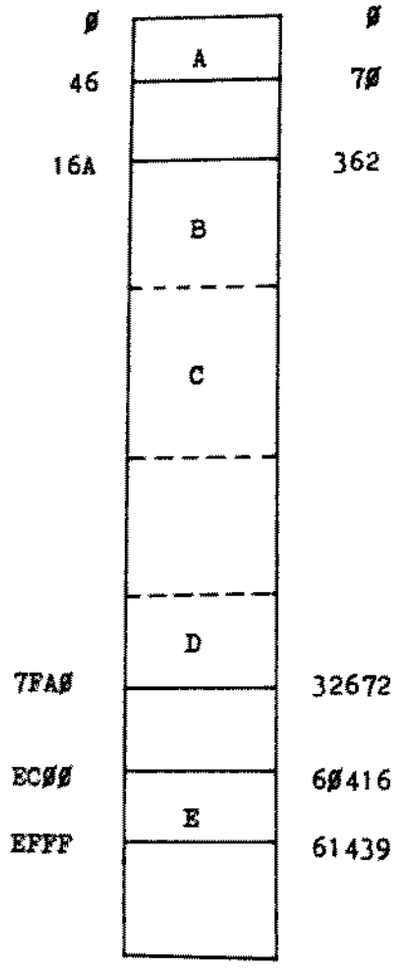
b/ Každé celé decimální číslo od Ø do 255 je v hexade-

cimální číselné soustavě maximálně dvojciferné.

Věnujte opět pozornost tabulce na str. 11, kde máte možnost porovnat decimální a hexadecimální vyjádření některých čísel.

2.3. Zjednodušená struktura paměti RAM počítače IQ 151

Struktura je následující:



Celá paměť RAM je rozdělena do určitých úseků, které mají své speciální poslání. Úseky jsou označeny písmeny A - E. Začátek a konec některých úseků je pevně stanoven, na našem schematu je taková hranice vyznačena plnou vodorovnou čarou. Hranice jiných úseků je vyznačena čárkovaně a počítač si ji volí sám třeba podle rozsahu programu, počtu proměnných, nebo ji musí zvolit programátor - čárkovaná hranice oblasti označené D.

Všimněme si, že některé úseky paměti nemusí být někdy plně využity; mezi úseky paměti, které jsou označeny písmeny,

se právě takové oblasti vyskytují - záleží ovšem na rozsahu programu, který je do počítače vložen apod. Hranice jednotlivých úseků paměti tvoří paměťová místa, jejichž adresy jsou rovněž na obrázku uvedeny - vlevo jsou tyto adresy vyjádřeny hexadecimálně, vpravo decimálně.

Význam jednotlivých úseků paměti RAM

A - oblast pro systémové proměnné, obsazuje ji sám počítač nebo překladač vyššího programovacího jazyka; některé adresy v této oblasti může obsadit i programátor a tím měnit některé pracovní parametry počítače;

B - oblast, v níž jsou uloženy programy v jazyce BASIC;

C - oblast paměti pro proměnné, ať již číselné, řetězcové nebo pro číselná a řetězcová pole;

D - oblast USR, využitelná pro programy ve strojovém kódu;

E - oblast paměti pro obrazovku - tzv. **VIDEORAM** .

3. Práce s počítačem v režimu MONITOR

Pod tímto termínem rozumíme práci při vypnutém překladači programovacího jazyka - v našem případě jazyka BASIC. Vypnutí překladače se děje stisknutím tlačítka

[BR]

v horní řadě bílých tlačítek počítače. Po stisknutí tohoto tlačítka se objeví na obrazovce následující symboly:

```
# F8BA
>
```

a vedle posledního znaku vidíme blikající kurzor.

V režimu MONITOR je možno pracovat pouze s tzv. příkazy monitoru. Jsou to určitá jednotlivá písmena, za nimiž mohou následovat některé číselné parametry podle povahy a funkce každého příkazu v režimu MONITOR. Písmena jsou následující:

D, M, F, S, W, L, R, C, G, X.

Funkci jednotlivých písmen - tedy příkazů monitoru si ozřejmíme v následujícím výkladu.

Důležité upozornění:

Při práci v režimu MONITOR je nutno vkládat veškeré číselné údaje v soustavě hexadecimální. Rovněž i informace, které se při práci v tomto režimu na obrazovce objeví, jsou v hexadecimální soustavě. V režimu BASIC naopak byla všechna čísla decimální.

3.1. Příkaz D

Pomocí tohoto příkazu můžeme na obrazovce získat výpis obsazení jednotlivých paměťových míst určitého úseku paměti

počínaje zvolenou adresou, která se uvede za písmeno D.

Příklad:

Jestliže bezprostředně po zapnutí počítače stiskneme tlačítko BR a napíšeme příkaz DØ, pak po jeho odeslání tlačítkem CR dostaneme na obrazovce tabulku, jejíž počáteční řádky jsou například následující:

```
ØØØØ  FF FF FF 69 FF 7F ØØ Ø7
ØØØ8  FE ØØ ØØ 2Ø D1 ED 17 ØE
ØØ1Ø  ØØ ØØ 5Ø 1E Ø2 BA F8 Ø4
```

V prvním sloupci jsou v hexadecimálním tvaru uvedeny adresy paměťových míst, dále na příslušných řádcích jsou v hexadecimálním tvaru uvedeny obsahy jednotlivých paměťových míst. Z tabulky vyplývá, že

na paměťovém místě s adresou	je uloženo číslo
ØØØØ	FF
ØØØ1	FF
ØØØ2	FF
ØØØ3	69
ØØØ4	FF
ØØØ5	7F
ØØØ6	ØØ
ØØØ7	Ø7
ØØØ8	FE
ØØØ9	ØØ
ØØØA	ØØ
ØØØB	2Ø

na paměťovém místě s adresou	je uloženo číslo
ØØØC	D1
ØØØD	ED
ØØØE	17
ØØØF	ØE
ØØ1Ø	ØØ
ØØ11	ØØ
ØØ12	5Ø

atd.

Takto můžeme na obrazovce vidět obsazení paměťových míst od adresy Ø až do adresy 6F /hexadecimálně/. Stiskneme-li nyní libovolný černý knoflík na klávesnici počítače, objeví se na obrazovce obsazení následující části paměti - přesněji od adresy 6Ø až do adresy DF /hexadecimálně/ a takto lze pokračovat.

Chceme-li ukončit výpis obsazení paměti, stiskneme tlačítko CR a počítač je schopen přijmout jiný příkaz v režimu MONITOR.

Chceme-li pořídit výpis obsazení paměti počínaje paměťovým místem s adresou 16A /hexadecimálně/, odešleme v režimu MONITOR tlačítkem CR příkaz

D16A .

3.2. Příkaz S

Pomocí tohoto příkazu můžeme v režimu MONITOR změnit obsah libovolného paměťového místa.

Příklad:

Paměťová místa s adresami od AØ1 do AØ5 /hexadecimálně/ obsaďte hexadecimálním číslem EF.

Postup:

V režimu MONITOR napíšeme příkaz S a za něj nejnižší adresu, tedy AØ1. Pak stiskneme tlačítko SP, nikoliv tedy CR.

Na obrazovce se objeví uvedená hexadecimální adresa a za ní původní obsah paměťového místa s touto adresou. Dále se v řádku objeví pomlčka a za ní blikající kurzor, tedy

ØAØ1 FF - □ .

Nyní napíšeme nový obsah zvoleného paměťového místa - tedy hexadecimální číslo EF a stiskneme opět tlačítko SP. Tím je provedeno obsazení a na obrazovce se nyní objeví následující adresa a její původní obsah analogicky, jako v předchozím případě. Opět tento obsah změníme na EF a tak pokračujeme až k adrese AØ5. Změníme-li obsah posledního zadaného paměťového místa, stiskneme tlačítko CR - nikoliv již SP, čímž se působení příkazu S přeruší a počítač je schopen akceptovat jiný příkaz režimu MONITOR.

