Programování ve strojním kódu pro Sinclair ZX Spectrum

3. díl

Obsah:

1. Na úvod

Již se znalostí strojních instrukcí Z80 je možno sestavovat plně fungující strojní programy. Někdy je však výhodné použít "polotovarů", které nám Spectrum nabízí jednak ve formě systémových proměnných, jednak ve formě podprogramů v ROM. Proto se v této části pokusím blíže vysvětlit využití hlavních systémových proměnných a některých podprogramů ROM. U systémových proměnných je to poměrně jednoduché, poněvadž jich není tolik. Obtížnější je to s podprogramy ROM, kterých je více a jsou složitější. Ten, kdo se chce vážně zabývat strojním programováním, by si měl bezpodmínečně opatřit a podrobně prostudovat některý z komentovaných výpisu ROM Spectra. Pro lepší představu: má asi 200 stran.

ZX Spectrum se vyrábí ve dvou verzích: se 16K RAM a s 48K RAM. Tato skutečnost však nemá vliv na str. programy. Pouze je možno ukládat str. programy na různé adresy. Všechny str. programy psané pro 16K verzi je možné bez jakýchkoliv úprav použít i pro 48K verzi, podstatně horší to bývá při pokusech přimět programy psané pro 48K verzi, aby fungovaly i na 16K verzi. Většinou nefungují, poněvadž nemají dostatek místa v paměti. Jejich drastické zkrácení je většinou krkolomná a nevděčná práce a výsledek obvykle neodpovídá vynaložené námaze. Krátké programy však lze upravit přeadresováním do jiného adresového prostoru.

Ještě než se pustíme do vlastní RAM a ROM, musíme se na chvíli vrátit k matematice, konkrétně ke způsobu, jakým Spectrum vyjadřuje číselné hodnoty.

2. Formy datových struktur

Každá číselná hodnota se dá vyjádřit více způsoby. Spectrum zná pouze 2 druhy čísel. Je to číslo typu integer t.j. celé číslo, a číslo typu floating point, t.j. číslo s pohyblivou řádovou čárkou. U čísel typu integer je to jednoduché. Jsou to čísla, se kterými se denně setkáváme: -23, 5, 1187, 23689 atp. Počítač je ukládá do jednotlivých pamětových míst ve formě 8 bitů, tvořících 1 bajt (jednotlivá pamětová místa jsou vlastně osmibitové registry schopné uchovávat hodnoty od 00000000B do 11111111B). Maximální hodnota kterou můžeme uložit do jednoho bajtu či pamětového místa je tudíž FFh = 255d. Větší čísla se ukládají do dvou bajtů v pořadí nižší bajt, vyšší bajt. U dvoubajtového čísla 5BFFh je nižším bajtem FFh a vyšším bajtem 5Bh. V tomto pořadí jsou také bajty uloženy do pamětových míst A a A+1. Maximální hodnota dvoubajtového čísla může být FFFFh = 65535d. Decimální hodnotu dvoubajtového čísla dostaneme tak, že k decimální hodnotě prvního bajtu přičteme 256-ti násobek decimální hodnoty druhého bajtu. Opačně t.j. při převádění decimální hodnoty čísla většího než 255, podělíme dané číslo 256, integer (t.j. celočíselnou část výsledku) uložíme do vyššího bajtu a zbytek čísla (t.j. číslo minus 256 krát integer, nikoliv zbytek dělení) uložíme do nižšího bajtu. Obě hodnoty po patřičném převodu do hex tvaru.

Příklad: obsah pamětového místa A je 7Eh a pamětového místa A+1 je 82h. Po převodu na decimální hodnoty je to 126d a 178d a obsah 2 bajtů je 126 + 256 * 178 = 45694. Naopak decimální číslo 23521d uložíme do 2 bajtů paměti takto: 23521 / 256 = 91.878 ..... integer je 91d = 5Bh a uložíme ho do bajtu A+1, zbytek, t.j. 23521 - 256 * 91 = 255d = FFh uložíme do bajtu A, hex vyjádření čísla 23521 je pak 5BFF a jeho uložení v paměti bude FFh 5Bh (je to sice opakování z 1.části, ale neuškodí).

