

**RUTINY ROM**

**24 SPECTRUM**

ČLENSKÁ  
KNŽNICE  
(609.)  
ZO  
SVAZARMU





Jan Šritter, Marcel Dauth

RUTINY ROM ZX SPECTRUM

Členská knižnice 602. ZO Svazarmu

Publikace je určena uživatelům počítačů a dalším zájemcům o práci s domácími mikropočítači Sinclair Spectrum i širšímu okruhu zájemců o malou výpočetní techniku



# **RUTINY ROM**

---

## **24 SPECTRUM**

PRAHA 1987



# R U T I N Y   R O M   Z X - S P E C T R U M

=====

Tato příručka je určena všem majitelům osobního počítače ZX-Spectrum (ZX-Spectrum+).

Každý programátor, který se časem propracoval k Assembleru se snaží vytvářet 'své' programy s maximálním využitím místa v paměti a to ho vede k využití podprogramů již sestrojených a přístupných - tedy podprogramů z ROM. Dalším častým důvodem je 'praktická nemožnost' sestrojít některé velmi složité programy (například výpočet funkce sinus).

Z těchto, i dalších důvodů vznikla příručka, kterou právě držíte v ruce. Tato příručka, první díl připravovaného popisu ZX-Spectra, Vám poskytne základní informaci o rutinách rozdělených do kapitol:

1. klávesnice
2. reproduktor
3. magnetofon
4. obrazovka
5. kalkulačka
6. různé

Při výběru popisovaných rutin byly upřednostněny takové rutiny, které zajišťují elementární operace (čtení klávesnice, styk s magnetofonem), základní aritmetické operace (například kalkulačka), atd. . Podprogramy, které vyžadují existenci mnoha systémových proměnných či celého BASIC-systému byly z aplikačních důvodů omezeny na minimum.

Popis rutin vyžaduje od čtenáře alespoň částečnou znalost ASSEMBLERU Z-80, hexadecimální a binární soustavy čísel.

Nyní je potřeba sjednotit význam některých značení používaných dále v textu. Každé číslo v textu je většinou zapsáno v hexadecimálním tvaru a za ním případně následuje číslo v dekadickém tvaru - 'FF [255]'. Adresy systémových proměnných jsou nahrazeny jejich názvy, například ERR-NR - 5C3A. Je-li název syst. prom. uzavřen do závorek, myslíme tím její obsah (tj. data či adresu; (ERR-NR) - FF). Symbol ((ERR-SP)) je obsahem (ERR-SP) (nepřímé adresování). Základní registry mikroprocesoru jsou značeny nečárkovaně, alternativní registry čárkovaně, např. H'=02 označuje operace; EXX ; LD H,02 A EXX.

U textu jsou užity zkratky;

S.p. - systémová proměnná; z.k. - Zásobník kalkulačky;  $\mu$ P - mikroprocesor

Nedoporučujeme přeskakování kapitol a vyhledávání rutin bez prvotního informativního přečtení, neboť v textu je využíváno již nabytých znalostí.

Příjemnou práci s příručkou přejí autoři:

© Jan Šritter & © Marcel Dauth  
602. ZO Svazarmu Praha 1987



# 1 . K L Á V E S N I C E =====

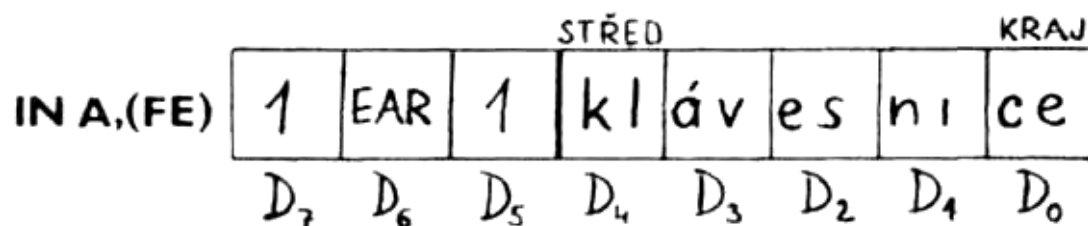
Klávesnice standardního počítače ZX-Spectrum je tvořena čtyřiceti tlačítky, která jsou rozdělena do 4 řad po deseti. Unitálně je však rozdělena na osm řádek po pěti tlačítkách (vzniklá půlením řádek klávesnice). Každou takovouto 'půlřádku' můžeme snadno přechít instrukcí IN. Nižší bajt adresy instrukce IN je vždy FE [254], vyšší bajt je v binárním zobrazení zastoupen jednou nulou a sedmi jedničkami (viz tabulka 1). Postavení nuly určuje čtení příslušné 'půlřádky' instrukcí IN.

Tabulka 1 - adresy kláves

Adresa	[dek.]	Klávesy	vyšší bajt
FEFE	65278	U,C,X,Z,CS	11111110
FDFE	65022	G,F,D,S,A	11111101
FBFE	64510	T,R,E,W,Q	11111011
F7FE	63486	5,4,3,2,1	11110111
EFFE	61438	6,7,8,9,0	11101111
DFFE	57342	Y,U,I,O,P	11011111
BFFE	49150	H,J,K,L,ENT	10111111
7FFE	32766	B,N,M,SS,SP	01111111

Instrukcí IN tak zaplníme registr (třeba A) osmibitovým datem, které má následující formát.

Obrázek 1 - formát informace o 'půlřádce'





Nejnižší bit obsahuje logickou jedničku jestliže tlačítko, které je z dané 'půlřádky' nejvíce u kraje, není stisknuto a hodnotu logické nuly je-li stisknuto. Bit D4 obsahuje informaci o stlačení klávesy v dané 'půlřádce' nejbližší středu klávesnice. Bit D6 'EAR' bude popsán v kapitole 3.

Stejným způsobem se 'čte' i klávesnice u počítače ZX Spectrum+. Určité klávesy nahrazují stisknutí jiných kláves společně s některým SHIFTem. Jsou to klávesy: ◀ (C.S.&5); ▶ (C.S.&6); ↓ (C.S.&6); ↑ (C-S.&7); TRUE VIDEO (C.S.&3); INU VIDEO (C.S.&4); CAPS LOCK (C.S.&2); EDIT (C.S.&1); GRAPH (C.S.&9); 'DELETE (C.S.&0) ; BREAK (C.S. &SPACE) ; EXTEND MODE (C.S.&S.S.) ; klávesa ';' (S.S.&0) ; klávesa ' "' (S.S.&P) ; klávesa '.' (S.S.&M) ; klávesa ',' (S.S.&N). Stiskneme-li například klávesu DELETE, počítač zaznamená stisknutí kláves CAPS SHIFT a 0, které tak i vyhodnotí.

K získání rychlé informace o stisknutí klávesy lze výhodné použít instrukcí: IN A, (18); IN A,<1A>; IN A,(1C) a IN A, (1E) v BASICu např. [IN 24, IN 26, IN 28, IN 30]. Výsledkem je logický součin informací všech 'půlřádek'.

Pomocí instrukcí IN A,(19); IN A,(1B); IN A,(1D) a IN A,(1F) lze získat náhodná čísla. Kódy generované uvedenými porty zjistíme prvním programem, který načítá získaný kód z portu do paměti. U programu je pro záznam vymezena paměť od 7530 do FFFF [30000 až 65535].

Nezapomeňte na 'CLEAR 29999' !!!

Uáš počítač může mít odchylky od popisovaného děje na zkušebním počítači, využijte proto možnost se přesvědčit o pravdivosti statistiky. První program uložený mimo oblast pro záznam (např. v paměti pro tiskárnu) zaznamená kódy na portu v čase kolem 2 sec. Na první prohlédnutí záznamu je patrné periodické opakování mezer a užitečných dat. Další program provede jednoduchou statistiku o výskytu užitečných kódů. U celém záznamu nebylo dle 'výskytu' zjištěno průměrně 69 kódů; namátkou uvedeme několik kódů, které na zkušebním počítači nebyly generovány: (14, 1C, 25, 26, 2C, 2F, 61, 71, 8A, A4, DC, EE, EF). Stejným programem byla na počítači ZX Spectrum+ zjištěna odchylka ve 'výskytu' u kódů 14 a 1C. Oblasti užitečných dat jsou dlouhé cca 02A8 [688] a 'nevyužitě' mezery <FF> cca 0243 [579] bajtů.

Program 1 - náhodné číslo

NávěštíAssembler		;Poznámka	Kódy hex. a dek.	
Rnd	LD HL,7530	;začátek záznamu	213075	[33,48,117]
	LD A, 00	;kód pro ukončení	3E00	[62,0]
	LD C, 19	;číslo portu	0E19	[14,25]
čtení	IN B, (C)	;čtení portu	ED40	[237,64]
	LD (HL),B	;uložení	70	[112]
	INC HL	;nové místo záznamu	23	[35]
	CP H	;konec záznamu?	BC	[188]
	RET Z	;návrat H=A, <0000>	C8	[200,3]
	JR čtení	;opakování	18F8	[24,248]

U dalších programech bude užíváno stejného formátu bez záhlaví.

## Program 2 - statistika kódů

Stat	LD	HL,5800	;Umazání tabulky	210058	[33,0,88	1
	LD	DE,5801	; 'výskytu'	110158	[17,1,88	1
	LD	BC,01FF		01FF01	[1,255,1	1
	LD	A, 00		3E00	[62,0	1
	LD	(HL),A		77	[119	1
	LDIR			EDB0	[237,192	1
	LD	DE,5800	; tabulka 'výskytu'	110058	[17,0,88	1
Nový	LD	HL,7530	; - " - dat (30000)	213075	[33,48,117	1
	LD	BC,8ACF	; délka dat (38535)	01CF8A	[1,207,138	1
Pokr	LD	A,E	; kód pro hledání	7B	[123	1
	CPIR		; hledá A=(HL)	EDB1	[237,193	1
	LD	A, 00		3E00	[62,0	1
	CP	B	; kontrola konce	B8	[184	1
	JR	C,Hodně	; bloku dat	3803	[56,3	1
	CP	C		B9	[185	1
	JR	Z,Nula		280D	[40,13	1
Hodně	EX	DE,HL	; další nalezený kód	EB	[235	1
	LD	A,(HL)		7E	[126	1
	CP	FF	; bude přeplněno ?	FEFF	[254,255	1
	JR	NZ,Npr1	; pokračuj od 'Npr1'	2003	[32,3	1
	INC	H		24	[36	1
	INC	(HL)	; vyšší bajt + 1	34	[52	1
	DEC	H		25	[37	1
Npr1	INC	(HL)	; zvětšení bajtu	34	[52	1
	EX	DE,HL	; 'výskytu'	EB	[235	1
	JR	Pokr	; hledej dál	18E8	[24,232	1
Nula	DEC	A		3D	[61	1
	CP	E	; konec dat ?	B8	[187	1
	RET	Z	; vrať se do BASICU	C8	[200	1
	INC	E	; další kód	1C	[28	1
	JR	Nový		18DC	[24,228	1

Program 'statistika kódů' byl uložen monitorem MON2 do bafru tiskárny a tam též spuštěn. Na adrese 5800 [22528] je pak uložen nižší bajt a na adrese o 0100 větší vyšší bajt počtu kódů 00 v 'záznamu', na adrese 5801 je nižší bajt počet kódů 01 vyskytujících se v datech ... atd. Tabulka končí četností kódu FF jejíž nižší bajt je na adrese 58FF a vyšší bajt na adrese 59FF.

## 028E KEY-SCAN

CALL     čtení klávesnice

Ustup:   -

Výstup: DE = kód kláves

A = D + 01

Příznak Z - v pořádku

NZ - nesmyslná kombinace

Mění:   BC, HL, příznaky

- - -

Nevyžaduje žádný vstupní parametr. Po návratu z podprogramu KEY-SCAN je registr E naplněn kódem stisknutého tlačítka 00 - 27 (podle tabulky 2), nebo kódem FF, není-li stisknuto žádné tlačítko. Registr D obsahuje kód stisknutých shiftových tlačítek a FF pro nestisknuté shifty.

U jednotlivých případech je hodnota v reg. DE uložena podle tabulky 3.

Hexadecimální kód klávesy je v levém horním rohu a dekadický v pravém rohu klávesy. Pro zajímavost si všimněte začátku (klávesa 'B') a šnekovité pokračování kódů.

Tabulka 2 - kódy kláves

<b>24</b> <b>1</b> 36	<b>1C</b> <b>2</b> 28	<b>14</b> <b>3</b> 20	<b>0C</b> <b>4</b> 12	<b>04</b> <b>5</b> 4	<b>03</b> <b>6</b> 3	<b>0B</b> <b>7</b> 11	<b>13</b> <b>8</b> 19	<b>1B</b> <b>9</b> 27	<b>23</b> <b>0</b> 35
<b>25</b> <b>Q</b> 37	<b>1D</b> <b>W</b> 29	<b>15</b> <b>E</b> 21	<b>0D</b> <b>R</b> 13	<b>05</b> <b>T</b> 5	<b>02</b> <b>Y</b> 2	<b>0A</b> <b>U</b> 10	<b>12</b> <b>I</b> 18	<b>1A</b> <b>O</b> 26	<b>22</b> <b>P</b> 34
<b>26</b> <b>A</b> 38	<b>1E</b> <b>S</b> 30	<b>16</b> <b>D</b> 22	<b>0E</b> <b>F</b> 14	<b>06</b> <b>G</b> 6	<b>01</b> <b>H</b> 1	<b>09</b> <b>J</b> 9	<b>11</b> <b>K</b> 17	<b>19</b> <b>L</b> 25	<b>21</b> <b>ENTER</b> 33
<b>27</b> <b>CAPS</b> <b>SHIFT</b> 39	<b>1F</b> <b>Z</b> 31	<b>17</b> <b>X</b> 23	<b>0F</b> <b>C</b> 15	<b>07</b> <b>V</b> 7	<b>00</b> <b>B</b> 0	<b>08</b> <b>N</b> 8	<b>10</b> <b>M</b> 16	<b>18</b> <b>SYM.</b> <b>SHIFT</b> 24	<b>20</b> <b>SPACE</b> 32

Tabulka 3 - výstupní tvar 'KEY-SCAN'

	reg. D	E	Příznak
žádná klávesa	FF	FF	Z
jediná klávesa	FF	kód	Z
jediná klávesa + SYMBOL SHIFT	18	kód	z
jediná klávesa + CAPS SHIFT	27	kód	z
pouze oba SHIFTY	27	18	z
právě dvě různé klávesy (D<E)	kód1	kód2	NZ
tři a více kláves	nemá	smysl	NZ

pozn.: Je-li příznak NZ, informace v reg. DE nemá smysl a nemusí se tudíž vyhodnocovat.

-----  
031E K-TEST

CALL dekodování kódu klávesy

Vstup: DE = stav klávesnice

Výstup: A = ASCII hodnota stisknutého tlačítka

B = kód stisknutého SHIFTu

Příznak C - v pořádku

NC - žádná klávesa nebo jeden SHIFT

Mění : C, D, H, L, příznaky (D=00|H=02)

Přiřazení hodnot do reg. A :

30 až 39 číslicím

41 až 5A písmenům

0D klávese ENTER

0E jsou-li stisklé oba SHIFTy

pozn.: Je výhodné používat posloupnost

CALL 028E,KEY-SCAN;kód

RET NZ ;návrat - nemá smysl

CALL 031E,K-TEST ;ASCII

RET NC ;návrat - žádné tlačítko

... ;pokračování  
-----

## 1F54 BREAK-KEY

CALL stav kláves 'BREAK'

Vstup: -

Výstup: Příznak NC - stav 'BREAK'

C - není stisknutý 'BREAK'

Mění : A, příznaky

- - -

Tato rutina rychle zjistí stav kláves 'BREAK', tj. klávesy CAPS  
SHIFT a klávesy SPACE.  
-----

XX

Použití rutin z podprogramu 'KEYBOARD DECODING' nedoporučujeme  
protože klávesnice je ovládána systémem v reálném čase a pro dvě  
shodné počáteční hodnoty vstupující např. do rutiny 'K-DECODE'  
nezaručuje stejnou výstupní hodnotu.

Např.: POKE 23658,8 vyvolá C-mód

POKE 23658,0 vyvolá L-mód

XX  
-----

## 2C88 ALPHANUM

CALL test na číslo a písmeno

Vstup: A - ASCII

Výstup: Příznak C - pro kód čísla (0 až 9)

nebo znaku (A až Z, a až z)

NC - kód jiného znaku

Mění : Příznaky

- - -

Rutina využívá podprogram '2D1B, NUMERIC'.  
-----

-----  
208D ALPHA

CALL test na písmeno

Vstup : A - ASCII

Výstup: Příznak C - pro kód písmene (A až Z, a až z)

NC - není písmeno

Mění : Příznaky  
-----

## 2D1B NUMERIC

CALL test na číslo

Vstup: A = ASCII

Výstup: Příznak NC - pro kód čísla (0 až 9)

C - není číslo

Mění : Příznaky  
=====Příklady:  
-----

1) Chceme, aby program čekal na stisk libovolné klávesy

```

Loop   IN A,(16)           ;všechny klávesy
        OR   E0            ;zamaskuj 'EAR'
        INC  A             ;nula pro žádnou klávesu
        JR   Z,Loop        ;čekej

```

2) Chceme, aby program čekal na stisk klávesy ENTER

```

Loop   CALL 028E,KEY-SCAN;klávesnice
        JR   NZ,Loop       ;nesmysl - zpět
        LD   A,21          ;kód ENTER
        CP   E             ;je kód ENTER?
        JR   NZ,Loop       ;není-li ENTER, zpět

```

3) Chceme, aby čekal na zadání libovolného písmene

```

Loop   CALL 028E,KEY-SCAN;klávesnice
        JR   NZ,Loop       ;nesmysl - zpět
        CALL 031E,K-TEST   ;ASCII => A
        JR   NC,Loop       ;žádná klávesa - zpět
        CALL 208D,ALPHA    ;kód písmena ?
        JR   NC,Loop       ;není-li písmeno - zpět
        dalši instrukce   ;v A je ASCII písmena

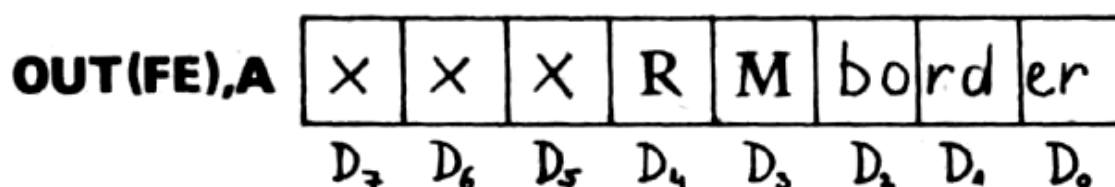
```

## 2 . R E P R O D U K T O R

=====

Reproduktor ve Vašem počítači je připojen na výstup portu FE [254] a přes něj je také ovládán. Chcete-li vyslat na port FE například z registru A informaci ("stavové slovo") můžete to provést třeba instrukcí OUT (FE),A. Každá takové stavové slovo musí mít následující formát.

Obrázek 2 - stavové slovo "BEEP"



X = zcela libovolný stav

D<sub>4</sub> = ovládá reproduktor (R)

\ 1 - vypnuto

D<sub>3</sub> - výstup pro Magnetofon (MIC) ( M )

/ 0 - aktivní stav

BORDER viz kapitola 4

Chceme-li vytvořit nějaký tón, je potřeba rozkmitat membránu reproduktoru (stejně jako u zvuku). To se provede střídáním Aktivního a pasivního stavu (tj. invertováním bitu D<sub>4</sub> stavového slova a jeho vyslání- na port 'FE'). Má-li mít tón určitou frekvenci, je třeba, aby bit D<sub>4</sub> na portu 'FE' byl invertován vždy po určitém, ale stejném časovém úseku.

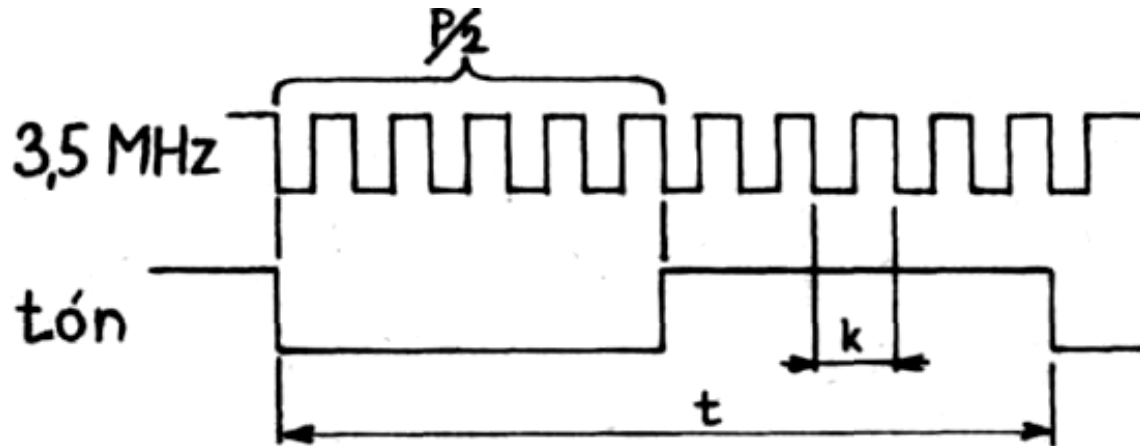
Nejvhodnějším měřídlem pro odstup jednotlivých změn stavového slova pro reproduktor jsou samotné "kroky" mikroprocesoru, který pracuje na kmitočtu 3.5 MHz což znamená, že za 1 sekundu projde 3500000 kroků (T-states). Jeden krok mikroprocesoru pak trvá

$$k = 1/3500000 \text{ sekundy.}$$

$$\{ k = 0.285714 \mu s \}$$

Zadaný tón má frekvenci  $f$ , jedna jeho perioda pak trvá  $t=1/f$  a počet kroků, ve kterých se musí uskutečnit 1 perioda tónu je

$$P = t/k = 3500000/f.$$

Obrázek 3 - frekvence tónu řízená kroky  $\mu P$ .

Jak je z periody tónu vidět, je třeba invertovat vždy dvakrát za jednu periodu  $t$ . Počet kroků je tedy poloviční  $P$ , tj.  $1750000/f$ .

#### 03B5 BEEPER

CALL tvorba tónu

Vstup:  $DE = f \cdot t$  (frekvence  $t$  čas) tj. doba

$HL = [(437\ 500/f) - 30]$

(počet kroků  $P/2$  dělený 4 zmenšený o 30)

pozn.: vydělíme-li počet kroků čtyřmi, dostaneme přibližně počet instrukcí; 30 se odečítá pravděpodobně pro zdržení, vzniklá spoluprací s obvodem ULA.

(s.p. BORDER) - barva bordru\*8

Výstup: Tón

Mění: A, BC, DE, HL, IX, příznaky

- - -

pozn.: U průběhu provádění podprogramu je zakázáno přerušení (DI), která bude povoleno na konci podprogramu (EI) !!