O tom, zda zadaná paměťová místa byla skutečně obsazena zvoleným číslem, se můžeme snadno přesvědčit pomocí výpisu, který dostaneme pomocí příkazu DAØ1.

Poznámky:

1/ Nechceme-li měnit obsah některého paměťového místa, stiskneme tlačítko SP, aniž bychom napsali nějaké

číslo za pomlčku.

2/ Mačkáme-li tlačítko SP, zobrazují se po řadě adresy, z nichž každá následující je o jednu větší než předchozí. Pokud bychom mačkali místo tlačítka SP bílé tlačítko pro ovládání posuvu kurzoru [↑] , každá další adresa by byla o jednu nižší.
Toho lze s výhodou použít, když se při obsazování paměťových míst spleteme a chceme chybu opravit, tedy se vrátit na nižší adresu.

Všimněme si, že příkaz S v režimu MONITOR provádí principiálně totéž, co příkaz POKE v režimu BASIC. Pouze je nutno respektovat upozornění ze stránky 24 - údaje hexadecimální !

Příkazu S se používá nejčastěji pro vkládání programů ve strojovém kódu na určitá místa paměti počítače.

3.3. Příkaz M

Tento příkaz přesune obsah paměťových míst o zadaných adresách na jiná paměťová místa, počínaje opět určenou adresou. Blíže to ozřejmí následující příklad.

Příklad:

Obsaďte paměťová místa počínaje hexadecimální adresou 16A až do 16F hexadecimálním číslem 41 a pak tento úsek paměti přesuňte tak, aby začínal na paměťovém místě s hexadecimální adresou 1AØ.

Řešení:

Obsazení paměťových míst od adresy 16A do 16F provedeme snadno pomocí příkazu S - viz předchozí článek. Pomocí příkazu D se můžeme rovněž přesvědčit, zda k žádanému obsazení došlo.

K přesunu obsahu tohoto úseku paměti na úsek počínající adresou 1AØ provedeme pomocí příkazu M, za nějž musíme doplnit celkem tři parametry vzájemně oddělené čárkou. Prvním parametrem je počáteční hexadecimální adresa úseku paměti, z níž dochází k přesunu, druhým je hexadecimální adresa posledního paměťového místa tohoto úseku. Třetím parametrem je první adresa paměťového místa, k němuž se obsah úseku paměti přesune.

Tedy v našem případě odešleme v režimu MONITOR tlačítkem CR následující příkaz:

M 16A,16F,1AØ .

Pomocí příkazu D16A se můžeme přesvědčit, že skutečně došlo k přesunu obsahu daného úseku paměti podle našich požadavků a rovněž vidíme, že tímto přesunem se nezměnilo původní obsazení úseku mezi adresami 16A a 16F. Obsah se jen překopíroval do jiné oblasti v paměti.

Příkazu M se používá nejčastěji při přesunu programu ve strojovém kódu do vhodného úseku paměti, kde nemůže být porušem programem v jazyce BASIC apod.

3.4. Příkaz F

Pomocí tohoto příkazu naplníme v režimu MONITOR daným hexadecimálním číslem z intervalu ⟨ØØ; FF⟩ úsek paměti, který začíná a končí zvolenými adresami, vystupujícími za příkazem F jako parametry. Parametry jsou samozřejmě hexadecimální a oddělené čárkou.

Příklad:

Obsaďte úsek paměti od hexadecimální adresy 3D9 do 3E2

hexadecimalním číslem 2E.

Řešení:

Obsazení provedeme pomocí příkazu F, za nímž budou tři parametry vzájemně oddělené čárkou. Prvním parametrem je počáteční adresa zvoleného úseku paměti, druhým je koncová adresa tohoto úseku a třetím je číslo, kterým budou jednotlivá místa ve zvoleném úseku paměti obsazena.

V našem případě tedy v režimu MONITOR odešleme tlačítkem CR příkaz

F 3D9,3E2,2E .

Že bylo obsazení provedeno podle našich požadavků, se můžeme přesvědčit pomocí příkazu D3D9.

3. 5. Příkaz R

Stiskneme-li v režimu MONITOR tlačítko s písmenem R, počítač se vrátí do režimu BASIC. Tedy tlačítko R působí opačně, než tlačítko BR.

3.6. Příkaz W

Pomocí tohoto příkazu lze obsah libovolného úseku paměti přehrát na magnetofonový pásek a pořídit tak trvalý záznam jeho obsazení. Úsek je pochopitelně opět určen počáteční a koncovou hexadecimální adresou hraničních paměťových míst, které tvoří parametry v příkazu W. Dále je nutno za příkaz W doplnit ještě další hexadecimální parametr, jehož význam vyplyne teprve z pozdějšího výkladu. Parametry za příkazem W jsou opět odděleny čárkami.

Příklad:

Obsaďte část paměti od adresy 3A2 do adresy 3A6 po řadě hexadecimálními čísly 19, 1A, 1B, 1C, 1D a pak nahrajte obsazení této části paměti na magnetofonový pásek.

Řešení:

Obsazení části paměti mezi adresami 3A2 a 3A6 danými hexadecimálními čísly provedeme v režimu MONITOR snadno pomocí příkazu S, který již známe.

Při pořizování záznamu úseku paměti si budeme počínat tak, jako když nahráváme program v jazyku BASIC jen s tím rozdílem, že budeme pracovat v režimu MONITOR a tlačítkem CR odešleme příkaz pro nahrávání ve tvaru

W 3A2,3A6,Ø .

Uvědomme si pouze, že tlačítko CR lze stisknout až když je zapnuto nahrávání magnetofonu alespoň po dobu 5 s, aby se na záznamu objevil určitý úsek pilotního kmitočtu. Ukončení přehrávky poznáme v režimu MONITOR tak, že se na obrazovce objeví blikající kurzor. Pořízením záznamu na magnetofonový pásek se obsazení daného úseku paměti nezmění ani nesmaže.

3.7. Příkaz L

Tento příkaz slouží k přehrání obsahu úseku paměti z magnetofonového záznamu do počítače. Příkaz L se zadává buď bez následného parametru, nebo za ním následuje hexadecimální číslo z intervalu ⟨ØØØØ; FFFF⟩ . L bez parametru má týž význam jako LØ. Funkce příkazu si nejlépe ozřejmíme na příkladu.

Příklad:

Přehrajte do paměti počítače magnetofonový záznam obsahu úseku paměti od adresy 3A2 do 3A6, který jste získali z minulého příkladu.

Řešení:

1/ Vypněte počítač síťovým vypínačem na jeho boku. Po několika okamžicích jej opět zapněte - smysl tohoto počínání vyplyne v závěru příkladu.

2/ Nastavte na magnetofonovém pásku začátek pilotního kmitočtu odpovídajícího záznamu.

3/ V režimu MONITOR dejte na obrazovku příkaz L.

4/ Zapněte přehrávání magnetofonu.

5/ Uslyšíte-li z příposlechu vyrovnaný pilotní tón, který bezprostředně předchází nahrávce obsahů paměťových míst, stiskněte tlačítko CR.

Dokončení přehrávky se projeví tak, že se na obrazovce objeví blikající kurzor. Pak můžeme vypnout magnetofon. Pomocí příkazu D3A2 se přesvědčíme, že v příkladu uvedené adresy byly přehrávkou obsazeny požadovaným způsobem.

Kdybychom místo příkazu L /tedy vlastně LØ/ použili při přehrávání příkazu L1, viděli bychom, že se paměť zaplnila předepsanými hodnotami nikoliv od adresy 3A2 do 3A6, ale od adresy 3A3 do 3A7. Význam hexadecimálního čísla za L a prvního hexadecimálního parametru za příkazem W je tedy takový, že jejich součet dává počáteční adresu úseku paměti, do něhož bude záznam přehrán. V našem případě je skutečně

3A2 + 1 = 3A3 .

Program ve strojovém kódu, který se takto může nahrát do určeného úseku paměti počítače, může být automaticky spuštěn. To se stane v případě, že třetí parametr za příkazem W není nula, ale nějaké jiné hexadecimální číslo z intervalu < ØØØØ; FFFF > . V takovém případě je program spuštěn od adresy, která je dána součtem dvou čísel - třetího hexadecimálního parametru za příkazem W a hexadecimálního čísla za příkazem L.