S čísly typu floating point je to poněkud složitější. Každé číslo můžeme vyjádřit ve tvaru A*B na n. A nazýváme mantisou čísla, B základem a n exponentem (vzpomeňme si na logaritmy ze školních časů). Zvolíme-li za základ číslo 2 a položíme-li podmínku 0.5=<A <1, můžeme každé číslo vyjádřit jako A*2 na n. Vrátíme-li se do 2. kapitoly 1. části k binárním číslům, zjistíme, že mantisu A v našem výrazu můžeme vyjádřit ve tvaru A1 * 2 na -1 + A2 * 2 na -2 + A3 * 2 na -3 + ... + An * 2 na -n, kde A1 až An jsou jednotlivé bity binárního zlomku. Např. 0,75d = 0,50 + 0,25 = 2 na -1 + 2 na -2, Binárně 1100. Jakékoliv číslo převedeme do tvaru A * 2 na n tak, že ho podělíme nejbližší vyšší mocninou dvou. Výsledek dělení je pak mantisa A a mocnina se objeví v exponentu n. Pro číslo 12d bude: 12/16=0,75 můžeme psát 12d = 0,7 5 * 2 na 4 (16 = 2 na 4. Kontrola je 0,75 * 2 na 4 = 0,75 * 16 = 12d). Pro číslo 425.725d je nejbližší vyšší mocnina dvou 512d = 2 na 9. Bude pak 425.725 /512 = 0,8314941 .... a můžeme psát 425,725d = 0,8314941 * 2 na 9. Mantisu převedeme z decimálního tvaru do binárního zlomku tak, že od daného decimálního čísla postupně odečítáme záporné mocniny dvou. Je-li v čísle daná mocnina dvou obsažena, bude v odpovídajícím bitu jednička, není-li mocnina obsažena, bude v bitu nula. V našich dvou příkladech to bude:

```
 0,75    decimální mantisa
-0,50    (2 na -1) - je obsaženo: .1
 0,25    mezisoučet
-0,25    (2 na -2) - je obsaženo: .11
 0,00    mezisoučet roven nule, nejsou žádné další mocniny
         dvou, v binárním zlomku budou následovat nuly.

 0,8314941    decimální mantisa
-0,5000000    (2 na -1) - je obsaženo: .1
 0,3314941    mezisoučet
-0,2500000    (2 na -2) - je obsaženo: .11
 0,0814941    mezisoučet
-0,1250000    (2 na -3) - je obsaženo: .110
 0,0814941    opakování mezisoučtu
-0,0625000    (2 na -4) - je obsaženo: .1101
 0,0189941    mezisoučet
-0,0312500    (2 na -5) - je obsaženo: .11010
 0,0189941    opakování mezisoučtu
-0,0156250    (2 na -6) - je obsaženo: .110101
 0,0033691    mezisoučet
```

a tak dál podle toho, s jakou přesností chceme dané číslo vyjádřit. Proto čím větší počet desetinných míst (bitů), tím přesnější vyjádření čísla. Ve většině případů je však výsledkem dělení (t.j. mantisa v decimálním tvaru) číslo s nekonečným

počtem desetinných míst a musíme pak náš převod někde zastavit. Dopouštíme se tím určité nepřesnosti, která závisí na počtu bitů, které máme k dispozici pro vyjádření binárního zlomku. Ve Spectru je vyhrazeno 32 bitů. Poněvadž byl zvolen systém, ve kterém mantisa může nabývat hodnot v intervalu od 0,5 do 1 víme, že mantisa je vždy větší než 0.5, t.j., že v mantise je vždy obsažena mocnina 2 na -1, t.j., že hodnota prvního bitu v binárním zlomku je vždy 1. Ve Spectru se této skutečnosti využívá tak, že první bit binárního zlomku určuje znaménko mantisy. Je-li roven 0, je mantisa kladná, je-li roven 1, je záporná.