Příklady:

-----

1) vytvoření tónu C o délce 1 sekundy.

tón C - frekvence 261.63 Hz, počet kroků je

$[1750000/261.63 - 6689]$

HL -  $(16689/4) - 30 = 1642] = 066A$

DE -  $f \cdot t = (261.63 * 1] = 0105$

```
LD  DE,0105      ; délka tónu
LD  HL,066A      ; výška tónu
CALL 03B5        ; tón
RET              ;
```

## 2) vytvoření zvyšujícího se tónu

## Program 3 - laser

	LD	HL,0300	;výška tónu	210003	[33,0,3 ]
Loop	LD	DE,0001	(délka - 1	110100	[17,1,0 ]
	PUSH	HL	;frekvence	E5	(229 3
	CALL	03B5,BEEPER	; tón	CDB503	[205,181,3]
	DI			F3	(253 3
	POP	HL		E1	[225 ]
	LD	DE,0002		110200	(17,2,0 3
	AND	A		A7	(167 3
	BBC	HL,DE	;frekv.-2	19	(25 3
	JR	NZ,Loop	;celý tón	20F0	(32,240 3
	EI			FB	(251 3
	RET			C9	(201 3

## 3) ozvučení kláves

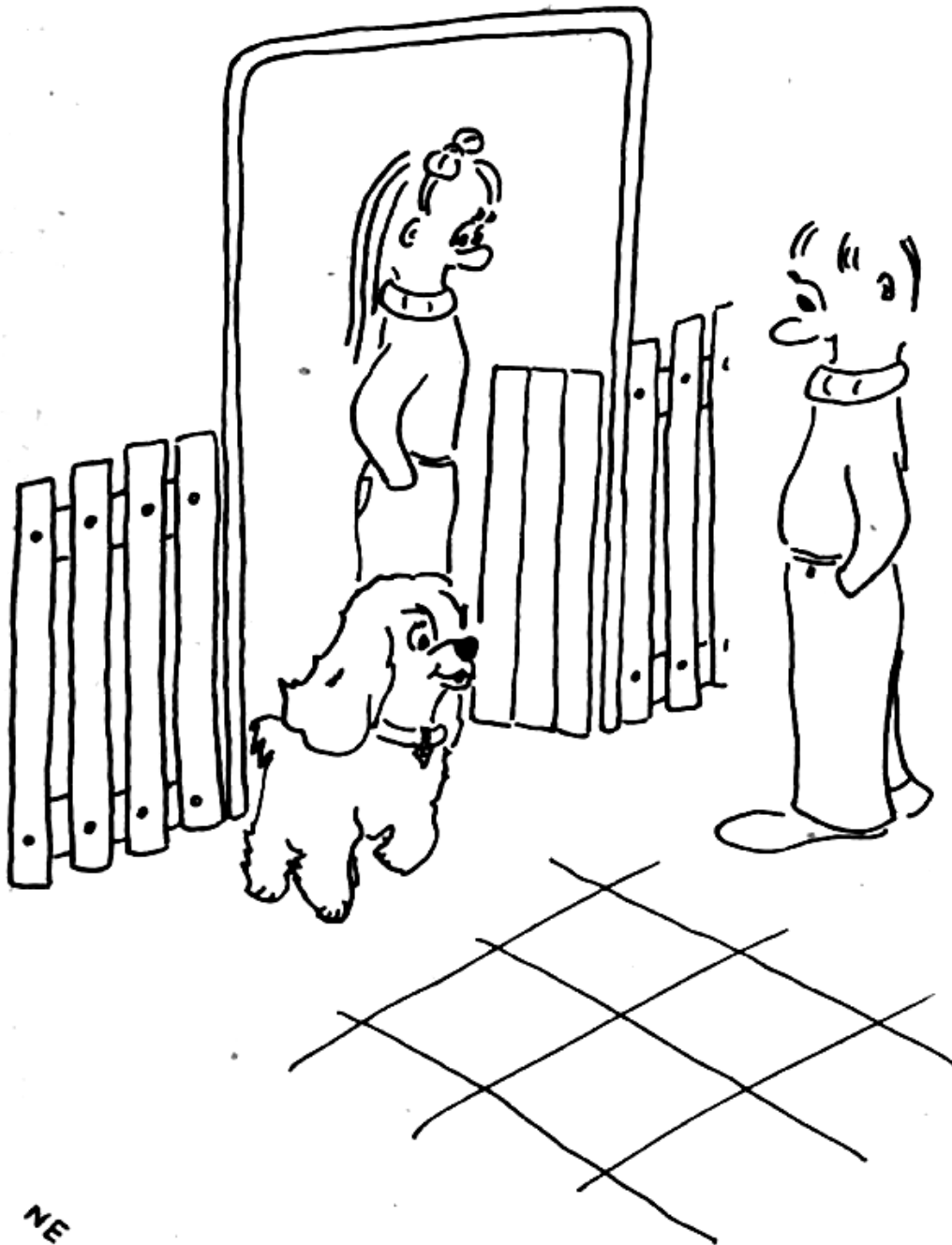
## Program 4 - key-bů

Key-bu	DI			F3	[243 ]
	LD	HL,0A00	;tón	21000A	[33,0,10 ]
	LD	DE,0008	; chybné	110800	[17,8,0 ]
	CALL	03B5	; kombinace	CDB503	[205,181,3]
	DI			F3	[243 ]
Key	CALL	028E	;čtení klávesnice	CDBE02	[205,142,2]
	JR	NZ,Key-bu	;chybná kombinace	20F0	[32,240 ]
	LD	A,D		7A	[122 ]
	CP	E		BB	[ 187 ]
	JR	Z,Key	;žádná klávesa	28F7	[40,247 ]
	CALL	1F34	;test BREAK	CD541F	[205,84,31]
	JR	C, Tón	;není stav 'BREAK'	3802	[56,2 ]
	EI			FB	[251 ]
	RET		;návrat BASIC	C9	[201 ]
Tón	LD	BC,0010		011000	[1,16,0 ]
	LD	L,E	;výška	6B	[107 ]
	LD	H,B	; tónu	60	[96 ]
	ADD	HL, BC	; podle	09	[9 ]
	ADD	HL,HL	; kódu	29	[41 ]
	ADD	HL, HL	; stisklé	29	[41 ]
	ADD	HL,HL	; klávesy	29	[41 ]
	LD	DE,0008	;délka tónu	110800	[17,8,0 ]
	CALL	03B5	;tón	CDB503	[205,181,3]
	DI			F3	[243 ]
	JR	Key	;opakuji	18DE	[24,222 ]

Pozn.: U programech 'laser' a 'key-bů' jsou použity instrukce DI (disable interrupt), aby se zamezilo čtení klávesnice v reálném čase a tudíž i zkreslení reprodukováných tónů.



PROSÍM VÁS, BYDLÍ TADY PAN BAJT?



NE, TO JSTE SI SPLETL ADRESU, ZDE ŽÁDNÝ BAJT NEBYDLÍ.

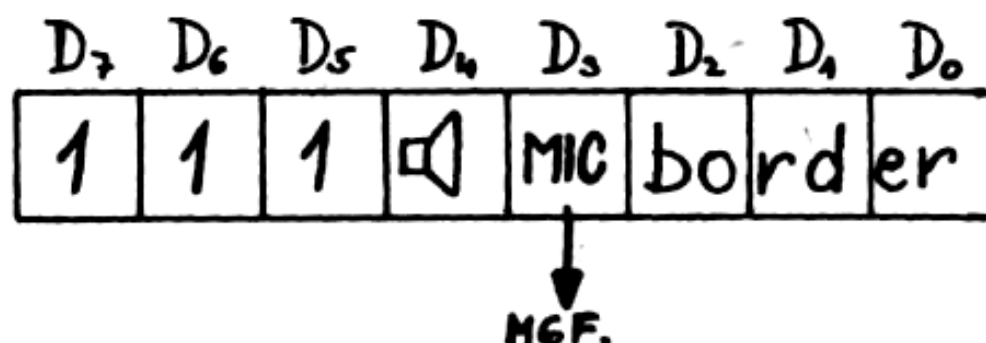
## 3 . M A G N E T O F O N

=====

Počítač spolupracuje s magnetofonem přes port 'FE'. Výstup z počítače (SAVE) na magnetofon přes jack MIC je ovládán pomocí bitu D3 datové sběrnice.

Obrázek 4 - stavové slovo "SAVE"

OUT (FE),A



Podle logické hodnoty bitu D3 je nastaveno napětí na jacku MIC pro vstup do magnetofonu. Podle hardware manuálu je napětí na pinu 20 obvodu ULA:

Tabulka 4 - ULA pin 20

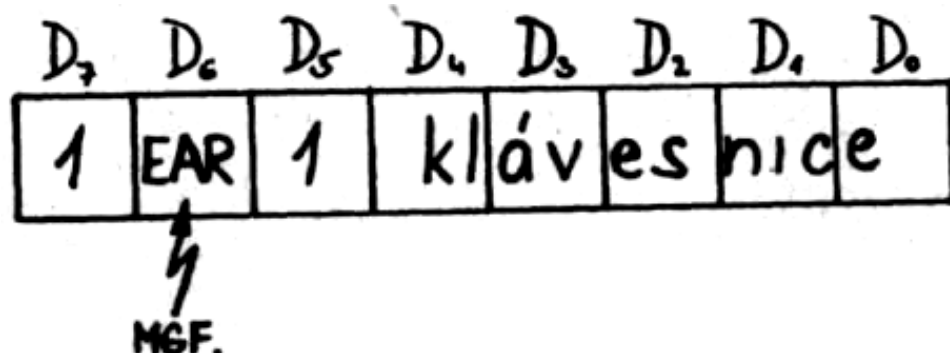
D4	D3	výstup
0	0	0.4 V
0	1	0.7 V
1	0	3.5 V
1	1	3.8 V

Logická jednička '1' je označována jako klidový stav a měla by být vždy posledním stavem vyslaným na magnetofon. Bity D2, D1, D0 viz kapitola 4 - obrazovka.

Ustup do počítače (LOAD) z magnetofonu je čten přes zdířku EAR. Pomocí instrukce IN z bitu D6 datové sběrnice.

Obrázek 5 - stavové slovo "LOAD"

IN A, (FE)



Signál, který byl zaznamenán na magnetofon, se nyní snímá a podle hodnoty na vstupu (EAR) je nastaven bit D4 stavového slova. Zde je nutno upozornit na rozdíly mezi jednotlivými modely počítače ZX Spectrum, způsobené použitými součástkami. Jednotlivé modely lze rozlišit podle tvaru chladiče a nejlépe přímo na desce plošných spojů (strana součástek), kde je nápis ISSUE 5 označení- typu. Například 'ISSUE 3'. Jinak je možné modely 2 a 3 rozpoznat například podle následující tabulky (pozor na možnost poškození vodičů klávesnice !).

Tabulka 5 - rozlišení ISSUE 2 a 3 (vše dekadicky)

napětí	B I T D6		I N 57342	
	ISSUE 2	ISSUE 3	ISSUE 2	ISSUE 3
0.4 V	0	0	191	191
0.7 V	1	0	255	191
3.5 V	1	1	255	299
3.8 V	1	1	255	299

Pozn.: Stejně hodnoty dostaneme i instrukcí IN A, (FE).

Pro převod signálu magnetofonu na logické stavy je dána rozhodovací úroveň podle použitých součástek. Napětí na vstupu nižší než rozhodovací úroveň jsou vyhodnocena jako logická nula. Napětí vyšší než rozhodovací úroveň jsou vyhodnocena jako logická jednička '1'.

Monitor používá pro spolupráci s magnetofonem dvě základní rutiny 'SA-BYTES' pro SAVE ; 'LD-BYTES' pro LOAD a VERIFY. Je nutné upozornit, že obě tyto rutiny se vrací do hlavního programu přes pomocný podprogram 'SA/LD-RET' !!!

Pozn.: Příkaz MERGE zde není zahrnut, protože vyžaduje existenci celého BASIC - systému.

-----  
053F SA/LD-RET

ukončení SAVE a LOAD rutin

Vstup : (BORDCR)

klávesa 'SPACE'

Výstup: při 'SPACE' - chyb. hlášení 'D BREAK - CONT repeats'

BORDER nastaven podle s.p.(BORDCR)

Mění : -

- - -

Podprogram proběhne vždy při návratu z rutin 'SA-BYTES' a 'LD-BYTES'. Podprogram postupně vykoná:

- ze s.p. (BORDCR) je obnoven původní BORDER.
- povolí přerušování (interrupt enable) - EI !!!
- je-li stisknuta klávesa 'SPACE' (tj. 'BREAK') provede se skok na chybové hlášení 'D BREAK - CONT repeats'.
- všechny výstupní informace z rutin 'LD-BYTES' nebo 'SA-BYTES' jsou zachovány pro další zpracování (reg. AF)
- návrat do hlavního programu - RET .

Zde uvádíme výpis této rutiny:

```
053F SA/LD-RET  PUSH AF                ;
                LD  A,(BORDCR)         ;
                AND 38                 ;
                RRCA                    ;
                RRCA                    ;
                RRCA                    ;
                OUT (FE),A              ; bod a)
                LD  A,7F                ;
                IN  A,(FE)              ; test 'BREAK'
                RRA                     ;
                EI                      ; bod b)
                JR  C,0554,BA/LD-END    ;
0552 REPORT-D  RST 0008,ERROR-1        ; bod c)
                DEFB 0C                 ;
0554 BA/LD-END  POP  AF                 ; bod d)
                RET                     ; bod e)
```

---

**04C2 SA-BYTES**

CALL uložení dat

Vstup: A = značkový bajt - 00 hlavička  
 \ FF blok dat

IX - počáteční adresa

DE = délka ukládaných dat

Výstup: vyslání dat na jack MIC a změny BORDRU  
 návrat přes rutinu 'SA/LD-RET'!!!  
 příznak C - v pořádku

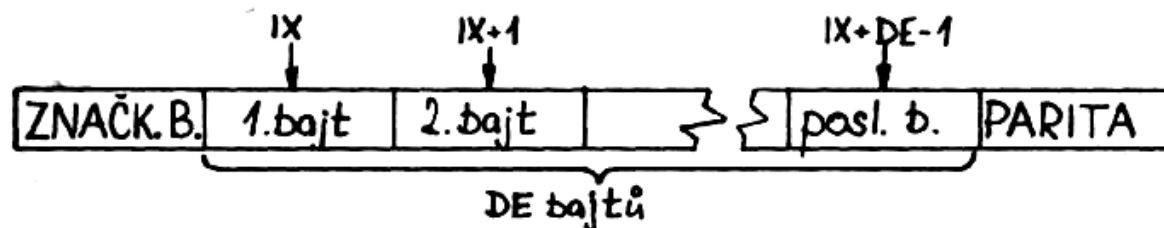
NC - BREAK, 'SA/LD-RET' volá chyb. hlášení

Mění : A, BC, DE, HL, IX, příznaky

- - -

Na magnetofon se uloží nejprve registr A jako značkový bajt, potom část paměti počínaje adresou IX o délce DE bajtů a nakonec bajt parity. Parita je kontrolní exklusivní součet (XOR) všech nahraných bajtů.

Obrázek 6 - záznam bloku dat



Pozn.: Velice často potřebujeme použít rutinu 'SA-BYTES', ale vytvořit si vlastní ošetření chyb (jako 'SA/LD-RET') bez volání chybových hlášení. Potom máme dvě možnosti:

1. vytvořit vlastní rutinu pro ošetření návratu; její adresu uložit do HL a pak volat rutinu 'SA-BYTES' s přeskočením instrukce na adrese '04C2 LD HL,SA/LD-RET', tj. CALL 04C5. Do této naší nové ošetřující rutiny se podprogram 'SA-BYTES' vrací s příznakem C - v pořádku nebo NC - chyba 'BREAK'. Nesmíme zapomenout povolit přerušení a RET !!!
2. vůbec nepoužít žádnou návratovou rutinu - pak přeskočíme instrukce '04C2 LD HL,SA/LD-RET' a '04C5 PUSH HL' a podprogram voláme CALL 04C6. Podprogram se pak rovnou vrací do hlavního programu, kde příznak C značí v pořádku nebo NC - nastala chyba 'BREAK'. Opět nesmíme zapomenout povolit přerušení !!!

Standardně se jako značkový bajt používá pouze 00 a FF, ale je možné použít i jakékoliv jiné libovolné číslo. Pro kódy '00' až '7F' se generuje zaváděcí tón délky 5 sekund; pro '80' až 'FF' tón délky 2 sekundy. Při nahrávání je podle tohoto bajtu provedeno řízení výběru bloku dat - viz rutina 'LD-BYTES'.

-----  
0556 LD-BYTES

CALL nahrání dat

Ustup: Příznak C - pro 'LOAD'

NC - pro 'VERIFY'

A - značkový bajt - 00 hlavička

\ FF blok dat

(výběr dat začínajících tímto bajtem) I

IX - počáteční adresa pro ukládání

DE - délka nahrávaných dat

Výstup: blok dat v paměti

vrací se přes 'SA/LD-RET' ('BREAK'-chyb. hlášení D)

příznak C - v pořádku

NC - ERROR, chyba (viz poznámka)

Mění : A, BC, DE, HL, IX, příznaky

- - -

Rutina porovná první přijatý bajt s hodnotou v reg. 'A'. Nesouhlasí-li, je nastaven příznak chyby a následuje návrat do výstupní nebo hlavní rutiny. Další bajty se ukládají jako data (LOAD), resp. jsou kontrolovány s obsahem paměti (VERIFY). Poslední přijatý bajt je parita nahrávky (neukládá se do paměti) a je porovnána s kontrolní vypočtenou paritou přijatých bajtů. Nesouhlasí-li, pak je nastaven příznak chyby a proveden návrat do výstupní nebo hlavní rutiny.

Chyba (NC) může nastat:

a) TYPE: nesouhlasí značkový bajt.

Ustupní hodnota v reg. A nesouhlasí s prvním nahraným bajtem.

b) VERIFY : nesouhlasí záznam.

Kontrolovaná nahrávka nesouhlasí s obsahem paměti.

c) TIME-UP: nenašel se impuls.

Porušen magnetofonový záznam (DROP OUT nebo rušení)

d) BREAK : přerušeni rutiny.

Byla stisknuta klávesa 'SPACE'.

e) PARITY-OVER: nesouhlasí parita.

Parita nahraných dat nesouhlasí se zaznamenanou paritou nebo některý bajt byl chybně načten.

Pozn.: Bohužel, zde uvedené druhy chyb při nahrávání nám nejsou programově dostupné. (Bohudík.)

Velice často potřebujeme použít rutinu 'LD-BYTES', ale vytvořit si vlastní rutinu ošetření chyb - např. podle vzniklá chyby rozhodnout o dalším průběhu programu nebo vyloučit volání chybových hlášení (tj. nepoužít 'SA/LD-RET').

I zde je potřeba přeskočit instrukce zajišťující návrat přes 'BA/LD-RET'. Protože tyto instrukce nejsou na počátku podprogramu 'LD-BYTES', vynechají se i některé další instrukce, které musíme před samotným vstupem do podprogramu nahradit.

Nahradit je potřeba instrukce :

```

0556 LD-BYTES      INC  D
                   EX   AF,A'F'
                   DEC  D
                   DI
                   LD   A,0F
                   OUT  (FE),A
-----
                   LD   HL,SA/LD-RET
0561                PUSH HL

```

Při programování vlastní chybové rutiny musíme postupovat podle následujících bodů:

a) příznak NZ' (pro normální průběh), jako U ROM

```

INC  D           ;reset Z
EX   AF,A'F'     ;reset Z' <A'= značkový bajt !!!>
DEC  D           ;původní hodnota

```

nebo příznak Z' - což zajistí neporovnání značkového bajtu s reg. 'A', ale jeho uložení jako první bajt dat do paměti.

```

CP   A           ;set Z
EX   AF,A'F'     ;set Z' <A'=značkový bajt !!!>

```

```

!!!!!!!!!!!!!!!!!!!!!!!! P O Z O R !!!!!!!!!!!!!!!!!!!!!!!
!!                                                                !!
!!   U takovém případě musí být délka dat v 'DE' o          !!
!!   jednu větší než délka udaná v rutině 'SA-BYTES'        !!
!!   a skutečný datový soubor je o jeden bajt delší!        !!
!!                                                                !!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

```

b) zakázat interrupci - DI !!!

c) nastavit bílý BORDER - není povinné

```

LD   A,0F
OUT  (FE),A

```

Po provedení bodů a), b) můžete zavolat rutinu 'LD-BYTES' instrukcí CALL 0561, existuje-li námi sestavená výstupní rutina a její adresa je uschována v HL, nebo instrukcí CALL 0562 jestliže výstupní rutina neexistuje. V obou případech pak nesmíme zapomenout na povolení přerušení.

-----  
0802 LD-BLOCK

CALL nahrání dat

Vstup : Příznak C - pro 'LOAD'

NC - pro 'VERIFY'

A - značkový bajt - 00

hlavička \ FF blok dat

(výběr dat začínajících tímto bajtem)

IX = počáteční adresa pro ukládání

DE = délka nahrávaných dat

Výstup: blok dat v paměti

vrací se přes 'SA/LD-RET' ('BREAK'-chyb. hlášení D)

příznak C - v pořádku

(NC- chyb. hlášení 'R Tape loading error')

Mění: A, BC, DE, HL, IX, příznaky

- - -

Rutina 'LD-BLOCK' využívá 'LD-BYTES', má shodné vstupní i výstupní parametry. Jakákoliv chyba spáchá chybové hlášení 'R Tape loading error'.

```

0802 LD-BLOCK    CALL 0556,LD-BYTES    ;load data
                RET  C                ;v pořádku -> návrat
0806 REPORT-R    RST 0008,ERROR-1     ;chyb. hlášení
                DEFB 1A              ;

```

Rutina 'LD-BYTES' používá pro nahrávání jednotlivých bitů rutin 'LD-EDGE', které měří délku impulsu z magnetofonu.

-----  
05E7 LD-EDGE-1

CALL měření doby impulsu

Vstup : B = časová konstanta

C = BORDER a typ posledního impulsu (viz obr.7)

Výstup: B = délka impulsu

C = BORDER a typ posledního impulsu (viz obr.7)

Příznak C - v B je požadovaná hodnota

NC - error, chyba:

Příznak Z - TIME-UP, impuls dlouhý

NZ - BREAK

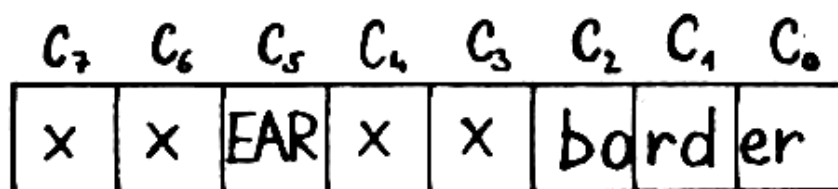
Mění : A, příznaky

- - -

Rutina 'LD-EDGE-1' měří dobu od vstupu do rutiny po přechod z jednoho stavu do druhého (obrázek 8).



Obrázek 7 - registr C



x - libovolný stav

BORDER - ( Viz 4. kapitola )

EAR - poslední stav přijatý z magnetofonu

Podprogram pracuje takto:

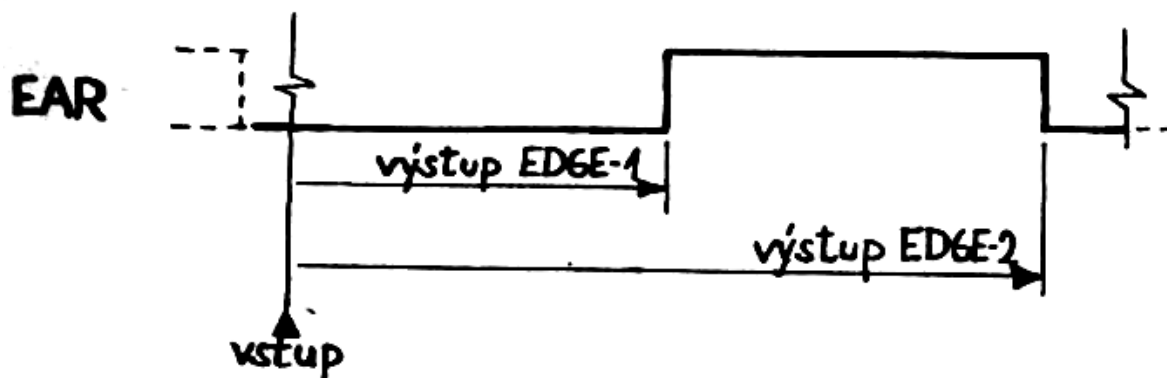
- obsah registru C podle obrázku 7
- v registru B je časová konstanta
- program přečte stav na zdířce 'EAR' a porovná jej s bitem  $C_5$  (poslední stav 'EAR'). Je-li stav stejný a reg. B není nulový, je B zvětšen o jednu a pokračuje bodem c). Je-li reg. B roven nule, jde o příliš dlouhý impuls a RET.
- při každém průchodu smyčkou kontroluje 'BREAK'
- Je-li impuls nelezen, nastaví se aktuální hodnota  $C_3$ , negována barva BORDERU 'bbb' a nový BORDER je vyslán na bránu FE (pruhy při nahrávání).

K reg. B je připočtena doba, po kterou se stav signálu na zdířce 'EAR' nezměnil.

Pozn.: Obrázek 7 není chybný. Rutiny 'LD-EDGE' mají v registru C (bity 0,1,2) uschování barvu BORDERU a v bitu  $C_5$  poslední stav 'EAR'. (V bitu  $C_3$  proto, že načtené stavové slovo z brány FE je rotováno a testována klávesa 'SPACE'. Tím se stav 'EAR' přesune do bitu  $C_5$ ).

Rutina '0SE3, LD-EDGE-2' změří dobu dvou po sobě jdoucích přechodů.

Obrázek 8 - rutiny LD-EDGE



Příklady :

1) Čtení stavu zdičky 'EAR' (i pro BASIC)

Program 5 - čtení vstupu 'EAR'

Ear	LD	BC,0000	;	010000	[1,0,0	]
	IN	A,(FE)	; čtení 'EAR'	DBFE	[219,234	]
	AND	40	; bit 'EAR'	E640	[230,64	]
	RET	Z	: 'EAR'=0, BC=0	C8	[200	]
	INC	C	: BC=1	0C	[12	]
	RET		; 'EAR'=1, BC=1	C9	[201	]

Pozn.: Uložíme-li uvedený program na adr. nn můžeme jednoduchým BASIC programem vypisovat stav zdičky 'EAR'. Nezapomeňte CLEAR nn-1 !