Máme-li úsek paměti mezi adresami 3A2 a 3A6 obsazen v důsledku přehrávky z magnetofonového záznamu, ukážeme si ještě jednu zajímavost, která souvisí s červeným tlačítkem RES. Používáme jej většinou tehdy, když chceme odstranit nějaké programy v jazyce BASIC z počítače.

Stiskneme tedy tlačítko RES a podíváme se znovu na obsah paměti mezi adresami 3A2 a 3A6 pomocí příkazu D3A2 v režimu MONITOR. Vidíme, že se obsahy všech uvažovaných paměťových míst po stisku tlačítka RES nezměnily ani nesmazaly. Chceme-li se tedy zbavit tohoto obsahu paměti, nezbývá, než počítač vypnout, což jsme prováděli hned v prvním bodě našeho příkladu. Tlačítko RES nemění /až na řídké výjimky/ obsahy jednotlivých paměťových míst. Jeho vztah k mazání programů v jazyce BASIC vysvětlíme později. Zapamatujme si prozatím, že programy ve strojovém kódu se nemohou mazat tlačítkem RES, jako běžné programy v jazyce BASIC.

Použití příkazů W a L - z výkladu již vyplynulo, že se používají při přehrávání programů ve strojovém kódu z počítače na magnetofonový pásek nebo naopak.

3.8. Příkazy C, G a X

Funkci těchto příkazů v režimu MONITOR bude možno ozřejmit až po výkladu partií o registrech počítače a programování ve strojovém kódu. Proto se jimi v tomto úvodním přehledu nebudeme podrobněji zabývat. Vrátíme se k nim příležitostně v příručce o programování ve strojovém kódu.

4. Významy důležitých adres paměti počítače

Jedná se zvláště o paměťová místa s adresami od ø do 7ø /decimálně/, tedy od ø do 46 /hexadecimálně/. Tyto adresy paměti RAM obsazuje počítač sám automaticky vhodnými hodnotami po zapnutí. V následujícím si ukážeme význam některých vybraných adres a jejich obsazení. Určité adresy této oblasti paměti RAM slouží jako registry počítače a využívají se při programování ve strojovém kódu, proto se o nich zmíníme až při výkladu takového programování.

4.1. Ovládání blikání kurzoru

Blikání kurzoru ovlivňuje obsah paměťového místa s adresou 7 /decimálně i hexadecimálně/. Jestliže je toto paměťové místo obsazeno celým číslem z intervalu ⟨1; 255⟩ /decimálně/, kurzor nebliká. Pokud je toto místo obsazeno číslem ø, kurzor bliká.

Zdálo by se tedy, že pokud provedeme v režimu BASIC odeslání tlačítkem CR příkazu

POKE 7, 1 ,

přestane kurzor blikat. Pokud ale provedeme tento postup na počítači, objeví se po vložení příkazu a jeho odeslání rovněž hlášení READY s blikajícím kurzorem. Znamená to, že po provedení příslušného příkazu si počítač opět sám automaticky nastavil výchozí parametry, tedy i paměťové místo s adresou 7 obsadil opět sám nulou. Chceme-li ukázat působení příkazu

POKE 7, 1 ,

musíme pak počítač na určitou dobu zdržet, abychom mohli pozorovat chování kurzoru až do doby, kdy se objeví READY a doj-

de k automatickému nastavení parametrů. Zdržení chodu počítače můžeme realizovat například pomocí příkazu pro pauzu

WAIT (5ØØ) ,

který způsobí pozastavení chodu počítače na dobu

5ØØ * Ø.1 s.

Odešleme-li tlačítkem CR příkaz

POKE 7, 1 : WAIT (5ØØ) ,

pak po dobu 5Ø sekund kurzor nebliká, teprve po uplynutí této doby se objeví hlášení READY s blikajícím kurzorem.
Odešleme-li tlačítkem CR příkaz

POKE 7, Ø : WAIT (5ØØ) ,

kurzor po dobu pauzy narozdíl od předchozího případu bliká.

Všimněte si rovněž, že odešlete-li tlačítkem CR příkaz

PRINT PEEK (7) ,

dostaneme na obrazovce jako výsledek vždy nenulové číslo, tedy paměťové místo s adresou 7 je obsazeno nenulovým číslem po dobu provádění tohoto příkazu, tedy při provádění příkazu kurzor nebliká.

4.2. Adresy s časovými informacemi

Na paměťových místech s adresami 8, 9, a 1Ø /decimálně/, tedy 8, 9 a A /hexadecimálně/ je tzv. časovač. Každých 2Ø ms - tedy po dobu 1 periody síťového napětí 5Ø Hz - se hodnota na paměťovém místě s adresou 8 zvýší o jednotku. Pokud tato hodnota přesáhne 255 /decimálně/, tedy FF /hexadecimálně/, zvýší se hodnota na adrese 9 o jedničku a hodnota na adrese 8 se vynuluje. Na adrese 9 se hodnoty mění 256krát pomaleji

než na adrese 8. Obdobný vztah je mezi hodnotami na paměťových místech s decimálními adresami 9 a 1Ø.

Pomocí časovače můžeme odhadovat dobu provádění některých příkazů, výpočtů, případně i částí programů. Jako ukázku určíme dobu, jaké je potřeba k výpočtu a napsání výsledku výrazu

PI / 18Ø + 7.5 * Ø.2 / 1Ø ^ 2 .

Vložme do počítače program

```
1Ø PRINT PEEK ( 8 ) ; PEEK ( 9 ) ; PEEK ( 1Ø )
2Ø PRINT PEEK ( 8 ) ; PEEK ( 9 ) ; PEEK ( 1Ø )
3Ø END
```

Když tento program několikrát za sebou opakovaně spustíme, vidíme podle čísel vypsaných na obrazovce, že mezi provedením řádku 1Ø a 2Ø uplynula doba 6 * 2Ø ms /nejčastěji/. Na obrazovce mají totiž hodnoty vypsané z paměťového místa o adrese 8 nejčastěji rozdíl 6.

Vložme nyní do programu řádek

15 A = PI / 18Ø + 7.5 * Ø.2 / 1Ø ^ 2 .

Spustíme-li doplněný program několikrát za sebou opakovaně, pak zjistíme, že rozdíl hodnot z adresy 8 je nejčastěji 1Ø. Doba zpracování vloženého řádku je proto

(1Ø - 6) * 2Ø ms.

Nyní řádek 15 zaměníme za tento:

15 PRINT PI / 18Ø + 7.5 * Ø.2 / 1Ø ^ 2 .

Spustíme-li opakovaně několikrát za sebou takto upravený program, je rozdíl hodnot z adresy 8 nejčastěji 15, doba provádění nynějšího příkazu na řádku 15 je

(15 - 6) * 2ø ms .

Příkaz PRINT se tedy provádí pomaleji, než příkaz LET.

4.3. Kód znaku, na nějž ukazuje kurzor

Tento kód znaku je uchováván v paměťovém místě s adresou 11 /decimálně/, což je B /hexadecimálně/. Znak kurzoru bývá většinou mezera, která se při blikání střídá se svým inverzním znakem - bílým čtverečkem. O tom se přesvědčíme tak, že tlačítkem CR odešleme příkaz

PRINT PEEK (11)

a jako výsledek dostaneme 32, což je decimální kód pro mezeru.

Zdálo by se, že stačí provést například

POKE 11, 77 ,

aby místo čtverečku blikalo písmeno M jakožto kurzor. Ale po odeslání příkazu tlačítkem CR a po jeho provedení opět počítač nastaví na obrazovku READY a kurzor přemístí na prázdnou část obrazovky, čímž se na paměťové místo s adresou 11 dostane opět kód mezery.