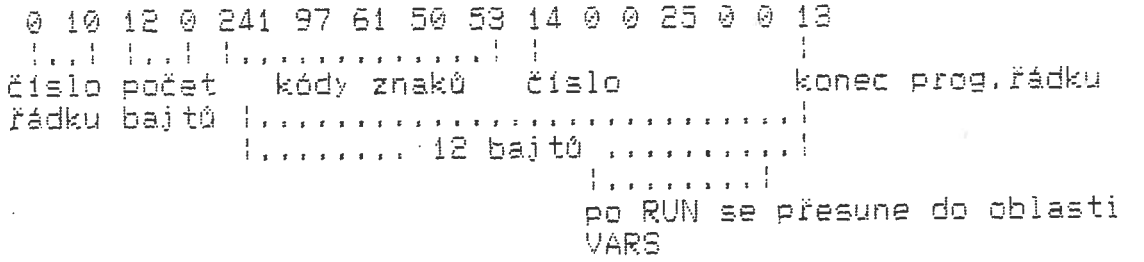
Spectrum pak ukládá čísla typu floating point do celkem pěti bajtů. První bajt obsahuje hex hodnotu exponentu, zvětšenou o 80h = 128d, následující čtyři bajty pak 32 bitů binárního zlomku, přičemž první bit vyjádřuje znaménko mantisy. Jěště k prvnímu (exponentovému) bajtu: je-li obsah tohoto bajtu FFh = 255d, je velikost exponentu rovna 255 - 128 = 127d = 7Fh. Je-li obsah exponentového bajtu rovna 0, je pak hodnota exponentu rovna 0 - 128 = -128d = -80h. Tím je také dána maximální hodnota čísla, které může Spectrum vyjádřit. Je to 0.999999999 ... * 2 na 127, prakticky 1 * 2 na 127, což je číslo s 38 nulami.

S tak velkými čísly však Spectrum neprovádí matematické operace a manuál uvádí, že největší číslo, které může zpracovat bez chyby je 4 294 967 295d. Číslo typu integer Spectrum ukládá do dvou bajtů, jak již bylo řečeno, ale číselné hodnoty přiřazené číselným proměnným ukládá do 5 bajtů následujícím způsobem: první bajt obsahuje 0, druhý bajt je znaménkový (0 - kladné, FFh - záporné). Ve třetím a čtvrtém bajtu je uložena číselná hodnota známým způsobem (v pořadí nižší bajt, vyšší bajt) a v pátém bajtu je uložena nula. Maximální velikost takto vyjádřeného čísla je +- FFFFh = +- 65535d.

Teď ještě detailnější pohled na způsob uložení proměnných v RAM Spectra (i když je to poměrně dobře vysvětleno v manuálu).

Programový řadek se ukládá do paměti v následujícím tvaru: V prvních dvou bajtech je číslo řádku v pořadí vyšší, nižší bajt. Je to výjímka ze zavedeného pravidla opačného ukládání bajtů a je to proto, poněvadž nejvyšší číslo řádku může být 9999d = 3915h. Podle hodnoty prvního bajtu - je-li nižší či rovno 39h - pozná počítač, že se jedná o programový řádek. Je-li hodnota prvního bajtu vyšší než 39h, znamená to pro počítač, že se již nejedná o programový řádek, ale o některou proměnnou. V následujících dvou bajtech je celkový počet bajtů programového řádku včetně závěrečného ENTER. Přičtením této hodnoty k adrese začátku programového řádku vypočte počítač adresu začátku dalšího řádku. V následujících bajtech jsou postupně uloženy kódy jednotlivých znaků v programovém řádku (klíčová slova a ostatní tokens t.j. slova vkládaná stlačením jediné klávesy, mají vždy jen jeden kód). Má-li proměnná přiřazenou číselnou hodnotu, následuje za kódem znaku číslo 14, signalizující, že následují pětibajtové série jednotlivých číselných hodnot obsažených v řádku. Na konci řádku je číslo 13, což je kód ENTER.

Příklad: programový řádek 10 LET a = 25 je v paměti počítače (t.j. v oblasti PROG RAM) uložen následovně (decimálně) :



Proměnné jsou jednak specifikovány již v programovém řádku po znaku pro číslo, t.j. 14, jednak se po spuštění programu přenesou do oblasti VARS, kde tvoří tzv. tabulku proměnných (proměnné zůstávají ve VARS i když po spuštění programu vymažeme všechny proměnné).