10 PRINT USR nn ;: POKE 23692,2: GO TO 10

2) standardně uložit bezhlavičkový soubor dat (screen)

Program 6 - uložení obrazovky

SA-SCR	LD	A,FF	; blok dat	3EFF	[62,255	]
	LD	IX,4000	; počát. adr.	DD210040	[221,33,0,64]	
	LD	DE,1B00	; délka dat	11001B	[17,0,27	]
	CALL	04C2	; 'SA-BYTES'	CDC204	[205,194,4	]
	RET		;	C9	[201	]

Pozn.: Rutina 'SA-BYTES' neuloží blok dat s hlavičkou jako příkaz SAVE v BASICU, ale uloží požadovaný počet bajtů jako blok nebo jako hlavičku - vytvoří tzv. bezhlavičkový (Header-less) soubor. Může tedy vytvořit hlavičku několik tisíc bajtů dlouhou.

3) nahrání souboru z předchozího příkladu Program 7 - nahrání obrazovky

LD-SCR	LD	A,FF	; blok dat	3EFF	[62,255	]
	LD	IX,4000	; adr. obr.	DD210040	[221,33,0,64]	
	LD	DE,1B00	; délka dat	11001B	[17,0,27	]
	SCF		; LOAD	37	[53	]
	CALL	0802	; 'LD-BLOCK'	CD0208	[205,2,8	]
	RET		;	C9	[201	]

4) nahrání libovolné standardní hlavičky

Program 8 - hlavičkář

FF00 Head	XOR	A	; hlavička	AF	[175	]
FF01	LD	IX,FF14	; [65300]	DD2114FF	[221,33,20,255]	
FF05	LD	DE,0011	; 17 bajtů	111100	[17,17,0	]
FF07	SCF		; LOAD	37	[55	]
FF08	CALL	0556	:	CD5605	[205,86,5	]
FF0B	JR	NC,Head	; chyba - zpět	30F2	[48,242	]
FF0C	RET		; OK.	C9	[201	]

Pozn.: Použijete-li program 8 přímo z BASICu, bude od adresy FF14 [65300] uloženo 17 bajtů hlavičky, které nesou informaci a následujícím bloku dat. Můžeme je snadno dekodovat podle tabulky 6. Program je přemístitelný na libovolnou adresu kromě FF14 až FF25 [65300 až 65317], kde je místo pro hlavičku. U našem příkladu je program uložen na adr. FF00 [65280]; CLEAR 65279 !!!

Tabulka 6 - Standardní hlavička

	<b>Program</b>	<b>Number array</b>	<b>Char. array</b>	<b>Bytes</b>	<b>IX + n</b>
1.	00	00	02	03	00
2.  až  11.	NÁZEV				01          0A
12.  13.	celková délka	DÉLKA			0B    0C
14.  15.	start LINE	x	x	počáteční  adresa	0D  0E
16.  17	délka BASICu	x	x	x	0F  10

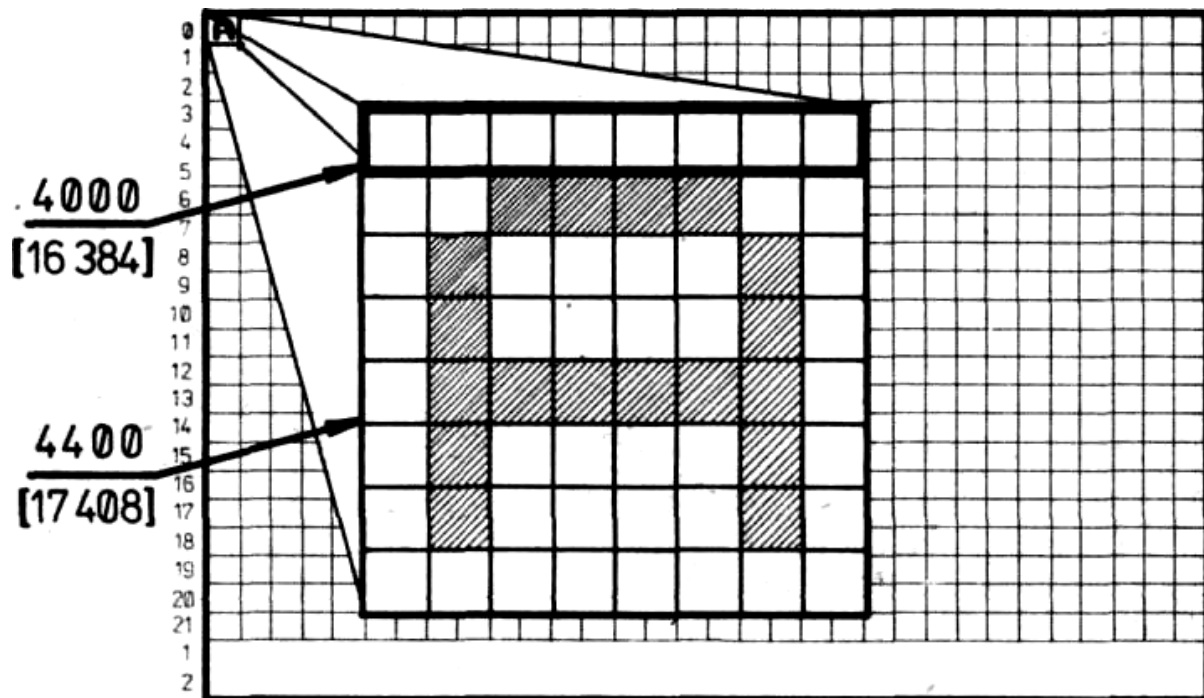
Pozn.: - x -> není vyhodnocováno  
 - není-li program nahrán s parametrem LINE obsahují bajty  
 14 - 00 [0] a bajt 15 - 80 [128].

#### 4 . O B R A Z O V K A =====

Obrazovka počítače ZX Spectrum je obsažena v 1600 bajtech [6912] na adresách 4000 až 5AFF [16384 až 23295]. Prvních 1800 [6144] bajtů 'VIDEO-BYTE' vytváří jakousi síť o stranách 256x192 bodů. Každé oko této sítě - bod na obrazovce, je v paměti počítače uchován jako jeden bit. Osm bitů ležících vodorovně vedle sebe je zaznamenáno do jednoho bajtu. Jednu bodovou řádku tvoří 20 [32] bajtů (tj. 256 bodů).

Osm bajtů pod sebou tvoří pixel ( 8x8 bodů) a do každého pixelu se zobrazí právě jedno písmeno. Pod sebe se na obrazovku vejde 18 [24] pixelů (tj. 192 bodů).

Obrázek 9 - obrazovka



Adresování obrazovky je následující - první bajt prvního pixelu na prvním řádku má adresu 4000 [16384] (levý horní roh), další adresa patří prvnímu bajtu druhého pixelu na první řádce a tak jsou postupně adresovány první bajty pixelů po řádkách zleva doprava až dojdeme k adrese prvního bajtu třicátého druhého pixelu osmé řádky, kde končí první třetina obrazovky. Další adresy přísluší druhým bajtům pixelů v první třetině (256 bajtů), třetím ... atd. až je dosažena adresa posledního bajtu první třetiny (8. bajt, 32. pixelu, 8. řádky). Následují adresy druhá a třetí třetiny, které jsou uspořádány podobně.

Tento složitý výklad nahradí názorná ukázka

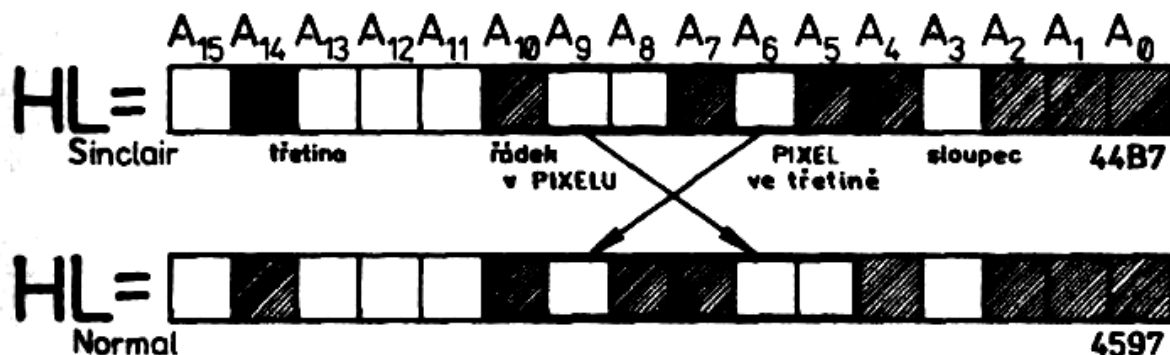
```

10 FOR F=16384 TO 22527
20 POKE F,254
30 PAUSE 10
40 NEXT F
RUN

```

Tento podivný způsob adresace se liší od adresace normální (tj. zleva doprava a shora dolů v řádkách) vzájemným prohozením adresových bitů A10,A9,A8 s bity A7,A6,A5.

Obrázek 10 - adresování Sinclair a 'normální'



Tuto výměnu 'tam' i 'zpět' vykoná krátký podprogram, jehož využití je uvedeno v příkladu. Ústup, výstup HL; mění A, E, přízn.

Program 9- change adres

CHANGE LDA,H	;	7C	[124	]
RRCA	;	0F	[15	]
RRCA	;	0F	[15	]
RRCA	;	0F	[15	]
LD H, A	;	67	[103	]
XOR L	;	AD	[173	]
AND E0	;	E6E0	[230,224	]
LD E,A	;	5F	[95	]
XOR L	;	AD	[173	]
LD L,A	;	6F	[111	]
LD A,E	;	7B	[123	]
XOR H	;	AC	[172	]
RLCA	;	07	[7	]
RLCA	;	07	[7	]
RLCA	;	07	[7	]
LD H,A	;	67	[103	]
RET	;	C9	[201	]

Příklad použití programu 9 v BASICU:

Tento program názorně ukáže 'normální' adresování obrazovky.

```

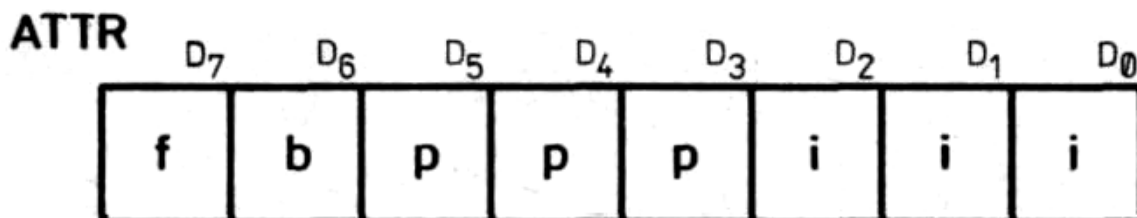
10 CLEAR 65279
20 GO SUB 1000
30 FOR F=16384 TO 22527
40 LET H=INT (F/256)
50 LET L=F-256*H
60 POKE 65281,L
70 POKE 65282,H
80 LET A=USR 65280
90 POKE A,254 100 PAUSE 10
110 NEXT F
120 BEEP .5,24
130 PAUSE 0: STOP
1000 FOR F=65280 TO 65302
1010 READ A: POKE F,A
1020 NEXT F: RETURN
1030 DATA 33,0,0,124,15,15,15,103
1040 DATA 173,230,224,95,173,111
1050 DATA 123,172,7,7,7,103,68,77
1060 DATA 201

```

V tomto programu je k rutině "CHANGE" přidána instrukce 'LD HL,nn', zajišťující přidání parametrů z BASICU. Na konci je rutina doplněna instrukcí 'LD B,H' a 'LD C,E' pro předání výsledku do BASICU.

Barevné podání obsahu VIDEO-BYTE je určováno obsahy ATRIBUTŮ na adresách 5800-5AFF [22520-23295] pro každý pixel 'normálním' způsobem tj. zleva - doprava a shora - dolů. Atribut nese informaci o barvě tisku (INK), podkladu (PAPER), světlosti (BRIGHT) a blikání (FLASH) ve formátu podle obrázku 11.

Obrázek 11 - atributy



- INK (i) - barva všech bodů, které jsou v příslušném pixelu vyjádřeny nastaveným bitem (jedničkou) ve VIDEO-BYTE.
- PAPER (p) - barva všech bodů, které jsou v příslušném pixelu vyjádřeny nulovým bitem ve VIDEO-BYTE.
- BRIGHT (b) - určuje světlost všech bodů příslušného pixelu tj. normální (b=0) nebo dvakrát světlejší (b=1).
- FLASH (f) - při nastaveném sedmém bitu jsou periodicky zaměňovány informace PAPER a INK, což způsobí blikání pixelu. Při f=0 je vše podle ATRIBUTŮ pixelu.

Tabulka 7 - barvy

D2(5) D1 (4) D0(3)

0	0	0	black	(černá)
0	0	1	blue	(modrá)
0	1	0	red	(červená)
0	1	1	magenta	(fialová)
1	0	0	green	(zelená)
1	0	1	cyan	(zelenomodrá)
1	1	0	yellow	(žlutá)
1	1	1	white	(bílá)

Tabulka 8 - adresy obrazovky

	Adresy 'VIDEO-BYTE'		Adresy 'ATTRIBUTU'	
První třetina	4000-47FF	[16384-18431]	5800-5BFF	[22528-22783]
Druhá třetina	4800-4FFF	[18432-20479]	5900-59FF	[22784-23039]
Třetí třetina	5000-57FF	[20480-22527]	5A00-5AFF	[23040-23295]

-----  
0E88 CL-ATTR

CALL najde adresu atributu k pixelu

Vstup : HL = adresa 'devátého' bajtu pixelu

Výstup: DE = adresa atributu pixelu

Mění : A, HL, BC a příznaky (BC= 32\*CB !!!)

- - -

Pozn.: 'devátý' bajt pixelu je adresa osmého bajtu pixelu zvětšená o 0100 [256] (instrukcí INC H). Chcete-li pouze operaci BC=32\*CB stačí tuto rutinu volat CALL 0E91.

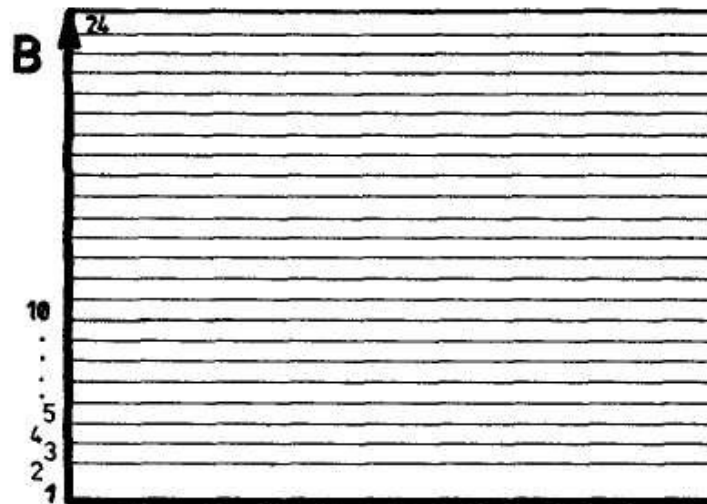
---

**0E9B CL-ADDR**

CALL najde adresu VIDEO-BYTE

Ustup : B = číslo řádky zdola (01-18 [1 až 24])  
 Výstup: HL = adresa 1. bajtu 1. pixelu řádky 'B'  
 Mění : A, D a příznaky (A=H)  
 - - -

Obrázek 12 - adresování řádků rutiny 'CL-ADDR'



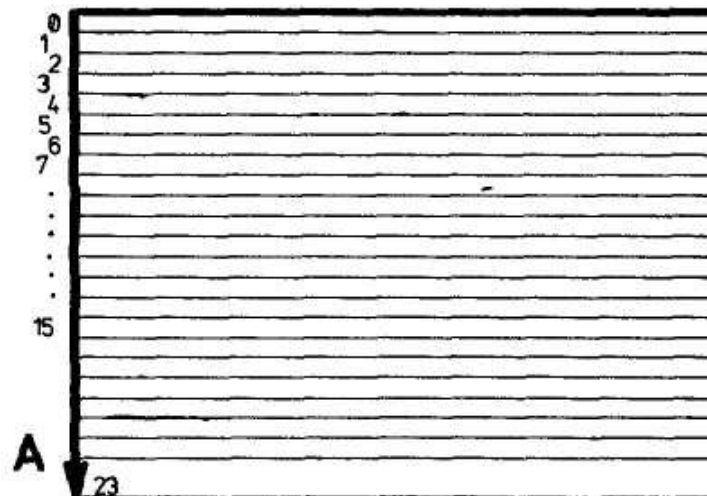

---

**0E9E CL-ADDRA**

CALL najde adresu VIDEO-BYTE - 'BASIC'

Ustup : A = číslo řádky shora (00-17 [0 až 23])  
 Výstup: HL = adresa 1.bajtu 1.pixelu řádky 'A'  
 Mění : A, D a příznaky (A=H)  
 - - -

Obrázek 13 - adresování řádků rutiny 'CL-ADDRA'





---

**22AA PIXEL-ADD**

CALL najde adresu videobitu (PLOT)

Vstup : C = souřadnice x <00;FF> [<0;255>]

B = souřadnice y <00;AF> [<0;175>] (viz obr.14)

Výstup: HL = adresa příslušného bajtu

A = pořadí bitu v bajtu <0 až 7>

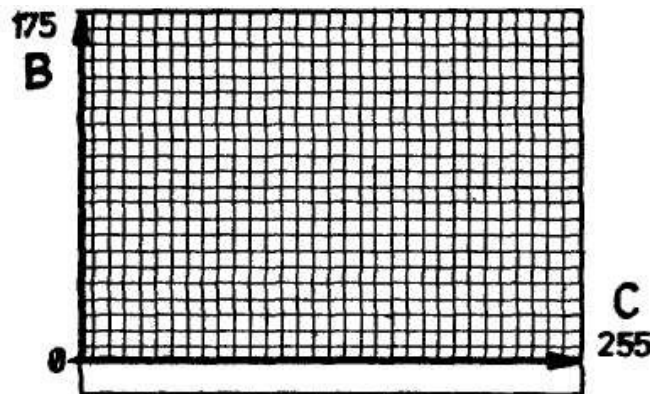
pro B > AF [175] ERROR 'B Integer out of range'

Mění : BC a příznaky

- - -

Rutina vypočítá pro dané souřadnice x,y adresu HL a bit A, kde se má příslušný bod zobrazit (jako PLOT x,y).

Obrázek 14 - souřadnice x,y rutiny 'PIXEL-ADD'



Pozn.: potřebujeme-li zjistit adresu bodu v dialogovém řádku nebo nám vyhovují tyto nové souřadnice, použijeme rutinu:

---

**22B0 PIXEL-ADDA**

CALL najde adresu videobitu

Vstup : C = souřadnice x <00;FF> [<0;255>]

A = souřadnice y <00;BF> [<0;191>] (viz obr. 15)

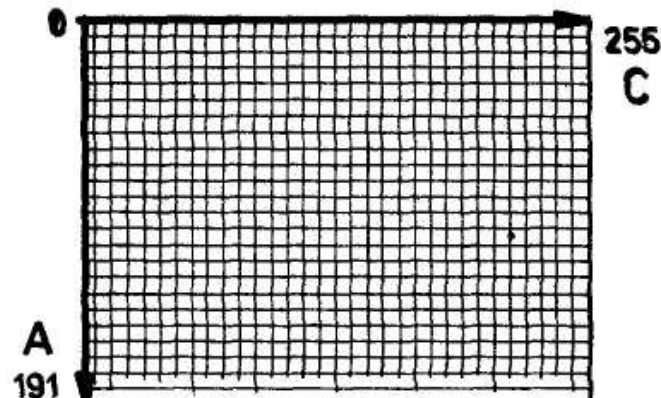
Výstup: HL = adresa příslušného bajtu

A = pořadí bitu v bajtu (0 až 7)

pro A > BF [191] - nesmyslná pozice

Mění : BC a příznaky

Obrázek 15 - souřadnice x,y včetně dialog. řádků



-----  
22CB POINT-SUB

CALL příkaz POINT

Vstup : dvě čísla kalkulačky (STK-TO-BC)

Výstup: číslo kalkulačky (STACK A)

Mění : A, B, příznaky a kalkulačka

- - -

Tato rutina vykoná příkaz BASICU POINT x,y. Souřadnice x,y jsou uložena jako čísla kalkulačky, která bude popsána v kapitole 5.

=====

Příklady:

-----

## 1) Smazání obrazovky

Program 10 - clear screen

CLS	LD	HL,4000	;odkud	210040	[33,0,64	1
	LD	DE,4001	;kam	110140	[17,1,64	]
	LD	BC,1800	;délka	010018	[1,0,24	]
	LD	(HL),L	;nastavení	75	[117	]
	LDIR		;1.bajtu	EDB0	[237,176	]
ATTR	LD	A,(5C8D)	;s.p. ATTR P	3A8D5C	[58,141,92	]
	LD	(HL),A	;nastav ATTR	77	[119	]
	LD	BC,02FF	;kolik	01FF02	[1,255,2	]
	LDIR		;clear ATTR	EDB0	[237,176	]
	RET		;koniec	C9	[201	]

Pozn.: První část (CLS) programu maže videobajty (nastavuje je na 00) a část druhá (ATTR) maže atributy (tj. vyplňuje atributy standardní hodnotou (ATTR P)).

U s.p. 'ATTR P' je uložena hodnota atributu pro standardní tisk.

Podprogramy podobné stavby mohou vyplňovat daným kódem libovolné oblasti v paměti. Instrukce 'LD (HL),L' se nahradí instrukcí 'LD <HL>,n', kde n je daný kód.

2) vytisknutí znaku v A na pozici B,C (PRINT AT B,C; CHR\$ A)

Program 11 - print na pozici (PRINT AT B,C; CHR\$ A)

PRINT	AND	7F	;	E67F	[230,127	]
	CP	20	; znaky ?	FE20	[254,32	]
	RET	C	; return (00-1F; 80-9F)	D8	[216	]
	LD	DE,3C00	; ASCII tabulka znaků	11003C	[17,0,60	]
	ADD	A,A	; - 0100	87	[135	]
	LD	L,A	;	6F	[111	]
	LD	H,E	;	63	[99	]
	ADD	HL, HL	; ASCII = 8*ASCII	29	[41	]
	ADD	HL, HL	; tabulka+8*ASCII	29	[41	]
	ADD	HL, DE	; adresa znaku v DE	19	[25	]
	EX	DE, HL	;	EB	[235	]
	LD	A,B	; řádka B=00-17 [0-23]	78	[120	]
	LD	B,D	; úschova D	42	[66	]
	CALL	0E9E,CL-ADDR	; první bajt	CD9E0E	[205,158	]
	LD	D,B	;	50	[14,80	]
	LD	B,00	;	0600	[6,0	]
	ADD	HL, BC	; připočten sloupec	09	[9	]
	LD	B,08	; 8 bajtů znaku	0608	[6,8	]
LOOP	LD	A,(DE)	; bajt z tabulky	1A	[26	]
	LD	(HL),A	; bajt na obrazovku	77	[119	]
	INC	DE	; další bajt tabulky	1C	[28	]
	INC	H	; další bajt pixelu!	24	[36	]
	DJNZ	LOOP	; celý znak	10F9	[16,249	]
			; HL='devátý' bajt			
	CALL	0E88,CL-ATTR	; adresa atributu	CD880E	[205,136	]
	LD	A,8E	; F=1, B=0, P=1, I=6	3E8E	[14,62	]
	LD	(DE),A	; nastavení atributu	12	[18	]
	RET		; konec	C9	[201	]

Pozn.: Tento program nahrazuje rutinu 'RST 0010, PRINT-A-1', která vyžaduje existenci mnoha s.p. a definic kanálů. Náš program 'Print' je zcela nezávislý a je tedy možné jej použít všude !!!

3) vykreslení bodu o souřadnicích C,B (PLOT C,B)

Program 12 - plot na pozici

PLOT	CALL	22AA	; adresa	CDAA22	[205,170,34	]
	INC	A	;	3C	[60	]
	LD	B,A	; počet cyklů	47	[71	]
	LD	A,01	; '1' rotuje	3E01	[62,1	]
LOOP	RRCA		; do správného	0F	[15	]
	DJNZ	LOOP	; bitu	10FD	[16,253	]
	OR	(HL)	; A = set bit	B6	[182	]
	LD	(HL),A	; nový bajt	77	[119	]
	RET		; konec	C9	[201	]

#### 4) zjištění bodu o souřadnicích C,B (POINT C,B) Program 13 - point pozice

POINT	CALL 22AA	; adresa	CDAA22	[205,170,34 ]
	INC A		3C	[60 ]
	LD B,A	; počet cyklů	47	[71 ]
	LD A,(HL)	; A = videobajt	7E	[126 ]
LOOP	RLCA	; D0 = test.	07	[7 ]
	DJNZ LOOP	; bit	10FD	[16,253 ]
	AND 01	; A=00 pro '0'	E601	[230,1 ]
	RET	; A=01 pro '1'	C9	[201 ]

Pozn.: Po vykonání této rutiny obsahuje reg. A číslo 0 (resp. 1) je-li bod o souřadnicích C,B Paper (resp. Ink).