Proto pro demonstraci obsazení paměťového místa s decimální adresou 11 je nutno použít jiného postupu. Na obrazovku napíšeme příkaz

PRINT PEEK(11) ,

ale neodesíláme jej. Vrátíme se kurzorem zpět na počátek tohoto příkazu, pak sjedeme kurzorem o textový řádek na obrazovce níže - stiskneme tedy dvakrát krátce tlačítko [↓] a na toto místo napíšeme nějaký znak, třeba písmeno R. Opět se kurzorem vrátíme na počátek tohoto řádku - provedeme posuv o jedno místo vlevo - a pak se kurzorem vrátíme na počátek

příkazu PRINT PEEK (11) , celý příkaz kurzorem projedeme a teprve potom stiskneme CR. Na obrazovce se objeví číslo 82, což je decimální kód písmena R.

Co se vlastně při takovém postupu děje?

Po odeslání příkazu tlačítkem CR kurzor automaticky přešel na počátek následujícího tiskového řádku na obrazovce a jeho poloha se ztotožnila s polohou písmena R, které jsme si na obrazovce připravili. V tom okamžiku se paměťové místo s decimální adresou 11 obsadilo kódem znaku R. Současně došlo k provedení příkazu, obsah jmenovaného paměťového místa byl vypsán na obrazovku. Pak teprve následovalo hlášení READY a automatické nastavení jiného obsahu paměťového místa s adresou 11.

Poznámka:

Všimněte si, že výše uvedeným způsobem bychom mohli k jednotlivým znakům najít jejich decimální kódy obdobně, jak to provádí funkce ASC.

4.4. Adresy kurzoru v oblasti VIDEORAM

Víme už, že VIDEORAM je ta část paměti RAM počítače, která se nám zobrazuje na televizní obrazovce. VIDEORAM počíná adresou

ECøø hexadecimálně, tj.

6ø416 decimálně

a končí adresou

EFFF hexadecimálně, tj.

61439 decimálně.

Jestliže počítač zapneme a v levém horním rohu se objeví písmeno B - počátek hlášení BASIC - znamená to, že první

paměťové místo VIDEORAMU s decimální adresou 6Ø416 je obsazena kódem písmena B, což je decimálně 66 a hexadecimálně 42, druhé s decimální adresou 6Ø417 je obsazeno kódem písmena A atd. Lze se o tom přesvědčit odesláním příkazu

PRINT PEEK (6Ø416) ; PEEK (6Ø417)

tlačítkem CR.

Spodní pravý roh obrazovky má adresu 61439 decimálně, což je EFFF hexadecimálně.

Protože na obrazovce je vždy na nějakém místě kurzor, musí existovat někde mimo VIDEORAM v paměti počítače uchovaný údaj o jeho okamžité poloze, tj. adresa, na níž kurzor zrovna je. Tato adresa se uchovává na paměťových místech s decimálními adresami 12 a 13, neboli C a D hexadecimálně. Protože na každém paměťovém místě může být jen číslo z decimálního intervalu ⟨Ø; 255⟩ a adresy VIDEORAMU jsou větší než decimální číslo 6Ø415, pak vidíme, že k uchovávání nějaké adresy videoramu skutečně nestačí jen jedna paměťová buňka, že takové číslo se musí rozložit do dvou paměťových míst.

Rozložení adresy VIDEORAMU do dvou paměťových míst s decimálními adresami 12 a 13 vypadá takto:

na adrese 13 - celé část z podílu adresy kurzoru a čísla 256

na adrese 12 - rozdíl mezi adresou kurzoru a součinem čísel, z nichž jedno je uloženo na adrese 13 a druhé je 256

Rozklad jsme provedli pro decimální čísla.

Tedy konkrétně:

Je-li kurzor v levém horním místě obrazovky, je na adrese 6Ø416 /decimálně/. Provedeme-li dělení

6Ø416/256 ,

pak celá část výsledku je 236 /decimálně/. Toto číslo se uloží do paměťového místa s adresou 13 /decimálně/. Nyní se provede

6Ø416 - 236 * 256 = Ø ,

což znamená, že na paměťovém místě s adresou 12 /decimálně/ bude uloženo číslo Ø.

Chceme-li mít na okamžik kurzor na místě písmena A v nápisu BASIC, musí být decimální adresa 13 obsazena decimálním číslem 236 a decimální adresa 12 obsazena decimálním číslem 1, neboť

6Ø417 = 236 * 256 + 1 .

Odešleme pomocí tlačítka CR v režimu BASIC příkaz

POKE 12, 1: POKE 13, 236: POKE 7, Ø : WAIT (1ØØ) .

Třetí příkaz POKE slouží k tomu, aby kurzor na uvedeném místě blikal po dobu určenou pauzou v příkaze WAIT.

Po odeslání uvedeného příkazu tlačítkem CR kurzor chvíli bliká na určeném místě, pak se objeví nápis READY a kurzor změní místo, protože při tomto hlášení počítač automaticky upravil parametry pro další práci.

4.5. Poloha kurzoru v řádku a sloupci

S předcházejícími adresami úzce souvisejí decimální adresy 14 a 15 - hexadecimálně E a F. Na první jmenované adrese je uloženo číslo sloupce na obrazovce, na němž je v daném okamžiku kurzor, na druhé adrese je uloženo číslo řádku na obrazovce, na němž je v daném okamžiku kurzor. Funkci

obou adres nám ozřejmí následující postup.

Stiskneme tlačítko RES a napíšeme příkaz

PRINT PEEK (14) , PEEK (15)

Po odeslání tlačítkem CR se na obrazovce objeví čísla Ø a 8.
Zdůvodnění je následující:

Po odeslání příkazu kurzor přešel na počátek následujícího řádku / a tedy do sloupce s pořadovým číslem Ø / a na osmý řádek obrazovky shora. Odpočítejte si řádky od nejhořejšího třeba tak, že posunujete kurzor směrem nahoru a uvědomíte si, že nejhořejší řádek obrazovky má pořadové číslo Ø. Při odeslání příkazu se decimální adresy 14 a 15 tedy obsadily po řadě čísly Ø a 8, která byla vzápětí vypsána na obrazovce v souladu se zadanými příkazy.

Pomocí obsazení paměťových míst s decimálními adresami 14 a 15 lze také zajistit tisk do libovolné polohy na obrazovce obdobně, jako to provádí příkaz PRINT & .

Příklad:

Na řádek 1Ø a sloupec 1Ø obrazovky napište písmeno A.

Řešení:

Napíšeme příkaz

POKE 14, 1Ø : POKE 15, 1Ø : PRINT "␣A"

a odešleme jej tlačítkem CR. Pozor - před písmenem A je v posledním příkazu ještě mezera, A tedy nenásleduje bezprostředně první úvozovky.
Po odeslání příkazu se písmeno A objeví v předepsané pozici na obrazovce.

4.6. Přepínání na grafické symboly

Na adrese 16 /decimálně/, což je 1Ø /hexadecimálně/, je buď sudé nebo liché číslo. Pokud je sudé, tisknou se na obrazovce písmena a čísla. Pokud je liché, tisknou se grafické znaky uvedené na příslušných černých tlačítkách klávesnice.

Příklad:

Příkaz

POKE 16, 15 : PRINT " D "

po svém odeslání tlačítkem CR vytiskne grafický symbol → , který je pod písmenem D na tlačítku uveden.

Příkaz

POKE 16, Ø : PRINT " D "

po svém odeslání tlačítkem CR vytiskne opět písmeno D.

Při hlášení READY počítač automaticky obsazuje paměťové místo s decimální adresou 16 číslem Ø.

4.7. Přepínání na inverzní tisk

Paměťové místo s adresou 17 /decimálně/, což je 11 /hexadecimálně/, řídí tisk pozitivních nebo negativních /inverzních/ znaků. Analogicky s minulým případem je pro sudá čísla na paměťovém místě s adresou 17 /decimálně/ tisk pozitivní, pro lichá čísla na uvedeném paměťovém místě negativní /inverzní/.

Příklad:

Příkaz

POKE 17, 13 : PRINT " D "

po odeslání tlačítkem CR vytiskne písmeno D inverzně.

Objeví-li se nápis READY, jmenovaná adresa je automaticky obsazena číslem Ø.