Prvním a u řetězcových proměnných také jediným znakem musí být písmeno. Aby počítač poznal, o jaký druh proměnné se jedná, je první bajt, přesněji řečeno jeho první 3 bity, následovně zakódován :

typ proměnné	první tři bity	první bajt
řetězec	010	41h až 5Ah
proměnná označena jediným písmenem	011	61h až 7Ah
číslicové pole	100	81h až 9Ah
proměnná označená více písmeny	101	A1h až BAh
řetězcové pole	110	C1h až DAh
řídící proměnná v příkazu FOR-NEXT	111	E1h až FAh

Ve všech případech zůstává v posledních 5 bitech prvního bajtu kód písmene malé abecedy minus 60h, což znamená, že pro a (nebo A - u proměnných se nerozlišují malá a velká písmena) bude v posledních 5 bitech hodnota 00001 = 1h, která se bude postupně zvyšovat až dosáhne hodnotu 11010b = 1Ah = 26d pro z. Nejnižší hodnota prvního bajtu řetězcové proměnné a$ je pak 01000001b = 41h a podle toho, že je to víc než 39h, pozná počítač, že opustil oblast, ve které je uložen program a nachází se v tabulce proměnných. Toto kódování provádí počítač tak, že ke kódu písmen malé abecedy přičítá určité hodnoty. Jsou to: -20h u řetězců, 0 u proměných s jednopísmenovým názvem, +20h u číslicových polí, +40h u proměnných s více písmenovým názvem, +60h u řetězcových polí a +80h u řídicích proměnných.

Typy proměnných

Ještě stručně ke způsobu uložení jednotlivých typů proměnných v tabulce proměnných (viz také kapitola 24 manuálu):

-proměnná s jednopísmenovým názvem (a):

celkový počet bajtů: 6
1. bajt - kód názvu
2.-6. bajt - pětibajtové vyjádření hodnoty proměnné

-proměnná s vícepísmenovým názvem (Adam):

celkový počet bajtů: počet bajtů názvu + 5
1. bajt - kód prvního písmene názvu + 40h
Další bajty až do celkové délky názvu - kódy jednotlivých písmen názvu. Kód posledního písmene je zvětšen o 80h, což je znamení pro počítač, že končí název a začíná hodnota (neuvedeno v manuálu).
Posledních pět bajtů - pětibajtové vyjádření hodnoty proměnné.

-řetězec (a$):

celkový počet bajtů: počet znaků v řetězci + 3 (bez $ a ").
1. bajt - kód názvu řetězce -20h
2. a 3. bajt - počet znaků v řetězci (bez uvozovek) v pořadí nižší bajt, vyšší bajt. V dalších bajtech kód jednotlivých znaků řetězce.

-řídící proměnná FOR-NEXT smyčky (FOR a = ... TO):

celkový počet bajtů: 19
1. bajt - kód názvu řídící proměnné + 80h
2. až 6. bajt - pětibajtové vyjádření počáteční hodnoty řídící proměnné. Při každém průchodu smyčkou se hodnota zvyšuje o velikost kroku, až dosáhne konečné hodnoty, t.j. hodnoty uvedené po TO, případně nejvyšší vyšší.
7. až 11. bajt - pětibytové vyjádření konečné hodnoty řídící proměnné.
12. až 16. bajt - pětibajtové vyjádření hodnoty kroku (1, není-li krok udán).
17. až 18. bajt - číslo řádku, na který se smyčka vrací po příkaze NEXT v obvyklém pořadí, t.j. nižší bajt, vyšší bajt.
19. bajt - číslo příkazu v daném řádku, na který se smyčka vrací po NEXT.

-číslicové pole (A(M,N)):

celkový počet bajtů: počet prvků (M*N) * 5 + počet dimenzí (v našem případě 2) * 2 + 4.
1. bajt - kód názvu pole + 20h
2. a 3. bajt - počet bajtů počínaje 4. bajtem do konce uložení
V dalších dvojicích bajtů jsou hodnoty jednotlivých dimenzí, t.j. hodnoty M a N.
V dalších M * n * 5 bytech jsou hodnoty jednotlivých prvků pole v pětibajtovém vyjádření, v prvních pěti bytech je hodnota prvku s indexem 1,1, následuje prvek s indexem 1,2 atd. až 1,N, pak následuje série prvků s indexy 2,1, 2,2 až 2,N a pak další série až po konečnou s indexy M,1, M,2 až M,N.

-řetězcové pole (A$(M,N))

celkový počet bajtů: počet prvků (M*N) + počet dimenzí (2) *2+4
1. bajt - kód názvu pole + 60h
2. a 3. bajt - počet bajtů počínaje 4. bajtem do konce uložení pole (t.j. celkový počet bajtů - 3).
4. bajt - počet dimenzí (2)
V dalších dvojicích bajtů jsou hodnoty jednotlivých dimenzí, t.j. hodnoty M a N.
V dalších bajtech jsou kódy znaků jednotlivých řetězců v pořadí 1. řetězec, 2. řetězec až M-tý řetězec.