#### 5) posuv obrazovky nahoru a dolů

##### Program 14 - roll up & down

Move	LD A,BF	; počet řádků	3EBF	[62,191 ]
	LD BC,0020	; počet bajtů	012000	[1,32,0 ]
	LD DE,5BE0	; print - buffer	11E05B	[17,224,91 ]
up-1	LD HL,4000	; pro 'up'	210040	[33,0,64 ]
	(LD HL,57E0) ;pro 'down' (21E057)		[33,224,87]	
	PUSH HL	; odlož adr.	E5	[229 ]
	LDIR	; copy 1.řádek	EDB0	[237,176 ]
loop	EX AF,A'F'	; A'=počet řádků	08	[8 ]
	POP DE		D1	[209 ]
	PUSH DE		D5	[213 ]
	LD H,D		62	[90 ]
	LD L,E	; HL = DE	6B	[107 ]
	CALL CHANGE	; normal. adr.	CDnnnn	[205,n,n ]
	LD C,20	I	0E20	[14,32 ]
	AND A	I	A7	[167 ]
up-2	ADD HL, BC &	NOP;inc pro 'up'	0900	[9,0 ]
	(SBC HL,BC) ;dec pro 'down' (ED42)		[237,66 ]	
	CALL CHANGE	; Spectrum adr.	CDnnnn	[205,n,n ]
	POP DE		D1	[209 ]
	PUSH HL		E5	[229 ]
	LDIR	; posun 1 řádek	EDB0	[237,176 ]
	EX AF,A'F'	I	08	[8 ]
	DEC A	; počet řádek -1	3D	[61 ]
	JR NZ,loop	; není celé	20E8	[32,232 ]
	POP DE	i	D1	[209 ]
	LD HL,5BE0	; print - buffer	21E05B	[33,224,91 ]
	LD C,20	; přenes	0E20	[14,32 ]
	LDIR	; poslední	EDB0	[237,176 ]
	RET	; řádek	C9	[201 ]

Pozn.: Uvedený program bez uvedených hodnot v závorkách pracuje jako program 'ROLL-UP'. Náhradíme-li instrukce na adresách) up-1, up-2 instrukcemi v závorkách vytvoříme program 'ROLL-DOWN'.

U návěští 'up-2' je instrukce NOP důležitá pro zachování Místa potřebného pro instrukci SBC ('ROLL-DOWN').

Program 'CHANGE', zde užitý, je uveden a popsán jako program číslo 9.

## 5 . K A L K U L A Č K A =====

"Co to je? Mám to vůbec v ROMce?" - i takto lze hovořit o Části operačního systému nazvaného "Kalkulačka". Mnoho majitelů ZX Spectra pokládá tuto oblast za nedůležitou! Ovšem opak je pravdou. Kalkulačka je přímo nezbytná a je "páteří" celého BASIC systému. Mezi nejdůležitější úkoly patří výpočty numerických funkcí (+, INT, ABS, SQR, SIN, LN) , předávání parametrů mezi příkazy, funkcemi a operacemi.

Popis "Kalkulačky" odpovídá její důležitosti. Velice podrobně bude vysvětlen systém její práce, zápis čísel, a oblasti paměti kalkulačkou používané. Ze samotných operací jsou některé vybrány pro osvětlení výkladu na příkladech.

Ostatní operace jsou stručně a jasně popsány, přičemž řetězové operace budou popsány v druhém dílu.

Kalkulačka je soubor 66 základních a mnoha dalších pomocných operací. V ROMce je rozmístěna od 2D4F až 386D a je označována FP-CALC, což v překladu znamená kalkulačka s plovoucí čárkou.

### 1) Č Í S L A

=====

Každá číselná hodnota je v kalkulačce zaznamenána do pěti bajtů a podle konkrétní hodnoty rozlišujeme čísla na jeden z typů 'INTEGER (short)' nebo 'REAL'.

#### a) INTEGER

-----

Je číslo, které je celé a leží v intervalu [ $-65533; 65333$ ]. Pět bajtů uchovávajících číslo typu INTEGER je pak nastaveno následovně:

1. bajt = 00

2. bajt = znaménko:    00 - kladné číslo  
                          FF - záporné číslo

3. a 4. bajt = standardně uložená hodnota čísla od 0000 do FFFF.  
Je-li číslo záporné (2. bajt=FF) pak je číslo uloženo v doplňku (doplňek  $x' = [65536] - |x|$ ). Hodnota je uložena v pořadí nižší - vyšší (LowHi).

5. bajt = 00

Příklady:	sgn	low	hi			
00	00	00	00	00	= >	0000 [ 0]
00	00	01	00	00	= >	0001 [ 1]
00	00	0A	00	00	= >	000A [ 10]
00	00	39	30	00	= >	3039 [12345]
00	FF	FF	FF	00	= >	-0001 [ -1]
00	FF	FF	FE	00	= >	-0100 [ -256]

## b) REAL

-----

je číslo, které není typu INTEGER. Je zapsáno v exponenciálním tvaru :

$$\pm m \cdot 2^{\pm e} \quad \begin{array}{l} \text{, kde 'm' je mantisa v intervalu } \langle 1/2; 1 \rangle \\ \text{a 'e' je exponent v intervalu } \langle -127; 127 \rangle \end{array}$$

Hodnota čísla je uložena takto:

1. bajt = exponent + 80 [128]

2. až 5.bajt = mantisa

Protože 1.bit (nejvyšší) mantisy je pro všechna 'm' roven 1, je tento bit využit pro záznam znaménka mantisy. Je-li tento bit roven jedné - je mantisa záporná a je-li nulový - mantisa je kladná a s bitem se počítá jako by byl vždy nastaven => (1/2). Popis je též v manuálu ZX Spectrum.

Příklady:

-----

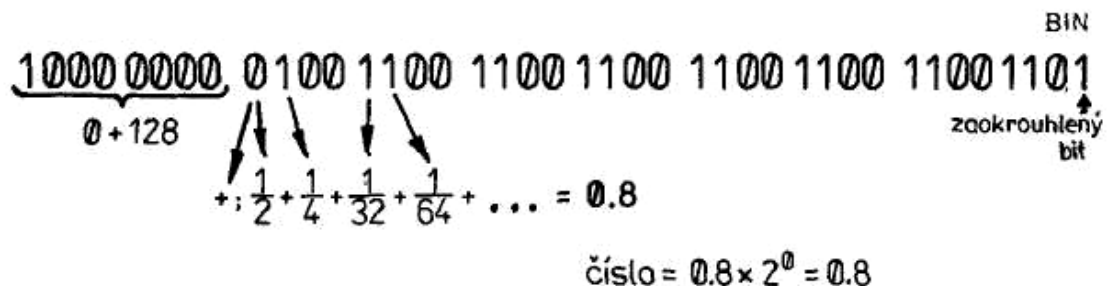
a) zápis čísla 0.125 jako čísla typu 'REAL'

Obrázek 16 - číslo typu REAL 0.125



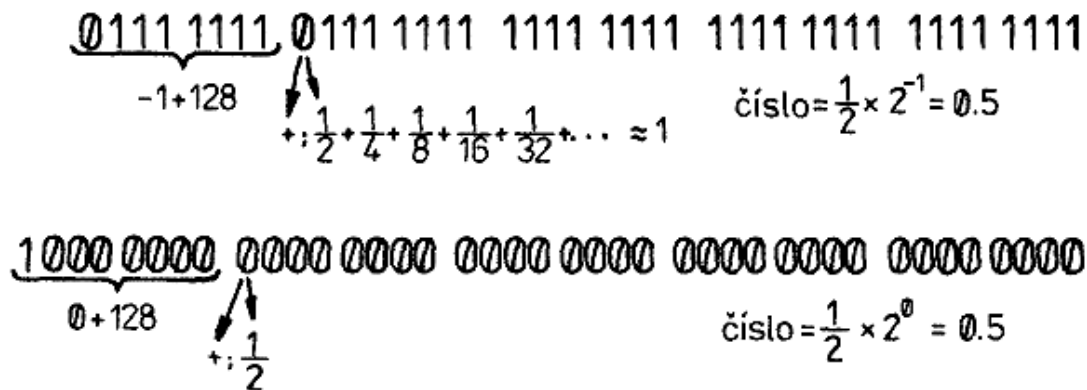
b) zápis čísla 0.8 jako čísla typu 'REAL'

Obrázek 17 - číslo typu REAL 0.8



c) pro číslo 0.5 existují dvě možnosti uložení:

Obrázek 18 - číslo typu REAL 0.5



Z dvojznačnosti uložení vyplývají i chyby: např.  $1/2$  není 0.5 (zkuste PRINT 1/2=0.5, výsledkem je 0).

V tomto uspořádání jsou ukládána čísla do z. k., registrů a také je nalezneme v BASIC programech za kódem 0E.

## 2) REGISTRY KALKULAČKY

=====

Ve standardním tvaru má kalkulačka 6 registrů: mem-0, mem-1, mem-2, mem-3, mem-4 a mem-5. Každý registr zabírá 5 bajtů a registry celkem 30 bajtů, které jsou v RAM uloženy v oblasti paměti 'MEMBOT', tj. 5C92-5CAF [23698-23727]. Protože je ale počáteční adresa 'MEMBOT' adresována s.p. 'MEM' můžeme pomocí její změny oblast registrů přemístit a zvětšit. Uzniknou tak další registry (mem-7 až mem-31).

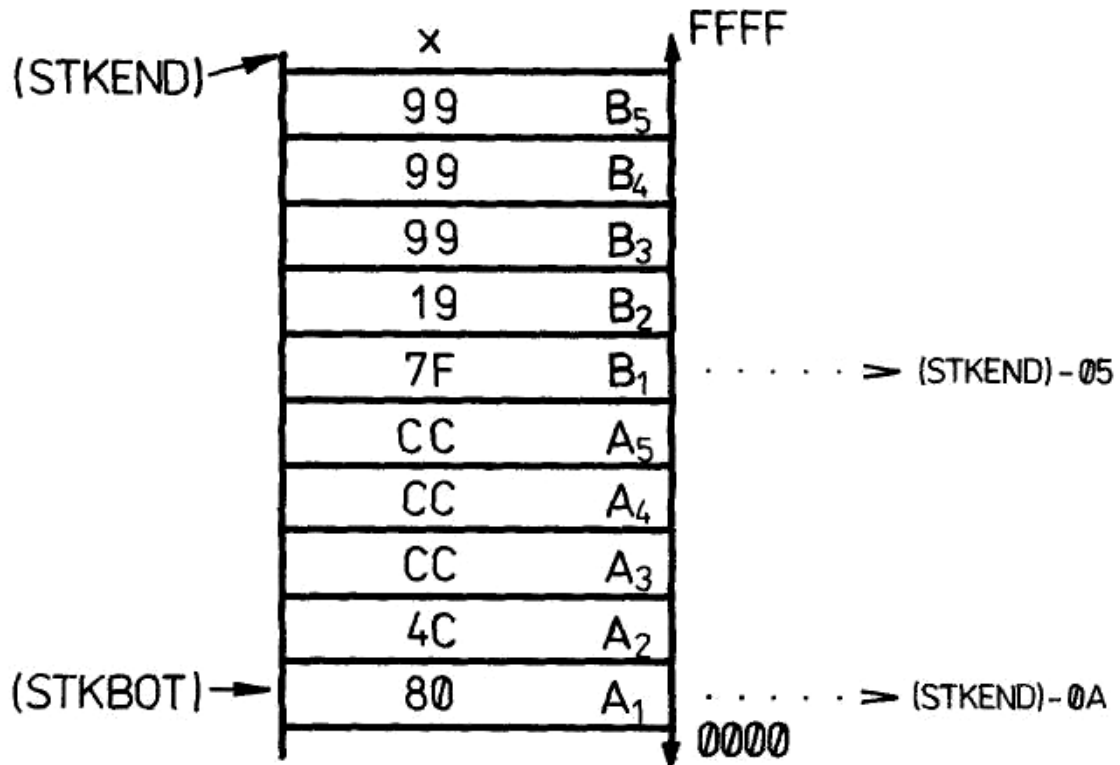
Práce s registry je popsána u rutin 'GET-MEM' a 'ST-MEM'.

## 3) ZÁSOBNÍK KALKULAČKY {z. k.}

=====

Obdobně jako u mikroprocesoru má i kalkulačka svůj zásobník. 'Dno' z. k. je na nižší adrese a 'vrchol' zásobníku se 'posouvá' k vyšším adresám. Na 'dno' zásobníku ukazuje s.p. 'STKBOT' {5C63- [23651]} a na 'vrchol' (první volný bajt za všemi čísly) ukazuje s.p. 'STKEND' {5C65 [23653]}. Označme si čísla v zásobníku 'A,B' (B je na 'vrcholu' zásobníku) a pro pořadí jejich bajtů s indexy pak platí (viz obrázek 19).

Obrázek 19 - zásobník kalkulačky



Z obrázku vidíme uspořádání čísel a jejich označení vzhledem k 'vrcholu' zásobníku, takže adresa (STKEND)-5 je adresou posledního čísla uloženého v zásobníku tj. 'B' a adresa [(STKEND)-103] je adresou předposledního čísla. Kdybychom přidávali do zásobníku další čísla, *mohlo* by se stát, že bychom mohli přemazat jiná důležitá data. Proto se před každým vložením nového čísla kontroluje, zda je pro něj dosti místa v paměti.

Paměť se testuje pomocí rutin:

#### 1F05 TEST-ROOM

CALL rozsah obsazené paměti

Vstup : BC = přírůstek nových bajtů

Výstup: HL = počet obsazených bajtů (i s novými)

Není místo = Error '4 Out of memory'

Mění : BC a příznaky

- - -

Rutina spočítá a otestuje rozsah obsazené paměti, přibude-li 'BC' nových bajtů. Jestliže by 'BC' novými bajty byla paměť přeplněna je voláno chybové hlášení '4 Out of memory'.

Pozn.: [HL=65536-sp+(STKEND)+68+BC] .



---

1F1A FREE-MEM

CALL rozsah obsazené paměti

Vstup : -

Výstup: BC = počet 'obsazených' bajtů

Mění : Příznaky, DE, HL (HL=BC)

- - -

Rutina zjistí rozsah 'obsazené' paměti (včetně chráněných bajtů, např. CLEAR 64994) přibude-li nulový počet bajtů (pomocí 'TEST-ROOM'). Výsledek uloží do 'BC', čímž je dosažitelný z BASICu.

Zkuste volnou paměť : 'PRINT 65536-USR 7962'.

---

33A9 TEST-5-SP

CALL místo pro číslo kalkulačky

Vstup : -

Výstup: není-li místo, chyb. hlášení '4 Out of memory'

Mění : Příznaky, BC (BC=0005)

- - -

Rutina spočítá rozsah obsazené paměti, přibude-li pět nových bajtů (tj. jedno číslo kalkulačky). Není-li pro nové číslo místo, je voláno chybové hlášení '4 Out of memory'. Je-li místa dostatek, vrací se rutina zpět bez parametrů.

Pozn.: využívá 'TEST-ROOM'; HL a DE zůstávají

---

4) INSTRUKCE

=====

Kalkulačka se volá 'RST 0028, FP-CALC nebo ekvivalentně 'CALL 335B, CALCULATE'. Ihned za tímto voláním následuje několik definovaných bajtů, které jsou vyhodnoceny jako instrukce pro kalkulačku (sečti, vynásob, ...). Kalkulačka prohlíží jednotlivé definované bajty (sekvenčně - za sebou) a podle nich provádí žádané operace. Posledním definovaným bajtem musí být vždy bajt 38 [56] (end-calc), který práci kalkulačky ukončí a provede RETURN na adresu následující za bajtem 38 [56]. Definovaný bajt s instrukcí pro kalkulačku nazýváme 'offset'.

Tabulka instrukcí kalkulačky s adresami rutin podle offsetů začíná na následující stránce

Tabulka 9 – offsety kalkulačky

offset	tab.	název	rutina	provede:
00	32D7	jump-true	368F	jestliže-skok
01	32D9	exchange	343C	zaměň posl. dvě čísla
02	32DB	delete	33A1	smaž poslední číslo
03	32DD	subtract	300F	odečti
04	32DF	multiply	30CA	vynásob
05	32E1	division	31AF	vyděl
06	32E3	to-power	3851	umocnění xty
07	32E5	or	351B	OR posl. dvou čísel
08	32E7	no-&-no	3524	AND posl. dvou čísel
09	32E9	no-l -eq	353B	-
0A	32EB	no-gr-eq	353B	-
0B	32ED	nos-neq	353B	-
0C	32EF	no-grtr	353B	-
0D	32F1	no-less	353B	-
0E	32F3	nos-eq	353B	-
0F	32F5	addition	3014	sečti
10	32F7	str-&-no	352D	-
11	32F9	str-l -eq	353B	-
12	32FB	str-gr-eq	353B	-
13	32FD	strs-neq	353B	-
14	32FF	str-grtr	353B	-
15	3301	str-less	353B	-
16	3303	str-eq	353B	-
17	3305	strs-add	359C	-
18	3307	val	35DE	-
19	3309	usr-\$	34BC	-
1A	330B	read-in	3645	-
1B	330D	negate	346E	opačná hodnota
1C	330F	code	3669	-
1D	3311	val	35DE	-
1E	3313	len	3674	-
1F	3315	sin	37B5	sinus
20	3317	cos	37AA	kosinus
21	3319	tan	37DA	tangens
22	331B	asn	3633	arkussinus
23	331D	acs	3643	arkuskosinus
24	331F	atn	37E2	arkustangens
25	3321	ln	3713	přirozený logaritmus
26	3323	exp	36C4	etx
27	3325	int	36AF	celá část čísla
28	3327	sqr	364A	odmocnina
29	3329	sgn	3492	znaménko
2A	332B	abs	346A	absolutní hodnota
2B	332D	peek	34AC	PEEK
2C	332F	in	34A5	IN
2D	3331	usr-no	34B3	spuštění assembleru
2E	3333	str\$	361F	-
2F	3335	chr\$	35C9	-
30	3337	not	3501	logická negace
31	3339	duplicate	33C0	kopie posl. čísla
32	333B	n-mod-m	36A0	celočíslné dělení
33	333D	jump	3686	skok
34	333F	stk-data	33C6	ulož data
35	3341	dec-jr-nz	367A	zmenši a skoč není-li nula
36	3343	less-0	3506	menší než nula
37	3345	greater-0	34F9	větší než nula

## Pokračování tabulky 9 - offsety kalkulačky

offset	tab.	název	rutina provede:	
38	3347	end-calc	369B	konec kalkulačky
39	3349	get-argt	3783	redukce argumentu sin a cos
3A	334B	truncate	3214	část bližší nule
3B	334D	fp-calc-2	33A2	opakuji operaci
3C	334F	e-to-fp	2D4F	-
3D	3351	re-stack	3297	číslo do typu REAL
3E	3353	series	3449	Chebyshevův polynom
3F	3355	stk-data	341B	konstanty
40	3357	st-mem	342D	uložení do registru
41	3359	get-mem	340F	vybrání z registru

offset 3E se zadává def. bajtem 86,88,8C pro série [6,8,12]  
 3F se zadává d.b. A0 až A4 pro konst. [0,1,1/2,3.14/2,10]  
 40 se zadává d.b. C0 až C5 pro uloř. do reg. 0 až 5  
 41 se zadává d.b. E0 až E5 pro vybrání z reg. 0 až 5

Proškrtnuté překlady offsetů budou uvedeny v druhém dílu.

Kalkulačka rozeznává tři typy offsetů:

- 1) Binární operace (např. sčítání) - operace do nichž vstupují dvě čísla (či řetězce) a výsledkem je číslo jedno (resp. řetězec). Toto má za následek, že zásobník kalkulačky se o jedno číslo zkrátí, {offset 00 až 17}
- 2) Unární operace (např. sinus) - operace, jejichž vstupem i výstupem je jedno číslo (např.  $x \rightarrow \sin x$ ). Počet čísel v zásobníku se nemění. (offset 18 až 3D)
- 3) Skupinové operace - operace series, stk-const, st-mem, get-mem. {offset 3E až 41}

Podle offsetu je vyhledána a provedena daná operace; schematicky lze toto znázornit obrázkem 20.

-----  
35BF STK-PNTRS

CALL Nastavení ukazatelů čísel

Vstup : -

Výstup: DE = (STKEND)

HL = (STKEND)-05

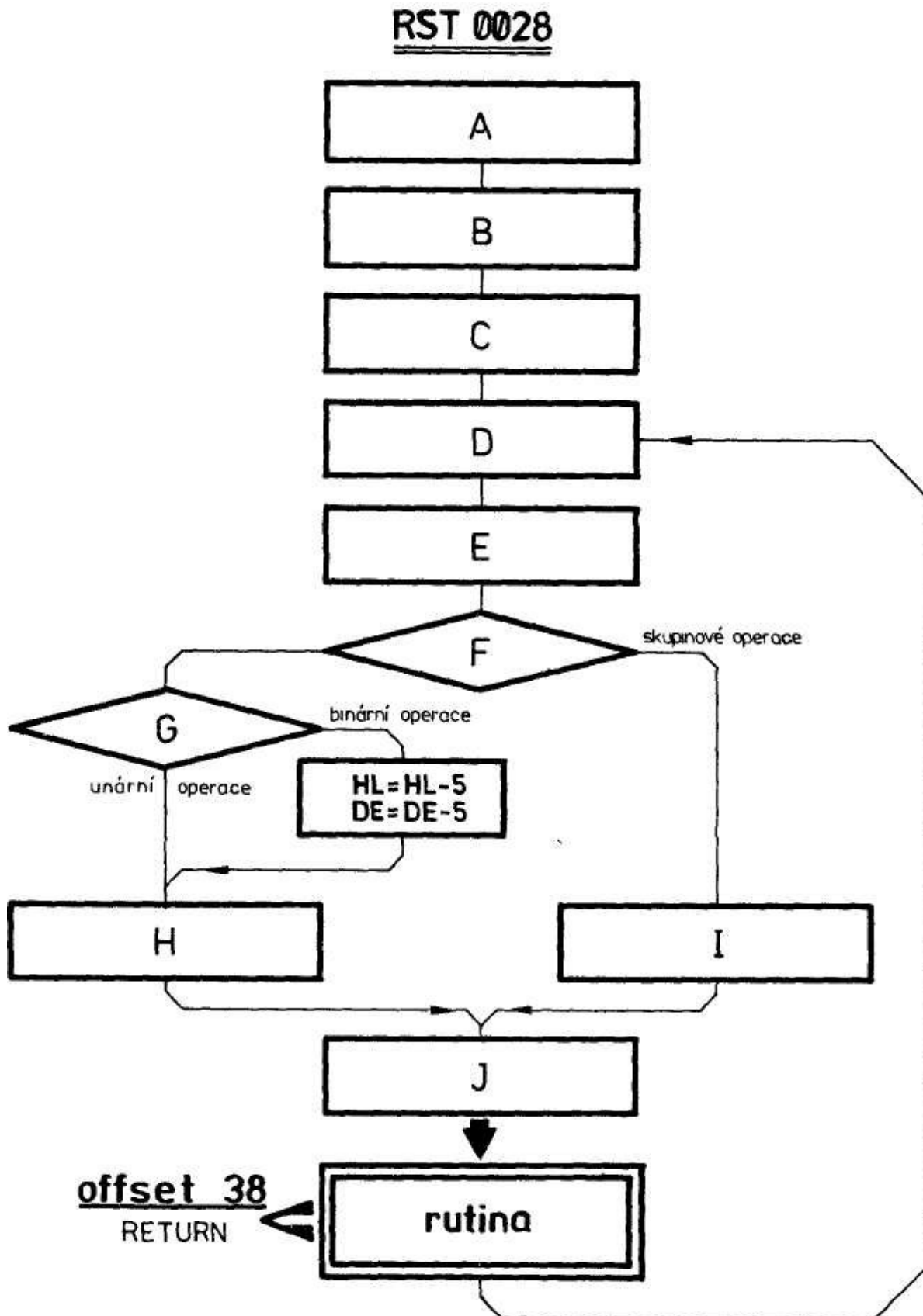
Mění : Příznaky

- - -

Pozn.: Tento podprogram nastavuje HL na poslední číslo z.k., tj. (STKEND)-5 a reg. DE na konec z.k., tj. (STKEND).