Vysvětlete sami, proč program

```
1Ø POKE 17, 1 : PRINT " D " : POKE 17, Ø :
   PRINT " D "
2Ø GOTO 1Ø
```

tiskne střídavě písmeno D pozitivně a negativně.

4.8. Délka stránky na obrazovce

Normálně lze na obrazovce umístit 32 řádků - řádky lze odpočítat pomocí svislého pohybu kurzoru. To, že se za normálních podmínek provozu počítače text objevuje na každé druhé, je věc, k níž se ještě vrátíme.

Lze však zajistit, že těchto řádek na obrazovce je méně. Provedeme to vhodným obsazením paměťového místa s adresou 19 /decimálně/, což je 13 /hexadecimálně/.

Provedeme-li například odeslání příkazu

```
POKE 19, 15
```

tlačítkem CR, je řádků na obrazovce pouze 16, spodní část obrazovky /od řádku s pořadovým číslem 15 až ke spodnímu okraji/ není zasažená posuvem řádků při práci s počítačem.

Názorně nám to předvede tento program, před jehož vložením do paměti počítače stiskněte pro všechny případy RES.

```
 5 CLS
1Ø PRINT & 2Ø, 2Ø " A "
12 PRINT & Ø, Ø
15 POKE 19, 15
2Ø PRINT 1
```



```
3Ø GOTO 2Ø
```

Příkaz na řádku 5 čistí obrazovku, příkaz na řádku 1Ø napíše písmeno A na řádek 2Ø a sloupec 2Ø, pak se zmenší stránka pomocí příkazu na řádku 15 a na takto zmenšenou stránku se píší neustále jedničky pomocí programových řádků 2Ø a 3Ø. Příkaz na řádku 12 pouze zajistí, že první jednička bude napsána do levého horního rohu obrazovky.

Po startu tohoto programu příkazem RUN vidíme, že písmeno A je neustále na stejném místě, zatímco jedničky se posouvají po prvních šestnácti řádcích.

Lze toho využít, chceme-li mít stabilně na obrazovce určitý obrázek nebo schema a současně provádět výpočty tak, aby obrázek nebyl poškozen nebo "neujel" nahoru.

Na paměťové místo s decimální adresou 19 dáváme tedy číslo posledního řádku, který na obrazovce chceme. Po zapnutí počítače je na této adrese decimální číslo 3Ø.

Všimněte si dále, že po hlášení READY nedojde ihned automaticky k původnímu obsazení adresy 19 /decimálně/, ale až po stisku červeného tlačítka RES.

4.9. Řádkování textu na obrazovce

Hustota řádků na obrazovce je určena číslem uloženým na paměťovém místě s adresou 2Ø /decimálně/, což je 14 /hexadecimálně/.

Po stisku tlačítka RES nebo po zapnutí počítače je automaticky jmenovaná adresa obsazena číslem 2 a proto počítač tiskne na obrazovce ob jeden řádek / na každý druhý /. Jestliže hodnotu na uvedené adrese změníme na 1, řádkování na obrazovce se zhustí, tisk je na každý následující řádek.

Obsadíme-li paměťové místo s decimální adresou 2Ø větším celým číslem, mezera mezi tiskovými řádky na obrazovce se úměrně zvětší. Zkuste sami účinek příkazů

```
POKE 2Ø, 1     nebo
POKE 2Ø, 5     apod.
```

Zadejte výpis nějakého programu za těchto podmínek.

Poznámka:

Obsazení paměťového místa s decimální adresou 2Ø se nemění po nápisu READY na obrazovce, číslem 2 se obsadí až po stisku RES nebo opětném zapnutí počítače automaticky.

4.10. Tóny a jejich délky

Protože o tomto tématu bylo již pojednáno podrobněji v příručce Jedlička, Feil: " Basic pro začátečníky " , nyní shrneme pouze pro úplnost základní poznatky.

Výška tónu je řízena obsazením paměťového místa s adresou 23 /decimálně/, což je 17 /hexadecimálně/. Délka tónu je řízena obsazením paměťového místa s adresou 24 /decimálně/, což je 18 /hexadecimálně/.

Po zapnutí počítače jsou uvedená paměťová místa obsazena po řadě decimálními hodnotami 4 a 18.

K vyvolání daného tónu nestačí pouze obsadit uvedená paměťová místa vhodnými parametry, je nutno tón vyvolat odesláním příkazu

```
CALL HEX ( F973)
```

tlačítkem CR. Jeden tón dosti odlišný od běžného akustického signálu počítače dostaneme odesláním příkazu

```
POKE 23, 9 : POKE 24, 121 : CALL HEX ( F973)
```


Všechny možné tóny nám předvede program

```
5 POKE 24, 18
1Ø FOR I = Ø TO 255
2Ø POKE 23, I
3Ø CALL HEX(F973)
4Ø NEXT I
```

Chcete-li tóny slyšet delší dobu, zaměníte programový řádek číslo 5 za řádek

```
5 POKE 24, 1ØØ
```

Jakmile se na obrazovce objeví hlášení READY, dochází automaticky k nastavení původních parametrů na decimálních adresách 23 a 24.

4.11. Určení meziblokové distance

Jistě jste již slyšeli nahrávku nějakého programu v jazyce BASIC na magnetofonovém pásku. Všimli jste si, že jednotlivým programovým řádkům odpovídá v nahrávce určitý "rachot" a mezi jednotlivými "rachoty" je mezera, kdy slyšíte pouze jednoduchý tón - tzv. pilotní kmitočet. Po dobu této mezery mezi jednotlivými "rachoty" odpovídajícími záznamu dvou sousedních řádků je řádek definitivně začleňován do paměti počítače. Pokud jsou programové řádky poměrně dlouhé - tedy mají délku skoro dva a půl řádku na obrazovce, víte, že často dochází k chybám při čtení takto dlouhých řádků z magnetofonového záznamu programu v jazyce BASIC. Chyby jsou způsobeny tím, že mezera mezi sousedními "rachoty" je malá a počítač během ní nestačí nahrávaný řádek správně zpracovat.

Tuto mezeru - meziblokovou distanci - lze regulovat

pomocí čísla, které je na paměťovém místě s decimální adresou 28, což je hexadecimálně 1C. Po zapnutí počítače je na uvedeném paměťovém místě automaticky nastaveno decimální číslo 33, což je hexadecimálně 21. Tento parametr je plně postačující v případě, že programové řádky nejsou extrémně dlouhé.

Máme-li však nahrát na magnetofonový pásek program, který má právě velmi dlouhé řádky, nutno raději číslo na uvedené adrese zvětšit a tím zajistit zvětšení meziblokové distance. Před pořízením přehrávky programu na magnetofonový pásek proto odešleme pomocí tlačítka CR příkaz

POKE 28, 39 ,

čímž jsme decimální adresu 28 obsadili decimálním číslem 39 /obsazení lze provést i v režimu MONITOR pomocí příkazu S - tedy adresu 1C obsadíme hexadecimálním číslem 27/. Pak teprve přehrajeme program z paměti počítače na magnetofonový pásek obvyklým způsobem, na nějž jsme zvyklí u programů v jazyce BASIC. Decimální adresu 28 můžeme obsadit i číslem větším, například i decimálním číslem 1ØØ, což je hexadecimálním číslem 64 - dojde opět ke znatelnému ještě většímu prodloužení meziblokové distance, ale již celkem zbytečnému.

Pokud jsme pořídili záznam programu na magnetofonový pásek při zvětšené meziblokové distancí a chceme tento program přehrát opět do paměti počítače, nemusíme údaj na adrese 28 nijak měnit - tedy počítač přijme záznam programu s libovolně zvětšenou meziblokovou distancí, i když na adrese 28 je decimální číslo 33, které automaticky nastavuje počítač po zapnutí.

Poznámka:

Zvětšení meziblokové distance nese s sebou i nepatrné

prodloužení doby přehrávání programu do paměti počítače - jeho délka na magnetofonovém pásku je větší v souladu se zvětšením meziblokové distance. Získáme však větší spolehlivost čtení programu počítačem při přehrávce.