V oblasti PROG, t.j. při ukládání programových řádků používá počítač po znaku 14 v pětibajtovém uložení hodnot všude tam, kde je to možné, hodnot typu integer. Pokud není dána hodnota integer, pak nastupuje vyjádření typu plovoucí řádová tečka.

V oblasti VARS, t.j. v tabulce proměnných, počítač používá zásadně vyjádření typu plovoucí řádová tečka (floating point), i když se jedná např. o číslo 1. Tato skutečnost je také jedním z důvodů poměrné pomalosti Basicu Spectra. Některé počítače mají možnost již při vložení čísla určit, jedná-li se o číslo typu integer. Počítač pak tomuto číslu vyhradí pouze 2 bajty. Spectrum pro jakékoliv číslo vyhradí vždy pět bajtů. Zvětšuje se tím tabulka proměnných a prodlužuje doba na její prohledávání. Dalším důvodem pomalosti je to, že u Spectra jsou pro délku řádku a pro délku řetězce vyhrazeny vždy dva bajty, což sice je výhodné z hlediska maximálního počtu znaků v řádku /řetězci/ (65535d), ale na druhé straně to opět prodlužuje dobu provádění basicových příkazů.

Ještě poznámku ke kalkulátoru (viz kapitola o ROM - RST #28). Kalkulátor má pseudo-kód (příkaz) 34, kterým se uloží do zásobníku kalkulátoru float point číslo v pětibajtovém tvaru. Aby nemusely být vypisovány všechny nulové body, je počet bajtů takového čísla zakódován v prvním (exponentovém) bajtu čísla, přičemž číslo musí mít nejméně dva bajty (další nuly doplní kalkulátor). Hodnotu upraveného prvního bajtu dostaneme odečtením 50h a přičtením 40h * (počet bajtů - 2). Např. číslo 2 má v pětibajtovém vyjádření tvar 82 00 00 00 00 (exponent a pět nul). Do tvaru stravitelného pro kalkulátor ho upravíme následovně: 1.bajt = 82h - 50h + 40h * (2 - 2) = 32h, 2.bajt = 00 (nemění se). Konečný tvar je 32 00 (t.j. pouze 2 bajty); číslo 0.8 je pětibajtově vyjádřeno: 80 4C CC CC CD, t.j. má všech pět bajtů, úprava pro kalkulátor: 1.bajt = 80h - 50h + 40h * (5 - 2) = F0h. Další bajty 4C CC CC CD se nemění. Konečná podoba: F0 4C CC CC CD. V obou případech si kalkulátor vypočte správnou hodnotu prvního bajtu, t.j. 82h, resp. 80h.

3. Rozdělení paměti RAM

RAM je zkratka anglického názvu Random Access Memory, což přeloženo do češtiny znamená paměť s libovolným výběrem. Znamená to, že do jednotlivých pamětových míst můžeme libovolně vkládat informace a také z nich informace vybírat. Pro RAM platí několik základních pravidel. Po zapnutí počítače jsou jednotlivá pamětová místa zaplněná zcela náhodnými hodnotami od 0 do FFh. Počítač proto v první fázi provede tzv. inicializaci systému, spočívající ve vložení nul do všech pamětových míst, vymezení hranic jednotlivých oblastí RAM a vložení počátečních hodnot do systémových proměnných. To je ta časová prodleva, než se po zapnutí počítače objeví na obrazovce Copyright (případně kurzor po příkazu NEW).

Další zásadou je to, že po zapsání jakékoliv hodnoty do pamětového místa tam tato hodnota zůstává tak dlouho, dokud není přepsána další hodnotou, pokud ovšem při jejím výběru nedojde k okamžitému vymazání některým podprogramem ROM.