-----

Obrázek 20 - schema práce kalkulačky



Podrobný popis práce kalkulačky: (podle obrázku 20)  
 =====

- A: Pomocí vyvolání podprogramu 'STK-PNTRS' se naplní HL a DE takto:  
 DE = (STKEND); HL = (STKEND)-5, tj. adresa posledního čísla.
- B: Registr B je uschován do s.p. 'BREG'.
- C: Obsah H'L' se zamění s obsahem sklípku mikroprocesoru {EX (SP),H'L'}. U H'L' je nyní návratová adresa a ve sklípku je uložena hodnota H'L'.
- D: Systémová proměnná (STKEND) je nastavena podle registru DE ! (velice důležité !!!). Návěští v ROM: 'RE-ENTRY'.
- E: Do akumulátoru je načten offset {LD A,(H'L')}, a poté je H'L' inkrementován a uložen do sklípku mikroprocesoru.
- F: Proveďte se test akumulátoru a je-li offset větší než 80 [128] (skupinové operace) pokračuje činnost kalkulačky bodem I.
- G: Proveďte se druhý test akumulátoru a je-li offset menší či roven 17 [23] (binární operace) jsou DE i HL zmenšeny o 5 takže DE=(STKEND)-5 { adresa posl. čísla } a HL=[(STKEND)-10] {adresa předposledního čísla}.
- H: Dvojnásobek offsetu je uložen do L' a pokračuje bodem J.
- I: Podle obsahu akumulátoru (86 až 8C; A0 až A4; C0 až C5; E0 až E5) se najde příslušný offset 3E,3F,40 či 41, který je zdvojnásoben a uložen do L'. Akumulátor obsahuje v bitech D4,D3,D2,D1,D0 hodnotu načteného offsetu pro pozdější rozlišení operací.
- J: K počáteční adrese tabulky adres rutin je přičteno L' a tím je zjištěna adresa v tabulce, kde je uložena adresa rutiny dané instrukce. Adresa rutiny je přečtena a uložena do D'E'. Ze sklípku  $\mu P$  je vybrána návratová adresa (uschovaná v bodě E) a je uložena do H'L'. Do sklípku se vloží návratová adresa 3365 {adresa bodu D}. Jako další je do sklípku uložena hodnota D'E', což je adresa rutiny, která se má provádět, a registr B je obnoven ze s.p. 'BREG'.

Provede se RET !!!

pozn.: Instrukce RET provede skok na hodnotu ze zásobníku  $\mu P$ . -tedy skočí na "naši" rutinu. Rutina je zakončena opět instrukcí RET - vrátí se na adresu '3365, RE-ENTRY' (uložené v bodě J), což je adresou bodu D. Je vybrán další offset a provedena příprava podle výše uvedených bodů až do načtení offsetu 3E, který práci kalkulačky ukončí.  
 S.p. 'BREG' je využívána pouze v podprogramech 'fp-calc-2' {offset 3B} a 'dec-jr-nz' {offset 35}.

Je-li tedy volána některá rutina kalkulačky mají registry tento obsah:

A	=	nižší 8 bitů offsetu (pro skupinové operace)
B	=	(BREG)
C	=	(STKEND - vyšší bajt)
		pro unary                      pro binary
DE	=	(STKEND)                      (STKEND) - 05
HL	=	(STKEND) - 05                      (STKEND) - 0A
B'C'	=	neporušeno
D'E'	=	adresa volané rutiny
H'L'	=	adresa návratu do hlavního programu



NEMÁTE NÁHODOU NÁHRADNÍ KALKULAČKU DO SPECTRA 2  
NEMÁME.

Před samotným spuštěním kalkulačky je většinou třeba uložit do z. k. nějaké vstupní parametry, nebo tyto parametry po ukončení kalkulačky přečíst a jinak použít. K tomuto účelu využijeme následujících rutin:

---

#### 2AB6 STK-STORE

CALL uložení čísla do z. k.

Vstup: A,E,D,C,B - čísla v pěti bajtech (registrech)

Ústup: čísla v z.k.

není-li místo - chyb. hlášení '4 Out of memory'

Mění : (STKEND)=(STKEND)+05; HL=(STKEND)

- - -

Podprogram otestuje, zda je v z. k. místo na nové číslo. Pomocí rutiny '2AB6, TEST-5-SP' a pak uloží registry A,E,D,C,B jako čísla kalkulačky od adresy (STKEND). Adresa (STKEND) je spolu s reg. HL zvětšena o pět.

---

#### 2DBE INT-STORE

CALL uloží DE od adresy HL jako číslo typu INTEGER

Vstup : HL = adresa uložení

DE = číslo

C = znaménko čísla DE - 00 kladné

\ FF záporné

Ústup: číslo DE od adresy HL typu INTEGER

Mění : A, příznaky

- - -

Podprogram uloží číslo v registru DE se znaménkem C jako číslo typu INTEGER od adresy v HL. Jestliže je číslo DE záporné (C=FF), je číslo uloženo v doplňku.

{00, (C), (E), (D), 00}

Např.: číslo -1 se uloží takto:

LD C,FF

LD HL,8000

LD DE,0001

CALL 2DBE,INT-STORE

Tyto instrukce vytvoří číslo -1 od adresy 8000 [32768] popsané v příkladech zápisu čísel typu INTEGER.

-----  
2D8C P-INT-STO

CALL uloží DE jako kladné číslo INT od adr. HL

Vstup : HL = adresa uložení  
DE = číslo

Výstup: kladné číslo typu INT od adr. HL

Mění : A, C a příznaky (C=00)

- - -

Podprogram nastaví C=00 a pokračuje rutinou '2D8E, INT-STORE'.  
Tím se DE uloží od adresy HL jako číslo kalkulačky typu INTEGER  
vždy kladné.

{00,00,(E),(D),00}

Např.: číslo 0A [10] uložíme:

LD HL,8000

LD DE,000A

CALL 2D8C, P-INT-STO

Tyto instrukce vytvoří číslo [10] od adresy 8000 [32768] popsané  
v příkladech zápisu čísel typu INTEGER.-----  
2D2B STACK-BC

CALL do z. k. uloží BC (typ INTEGER)

Vstup : BC = číslo

Výstup: kladné číslo v z. k. typu INTEGER

Mění : IY (=5C3A), kalk., A, příznak (NC)

- - -

Podprogram vypadá následovně:

```

2D2B STACK-BC    LD    IY,5C3A
                  XOR    A
                  LD     E,A
                  LD     D,C
                  LD     C,B
                  LD     B,A
                  CALL 2AB6,STK-STORE
                  RST    0028,FP-CALC
                  DEFB 38,end-calc
                  AND    A
                  RET

```

Registr IY je nastaven na hodnotu 5C3A, číslo z BC je transformováno do CD, registry A,E,B jsou vynulovány a pro zápis je použit podprogram '2AB6, STK-STORE' (viz jeho formát). Potom je zavolána a ihned ukončena kalkulačka. To způsobí aktualizaci HL,DE,(STKEND) po přijmutí nového čísla. Na závěr je nulován příznak Carry.

U kolonce "Mění" znamená 'kalkulačka' přepsání všech registrů, popsaných v rutině '335B,CALCULATE'. Toto označení budeme používat v popisu všech následujících rutin kalkulačky.



-----  
2D28 STACK-A

CALL uloží A do z. k. typu INTEGER

Vstup : A = číslo

Výstup: kladné číslo v z. k. typu INTEGER

Mění : IY (=5C3A), kalk., A, příznak (NC)

- - -

Podprogram přesune A do BC (B=00,C=A) a k uložení čísla použije rutinu '2D2B, STACK-BC'.

-----  
2D22 STK-DIGIT

CALL uloží č. 0 až 9 do z. k.

Vstup : A = ASCII čísla 0 až 9

Výstup: (A=ASCII 0-9) -&gt; číslo v z. k., příznak NC

(A&lt;&gt;ASCII 0-9) -&gt; chyba, příznak C

Mění : IY ( = 5C3A), kalkulačka, A, příznaky

- - -

Podprogram zjistí pomocí rutiny '2D1B, NUMERIC hodnotu v A a není-li ASCII numerické vrátí se s nastaveným příznakem Carry a číslo se neuloží do z. k.. Jinak je číslo uloženo rutinami '2D2B, STACK-A' a '2D2B, STACK-BC' na vrchol zásobníku a příznak Carry nulován.

=====

Pro vybrání čísla ze z. k. použijeme těchto podprogramů:

-----  
2D7F INT-FETCH

CALL číslo INT. z kalkulačky uloží do DE (C- znaménko)

Vstup : HL = adresa čísla INTEGER kalkulačky

Výstup: DE = číslo

C = znaménko - 00 kladné číslo

FF záporné číslo

Mění : Příznaky, A, HL (HL=HL+3)

- - -

Podprogram je přesně inverzní k rutině '2D8E, INT-STORE'

Např.: číslo z adresy HL uloží do DE takto:

HL: 00 FF FF FF 00

C=FF, DE=0081

-----  
2DA2 FP-TO-BC

CALL číslo ze z. k. do BC

Vstup : číslo v z. k.

Výstup: BC = číslu ze z. k. v rozsahu [(-65535.5;65535.5)]

a příznak NC - v pořádku

C - ERROR, číslo mimo rozsah

NZ - záporné číslo v BC

Z - kladné číslo v BC

Mění : kalkulačka, A=C, příznaky

- - -

Prohlédneme si podprogram podrobněji:

```

2DA2 FP-TO-BC    RST    0028,FP-CALC
                  DEFB   38,end-calc      ;HL=(STKEND)-5
                  LD     A,HL              ;A=1.bajt čísla =00 INT
                                          ;                jinak REAL
                  AND    A                ;test typu čísla
                  JR     Z,2DAD,FP-DELETE ;skok pro číslo INT.
                  RST    0028,FP-CALC
                  DEFB   A2,stk-half
                  DEFB   0F,addition
                  DEFB   27,int
                  DEFB   38,end-calc      ;x=INT(x+.5) zaokrouhlení
2DAD FP-DELETE   RST    0028,FP-CALC
                  DEFB   02,delete        ;vymazáno (formálně)
                  DEFB   38,end-calc      ;číslo v z. k.
                  PUSH   HL
                  PUSH   DE
                  EX     DE,HL
                  LD     B,(HL)           ; 1.bajt do B
                  CALL   2D7F,INT-FETCH   ;2.,3. a 4.bajt do C,E
                                          ; a D
                  XOR    A
                  SUB    B                ;nastaví C je-li B<>00
                                          ; (číslo není typu INT.)
                  BIT    7,C              ;NZ při záporném čísle
                  LD     B,D
                  LD     C,E              ;DE do BC
                  LD     A,E              ;nižší bajt do A
                  POP    DE
                  POP    HL
                  RET

```

Podprogram tedy uloží, po zaokrouhlení, číslo  
[<-65535.5;65535.5)] do BC.

-----  
2DD5 FP-TO-A

CALL uloží číslo ze z. k. do A

Ustup : číslo v z. k.

Výstup: A = číslo ze z. k. je v rozsahu [<-255.5; 255.5)]

a příznak NC - v pořádku

C - ERROR, číslo je mimo rozsah

NZ - záporné číslo v A

Z - kladné číslo v A

Mění : kalkulačka

- - -

I na tuto rutinu se podívejme podrobně:

```

2DD5 FP-TO-A    CALL   2DA2,FP-TO-BC    ;ulož číslo do BC
                  RET     C                ;číslo je mimo rozsah
                  PUSH   AF              ;uschovej příznaky
                  DEC    B                ;
                  INC    B                ;je-li B=00, pak 'FP-A-END'
                  JR     Z,2DE1,FP-A-END ;jinak se číslo do A nevejde
                  POP    AF              ;
                  SCF                     ;nastav příznak chyby
                  RET                     ;
2DE1 FP-A-END   POP    AF                ;vyber příznaky
                  RET                     ;

```

-----  
2314 STK-TO-A

CALL číslo ze z. k. do A

Vstup : číslo v z. k.

Výstup: A = číslo ze z. k. je v rozsahu [ $-255.5$ ;  $255.5$ ]

C = znaménko - 01 číslo v A kladné

\ FF číslo v A záporné

'B Integer out of range', je-li číslo mimo rozsah

Mění ; kalkulačka, příznaky z rutiny '2DD5, FP-TO-A'

- - -

Pozn.; Upište si program sami a vezměte si na pomoc popis rutiny  
'2DD5, FP-TO-A', který rutina využívá.-----  
2307 STK-TO-BC

CALL dvě čísla ze z. k. do B a C

Vstup : dvě čísla v z. k. v rozsahu [ $-255.5$ ;  $255.5$ ]

Výstup: B = 1. číslo (z vrcholu z. k.)

D = znaménko 'B' 01 - kladné

FF - záporné

C = 2. číslo z. k.

E = znaménko 'C' 01 - kladné

FF - záporné

'B Integer out of range', je-li kterékoli číslo mimo uvedený  
rozsah

Mění : kalkulačka, A, příznaky (A=C)

- - -

Podprogram volá dvakrát rutinu '2314, STK-TO-A' a výsledky  
ukládá do registrů B,C,D,E.Uvědomte si vzájemné využívání jednotlivých rutin 'STK-TO-  
BC' volá 'STK-TO-A', ta pak 'FP-TO-A', a 'FP-TO-A' volá 'FP-  
TO-BC', která maže čtená čísla ze zásobníku kalk.. Z toho  
vyplývá, že i rutina 'STK-TO-BC' po přečtení čísel do B a C  
tyto čísla ze zásobníku kalk. 'vymaže'.-----  
1E94 FIND-INT1

CALL kladné číslo ze z. k. do A

Vstup : číslo v z. k.

Výstup: A = číslo ze z. k. v rozsahu &lt;00;FF&gt;

'B Integer out of range', je-li číslo mimo rozsah

Mění : kalkulačka, příznaky

- - -

Podprogram využívá rutinu '2DD5, FP-TO-A', po jejímž  
návratu testuje příznaky C (mimo rozsah) a NZ (záporné) s  
případným skokem na chybové hlášení 'B'.

---

1E99 FIND-INT2

CALL kladné číslo ze z. k. do BC

Ustup : číslo v z. k.

Ústup: BC = číslo ze z. k. v rozsahu <0000;FFFF>

'B Integer out of range', je-li číslo mimo rozsah

Mění : kalkulačka, příznaky

- - -

Podprogram využívá rutinu '2DA2, FP-TO-BC', po jejímž návratu testuje příznaky C (mimo rozsah) a NZ (záporné) s případným skokem na chybové hlášení 'B'.

---

Podprogramy FIND-INT1 a FIND-INT2 jsou výhodné při předávání parametrů, vyjadřujících obsah bajtu (datové slovo) nebo adresu (např. v příkazech POKE, OUT, IN, PEEK, PLOT, ...).

---

## 1E85 TWO-PARAM

CALL dvě čísla ze z. k. do A a BC

Ustup ; dvě čísla ze z. k.

Ústup: A = 1. číslo ze z. k. v rozsahu <-FF;FF>, je-li toto

číslo záporné, pak se do A uloží jeho doplněk

BC = 2. číslo ze z. k. (viz 'FIND-INT2')

'B Integer out of range', je-li některé z čísel mimo uvedený rozsah

Mění : kalkulačka, příznaky

- - -

Tato rutina se používá v příkazech POKE a OUT, kde můžeme vkládat i záporná čísla (vytvoří se doplněk).

Např.: v BASICu jsou tyto dva příkazy identické (stejně):

POKE nn,-1           <=>           POKE nn,255

---

1E80 POKE

CALL BASIC příkaz POKE

Ustup : dvě čísla ze z. k.

Ústup: POKE BC,A (formát čísel viz 'TWO-PARAM')

Mění : kalkulačka, příznaky

- - -

Úpis: 1E80 POKE CALL 1E85,TWO-PARAM

LD (BC),A

RET

---

1E7A OUT

CALL BASIC příkaz OUT

Ustup : dvě čísla ze z. k.

Ústup: OUT (C), A (formát čísel viz 'TWO-PARAM')

Mění : kalkulačka, příznaky

- - -

Úpis: 1E7A OUT CALL 1E85,TWO-PARAM

OUT (C),A

RET

Nyní trochu teorie:

Při provádění všech funkcí nebo příkazů jsou systémem jejich parametry vloženy do z. k. a potom se teprve provádí rutina daného příkazu (resp. funkce). Napíšeme-li 'POKE 16384,-10', jsou nejdříve tato čísla uložena do zásobníku kalkulačky. Pak je systémem nalezen podprogram 'POKE' a ten si už vyvolá parametry pomocí rutiny 'TWO-PARAM'. Zjistí, že A je záporné a vytvoří jeho doplněk [NEG 10=2463 a číslo F6 uloží na adresu 4000.

Podobné probíhají i ostatní příkazy - např. BEEP t,f.

A teď už k samotným operacím kalkulačky. Operace budou popisovány od nejjednodušších ku složitějším, protože složité operace mohou využívat jednodušších.

-----  
offset: 02            delete  
33A1                    vymazání čísla

Vstup : počet čísel z. k.

Výstup: počet čísel z. k. méně jedno číslo

Mění : vypouští poslední číslo z. k.

- - -

Operace '02-delete' je tvořena pouze jedinou instrukcí, a to RET. Proč je tedy výsledkem rutiny vymazání posledního čísla ?

Offset 02 patří ke skupině binárních operací. Proto je DE = adresa posledního čísla [(STKEND)-5] a HL = adresa předposledního čísla [(STKEND)-10]. Zavoláním operace 'delete' dojde k návratu (RET) a podle adresy ze sklíčku skočí  $\mu P$ . na adresu '3365, RE-ENTRY' ( v našem označení do bodu D ). První instrukce nastaví (STKEND) podle hodnoty v DE a protože původní hodnota v DE = (STKEND)-5 je tímto krokem i vrchol z. k. zmenšen o 5 (jedno číslo). Uzhledem k nové hodnotě (STKEND) je nyní DE = (STKEND) a HL = (STKEND)-5, což je standardní naplnění registrů DE a HL před načtením nového offsetu (např. jako v bodě A).

Všechny ukazatele HL, DE i (STKEND) byly zmenšeny oproti povodnímu stavu o pět, čímž je poslední číslo vymazáno (přeskočeno ukazateli) a kalkulačka je již nedostupná !!!

Avšak v paměti počítače zůstává v nezměněné podobě; podobně je tomu u vybrání čísla ze zásobníku  $\mu P$ . instrukcí POP - i zde je číslo vybráno a už jej nelze použít, protože jakoukoliv instrukcí PUSH nebo CALL je zničeno zcela, ale v případě nutnosti ho lze do zásobníku vrátit instrukcemi DEC SP, DEC SP.

Upozornění:

Zápis o rutině "Vstup, Výstup a Mění" je zde zcela formální a nejsou zde zaznamenány změny registrů  $\mu P$ , protože jsou pro nás nepodstatné. zaznamenány jsou změny týkající se kalkulačky samotné. Je-li přesto potřeba zjistit stav některého z registrů  $\mu P$ , pak před provedením offsetu jsou registry naplněny jako výstup rutiny '3365, CALCULATE' (popsáno dříve) a po provedení rutiny se ukazatele HL, DE a (STKEND) zvětší (resp. zmenší) o pět a v z. k. je nyní o jedno číslo více (resp. méně). Ostatní registry jsou následující smyčkou kalkulačky přemazány, takže většinou nenabývají velkého významu.

---

```
offset: 31    duplicate
          33C0 CALL          kopie čísla
```

```
Ustup : z. k.
Výstup: nové číslo do z. k. (kopie posl. č.)
Mění : -
- - -
```

Rutina je složena ze tří instrukcí:

```
33C0 MOVE-PP    CALL 33A9,TEST-5-SP    ; BC=5,HL & DE standard
                LDIR                  ; (STKEND)=>(<STKEND)-5) 5x
                RET                   ; HL=HL+5, DE=DE+5
```

Tím se vytvoří nové číslo, které je kopií posledního čísla v z. k. a současně se aktualizují ukazatelé DE a HL.

Pozn.: Rutinu lze volat i obecně: CALL 33C0,MOVE-PP - výsledkem je potom přenesení pěti bajtů z adresy HL na adresu DE po předchozím provedení testu místa v z. k..

---

```
offset: 01    exchange
          343C CALL          výměna posledních dvou čísel
```

```
Ustup : z. k.
Výstup: záměna posledních dvou čísel v z. k.
Mění:-
- - -
```

Rutina je binární operací, proto je DE = adr. posl. čísla a HL = adr. předp. čísla. Obsahy 5 bajtů z adres HL a DE se vymění. HL a DE se tímto postupem zvětší o 5 - což je opět standardní hodnota - počet čísel ve sklípku zůstává stejný.

Pozn.: Rutinu lze volat i obecně: CALL 343E,SWAP-BYTE. Ustupem je reg. B - počet vyměňovaných bajtů. Rutina vymění B bajtů z adres HL a DE. Po provedení je HL i DE zvětšeno o B a je-li B sudé jsou hodnoty HL s DE zaměněny !!!

---

Před popisováním offsetů pracujících s pamětí kalkulačky je nutno popsat následující rutinu:

---

```
3406 LOC-MEM
CALL    vyhledání v paměti
```

```
Ustup : HL = adresa paměťového pole kalkulačky
        A = pořadí čísla v paměťovém poli kalk.
Výstup: HL = adresa A-tého čísla v paměťovém poli kalk.
Mění : A, C, B, příznaky (A=A*5, C=A, B=00)
- - -
```

Hledáme-li A-té číslo od adresy HL, pak jeho adresa je "HL=HL + 5\*A".

---

```
offsety: C0-C5  st-mem
342D          uložení čísla z. k. do registru kalk.
```

```
Ustup : číslo na vrcholu z. k.
Výstup: číslo v reg. kalk. 0 až 5
Mění : -      (číslo v z. k. zůstává zachováno)
- - -
```

Rutina kopíruje číslo na vrcholu z. k. do reg. kalk. daného offsetem. Při volání této operace (skupinové) je v bitech D4 až D0 reg. A uchován offset pro volbu reg. kalk.

Je-li oblast registrů kalk. definovaná s. p. MEM standardně na adresu 5C92 [23698] - MEHBOT, pak smíme používat pouze registry 0 až 5 (offsety C0 až C5).

Předdefinujeme-li začátek registrů kalkulačky změnou s. p. MEM do volné paměti můžeme používat plný rozsah registrů 00 až IF [0 až 31] s offsety C0 až DF (plný rozsah registrů zabere A0 [160] bajtů paměti!). Délka paměťového místa kalkulačky není ničím omezena a použijete-li ve standardním nastavení s. p. MEM např. offset C6 podaří se Vám tak 'elegantním' způsobem přepsat s. p. RANTOP, což je velmi hazardní krok.

---

```
offsety : E0-E5 get-mem
340F z registru kalk. na vrchol z. k.
```

```
Ustup : registr kalk.
Výstup: k z. k. přibude číslo z reg. kalk.
Mění : - (kopírovaný registr se nemění)
- - -
```

Rutina uloží číslo z registru kalk. (offset v A na bitech D4 až D0) na vrchol z. k. (nové číslo). Rozšíření registrů viz operace 'st-mem'.

---

```
offsety: A0-A4  stk-const
341B          uložení ROM konstanty do z. k.
```

Ustup : čísla definovaná v ROM na adrese 32C5 [12997] Výstup: k z. k. přibude ROM konstanta :

	offset	číslo	název	konstanty	def.b.
A0	0	(zero)	00 B0 00		
A1	1	(one)	40 B0 00 01		
A2	0.5	(a half)	30 00		
A3	PI/2	(a half of pi)	F1 49 0F DA A2		
A4	10	(ten)	40 B0 00 0A		

```
Mění : -
- - -
```

Rutina doplní z. k. některou ROM konstantou podle offsetu.

---

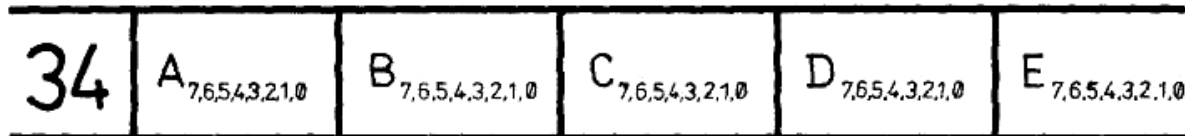
```
offset: 34      stk-data
330B          uložení vlastní konstanty do z. k.
```

```
Ustup : konstanta určená několika bajty za offsetem 34 [52]
Výstup: k z. k. přibude vlastní konstanta
Mění : HL=HL+5, DE=DE+5, B=0 ,H'L'
- - -
```

POZOR !!!  
=====

Konstanty takto zapisované (definované za offsetem 34 mají zcela jiný formát (viz 'stk-const'; z důvodu zkrácení zápisu). Označme bajty za offsetem 34 postupně písmeny A,B,C,D,... atd., a jejich bity indexované (A7,A6,...,A0,B7,B6,B5,...,B0,C7,C6,... atd.).

Obrázek 21 - indexované bajty za offsetem 34



Typ čísel INTEGER :

=====

bity A5,A4,A3,A2,A1,A0 jsou nulové !!!

Bity A7,A6 určují po zvětšení o jedničku počet definovaných bajtů, kromě bajtu B, které budou přesně zkopírovány (C,...) Zbylé, nedefinované bajty (do pěti) jsou nuly.

Bajt B je zvětšen o 50 [80] a tvoří první bajt čísla. Protože je u čísel typu INTEGER první bajt nulový musí být bajt B vždy roven B0 [176].