5. Uložení programu a proměnných v paměti počítače

V této partii si něco řekneme o uložení programu, který je v jazyce BASIC, do paměti počítače a rovněž, jakým způsobem se do paměti počítače zanášejí hodnoty proměnných, ať již číselných nebo řetězcových. Budeme potřebovat některé dovednosti z předchozích kapitol, zejména:

a/ příkazy v režimu MONITOR, zvláště D a S;

b/ kódy jednotlivých znaků;

c/ decimální a hexadecimální čísla, jejich převody.

5.1. Uložení programu v jazyce BASIC do paměti počítače

Pokud máme počítač zapnutý a pracovali jsme s ním, vypneme jej a po několika okamžicích ho opět zapneme. Důvod již známe - paměť se vyčistí od zbytků proměnných, programů apod., což nelze udělat pouze stisknutím červeného tlačítka RES.

Nyní v režimu MONITOR si prohlédneme část paměti počínaje adresou 16A. V režimu MONITOR odešleme tedy tlačítkem CR příkaz

D16A .

Na obrazovce se objeví oblasti paměti obsazené buď čísly ØØ /od adresy 16A do 17F - hexadecimálně/, nebo obsazené hexadecimálním číslem FF /od adresy 18Ø do 1BF - hexadecimálně/. Pokud by se na jednotlivých paměťových místech objevila místy i jiná čísla, mají svůj původ v akceptování různých poruch při zapnutí počítače. Tak tedy vypadá čistá programová paměť počítače. Nyní se vrátíme pomocí příkazu R do režimu BASIC. V režimu BASIC vložíme následující jednoduchý program:


```
1ØØØ REM AAAAA
1Ø1Ø REM BBBBB
1Ø2Ø REM CCCCC
1Ø3Ø REM DDDDD
```

Po vložení programu do počítače se opět podíváme v režimu MONITOR na nynější obsazení úseku paměti RAM od hexadecimální adresy 16A. Vidíme, že úsek vypadá nyní takto:

```
Ø16A  75 Ø1 E8 Ø3 8E 41
Ø17Ø  41 41 41 41 ØØ 8Ø Ø1 F2
Ø178  Ø3 8E 42 42 42 42 42 ØØ
Ø18Ø  8B Ø1 FC Ø3 8E 43 43 43
Ø188  43 43 ØØ 96 Ø1 Ø6 Ø4 8E
Ø19Ø  44 44 44 44 44 ØØ ØØ ØØ
Ø198  ØØ FF FF FF FF FF FF FF
Ø1AØ  FF FF FF FF FF FF FF FF
.............
........
```

Vidíme, že náš program je uložen v paměti počínaje hexadecimální adresou 16A až do hexadecimální adresy 198 - jinde zůstalo zachováno původní obsazení paměťových míst.
První programový řádek

1ØØØ REM AAAAA

je zobrazen na paměťových místech s hexadecimálními adresami v rozmezí od 16A do 174 a vypadá

75 Ø1 E8 Ø3 8E 41 41 41 41 41 ØØ .

Co jednotlivá hexadecimální čísla říkají ?

Všimneme si nejprve, že se v řádku objevuje celkem

pětkrát hexadecimální číslo 41, což odpovídá hexadecimálnímu kódu písmana A za příkazem REM v našem programu. Dále si všimneme, že řádek končí vždy ØØ.

Nyní se podíváme, co znamenají první dvě hexadecimální čísla 75 a Ø1. Napíšeme-li je v obráceném pořadí, dostaneme hexadecimální číslo Ø175 a toto číslo nám určuje adresu, na níž začíná druhý programový řádek. Podíváme-li se na část paměti počínaje adresou Ø175, zjistíme, že je to skutečně adresa prvního paměťového místa za místem obsazeným ØØ jakožto koncem záznamu prvního programového řádku.

Tedy v prvních dvou paměťových místech záznamu každého programového řádku je současně odkaz na začátek následujícího programového řádku. V této souvislosti si všimneme, že na začátku záznamu druhého programového řádku jsou čísla 8Ø a Ø1 a hexadecimální číslo Ø18Ø udává adresu, na níž začíná záznam třetího programového řádku atd.

Na třetím a čtvrtém místě záznamu prvního programového řádku jsou čísla E8 a Ø3 /hexadecimálně/. Tato čísla v sobě skrývají informaci o čísle programového řádku. Musíme však jejich pořadí opět prohodit, čímž dostaneme hexadecimální číslo Ø3E8, které když převedeme na decimální tvar, dostaneme číslo prvního programového řádku - tedy 1ØØØ. Pro zopakování převod probíhá takto:

$$3 \qquad E \qquad 8$$

$$3 * 16^2 + 14 * 16^1 + 8 * 16^Ø \quad .$$

Na pátém místě záznamu prvního programového řádku je hexadecimální číslo 8E, což odpovídá kódování instrukce REM. Všimněte si, že před sadou stejných čísel v záznamu každého pro-

gramového řádku je opět toto hexadecimální číslo 8E odpovídající příkazu REM v jednotlivých programových řádcích.

Obraťme nyní pozornost na záznam posledního programového řádku našeho programu. Je ve tvaru

96 Ø1 Ø6 Ø4 8E 44 44 44 44 44 ØØ .

Číslo posledního programového řádku je určeno hexadecimálně číslem Ø4Ø6, po jeho převedení do decimální číselné soustavy dostaneme 1Ø3Ø, první dvě hexadecimální čísla záznamu řádku odkazují na adresu Ø196, kde by měl začínat další programový řádek. Protože náš program již další řádek nemá, je paměťové místo s touto adresou a následující obsazeno pouze ØØ.

Konec zápisu programu v paměti počítače poznáme tak, že ho tvoří 4 paměťová místa za sebou obsazená ØØ.

Protože umíme pracovat v režimu MONITOR pomocí příkazu S - dosazování nových čísel na určitá paměťová místa - zkusme změnit číslo prvního programového řádku z 1ØØØ na 1ØØ1 a rovněž písmeno A za příkazem REM změníme na písmeno X. Změnu nebudeme provádět v režimu BASIC, jak jsme dosud zvyklí, ale změnu provedeme pomocí příkazu S v režimu MONITOR.

Protože číslo prvního programového řádku je dáno hexadecimálními čísly E8 a Ø3 na hexadecimálních adresách 16C a 16D, pak stačí pomocí příkazu S nahradit na adrese 16C číslo E8 hexadecimálním číslem o jednu větším, tedy E9, neboť hexadecimálně Ø3E9 je decimálně 1ØØ1. Prvnímu písmenu A v prvním programovém řádku odpovídá hexadecimální číslo 41 na paměťovém místě s hexadecimální adresou 16F. Pokud písmeno A na tomto místě chceme nahradit písmenem X, jehož hexadecimální kód je 58, provedeme to opět pomocí příkazu S v režimu MONITOR tak, že číslo 41 na uvedené adrese nahradíme číslem 58. Po provede-

ní obou změn hodnot na uvedených adresách se pomocí příkazu R v režimu MONITOR vrátíme do režimu BASIC a pomocí příkazu LIST se přesvědčíme, že všechny požadované změny v programu byly realizovány.

5.2. Funkce tlačítka RES ve vztahu k programům v jazyce BASIC

Máme-li program z předchozího článku napsaný na obrazovce v režimu BASIC, stiskneme nyní tlačítko RES. Víme, že náš program z obrazovky zmizí, nedá se již vyvolat následným příkazem LIST. Vraťme se pomocí tlačítka BR do režimu MONITOR a pomocí příkazu D se podíváme na úsek paměti od hexadecimální adresy 16A. Vidíme, že se záznam našeho programu působením tlačítka RES změnil pouze v tom, že na hexadecimálních adresách 16A a 16B jsou nyní čísla ØØ, jinak nedošlo k žádnému jinému poškození záznamu programu. Tlačítko RES obsadilo nulami pouze první dvě adresy programového záznamu.