Spectrum v 16k verzi má celkem 16384 pamětových míst v RAM, 48k verze má v RAM celkem 49152 pamětových míst. RAM je rozdělena do jednotlivých oblastí, do kterých se ukládají vždy určité typy informací. Toto rozdělení je na schématu na této straně. V obou variantách Spectra začíná RAM na adrese 16384 a první čtyři oblasti RAM mají pevné adresy. Poslední pevnou adresou v RAM je 23734, kde končí oblast systémových proměnných a hranice všech dalších oblastí jsou pohyblivé a mění se podle množství informací v jednotlivých fázích práce počítače dokonce hranice některých oblastí splývají, t.j., některé oblasti neexistují, resp. mají nulový obsah. Adresy těchto pohyblivých hranic oblastí jsou uloženy v systémových proměnných, jejichž přehled je uveden v kap. 25 manuálu. K některým systémovým proměnným se ještě vrátíme a probereme je podrobněji. Napřed se ale musíme zabývat jednotlivými oblastmi RAM.

Schéma paměti RAM

oblast: název	oblast: adresa začátku/délka		systémová proměnná: název	systémová proměnná: adresa dec-hex
Paměť ROM	00000	#0000		
	16384	#4000		
Registr obrazovky	16384	#4000		
	6144	#1800		
Registr atributů	22528	#5800		
	768	#0300		
Registr tiskárny	23296	#5B00		
	256	#0100		
Systémové proměnné	23552	#5C00		
	182	#00B6		
(Dodat.syst.proměnné)	23734	#5CB6		
	58	#003A		
(Mapy microdrive)	23792	#5CF0		
	různé			

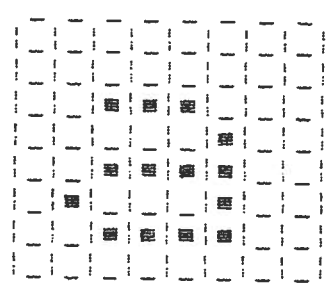
Kanálova data + 80h	[23734	#5CB6]	CHANS	23631	#5C4F
	[21	#0015]			
Basicový program	[23755	#5CCB]	PROG	23635	#5C53
Tabulka proměnných + 80h			VARS	23627	#5C48
Vložený řádek + ENTER +80h			E_LINE	23641	#5C59
Inputová data + ENTER			WORKSP	23649	#5C61
Pracovní prostor					
Zásobník kalkulátoru			STKBOT	23651	#5C63
Volná paměť			STKEND	23653	#5C65
Zásobník			SP-registr mikroprocesoru		
Zásobník GOSUB					
3Eh	[65367	#FF57]	RAMTOP	23730	#5CB2
(Strojní program)					
Definované znaky	[65368	#FF58]	UDG	23675	#5C7B
	168	#00A8			
Konec RAM	65535	#FFFF	P_RAMT	23732	#5CB4

adresy označené [] nejsou pevně stanoveny, mohou se měnit

Display file

Oblast od adresy 16384 po adresu 22527 (4000h-57FFh) je tzv. Display file, čili oblast pro uložení obrazové informace. Jsou zde uloženy znaky, které počítač vypíše na obrazovku. Tato oblast zaujímá celkem 6144 bajtů. Proč zrovna 6144? Obrazovka má 24 řádků po 32 sloupcích, celkem 768 znaků. Každý znak je uložen v osmi bajtech, tudíž pro uložení všech znaků obrazovky je třeba 768 * 8 = 6144 bajtů. Teď se musíme podrobněji podívat na tuto oblast, resp. na způsob uložení znaku v ní.

Každý z 8 bajtů tvořících znak vyjadřuje jeden řádek znaků na obrazovce. Např. písmeno "a" je vytvořeno následovně:



Bin	Hex
0000 0000	00
0000 0000	00
0011 1000	38
0000 0100	04
0011 1100	3C
0100 0100	44
0011 1100	3C
0000 0000	00

V binárním vyjádření znamená 0 paper a 1 ink.