Příklady:

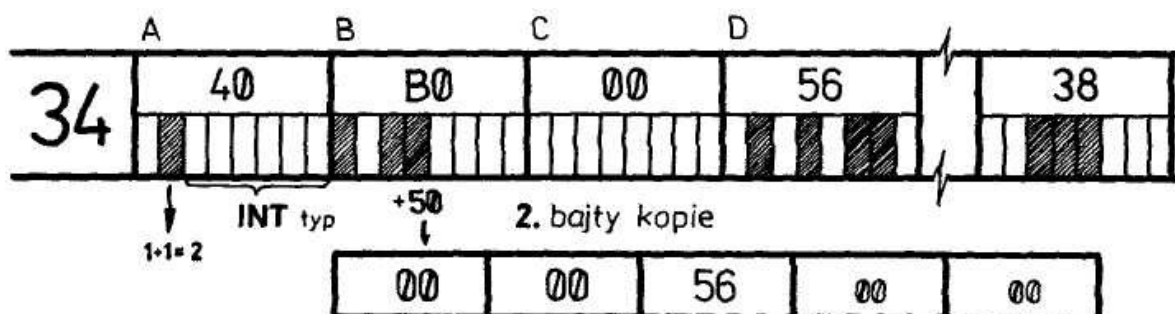
a) zapsání konstanty 0056 [86] do z. k.

```

RST 0028,FP-CALC ;
... ;
DEFB 34,stk-data ;
DEFB 40 ; bajt A - 1 číslo COPY
DEFB B0 ; bajt B
DEFB 00 ; znaménko
DEFB 56 ; nižší bajt
DEFB . . ; další offset kalk.
DEFB 38,end-calc ;

```

Obrázek 22 - zkrácený zápis čísla INT 86





b) zapsání konstanty 3039 [12345] do z. k.

```

RST 0028,FP-CALC ; z. k. = x,y
...
DEFB 34,stk-data ;
DEFB 80          ; bajt A - 3 čísla COPY
DEFB B0          ;                                     (C,D,E)
DEFB 00          ; znaménko
DEFB 39          ; low
DEFB 30          ; hi , z. k. = x,y,3039
DEFB ..          ; další offset
DEFB 38,end-calc ;

```

Pozn.: Pokuste se nyní zapsat pomocí def. bajtů za offsetem 34 konstanty - 0, 1, 10. Řešení najdete u offsetu A0, 'stk-const'.

Typ čísel REAL :

=====

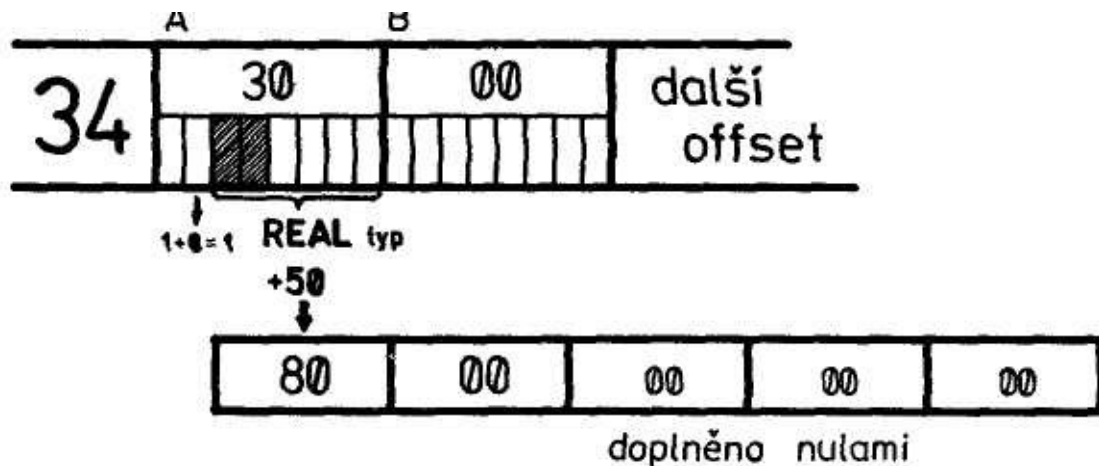
bity A5,A4,A3,A2,A1,A0 nejsou nulové !!!

Bajt A ve tvaru 0,0,A5,A4,A5,A2,A1,A0 je zvětšen o 50 [80] a tvoří první bajt čísla kalkulačky (exponent). Bity A7,A6 po přičtení jedničky určují počet definovaných bajtů které ještě vyjadřují zadávanou konstantu a budou přímo zkopírovány (bajty B,C,...).

Zbylé nedefinované bajty (do pěti) jsou automaticky nulovány.

Příklady:

a) a) zapsání čísla 0.5 {DEFB 34,30,00} Obrázek 23t zkrácený zápis čísla REAL 0.5



b) jak zapíšete číslo 40.4762 jako definovanou konstantu ?  
(výsledek DEFB 34, 76, 21, E1)

Pro další popis instrukcí je nutno vysvětlit následující podprogramy :

-----  
34E9 TEST-ZERO

CALL test je-li číslo nula

Ustup : číslo kalk. na adrese HL

Uýstup: příznak C - číslo je nula

NC - číslo není nula

Mění : A, příznaky

- - -

Podprogram testuje číslo kalkulačky začínající na adrese HL.

-----  
350B FP-0/1

CALL nula nebo jedna

Ustup : příznak Carry

Uýstup: od adresy HL vytvoří číslo kalkulačky!

00,00,00,00,00 je-li příznak NC 00,00,01,00,00 je-li příznak C

Mění : A=00, příznaky kromě carry

- - -

Podprogram vytvoří od adresy HL číslo rovné příznaku Carry.

=====

offset: 30 not

3501 logická negace

Ustup : číslo v z. k.

Uýstup: logická negace posledního čísla v z. k.

Mění : -

- - -

Podprogram je vytvořen takto:

3501 NOT CALL 34E9,TEST-ZERO ;test nuly => Carry

JR 350B,FP-0/1 ;Carry => číslo do z. k.

To znamená, že pro nulu je výsledkem jednička a pro všechna ostatní čísla různá od nuly je výsledkem nula.

-----  
offset: 36 less-0

3506 menší než nula

Ustup : číslo v z. k.

Uýstup: do z. k. číslo 0 je-li číslo nezáporné

1 je-li číslo záporné

Mění : testované číslo v z. k.

- - -

Do Carry se uloží 7 bit druhého bajtu čísla (znaménko čísla typu INTEGER i REAL) a pokračuje se rutinou FP-0/1. Tím je poslední číslo v z. k. rovno jedné - bylo-li záporné, nebo rovno nule - bylo-li kladné či nula.

---

```
offset: 37      greater-0
34F9           větší než nula
```

```
Ustup : číslo v z. k.
Výstup: číslo      0 pro všechna nekladná čísla
          1 pro všechna kladná čísla
Mění : testované číslo
- - -
```

Tato rutina testuje číslo v z. k. a výsledkem je číslo 1 v z. k. jestliže bylo testované číslo kladné nebo nula, bylo-li číslo v z. k. záporné je výsledkem číslo nula v z. k.. Podprogram využívá rutiny: '34E9, TEST-ZERO', '3507, LESS-0', '350B, FP-0/1'.

---

```
offset: 1B      negate
346E           číselné minus
```

```
Ustup : číslo v z. k.
Výstup: číslo v z. k. s opačným znaménkem
Mění : -
- - -
```

Neguje číslo v zásobníku. Je-li číslo INTEGER pak využívá podprogramů '2D7F, INT-FETCH' a '2D8E, INT-STORE'.

---

```
offset: 2A      abs
346A           absolutní hodnota
```

```
Ustup : číslo v z. k.
Výstup: absolutní hodnota čísla v z. k.
Mění : -
- - -
```

Z čísla v zásobníku je vytvořena jeho absolutní hodnota.

Při psaní programů pro kalkulačku je výhodné místo poznámek přímo znázornit zásobník. Znázornění zásobníku Může pro již probrané operace vypadat například takto:

```
{ Ustup = x,y }
```

```
DEMO  RST  0028,FP-CALC ;x,y
      DEFB C0,st-mem-0  ;x,y          (mem-0=y)
      DEFB 01,exchange ;y,x
      DEFB A4,stk-ten   ;y,x,0A
      DEFB 01,exchange ;y,0A,x
      DEFB 31,duplicate ;y,0A,x,x
      DEFB 02,delete    ;y,0A,x
      DEFB E0,get-Mem-0 ;y,0A,x,y
      DEFB ..           ;
      DEFB 38,end-calc  ;
```

---

```
offset: 3A      truncate
3214          celá část bližší nule
```

```
Ustup : číslo v z. k.
Výstup: číslo bližší nule (viz dále)
Mění : argument funkce
- - -
```

Celá část čísla bližší nule je číslo, které vznikne odtržením všech desetinných míst bez ohledu na znaménko. Např.:

```
| (3.5) = 3      | (-2.4) = -2      atd.
```

P O Z O R ! ! !

----- U této rutině je softwarová chyba !!

Pro argument -65536 je chybný výsledek -1 ! Tato chyba ovlivňuje i funkci INT BASICového interpretru. Zkuste:

```
'PRINT INT (-65536)'
```

---

```
offset: 27      int
36AF          celá část čísla
```

```
Ustup : číslo v z. k.
Výstup: celá část čísla
Mění : argument funkce, registr mem-0
- - -
```

Nechť  $x$  je libovolné číslo, pak celou částí čísla  $x$  rozumíme celé číslo  $n$  pro které platí :

$$n \leq x < n+1.$$

Podprogram využívá rutinu ' 3214 , TRUNCATE ' a rutinu '368F, JUMP-TRUE', která bude vysvětlena později.

---

```
offset: 29      sgn
3492          znaménko
```

```
Ustup : číslo v z. k.
Výstup: znaménko - číslo v z. k.      -1 pro záporná čísla
                                         0 pro nulu
                                         1 pro kladná čísla
```

```
Mění : testované číslo
- - -
```

Rutina vykonává funkci známou z BASICu. 'LET x = SGN x'.

---

```
offset: 2B      peek
34AC CALL      PEEK
```

Ustup : adresa uložená v zásobníku Výstup: číslo v z. k. ( = PEEK adresa )

```
Mění : argument funkce
- - -
```

Protože již známe rutiny zde využitě, uvádíme výpis rutiny:

```
34AC peek CALL 1E99,FIND-INT2      ;
LD A,(BC)                        ;
JP 2D28,STACK-A                  ;RET přes rutinu STACK-A
```

Do BC se uloží adresa vybraná z kalkulačky, provede se PEEK assemblerovskou instrukcí 'LD A,(BC)' a výsledek se opět uloží do zásobníku kalkulačky. Rutinu lze volat i příkazem  
CALL 34AC,peek.

-----  
offset: 2C            in  
34A5 CALL            IN

Ustup : adresa uložení v z. k.  
Výstup: Číslo v z. k. (IN adr.)  
Mění : argument funkce  
- - -

Rutina pracuje zcela analogicky s rutinou PEEK. Ze z. k. je vybrána adresa, provede se IN a výsledek je opět uložen do z. k.. Rutinu lze volat i příkazem 'CALL 34A5'.

-----  
offset:        2D usr-no  
34B3            spuštění strojového kódu

Ustup : adresa uložená v z. k.  
Výstup: číslo kalkulačky = BC  
Mění : HL=2D2B  
- - -

Pro důležitost této rutiny přinášíme její výpis s podrobnějším popisem:

34B3	usr-no	CALL 1E99,FIND-INT2	;BC= adr. assembleru
		LD HL,2D2B	;2D2B to je STACK-BC
		PUSH HL	;návrat přes STACK-BC
		PUSH BC	;nepřímý skok na adresu
		RET	;uschovanou ve sklípku (BC)

Před samotným spuštěním assembleru se do zásobníku  $\mu P$ . uloží adresa rutiny '2D2B, STACK-BC aby mohl být předán obsah registru BC. Podprogram 'STACK-BC' uloží BC do z. k. a provede inicializaci některých ukazatelů (viz popis). U průběhu činnosti Vašeho programu v assembleru nesmí být změněn reg. H'L', kde je uložena návratová adresa (případně obsah uschován a vrácen {PUSH, POP}).

Voláme-li tedy z BASICU nějaký assemblerovský program, pak na jeho počátku jsou registry naplněny takto:

A	= ???
BC	= adresa začátku assembleru (výhodné !!)
DE	= (STKEND)
HL	= (STKEND) -5
B'C'	= ???
D'E'	= ???
H'L'	= návratová adresa systému
IX	= ???
IY	= 5C3A
(SP)	= 2D2B 'STACK-BC'
(SP)'	= 3365 'RE-ENTRY' ;druhá hodnota sklípku

---

```
offset: 33      jump
3686           skok
```

Ustup : definovaný bajt za offsetem 33

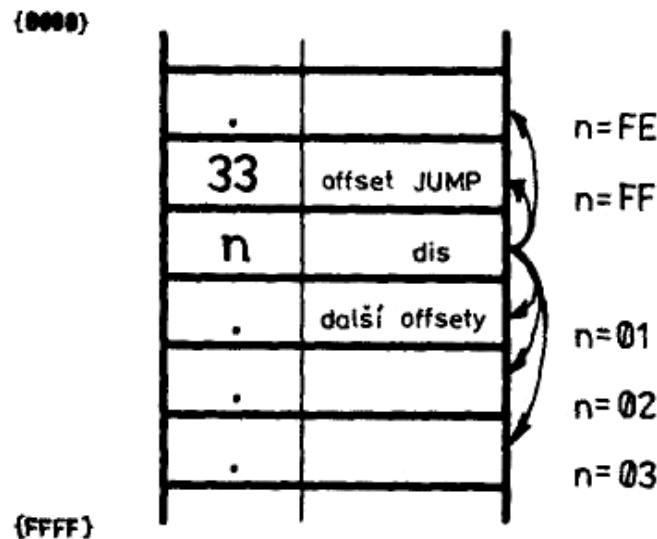
Ústup: relativní skok v offsetech

Mění : -

- - -

Rutina připočte k H'L' (čítač offsetů) číslo následující za offsetem 33, čímž se provede skok (stejně jako instrukce relativního skoku v assembleru 'JR dis') - viz obrázek 24.

Obrázek 24- relativní skok JUMP




---

```
offset: 00      jump-true
368F           podmíněný skok
```

Ustup : číslo v z. k. (0/1)

definovaný bajt za offsetem 00

Ústup: skok jestliže je číslo v z. k. různé od nuly  
přeskočení parametru skoku je-li číslo z. k. nula

Mění : rozhodovací číslo z. k. (0/1)

- - -

Protože offset 00 patří do skupiny binárních operací je poslední číslo v z. k. (0/1) návratem vymazáno. Je-li toto rozhodovací číslo v z. k. různé od nuly provede se skok daný následujícím parametrem (shodně jako offset 33). V opačném případě je parametr skoku přeskočen (INC H'L') a načten následující offset.

---

```
offset: 35      dec-jr-nz
367A           podmíněná smyčka
```

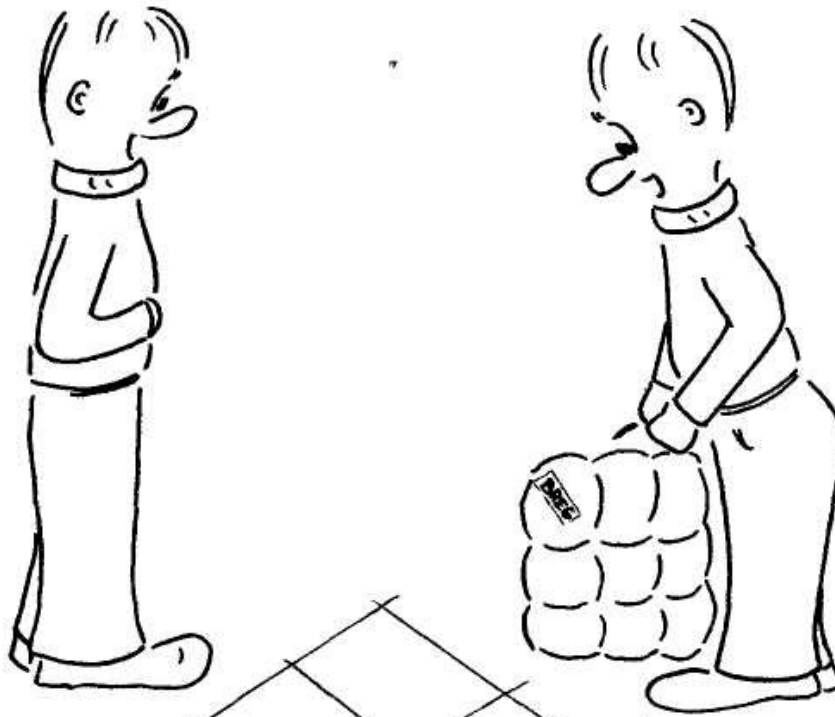
Ustup : s.p. (BREG) - počet cyklů  
           definovaný bajt za offsetem 35

Výstup: je-li (BREG) různý od nuly provede se skok  
           je-li (BREG) roven nule parametr skoku se ignoruje

Mění : s. p. (BREG) = (BREG) - 1

- - -

Tato operace je používána v podprogramu 'SERIES-GENERATOR', který bude popsán později. U případě obsahu s. p. (BREG) různého od nuly se provede skok popsáný v offsetu 33.



KARLE, JESTLI NEJSI „NULA“, HOĎ TEN BALÍK NA MOJÍ ADRESU.

---

```
offset: 07      or
351B          x OR y
```

Ustup : čísla x,y v z. k. (x je na vrcholu z. k.)

Uýstup: číslo x OR y

Mění : čísla x,y v z. k.

- - -

Rutina vytvoří operaci OR podle tabulky:

Tabulka 10 - operace OR

X	Y	X OR Y
0	0	Y
0	1	Y
1	0	číslo 1
1	1	číslo 1

0 - značí číslo 0  
1 - značí číslo  
různé od nuly

Podprogram pracuje takto: pomocí '34E9, TEST-ZERO', testuje poslední číslo (tj. x) a je-li rovno nule vrací se rutina s výsledkem B. Je-li ale různé od nuly je nastaven příznak Carry (SCF) a volán podprogram '350B, FP-0/1', který vytvoří číslo 1.

Pozn.: Jestliže pracujeme pouze s nulami a jedničkami pak je operace zcela shodná s operací OR Booleho algebry.

---

```
offset: 08      no-&-no
3524          x AND y
```

Ustup : čísla x,y v z. k. (x poslední)

Uýstup: číslo x AND y

Mění : čísla x,y ze zásobníku

- - -

Podprogram vytvoří operaci x AND B podle tabulky:

Tabulka 11 - operace AND

x	y	x AND y
0	0	0
0	1	0
1	0	Y
1	1	Y

0 - značí číslo 0  
1 - značí číslo  
různé od nuly

Podprogram pracuje analogicky s rutinou x OR y, pomocí rutin '34E9, TEST-ZERO' a '350B, FP-0/1'.

Pozn.: Jestliže pracujeme pouze s nulami a jedničkami je operace zcela shodná s operací AND Booleho algebry.



-----  
 offset: 0F            addition  
 3014                    sečtení

Ustup : čísla x,y v z. k.

Výstup: číslo x+y

při přeplnění chyb. hláš. '6 Number too big'

Mění : čísla x,y ze z. k.

- - -

Po programové stránce je tento podprogram již poměrně složitý a tak se spokojme s tím, že podprogram sečte poslední čísla v z. k., obě čísla vymaže a výsledek uloží opět do z. k.

-----  
 offset: 03            subtract  
 300F                    odečtení

Ustup : čísla x,y v z. k. (x je poslední)

Výstup: číslo y-x

Mění : čísla x,y ze z. k.

- - -

Podprogram poslední číslo zneguje (offset 1B, negate) a pak obě čísla sečte (offset 0F, addition).

-----  
 Dále popsaná operace násobení a dělení využívají těchto rutin :

-----  
 30A9 HL=HL\*DE  
 CALL násobení 16 x 16 bit

Ustup : HL, DE

Výstup: HL=HL\*DE

příznak NC - v pořádku

C - HL je přeplněno (výsledek příliš velký)

Mění : DE, A, příznaky

- - -

Použití této rutiny je zřejmé : do registrového páru je uložen výsledek součinu HL \* DE.

-----  
 30C0        PREP-M/D  
 CALL odstranění nuly

Ustup : Číslo v z. k.

Výstup: příznak    C - Číslo je nula

NC - READY

A - XOR s 2.bajtem Čísla (znaménko)

7.bit znaménkového bajtu je nastaven

Mění : A, příznaky

-----  
2FBA FETCH-TWO

CALL dvě čísla do registrů

Vstup : dvě čísla kalk. na adrese HL (a HL+5)

Výstup: H',B',C',C, B - první číslo kalk.

L',D',E',D,E - druhé číslo kalk.

Mění : - (A, HL i příznaky uchovány)

=====

offset: 04 multiply

30CA násobení

Vstup : čísla x,y v z. k.

Výstup: číslo x\*y v z. k.

při přeplnění chyb. hlášení '6 Number too big'

Mění : čísla x,y ze zásobníku

-----

offset: 05 division

31AF dělení

Vstup : čísla x,y v z. k. (x je na vrcholu)

Výstup: číslo y/x v z. k.

je-li x=0 nastává chyba '6 Number too big'

Mění : čísla x,y ze z. k.

- - -

Obě operace násobení i dělení probíhají následovně: čísla x a y jsou vyzvednuta ze z. k. (smazána), provedena operace a její výsledek je uložen opět do zásobníku.

-----

offset: 3D re-stack

3297 INTEGER -&gt; REAL

Vstup : číslo kalkulačky na adrese HL

Výstup: číslo je převedeno na tvar REAL

Mění : -

- - -

Je-li číslo typu REAL, rutina je ukončena. Je-li typu INTEGER je načteno do DE (pomocí rutiny '2D7F, INT-FETCH') upraveno na tvar REAL a vráceno na adresu HL.

-----

offset: 86,88,8C series

3449 Čebyševův polynom

Vstup : číslo kalkulačky, koeficienty polynomu

Výstup: hodnota Čebyševova polynomu

Mění : mem-0,1,2

- - -

Tyto offsety patří mezi skupinové operace. U ROM - monitoru se vyskytují pouze offsety 86, 88, 8C. Tuto rutinu používají podprogramy pro výpočet funkcí: SIN, ATN, LN, EXP a funkce od nich odvozené COS, TAN, ASN, ACS, ↑ a SQR.

Čebyševův polynom je postupně generován pro n=1, 2, ... rekurentním vztahem  $T_{n+1}(z) = 2zT_n(z) - T_n(z) - T_{n-1}(z)$ , kde  $T_n(z)$  je Čebyševův polynom n-tého stupně a jednotlivé

funkce mají 'n' rovno 6 pro SIN, 8 pro EXP a 12 dek. pro LN a ATN. Přesný matematický popis najdete v knize "Handbook of Mathematical Functions" (M. Abramowitz, I. A. Stegun - Dover 1965, str. 795).

Práce generátoru je též ukázána v knize "The Complete Spectrum ROM Disassembly" (Dr. Ian Logan, Dr. Frank O'Hara na str. 222).

Např. ve výpočtu funkce SIN je použit tento 'SERIES GENERÁTOR'

```
37BE      DEFB 86,series 06
          1.  DEFB 14,exponent 64
              DEFB E6,(00,00,00)
          2.  DEFB 5C,exponent 6C
              DEFB 1F,0B,(00,00)
              ..
              ..
              ..
          6.  DEFB F1,exponent 81
              DEFB 23,50,1B,EA
              ..
```

Přibližně (zjednodušeně lze vysvětlit práci této rutiny tak, že dané funkce (např. SIN) je nahrazena algebraickým polynomem n-tého stupně, jemuž zadáváme koeficienty definovanými bajty za voláním rutiny 'SERIES GENERATOR'. Stupeň polynomu (počet koeficientů) je určen bity A5 až A0 z offsetu generátoru. Je-li offset

```
86 = 6 dat, 6. stupeň
88 = 8 dat, 8. stupeň
8C = 12 dat, 12. stupeň
```

```
-----
offset: 39      get-argt
3783           redukce argumentu SIN a COS
```

Vstup : argument funkce SIN nebo COS v z. k.  
 Výstup: redukce argumentu do daného intervalu  
           mem-0 = výsledek testu 'greater-0'  
 Mění : mem-0

- - -

U mem-0 je uschován výsledek testu 'greater-0' pro hodnotu  $\text{ABS}(4*Y)-1$ , který je využit rutinou 'cos'.

Je zredukována osa x. Interval  $\langle -\pi, \pi \rangle$  je zredukován na interval  $\langle -0.5; 0.5 \rangle$  pro výpočet čebyševovým polynomem. Hodnota funkce je upravována tak dlouho, dokud není v potřebném intervalu.