Nyní pomocí příkazu S v režimu MONITOR dosadíme na tyto dvě adresy původní hexadecimální čísla - po řadě 75 a Ø1. Pomocí příkazu D se můžeme přesvědčit, že k obsazení v režimu MONITOR došlo a pak se pomocí příkazu R v řežimu MONITOR vrátíme do režimu BASIC. Odešleme-li nyní příkaz LIST tlačítkem CR, program se opět objeví na obrazovce a můžeme jej i po předchozím stisku tlačítka RES minimálně z obrazovky opsat.

Poznámka:

Uvedeným způsobem je možno "zachránit" programy v jazyce BASIC i po náhodném nebo úmyslném stisku tlačítka RES; druhý případ může nastat, pokud se během chodu programu program stane neovladatelný a přesto bychom chtěli získat dodatečně jeho přesný výpis, případně jej přehrát na magn. pásek.

Takto jednoduchý program jde i spustit příkazem RUN. Většinou však u složitějších programů takové spuštění programu vede k jejich nenávratnému poškození a ztrátě, protože nebyl ošetřen i konec programu. Proto chceme-li obecně takto "zachráněný" program spustit, provádíme navíc tyto operace:

a/ Ve výpisu "zachráněného" programu objedeme kurzorem poslední řádek a odešleme ho tlačítkem CR.

b/ Za poslední řádek doplníme ještě další, který obsahuje příkaz END apod.

Jako další příklad uložení programu v jazyce BASIC do paměti počítače si ukážeme tento program:

```
1Ø FOR I = 1 TO 1Ø
2Ø PRINT I, I ^ 2
3Ø NEXT I
```

Před vložením tohoto programu do počítače opět počítač na několik okamžiků vypněte, aby vás nezmátly některé údaje pocházející z minulého programu. Po vložení našeho druhého programu do počítače přejdeme pomocí tlačítka BR do režimu MONITOR a pomocí příkazu D zjistíme, že záznam našeho programu vypadá takto:

Záznam prvního řádku

76 Ø1	počáteční adresa druhého řádku - Ø176
ØA ØØ	číslo prvního řádku - ØØØA

81	kód instrukce FOR
49	kód písmena I
C1	kód instrukce přiřazení identifikátoru I /nikoliv tedy rovnítko a jeho kód/
31	kód čísla 1
B3	kód instrukce TO
31	kód čísla 1
3Ø	kód čísla Ø
ØØ	konec prvního programového řádku

Záznam druhého řádku

81 Ø1	počáteční adresa 3. řádku - Ø181
14 ØØ	číslo druhého programového řádku
94	kód instrukce PRINT
49	kód písmena I
2C	kód čárky
49	kód písmena I
BD	kód operace mocnění
32	kód čísla 2
ØØ	konec druhého programového řádku

Záznam třetího řádku

88 Ø1	počáteční adresa 4. programového řádku, pokud by existoval - Ø188
1E ØØ	číslo třetího programového řádku
82	kód instrukce NEXT
49	kód písmena I

ØØ	konec třetího programového řádku

Dále v záznamu programu následují skupiny nul, které program uzavírají.

Poznámka:

Jistě jste si všimli, že v prvním řádku programu se vyskytuje rovnítko, které se ale v záznamu programu neprojeví prostřednictvím hexadecimálního kódu rovnítka, tedy 3D, jak by se na první pohled zdálo, ale projeví se pomocí hexadecimálního čísla C1. Rovnítko zde má pouze symbolický význam, v zásadě jde o přiřazení čísla 1 identifikátoru I.

Po stisku tlačítka RES a znovuobsazení paměťových míst s hexadecimálními adresami 16A a 16B a po návratu do režimu BASIC je možno "zachráněný" program vypsat na obrazovku pomocí příkazu LIST.

5.3. Záznam číselných proměnných v paměti počítače

Máme-li počítač zapnutý, vypneme jej a po několika okamžicích jej opět zapneme z již známých důvodů. Do počítače vložíme v režimu BASIC následující jednoduchý program:

```
1  A = 2345
```

aniž ho zatím spustíte. Rozsah volné paměti počítače zjistíte snadno pomocí příkazu MEM v režimu BASIC - objeví se vám číslo 32200 /decimální/. Nyní spustíte tento program příkazem RUN a pomocí příkazu MEM zjistíte, že rozsah volné paměti se snížil o 6 paměťových míst.

Nyní přejdeme pomocí tlačítka BR do režimu MONITOR a pomocí příkazu D se podíváme na obsazení paměťových míst od adresy 16A /hexadecimálně/. Náš jednoduchý program je v oblasti paměťových míst počínaje hexadecimální adresou 16A a za programem je následujících 6 údajů na hexadecimálních adresách od 0177 do 017C:

```
00 41 00 90 12 8C .
```

Na prvních dvou místech jsou kódy identifikátorů - kdyby náš identifikator byl AB a nikoliv jen A, první dvě místa by byla obsazena hexadecimálními čísly 42 a 41 . Proto také IQ 151 může používat nejvýše dvouznakových identifikátorů . Kody jednotlivých znaků dvouznakového identifikátoru jsou uloženy v paměti v opačném pořadí - pro identifikátor AB je v záznamu nejdříve 42 jakožto hexadecimální kód písmena B a pak teprve 41 jakožto hexadecimální kód písmena A.

Další čtyři hexadecimální čísla vyjadřují dohromady číslo, které je identifikátoru přiřazeno, tedy v našem případě decimální číslo 2345. Zobrazení je však složitější; nejdříve

si sestavíme tabulku, jak se na uvedených čtyřech paměťových místech zobrazují celá dekadická čísla od -20 do 20. Pozor - na paměťových místech jsou v tabulce čísla hexadecimální.

decimální číslo	obsazení 4 pam. míst	decimální číslo	obsazení 4 pam. míst
-20	00 00 A0 85	0	AA 00 00 00
-19	00 00 98 85	1	00 00 00 81
-18	00 00 90 85	2	00 00 00 82
-17	00 00 88 85	3	00 00 40 82
-16	00 00 80 85	4	00 00 00 83
-15	00 00 F0 84	5	00 00 20 83
-14	00 00 E0 84	6	00 00 40 83
-13	00 00 D0 84	7	00 00 60 83
-12	00 00 C0 84	8	00 00 00 84
-11	00 00 B0 84	9	00 00 10 84
-10	00 00 A0 84	10	00 00 20 84
-9	00 00 90 84	11	00 00 30 84
-8	00 00 80 84	12	00 00 40 84
-7	00 00 E0 83	13	00 00 50 84
-6	00 00 C0 83	14	00 00 60 84
-5	00 00 A0 83	15	00 00 70 84
-4	00 00 80 83	16	00 00 00 85
-3	00 00 C0 82	17	00 00 00 85
-2	00 00 80 82	18	00 00 10 85
-1	00 00 80 81	19	00 00 10 85
		20	00 00 20 85

Tabulka zobrazení vybraných decimálních čísel

Na první pohled z předchozí tabulky nějakou jednoduchou souvislost mezi dekadickým číslem a jeho zobrazením v počítači nevysoudíme, všimněme si ale nejdříve, že kladná decimální čísla větší než jedna, která lze navíc vyjádřit ve tvaru 2^x se zobrazují jako čtveřice

ØØ ØØ ØØ $\underbrace{81 + X}$

součet v hexadecimální číselné soustavě .

Druhé hexadecimální číslo zprava rozdělí interval mezi dekadickými čísly tvaru 2^x a 2^{x-1} na 128 dílů. Je-li toto druhé hexadecimální číslo z intervalu

$\langle ØØ ; 7F \rangle$,

je vložené dekadické číslo kladné, pokud je z intervalu

$\langle 8Ø ; FF \rangle$,

je vložené dekadické číslo záporné. Všimněte si, že v obou uvedených intervalech je právě 128 hexadecimálních čísel.