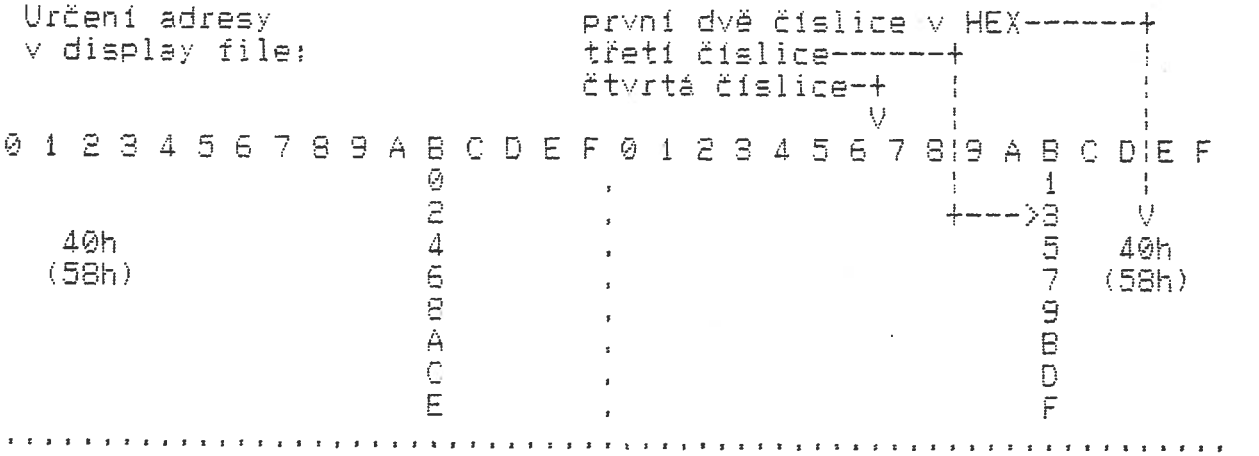
Obrazovka je rozdělena do tří částí po osmi řádcích textu. V display file jsou bajty jednotlivých znaků uloženy následovně počínaje prvním bajtem display file: 1 bajt prvního znaku řádku 0, 1 bajt druhého znaku řádku 0, 1 bajt třetího znaku řádku 0 atd. až první bajt posledního znaku řádku 0, 1 bajt prvního znaku řádku 1, 1 bajt druhého znaku řádku 1, až 1 bajt posledního znaku řádku 1. Obdobně pokračují všechny první bajty dalších řádků až po řádek 7 a pokračuje: 2 bajt prvního znaku řádku 0, 2 bajt druhého znaku řádku 0, až 2 bajt posledního

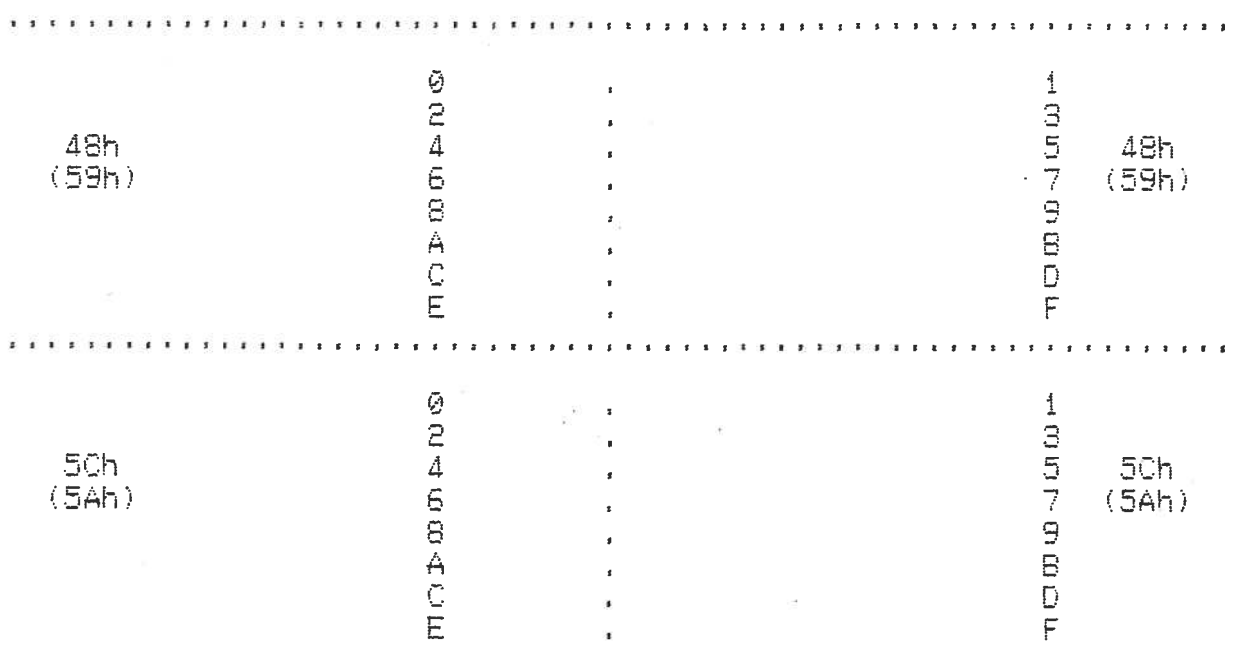
znaku řádku 0. Pak pokračují postupně druhé bajty znaku na řádcích 1 až 7, pak podobným způsobem třetí bajty řádků 0 až 7, čtvrté bajty řádku 0 až 7, pak postupně další bajty, až nakonec osmé bajty řádku 0 až 7. Pak se tento postup opakuje pro řádky 8 až 15 a nakonec pro řádky 16 až 23. Nejlépe je tento způsob vidět na kreslení screen$ (obrazu) u her.