Např.:         $\text{SIN } 720 = \text{SIN } 360 = \text{SIN } 0$   
                $\text{SIN } 5/2\pi = \text{SIN } \pi/2$   
                $\text{SIN } 1.5 = \text{SIN } 0.5$  (!!! v jednotkách  
   pro Čebyševův polynom)

Takže výsledkem (redukci argumentu x) je číslo U takové, že :

a)  $U=4*Y$         jestliže  $-1 \leq 4*Y \leq 1$   
 b)  $U=2-4*Y$     jestliže  $1 < 4*Y < 2$   
 c)  $U=-4*Y-2$    jestliže  $-2 \leq 4*Y < -1$   
 , kde  $Y=X/\langle 2\pi \rangle - \text{INT}\langle X/\langle 2\pi \rangle \rangle + 0.5$

---

```
offset: 1F      sin
375B           sinus
```

Ustup : číslo x v z. k.

Ústup: číslo SIN x v z. k.

Mění : číslo x ze z. k. mem=0,1,2

- - -

Rutina vypočte sinus zadaného čísla pomocí Čebyševova polynomu.  
Používá 'get-argt' a 'series'.

---

```
offset: 20      cos
37AA           kosinus
```

Ustup : číslo x v z. k.

Ústup: číslo COS x v z. k.

Mění : číslo x ze z. k. mem=0,1,2

- - -

Rutina vypočte kosinus zadaného čísla v zásobníku pomocí  
vztahu  $\cos x = \sin(x + \pi/2)$ . Používá rutinu SIN.

---

```
offset: 21      tan
37DA           tangens
```

Ustup : číslo x v z. k.

Ústup: číslo TAN x v z. k.

chyb. hlášení '6 Number too big', jestliže

číslo x není v rozsahu  $x \in (2k+1)\pi/2$

Mění : číslo x ze z. k. mem=0,1,2

- - -

Rutina vypočte tangens zadaného čísla pomocí vztahu  $\tan x = \sin x / \cos x$ .

Ukažme si postup výpočtu na výpisu:

```
37DA tan      RST 0028,FP-CALC           ;x
              DEFB 31,duplicate          ;x, x
              DEFB 1F,sin                 ;x, sin x
              DEFB 01,exchange            ;sin x, x
              DEFB 20,cos                  ;sin x, cos x
              DEFB 05,division ;sin x/cos x (= tan x)
              error: cos x = 0
              DEFB 38,end-calc ;
              RET                          ;konec rutiny
```

Postup výpočtu je zcela jasný i názorný.

---

```
offset: 24      atn
37E2           arkustangens
```

```
Ustup : číslo x v z. k.
Výstup: číslo ATN x v z. k.
Mění : číslo x ze z. k. mem-0,1,2
- - -
```

Rutina vypočítá ATN x pomocí Čebyševova polynomu. Používá se rutina '3297, RE-STACK'.

---

```
offset: 22      asn
3833           arkussinus
```

```
Ustup : číslo x v z. k.
Výstup: číslo ASN x v z. k.
Mění : číslo x ze z. k. mem-0,1,2
- - -
```

Rutina vypočítá ASN x pomocí vztahu mezi ASN x a ATN x.

---

```
offset: 23      acs
3843           arkuskosinus
```

```
Ustup : číslo x v z. k.
Výstup: číslo ACS x v z. k.
Mění : číslo x ze z. k. mem-0,1,2
- - -
```

Rutina vypočítá ACS x pomocí vztahu  $ACS\ x = \pi/2 - ASN\ x$

---

```
offset: 25      ln
3713           logaritmus
```

```
Ustup : číslo x v z. k.
Výstup: číslo LN x v z. k.
Mění : číslo x ze z. k. mem-0,1,2
- - -
```

Rutina vypočítá přirozený logaritmus pomocí Čebyševova polynomu. Používá rutinu '3297, RE-STACK'.

---

```
offset: 26      exp
36C4           exponencial
```

```
Ustup : číslo x v z. k.
Výstup: číslo EXP x v z. k.
Mění : číslo x ze z. k. mem-0,1,2,3
- - -
```

Rutina vypočítá exponenciálu pomocí Čebyševova polynomu. Používá rutinu '3297, RE-STACK'. {  $X=e^x$  }

---

```
offset: 06      to-power
3851           umocnění
```

Ustup : čísla x,y v z. k. (x je poslední číslo)

Ústup: číslo  $Y^x$  v z. k.

Mění : čísla x,y ze z. k. mem-0,1,2

- - -

Rutina vypočítá  $Y^x$  podle vztahu  $Y^x = e^{x \cdot \text{LN } Y}$

Pozn.: Tento vztah dokážeme snadno zlogaritmováním

$$x \cdot \text{LN } y = x \cdot \text{LN } y \cdot \text{LN } e \quad (\text{LN } e = 1)$$

---

```
offset: 28      sqr
384A           odmocnina
```

Ustup : číslo x v z. k.

Ústup: číslo  $\text{SQR } x$  v z. k.

Mění : číslo x ze z. k. mem-0,1,2

- - -

Rutina odmocní x s použitím vztahu  $\text{SQR } x = x \uparrow 0.5$  a použije rutinu '3851, TO-POWER'.

---

```
offset: 32      n-mod-m
38A0           celočíselné dělení
```

Ustup : čísla N,M (INTEGER) v z. k. (M je posl. č.)

Ústup: čísla  $N-M \cdot \text{INT}(N/M)$  a  $\text{INT}(N/M)$  na vrcholu? v z. k.

Mění : čísla N,M ze z. k. mem-0

- - -

Výsledkem celočíselného dělení jsou dvě čísla : celá část podílu a zbytek podílu.

---

```
offset: 3B      fp-calc-2
33A2           opakování operace
```

Ustup : (BREG)

Ústup: operace offsetu v (BREG)

Mění : -

- - -

Tato rutina provede operaci, jejíž offset je zapsán v registru (BREG). Tato rutina je volána pouze z adresy '2757, (SCANNING)'.

---

```

offset: 38 end-calc
369B ukončení kalkulačky
Vstup :-
Výstup: ukončení činnosti FP-CALC
návrát 'zpět', za offset 38
obnoví se obsah H'L' Mění : -

```

Tento offset ukončí práci kalkulačky a program pokračuje další instrukcí za posledním definovaným bajtem. Podívejme se blíže na ukončení práce kalkulačky :

```

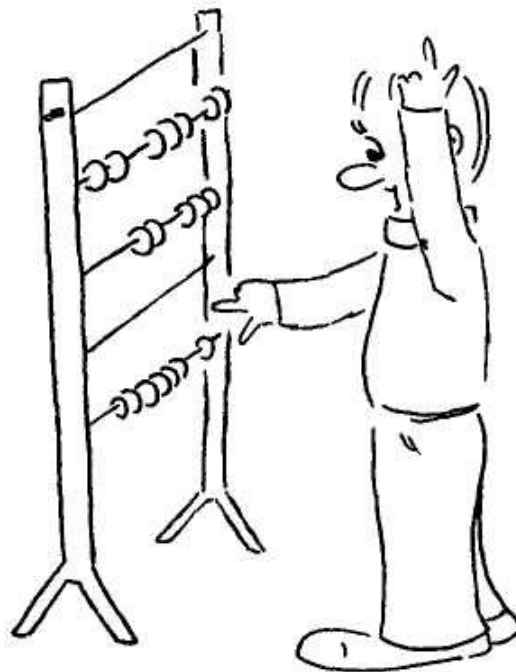
369B end-calc    POP    AF          ;znič návratovou adresu
                  EXX          ;'RE-ENTRY'
                  EX    (SP),HL    ;návrát H'L'a zpáteční adresy
                  EXX          ;
                  RET             ;skok zpět

```

Registry A, BC, DE, HL, B'C', D'E' odpovídají počátečnímu nastavení rutiny '335B, CALCULATE' při výstupu.

---

KOMUNIKACE S POČÍTAČEM 3,5 GENERACE  
TRIVIALNÍ !!  
JE ZCELA .



Příklad:

=====

Na závěr uvádíme program pro kalkulačku, který kreslí graf SINUS. Přibližně odpovídá programu v BASICu:

```
10 FOR a=0 TO 255 LET x=80*SIN(a/40.4762)+86
30 PLOT a,x
40 NEXT a
```

V assembleru je použita rutina '22E5, PLOT-SUB', která vykreslí bod o souřadnicích C,B (vysvětlení ve II.dílu).

Program 15 - sinusovka

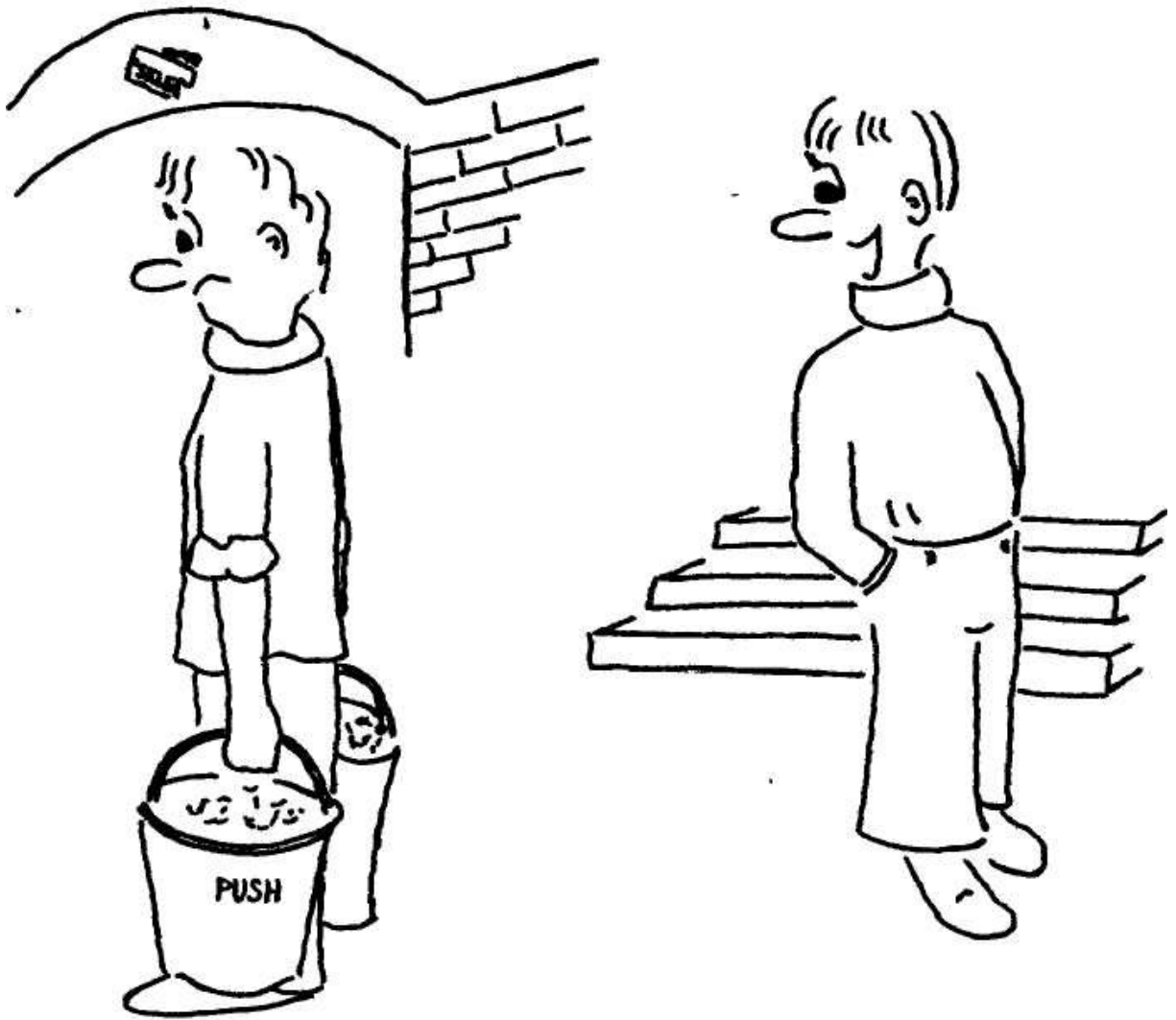
SINUS	XOR	A	;A začíná nulou	AF	[175	]
Loop	PUSH	AF		F5	[243	]
	CALL	2D28,STACK-A	;A do z. k.	CD282D	[205,40,43	]
	RST	0028,FP-CALC	;kalkulačka	EF	[239	]
	DEFB	34,stk-data	;konstanta	34	[52	]
exp	DEFB	76,21,E1	;40.4762 ->	7621E1	[118,33,225	]
	DEFB	05,division	; x/k	05	[5	]
fce	DEFB	1F,sin	; sin x/k ->	1F	[31	]
	DEFB	34,stk-data	;konst. 80	34	[52	]
	DEFB	40,B0,00,50		40B00050	[64,176,0,80]	
	DEFB	04,multiply	; 80*sin x/k	04	[4	]
	DEFB	34,stk-data		34	[52	]
	DEFB	40,B0,00,56	;80*sin x/k, 86	40B00056	[64,176,0,86]	
	DEFB	0F,addition	;80*sin x/k + 86	0F	[15	]
	DEFB	38,end-calc		38	[36	]
	CALL	1E94,FIND-INT-1;	(reg.A- z. k.	CD941E	[205,148,30	]
	LD	B,A	; kopie do B	47	[71	]
	POP	AF		F1	[241	]
	PUSH	AF	;A = x - souřadnice	F3	[245	]
	LD	C,A	;C = x - souřadnice	4F	[79	]
	CALL	22E5,PLOT-SUB;PLOT		CDE322	[205,229,34	]
	POP	AF		F1	[241	]
	INC	A	;další bod	3C	[60	]
	JP	NZ,Loop	;kreslí další bod	C202FF	[194,2,255	]
	RET		;konec -> BASIC	C9	[201	]

Program je umístěn na adrese FF00 [65280]. Je možné jej zkonstruovat mnoha způsoby (např. přes smyčku {offset 35 -dec-jr-nz}), toto je pouze jeden z možných způsobů.

Podle výpisu bude na obrazovce pouze jedna perioda funkce. Pro větší počet period stačí změnit exponent konstanty (návěští exp) na adrese FF07 [65287]. Zvětší-li se například o jedničku {POKE 65287,119} pak je exponent dvakrát větší, konstanta se zdvojnásobí a na obrazovku se zobrazí dvě periody funkce. Pokud Vás omrzí graf funkce sinus, nabízíme zcela neotřelý pohled na graf funkce kosinus změnou funkce ve výpočtu (druhé návěští fce {POKE 63291,32}).

Nepatrně rychleji bude rutina kreslit průběh funkce zakážeme-li přerušování <DI>. Samozřejmě je dobré přerušování na konci opět povolit (aby počítač nezapomněl brát v úvahu naše příkazy).





KARLE, ODNEŠ TO SVINŠTVO DO SKLÍPKU!

## 6 . R Ů Z N Ě =====

### R E S T A R T Y \*\*\*\*\*

RST 0028, FP-CALC již byl popsán.

RST 0008, ERROR-1

-----  
Tento restart je volán, nastane-li  
kdekoli v systému chyba.

Definovaným bajtem následujícím za RST 0008 je později určována,  
která z chyb nastala (viz tabulka 12).

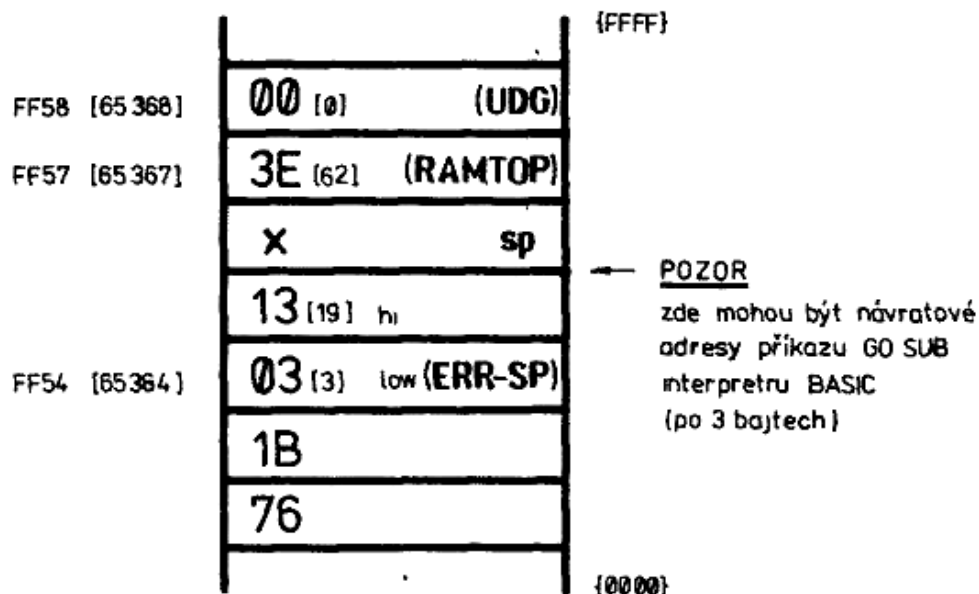
-----  
0008 ERROR-1

RST chybové hlášení

Provede: a) LD (X-PTR), (CH-ADD) ; (zde nastala chyba  
b) LD (STKBOT), (STKEND) ; vynulování kalkulačky  
c) LD (MEH), 5C92 ; standardní reg. kalk.  
d) LD (ERR-NR), kód chyby ;  
e) LD HL, (STKBOT) ;  
f) LD SP, (ERR-SP) a RET ; skok na ((ERR-SP))

Adresa (ERR-SP) je většinou o 3 Menší než (RAMTOP). Naplníme-li  
((ERR-SP)) adresou nějaké rutiny, pak je proveden 'skok' na tuto  
rutinu při každém chyb. hlášení (RST 0008).

Obrázek 25 - vrchol paměti pro systém (RAMTOP a ERR-SP)



Tabulka 12 - kódy chyb RST 0008

Adresa	Zpráva	Kód hlášení	Výskyt
1391	DEFB 80	- inicializační bajt	
1392	Report 0	FF OK.	1B50
1394	Report 1	00 NEXT without FOR	1DD8
13A4	Report 2	81 Variable not found	0670; 1C2E
13B6	Report 3	02 Subscript wrong	2A20
13C6	Report 4	03 Out of memory	! 1F15
13D2	Report 5	04 Out of Screen	0C86
13DF	Report 6	es Number too big	31AD; 3703
13ED	Report 7	86 RETURN without GO SUB	1F38
1401	Report 8	07 End of file	15E4
140C	Report 9	08 STOP statement	1CEE
141A	Report A	09 Invalid argument	34E7; 371A
142A	Report B	8A Integer out of range	046C; 1E9F; 24F9; 35DC
143E	Report C	0B Nonsense in BASIC	1C8A; 21CE
144F	Report D	0C BREAK - CONT repeats	0552; 0D00; 0F0A
1463	Report E	0D OUT of DATA	1E08
146E	Report F	0E Invalid file name	0642; 1765
147F	Report G	0F No room for line	! 1555
148F	Report H	10 STOP in INPUT	21D4
149C	Report I	11 FOR without NEXT	1D84
14AC	Report J	12 Invalid I/O device	15C4
14BE	Report K	13 Invalid colour	2244
14CC	Report L	14 BREAK into program	1B7B
14DE	Report M	15 RAMTOP no good	1EDA
14EC	Report N	16 Statement lost	1BEC
14FA	Report O	17 Invalid stream	160E; 1725
1508	Report P	18 FN without DEF	2812
1516	Report Q	19 Parameter error	188B
1525	Report R	1A Tape loading error	0806
1537		1B „_“	
1539		1C 1982 Sinclair Research Ltd (1295)	

Adresy 'výskytu' s vykřičníkem není možné použít přímo. Efektu nahrazení chybového hlášení vlastní ošetřující rutinou využívá většina programů typu: ON-ERROR-USR, ON-ERROR-GOTO, ON-BREAK-GOTO.

Popíšeme nyní program ON-ERROR-USR. Až nastane jakékoliv chyba, spustí se náš podprogram třeba na adrese nn, kterou musíme zapsat do s. p. ERR-SP:

Program 16 - inicializace ON-ERROR-USR

```

INIT    LD  DE,nn                ; adresa začátku chybové rutiny
        LD  HL,(5C3D)            ; HL=(ERR-SP)
        LD  (HL),E               ;
        INC HL                   ;
        LD  (HL),D               ;
        RET                     ; ((ERR-SP))=nn

```

Na adrese 'nn' umístíme podprogram, který může reagovat různě podle kódu chyby (tabulka 12) v s. p. ERR-NR, nebo bez ohledu na kód chyby počítač nastavit na počáteční stav (JP 0000).

Jak jsou tyto programy řešeny, najdete ve sbírce programů 'TOOLKIT - SUPERCODE'.

```
-----
0010      PRINT-A-1
RST          rutina otevřeného kanálu
```

Ustup : A - kód ASCII  
 Výstup: Obrazovka, tiskárna, ...  
 Mění : ...  
 - - -

Rutina zjistí a řízení předá otevřenému kanálovému přenosu, a je využívána příkazy : PRINT, LPRINT, LIST, LLIST, INPUT.

Všechny tyto rutiny můžeme nahradit rutinou vlastní, jejíž adresu vložíme do (CURCHL). Podrobný popis této rutiny vzhledem k rozsáhlosti popisu jednotlivých kanálů jsme nuceni 'odložit' do II. dílu.

Restarty 0018, GET-CHAR; 0020, NEXT-CHAR se vážou k výběru znaků pro restart 0010, PRINT-A-1 a budou popsány společně s tímto restartem.

```
=====
```

Ještě několik užitečných podprogramů, které nebyly popsány:

```
-----
0041      PO-SEARCH
CALL          vyhledání v tabulce
```

Ustup : DE - začátek tabulky  
           (první znak invertován, tzn.: + 80 [128])  
           A - pořadové číslo slova (00 až FF)  
 Výstup: DE = adresa začátku A-tého slova.  
 Mění : Příznaky  
 - - -

Podprogram vyhledá v tabulce od adresy DE (začínající invertovaným znakem) adresu A-tého slova. Každé slovo (výraz) musí končit invertovaným znakem. Například jsou to tabulky na adresách 1391 a 0095.

```
-----
16DC      INDEXER
CALL          prohledávání tabulky
```

Ustup : HL - adresa počátku dvoupoložkové tabulky  
           C - hledaná (první) položka  
 Výstup: příznak C - položka byla nalezena  
           HL - adresa nalezené (druhé) položky  
           NC - položka nebyla nalezena  
 Mění : Příznaky  
 - - -

V dvoupoložkové tabulce začínající na adrese HL prohledává jestli je první položka rovna registru C. Najde-li ji, pak je nastaven příznak Carry a v HL je adresa druhé položky. Prohledávání končí i když je konec tabulky, první položka je 88. Carry je nulován.

Příklad:

Obrázek 26 - dvoupoložková tabulka

(HL) →	07	02	(HL+01)
(HL+02)	71	03	(HL+03)
(HL+04)	FA	07	(HL+05)
(HL+06)	00	x	

---

```

19DD      DIFFER
      CALL      rozdíl

```

Ustup : HL, DE

Výstup: BC = HL - DE

Mění : vzájemná výměna hodnot HL,DE (EX DE,HL)

- - -

Rutina uloží do BC rozdíl (diferenci) mezi HL a DE.

---

```

2AEE      DE, (DE+1)
      CALL

```

Ustup : DE

Výstup: DE = (DE + 1)

Mění : HL = DE + 2 (původní vstupní obsah DE + 2)

- - -

Rutina naplní reg. pár DE číslem z adres DE+1 a DE+2 ( LD E, (DE+1) & LD D, (DE+2) ). HL je rovno původnímu obsahu DE zvětšenému o dvě.

---

```

2F8B      CA=10*A+C
      CALL

```

Ustup : A, C

Výstup: CA = 10 \* A + C

Mění : HL = CA

- - -

Rutina uloží do registrů CA desetinásobek A a přičte C.

---

---

```

2BF1      STK-FETCH
      CALL      číslo z. k. do registrů

```

Ustup : poslední číslo v z. k.  
 Výstup: přenos do registrů A,E,D,C,B  
 Mění : HL, (STKEND)=(STKEND)-5  
 - - -

Rutina načte číslo na vrcholu z. k. do registrů, takže v reg. A  
 nalezneme první bajt čísla ... atd. přesně podle popisu v  
 kapitole 5.-kalkulačka.

---

```

2D38      INT-TO-FP
      CALL      celé číslo ASCII do z. k.

```

Ustup : (CH-ADD) - adresa ASCII čísel  
           A = ASCII prvního čísla  
 Výstup: číslo INT v z. k.  
           příznak C  
 Mění : A, přízn., (CH-ADD) (+počet čísel), HL=(CH-ADD)  
 - - -

Rutina převede ASCII hodnotu celého čísla do z. k. pomocí rutiny  
 '2D22,STK-DIGIT' a pokračuje načtením ASCII z adresy podle s. p.  
 (CH-ADD). Jednotlivé číslice přičítá k součtu hodnoty, kterou  
 předtím násobí deseti. Je-li v registru A nenumerický ASCII znak,  
 rutina končí s příznakem Carry.