Z následujících tabulek vyplyne význam třetího a čtvrtého hexadecimálního čísla /počítáno vždy zprava/. Třetí hexadecimální číslo zprava dělí opět interval mezi dvěma dekadickými čísly, jejichž záznam se liší o jednotku na druhém místě zprava na 256 dílků, čtvrté dělí dále interval mezi decimálními čísly, jejichž záznam se liší o jednotku na třetím místě zprava, na dalších 256 dílků. Názorně to ukazují následující tabulky.

decimální číslo		obsazení 4 pam. míst
2^{15}	32768	ØØ ØØ ØØ 9Ø
128 diferencí po 256	33Ø24	ØØ ØØ Ø1 9Ø
	3328Ø	ØØ ØØ Ø2 9Ø
	33536	ØØ ØØ Ø3 9Ø
	33792	ØØ ØØ Ø4 9Ø
	.....	
	.....	
	6528Ø	ØØ ØØ 7F 9Ø
2^{16}	65536	ØØ ØØ ØØ 91

decimální číslo		obsazení 4 pam. míst
256 diferencí po 1	33Ø24	ØØ ØØ Ø1 9Ø
	33Ø25	ØØ Ø1 Ø1 9Ø
	33Ø26	ØØ Ø2 Ø1 9Ø
	33Ø27	ØØ Ø3 Ø1 9Ø
	.....	
	.....	
	33279	ØØ FF Ø1 9Ø
	3328Ø	ØØ ØØ Ø2 9Ø

Tabulky zobrazení decimálních čísel

Poznámka: Uvědomte si, že v hexadecimální číselné soustavě platí 9Ø = 81 + F .

decimální číslo	obsazení 4 pam. míst
33Ø25	ØØ Ø1 Ø1 9Ø
33Ø25.25	4Ø Ø1 Ø1 9Ø
33Ø25.5	8Ø Ø1 Ø1 9Ø
33Ø25.75	CØ Ø1 Ø1 9Ø
33Ø26	ØØ Ø2 Ø1 9Ø

Tabulka demonstrující zobrazení necelých čísel

Podle uvedených tabulek a našich dosavadních znalostí již dovedeme určit, že decimální číslo -33Ø25 se zobrazí jako hexadecimální čtveřice

ØØ Ø1 81 9Ø

apod.

Ještě si v poslední tabulce ukážeme, jak se zobrazují decimální čísla mezi nulou a jedničkou:

decimální číslo	obsazení 4 pam. míst
1	ØØ ØØ ØØ 81
Ø.5	ØØ ØØ ØØ 8Ø
Ø.25	ØØ ØØ ØØ 7F
Ø.125	ØØ ØØ ØØ 7E
.....	
.....	

Snadno již sami odvodíte, že decimální číslo -Ø.5 se zobrazí jako čtveřice

ØØ ØØ 8Ø 8Ø

a hexadecimální čtveřice

ØØ ØØ 4Ø 8Ø

odpovídá decimálnímu číslu Ø.75.

5.4. Význam funkce PTR

Vrátíme se k našemu předchozímu příkladu, kdy jsme do čisté paměti počítače zavedli pomocí startu programu

```
1  A = 2345
```

proměnnou A .

V režimu BASIC odešleme tlačítkem CR příkaz

```
PRINT PTR(A)
```

a na obrazovce se objeví decimální číslo 377, což je po převedení do hexadecimální číselné soustavy 179. Pomocí tlačítka BR přejdeme nyní do režimu MONITOR a podíváme se na záznam našeho programu a zavedené proměnné v paměti počítače. Vidíme, že právě na hexadecimální adrese 179 je první paměťové místo ze čtyř, do nichž se zobrazilo číslo 2345.

Vidíme rovněž, že za záznam programu v paměti počítače se ukládají hodnoty číselných proměnných. Tato oblast paměti není určena jednoznačně, záleží vždy na délce programu v jazyce BASIC.

5.5. Záznam řetězcových proměnných v paměti počítače

Vyčistíme paměť počítače pomocí jeho krátkého vypnutí a vložíme v režimu BASIC tento jednoduchý program:

```
1 AØ = "ABCDEF"
```

Pomocí příkazu MEM v režimu BASIC zjistíme, že je dosud volných 32195 paměťových míst. Spustíme program pomocí příkazu RUN v režimu BASIC a zjistíme, že se počet volných míst snížil o 6. Počítač si tedy tuto řetězcovou proměnnou "zapamatoval" na 6 paměťových místech. Nyní pomocí tlačítka BR přejdeme do režimu MONITOR a pomocí příkazu D se podíváme, jak vypadá obsazení části paměti od adresy 16Ø. Poznáme jednak záznam programu, jednak vidíme, že řetězec byl zaznamenán na paměťové místa od adresy 17C do 181 /hexadecimálně/ ve tvaru:

```
ØF 41 Ø6 FF 72 Ø1
```

První číslo zleva znamená, že jde o řetězec, druhé je hexadecimálním kodem příslušného identifikátoru řetězcové proměnné, třetí hexadecimální číslo zleva popisuje délku řetězce. Když poslední dvě hexadecimální čísla napíšeme v opačném pořadí, dostaneme hexadecimální číslo Ø172, což je první adresa, na níž začíná v záznamu programu řetězec.

Poznámka:

Je-li identifikátor pro řetězcovou proměnnou dvojpísmenový - např. ABØ - pak na prvním místě zleva v záznamu této zavedené proměnné je součet hexadecimálního kodu druhého písmena - tedy v našem případě B - a hexadecimálního čísla 8Ø .

Změníme nyní v režimu MONITOR pomocí příkazu S délku zavedeného řetězce z Ø6 na Ø4. Pak se vrátíme pomocí příkazu R do režimu BASIC a tlačítkem CR odešleme příkaz

```
PRINT AØ
```


Snadno sami vysvětlíte, proč se na obrazovce objeví pouze písmena

```
ABCD
```

Nyní zkusme ještě v režimu MONITOR změnit adresu v záznamu řetězcové proměnné z hexadecimálního čísla Ø172 na Ø173. Po návratu do režimu BASIC a odeslání příkazu

```
PRINT AØ
```

pomocí tlačítka CR se na obrazovce objeví písmena

```
BCDE
```

Tuto skutečnost opět snadno sami zdůvodníte, pokud jste pozorně četli význam jednotlivých hexadecimálních čísel v záznamu řetězcové proměnné.

5.6. Význam funkce PTR pro řetězcové proměnné

Pomocí příkazu RUN v režimu BASIC spustíme nyní náš předchozí malý program

```
1 AØ = "ABCDEF"
```

a po odeslání příkazu

```
PRINT PTR ( AØ )
```

tlačítkem CR v režimu BASIC dostaneme decimální číslo 382, po převedení do hexadecimální číselné soustavy je to 17E. Na paměťovém místě s touto adresou je uložen údaj o délce řetězce AØ v záznamu uvedené proměnné v paměti počítače.

Výslovnost anglických slov

AND	end
BASIC	bejzik
BREAK	brejk
BYE	baj
bytes free	bajts frí
CALL	kól
CLEAR	klír
clear screen /CLS/	klír skrín
continue /CONT/	kentynjú
FREE	frí
GOSUB	gousab
GOTO	goutu
LOAD	loud
POKE	pouk
READ	ríd
READY	redy
RESTORE	ristór
RETURN	ritérn
RIGHT	rajt
RUN	ran
SAVE	sejv
SCRATCH	skreč
space /SPC/	spejs
square root /SQR/	skvér rút
TO	tu
UNPLOT	anplot
user /USR/	júzr
WAIT	wejt
WORD	word

$ /string/	string

Výslovnost ostatních neuváděných slov:

čtou se tak, jak jsou napsána.

Např.: STOP, END

OBSAH

Monitor IQ 151

příručka pro začátečníky

Autor: RNDr. Miloslav Feil, CSc.

Recenze: RNDr. František Lustig

PaedDr. Jan Kuchař

Schválilo ministerstvo školství ČSR dne 6. listopadu 1985, č.j.: 31005/85 - 211 jako učební pomůcku pro základní a střední školy

Vydalo Komenium, n. p. jako učební pomůcku pro základní a střední školy pod ČKL 02069 v roce 1985

Odpovědná redaktorka: PhDr. Emilie Petrášková

Technická redaktorka: Martina Mášová

Číslo publikace: 5716786 , náklad 20 000 výtisků

Vydání 1.