---

```

2BA6      L-ENTER
      CALL      řízený blokový přenos
      Ustup : HL - kam
              DE - odkud
              BC - délka (je-li BC=0 přenos neproběhne)
      Výstup: příznak NZ - BC>0, kopie bloku z DE na HL délky BC
              Z - BC=0, přenos neproběhne
      Mění : BC=0, příznak NZ - HL zůstane, DE=DE+BC (délka)
              Z - záměna DE a HL (EX DE,HL)
      - - -

```

Rutina provede blokový přenos, je-li BC>0 z adresy DE na adresu  
 HL o BC délce bajtů (zachová registr HL). Je-li BC=0 vrací se s  
 nastaveným příznakem Z a blokový přenos neprovede.

Příklad:

-----

Pokuste se vysvětlit následující efekt:

```

10 CLEAR 65000
20 FOR f=65280 TO 65292
30 READ x:POKE f,x
40 NEXT f
100 FOR f=0 TO 255
110 POKE 65285,f
120 RANDOMIZE USR 65280
130 NEXT f 200 FOR f=0 TO 255
210 POKE 65284,f
220 RANDOMIZE USR 65280
230 NEXT f 240 GOTO 100
1000 DATA 33,0,64,17,0,0,1,0,24,
      205,166,43,201

```

Tento program je zcela bezpečný a nemusíte se obávat hlášení výrobce počítače, kdy a kdo jej uvedl na trh.

## 7 . Z Á V Ě R E M ( E P I L O G )

=====

Závěrem se chceme omluvit všem čtenářům této publikace za případnou nesrozumitelnost výkladu činnosti rutin. Nejlépe je možné porozumět vysvětlovaným rutinám na příkladech. Z našeho hlediska dělíme čtenáře do tří kategorií:

a) začínající programátoři v assembleru Z80, kterým se tato publikace bude zdát příliš složitá. b) programátory "naší" úrovně, kteří tuto příručku berou jako opakování svých znalostí, c) geniální osobnosti, které tyto rutiny znají z paměti a "naše" příručka se jím zdá zcela triviální.

Cílem této příručky bylo popsat a vysvětlit práci základních rutin a umožnit jejich všeobecné (libovolné) použití. Nikoliv dát pouze strohý popis, co která rutina dělá. Nakolik byl tento záměr úspěšný posuďte sami.

Nechceme čtenáře lekat a proto až v závěru Vám sdělujeme, že tato příručka je jakousi 'bombou - průzkumnicí' a že podle ohlasu (kladného) připravíme díl druhý, který vedle slibů uvedených již v příručce by měl podat informaci o BASIC systému a INTERPRETRU.

Na konci této brožury najdete vzor objednáčního lístku na díl druhý, jehož vydání bychom chtěli připravit na únor 1988.

Rádi bychom posoudili Vaši pozornost a zaujetí nad příručkou, předkládáme Vám proto program v BASICu jehož neznámá funkce čeká na Vaše rozluštění. Napište nám jak program pracuje a využívá-li některých poznatků z příručky.

Dopisy s výsledky Vašeho zkoušení - bádání nebo s připomínkami k výsledku "naší" práce (slova díků či kritiky) očekáváme na adrese:

Značka "Rutiny I."

Poste restante, schránka 26, 182 00 Praha

Máme zájem o spolupráci se čtenáři kategorie " c " při přípravě II. dílu této potřebné příručky (jak si myslíme).

A u t o ř i



A zde již slíbený tajemný program:

```

10 CLEAR 65000
20 LET s=0
30 FOR f=65280 TO 65402
40 READ x
50 LET s=s+x
60 POKE f,x
70 NEXT f
80 IF s<>14987 THEN STOP
90 RANDOMIZE USR 65320
1000 DATA 127,7,74,127,19,82,191
1001 DATA 32,90,127,46,90,191,61
1002 DATA 98,127,77,98,253,93,98
1003 DATA 254,111,98,254,129,90
1004 DATA 253,149,82,254,170,82
1005 DATA 253,192,74,254,216,74
1006 DATA 0,243,62,255,14,254,30
1007 DATA 199,87,58,39,255,38
1008 DATA 255,111,126,50,89,255
1009 DATA 35,126,50,91,255,35
1010 DATA 126,50,95,255,35,62,39
1011 DATA 189,40,3,125,24,1,175
1012 DATA 30,39,255,122,205,88
1013 DATA 255,195,47,255,6,0,46
1014 DATA 0,237,80,203,0,192,99
1015 DATA 37,32,253,101,37,32
1016 DATA 253,230,239,237,121,99
1017 DATA 37,32,253,101,37,32
1018 DATA 253,246,16,237,121,24
1019 DATA 225

```

Tabulka 13 - adresný seznam rutin

Adresa	Název	Překlad	Volání	Kap.	Str.
0008	ERROR-1	chybové hlášení	RST	6	68
0010	PRINT-A-1	rutina otevřeného kanálu	RST	6	70
0028	FP-CALC	restart pro kalkulačku	RST	5	31
028E	KEY-SCAN	čtení klávesnice	CALL	1	4
031E	K-TEST	dekódování kódu klávesnice	CALL	1	6
03B5	BEEPER	tvorba tónu	CALL	2	9
04C2	SA-BYTES	uložení dat	CALL	3	15
053F	SA/LD-RET	ukončení SAVE a LOAD		3	14
0556	LD-BYTES	nahrání dat	CALL	3	16
05E3	LD-EDGE2	změří dobu dvou impulsů	CALL	3	19
05E7	LD-EDGE1	změří dobu impulsu	CALL	3	18
0602	LD-BLOCK	nahrání dat	CALL	3	18
0C41	PO-SEARCH	vyhledání v tabulce	CALL	6	70
0E68	CL-ATTR	najde adresu atributu	CALL	4	25
0E9B	CL-ADDR	najde adresu videobajtu	CALL	4	26
0E9E	CL-ADDRA	najde adresu videobajtu (BASIC)	CALL	4	26
16DC	INDEXER	prohledávání tabulky	CALL	6	70
19DD	DIF-FER	rozdíl	CALL	6	71
1E7A	OUT	OUT - příkaz BASICU	CALL	5	46
1E80	POKE	POKE - příkaz BASICU	CALL	5	46
1E85	TWO-PARAM	dvě čísla z. k. do A a BC	CALL	5	46
1E94	FIND-INT-1	číslo z. k. do A	CALL	5	45
1E99	FIND-INT-2	číslo z. k. do BC	CALL	5	46
1F05	TEST-ROOM	rozsah obsazené paměti	CALL	5	34
1F1A	FREE-MEM	obsazená paměť	CALL	5	35
1F54	BREAK-KEY	stav kláves "BREAK"	CALL	1	6
22AA	PIXEL-ADD	najde adresu videobitu (PLOT)	CALL	4	27
22B0	PIXEL-ADDA	najde adresu videobitu	CALL	4	27
22CB	POINT-SUB	příkaz POINT	CALL	4	28
2307	STK-TO-BC	dvě čísla z. k. do B a C	CALL	5	45
2314	STK-TO-A	číslo ze z. k. do A	CALL	5	45
2AB6	STK-STORE	číslo do kalk.	CALL	5	41
2AEE	DE, (DE+1)		CALL	6	71
2BA6	L-ENTER	řízený blokový přenos	CALL	6	72
2BF1	STK-FETCH	číslo ze z. k. do registrů	CALL	6	72
2C88	ALPHANUM	test čísla a písmene	CALL	1	6
2C8D	ALPHA	test písmeno	CALL	1	7
2D1B	NUMERIC	test čísla	CALL	1	7
2D22	STK-DIGIT	ASCII 0-9 do z. k. jako +INT	CALL	5	43
2D28	STACK-A	A do z. k. jako + INT	CALL	5	43
2D2B	STACK-BC	BC do z. k.	CALL	5	42
2D38	INT-TO-FP	celé ASCII číslo do z. k.	CALL	6	72
2D7F	INT-FETCH	č. kalk. INT do DE, C-znam.	CALL	5	43
2D8C	P-INT-STO	DE od HL číslo kalk. +INT	CALL	5	42
2D8E	INT-STORE	DE od HL číslo kalk. INT	CALL	5	41
2DA2	FP-TO-BC	kladné číslo ze z. k. do BC	CALL	5	43
2DD5	FP-TO-A	kladné číslo ze z. k. do A	CALL	5	44
2F8B	CA=10*A+C		CALL	6	71
2FBA	FETCH-TWO	natažení dvou čísel do reg.	CALL	5	60
300F	subtract	odečtení	03	5	59
3014	addition	sečtení	0F	5	59
30A9	HL=HL*DE	násobení 16x16 bit	CALL	5	59
30C0	PREP-M/D	odstranění nuly	CALL	5	59
30CA	multiply	násobení	04	5	60
31AF	division	dělení	05	5	60
3214	truncate	celá část bližší nule	3A	5	54

Adresa	Název	Překlad	(pokr.)	Volání	Kap.	Str.
3297	re-stack	INTEGER->REAL		3D	5	60
335B	CALCULATE	kalkulačka		RST,CALL	5	31
33A1	delete	vymazání posl. čísla z. k.		02	5	47
33A2	fp-calc-2	opakování operace		3B	5	64
33A9	TEST-S-SP	Místo pro číslo kalk.		CALL	5	35
33C0	MOVE-FP	přesun 5 bajtu v z. k.		CALL	5	48
33C0	duplicate	kopie čísla v z. k.		31	5	48
33C6	stk-data	vlastní konstanta do z. k.		34	5	49
3406	LOCK-MEM	vyhledání adresy čísla		CALL	5	48
340F	get-mem	reg. kalk. do z. k.		E0	5	49
341B	stk-const	kopie ROM konstant do z. k.		A0	5	49
342D	st-mem	číslo z. k. do reg. kalk.		C0	5	49
343C	exchange	výměna posl. dvou čísel z. k.		01	5	48
3449	series	Čebyševův polynom		86	5	60
346A	abs	absolutní hodnota		2A	5	53
346E	negate	číselné minus		1B	5	53
3492	sgn	znaménko		29	5	54
34A5	in	IN		CALL-2C	5	55
34AC	peek	PEEK		CALL-2B	5	54
34B3	usr-no	spuštění assembleru		2D	5	55
34E9	TEST-ZERO	test na nulu		CALL	5	52
34F9	greater-0	větší než nula		37	5	53
3501	not	logická negace		30	5	52
3506	less-0	menší než nula		36	5	52
350B	FP-0/1	nula nebo jedna		CALL	5	52
351B	or	x OR y		07	5	58
3524	no-&-no	x AND y		08	5	58
35BF	STK-PNTRS	nastavení ukaz. čísel v z. k.		CALL	5	37
367A	dec-jr-nz	DJNZ v offsetech		35	5	57
3686	jump	skok		33	5	56
368F	jump-true	podmíněný skok		00	5	56
369B	end-calc	konec offsetů		38	5	65
36A0	n-mod-m	celočíslné dělení		32	5	64
36AF	int	celá část čísla		27	5	54
36C4	exp	exponenciál		26	5	63
3713	ln	logaritmus		25	5	63
3783	get-argt	redukce argumentu SIN a COS		39	5	61
37AA	cos	kosinus		20	5	62
37B5	sin	sinus		1F	5	62
37DA	tan	tangens		21	5	62
37E2	atn	arkustangens		24	5	63
3833	asn	arkussinus		22	5	63
3843	acs	arkuskosinus		23	1	63
384A	sqr	odmocnina		28	5	64
3851	to-power	umocnění		06	5	64

Tabulka 14 - abecední seznam rutin

Adresa	Název	Překlad	Volání	Kap.	Str.
346A	abs	absolutní hodnota	2A	5	53
3843	acs	arkuskosinus	23	5	63
3014	addition	sečtení	0F	5	59
2C8D	ALPHA	test písmene	CALL	1	7
2C88	ALPHANUM	test čísla a písmene	CALL	1	6
3833	asn	arkussinus	22	5	63
37E2	atn	arkustangens	24	5	63
03B5	BEEPER	tvorba tonu	CALL	2	9
1F54	BREAK-KEY	stav kláves "BREAK"	CALL	1	6
2F8B	CA=10*A+C		CALL	6	71
335B	CALCULATE	kalkulačka	RST, CALL	5	31
0E88	CL-ATTR	najde adresu atributu	CALL	4	25
0E9B	CL-ADDR	najde adresu videobajtu	CALL	4	26
0E9E	CL-ADDRA	najde adresu videobajty (BASIC)	CALL	4	26
37AA	cos	kosinus	20	5	62
2AEE	DE, (DE+1)		CALL	6	71
367A	dec-jr-nz	DJNZ v offsetech	35	5	57
33A1	delete	vymazání posl. čísla z. k.	02	5	47
19DD	DIF-FER	rozdíl	CALL	6	71
31AF	division	dělení	05	5	60
33C0	duplicate	kopie čísla v z. k.	31	5	48
369B	end-calc	konec offsetů	38	5	65
0008	ERROR-1	chybové hlášení	RST	6	68
343C	exchange	výměna posl. dvou čísel v z. k.	01	5	48
36C4	exp	exponenciál	26	5	63
2FBA	FETCH-TWO	natažení dvou čísel do reg.	CALL	5	60
1E94	FIND-INT-1	číslo z. k. do A	CALL	5	45
1E99	FIND-INT-2	číslo z. k. do BC	CALL	5	46
350B	FP-0/1	nula nebo jedna	CALL	5	52
0028	FP-CALC	restart pro kalkulačku	RST	5	31
33A2	fp-calc-2	opakování operace	3B	5	64
2DD5	FP-TO-A	kladné číslo ze z. k. do A	CALL	5	44
2DA2	FP-TO-BC	kladné číslo ze z. k. do BC	CALL	5	43
1F1A	FREE-MEM	obsazená paměť	CALL	5	35
340F	get-mem	reg. kalk, do z. k.	E0	5	49
3783	get-argt	redukce argumentu SIN a COS	39	5	61
34F9	greater-0	větší než nula	37	5	53
30A9	HL=HL*DE	násobení 16x16 bit	CALL	5	59
34A5	in	IN	CALL-2C	5	55
16DC	INDEXER	prohledání tabulky	CALL	6	70
36AF	int	celá část čísla	27	5	54
2D7F	INT-FETCH	číslo z. k. INT do DE	CALL	5	43
2D38	INT-TO-FP	celé ASCII číslo do z. k	CALL	6	72
2D8E	INT-STORE	DE od HL jako INT kalk.	CALL	5	41
3686	jump	skok	33	5	56
368F	jump-true	podmíněný skok	00	5	56
031E	K-TEST	dekódování kódu klávesnice	CALL	1	6
028E	KEY-SCAN	čtení klávesnice	CALL	1	4
2BA6	L-ENTER	řízený blokový přenos	CALL	6	72
0802	LD-BLOCK	nahrání dat	CALL	3	18
0556	LD-BYTES	nahrání dat	CALL	3	16
05E7	LD-EDGE1	změří dobu impulsu	CALL	3	18
05E3	LD-EDGE2	změří dobu dvou impulsů	CALL	3	19
3506	less-0	menší než nula	36	5	52
3713	ln	logaritmus	25	5	63
3406	LOCK-MEM	vyhledání adresy čísla	CALL	5	48

Adresa	Název	Překlad (pokr.)	Volání	Kap.	Str.
33C0	MOVE-FP	přesun 5 bajtů v z. k.	CALL	5	48
30CA	Multiply	násobení	04	5	60
36A0	n-mod-m	celočíslné dělení	32	5	64
346E	negate	číselné Minus	1B	5	53
3524	no-&-no	x AND y	08	5	58
3501	Not	logická negace	30	5	52
2D1B	NUMERIC	test čísla	CALL	1	7
351B	or	x OR y	07	5	58
1E7A	OUT	OUT - příkaz BASICu	CALL	5	46
2D8C	P-INT-STO	DE +INT kalk. od HL	CALL	5	42
34AC	peek	PEEK	CALL-2B	5	54
22AA	PIXEL-ADD	najde adresu videobitu (PLOT)	CALL	4	27
22B0	PIXEL-ADDA	najde adresu videobitu	CALL	4	27
0C41	PO-SEARCH	vyhledání v tabulce	CALL	6	70
22CB	POINT-SUB	příkaz POINT	CALL	4	28
1E80	POKE	POKE - příkaz BASICu	CALL	5	46
30C0	PREP-M/D	odstranění nuly	CALL	5	59
0010	PRINT-A-1	rutina otevřeného kanálu	RST	6	70
3297	re-stack	INTEGER->REAL	3D	5	60
04C2	SA-BYTES	uložení dat	CALL	3	15
053F	SA/LD-RET	ukončení SAVE a LOAD		3	14
3449	series	Chebyshevův polynom	86	5	60
3492	sgn	znaménko	29	5	54
37B5	sin	sinus	1F	5	62
384A	sqr	odmocnina	28	5	64
342D	st-nem	Číslo ze z. k. do reg. kalk.	C0	5	49
2D28	STACK-A	A do z. k. jako + INT	CALL	5	43
2D2B	STACK-BC	BC do z. k. jako + INT	CALL	5	42
341B	stk-const	kopie ROM konstant do z. k.	A0	5	49
33C6	stk-data	vlastní konstanta do z. k.	34	5	49
2D22	STK-DIGIT	ASCII 0-9 do z. k. jako +INT	CALL	5	43
2BF1	STK-FETCH	číslo ze z. k. do reg. kalk.	CALL	6	72
35BF	STK-PNTRS	nastavení ukaz. čísel v z. k.	CALL	5	37
2AB6	STK-STORE	číslo do z. k.	CALL	5	41
2314	STK-TO-A	číslo ze z. k. do A	CALL	5	45
2307	STK-TO-BC	dvě čísla z. k. do B a C	CALL	5	45
300F	subtract	odečítání	03	5	59
37DA	tan	tangens	21	5	62
33A9	TEST-S-SP	místo pro číslo kalk.	CALL	5	35
1F05	TEST-ROOM	rozsah obsazené paměti	CALL	5	34
34E9	TEST-ZERO	test na nulu	CALL	5	52
3851	to-power	umocnění	06	5	64
3214	truncate	celá část bližší nule	3A	5	54
1E85	TWO-PARAM	dvě čísla z. k. do A a BC	CALL	5	46
34B3	usr-no	spuštění assembleru	2D	5	55

Tabulka 15 seznam programů

číslo	strana	název
1	3	náhodné číslo
2	4	statistika kódů
3	10	laser
4	10	key-bů
5	20	čtení vstupu 'EAR'
6	20	uložení obrazovky
7	20	nahrání obrazovky
8	20	nahrání libovolné standardní hlavičky
9	23	change adres
10	28	clear screen
11	29	print na pozici
12	29	plot na pozici
13	30	point pozice
14	30	rol 1 up li down
15	66	sinusovka
16	69	inicializace ON-ERROR-USR

Tabulka 16 - seznam obrázků

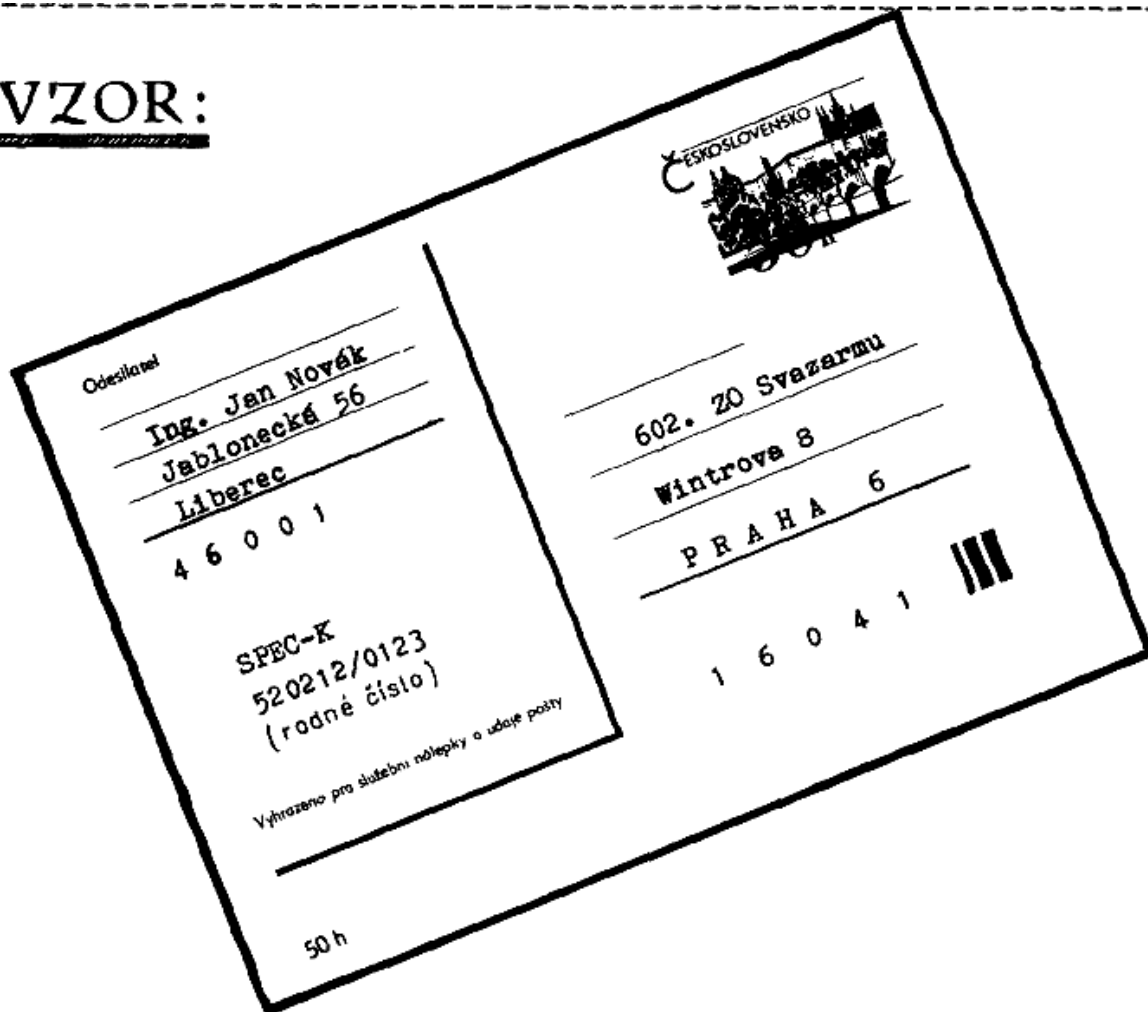
číslo	strana	název
1	2	formát informace o 'půlřádce'
2	8	stavové slovo -BEEP"
3	9	frekvence tonu řízená kroky $\mu P$
4	12	stavové slovo "SAVE"
5	13	stavové slovo "LOAD"
6	15	záznam bloku dat
7	19	registr C
8	19	rutiny LD-EDGE
9	22	obrazovka
10	23	adresování Sinclair a 'normální'
11	24	atributy
12	26	adresování řádků rutiny 'CL-ADDR'
13	26	" " " 'CL-ADDRA'
14	27	souřadnice x,y rutiny 'PIXEL-ADD'
15	27	" x,y včetně dialogových řádků
16	32	číslo typu REAL 0.125
17	32	" " " 0.6
18	33	" " " 0.5
19	34	zásobník kalkulačky
20	38	schéma práce kalkulačky
21	50	indexované bajty za offsetem 34
22	50	zkrácený zápis čísla INT 86
23	51	" " " REAL 0.5
24	56	relativní skok JUMP
25	68	vrchol paměti pro systém (RAMTOP)
26	71	dvoupoložková tabulka pro INDEXER

Tabulka 17 - převodní HEXA - DEK.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
	0	1	2	3	4	5	6	7	8	9	A	B	c	D	E	F

Tabulka 18 - seznam tabulek

číslo	strana	název
1	2	adresy kláves
2	5	kódy kláves
3	5	výstupní tvar 'KEY-SCAN'
4	12	ULA pin 28
5	13	rozlišení ISSUE 2 a 3
6	21	standardní hlavička
7	25	barvy
8	25	adresy obrazovky
9	36	offsety kalkulačky
10	58	operace OR
11	58	" AND
12	69	kódy chyb RST 0008
13	76	adresný seznam rutin
14	78	abecední " "
15	80	seznam programů
16	80	obrázků
17	81	převodní HEXA - DEK.
18	82	seznam tabulek

**VZOR:**





**Jan Šritter, Marcel Dauth**

**RUTINY ROM ZX SPECTRUM**

Vydala ZO Svazarmu č. 4006/602, Wintrova 8, 160 41 Praha 6  
pro potřeby vlastního aktivu v prosinci 1987

Zodpovědný redaktor PhDr. Jindřich Jirka  
Technická redaktorka Daniela Prokopová  
Obálku navrhl Josef Svoboda

Tisk Tiskařské závody, n. p. Praha, náklad 3000 výtisků

Povoleno MK ČSR pod číslem 59-201-87

**Cena Kčs 18,-**