Adresa prvního bajtu 1. znaku je 16384, adresa druhého bajtu téhož znaku je 16384 + 32 * 8 = 16640, adresa třetího bajtu obdobně 16640 + 32 * 8 = 16896 atd. První znak je tudíž uložen v následujících bajtech: 16384, 16640, 16896, 17152, 17408, 17664, 17920 a 18176. Druhý znak je uložen v bajtech o jednu vyšších. V decimálních hodnotách to vypadá dost divoce, ale podívejme se na to v hex vyjádření: pro adresy bajtu 1. znaku obrazovky dostaneme: 4000, 4100, 4200, 4300, 4400, 4500, 4600, 4700. Adresy druhého znaku budou: 4001, 4101, 4201, ...

Použijeme-li pro adresování jednotlivých bajtů v Display file registru DE, pak platí: Je-li v DE adresa prvního bajtu některého znaku, pak adresy dalších bajtů tohoto znaku dostaneme postupným opakováním 8 * inc D, zatímco adresu prvního bajtu následujícího znaku dostaneme inc DE. Toto inc DE platí ovšem jen v jednotlivých třetinách obrazovky. Při přechodu z jedné třetiny do druhé musíme použít následující sekvenci strojních instrukcí: RR D; RR D; RR D; INC DE; RL D; RL D; RL D. Je to proto, že adresa prvního bajtu posledního znaku řádku 7 je 40FFh a adresa prvního bajtu prvního znaku řádku 8 je 4800h (nikoliv 4100h, což je adresa 2. bajtu prvního znaku). Uvedená sekvence strojních instrukcí dá v každém případě správnou adresu prvního bajtu následujícího znaku. Adresu prvního bajtu libovolného znaku na obrazovce určíme pomocí tabulky (čísla v závorkách platí pro adresy atributů). Do jednotlivých adres Display file můžeme pomocí POKE vkládat libovolné hodnoty. Zkuste například: LD HL,#40A4; LD (HL),#FF; RET. Po spuštění tohoto strojního programu se na obrazovce v místě odpovídajícím PRINT AT 5,4 objeví tenká čárka. Je to proto, poněvadž adresa 40A4h odpovídá podle tabulky: 40 - první třetině obrazovky, A - 5. řádek, 4 - 4. sloupec.

Pro určení atributů platí pro první dvě číslice hodnoty v závorkách, další dvě číslice podle vzoru nahoře.





Atributy

Oblast od adresy 22528 po adresu 22295 (5800h až 5AFFh) je vyhrazena pro uložení atributů znaků zobrazovaných na obrazovce. Jako atributy označuje manuál: barvy pro paper a ink, bright a flash.

Každému znaku přisluší jeden bajt v pořadí 1. znak prvního řádku, 2. znak prvního řádku až poslední znak prvního řádku. Pak následují obdobným způsobem postupně všechny ostatní řádky. (Píši-li znaky první řádky, jsou tím míněny znaky řádku 0). Tato skutečnost je důvodem, proč Spectrum nemá "plnohodnotnou bodovou" barevnou grafiku (nemůže např. zobrazit horní polovinu znaku jinou barvou než jeho spodní polovinu), ale celý znak musí mít stejnou barvu (znakem rozumíme vždy osm bajtů vyjadřujících znak, takže znakem je i mezera), at už se jedná o ink nebo paper. Adresy, v nichž jsou uloženy atributy odpovídající jednotlivým pozicím na obrazovce, zjistíme z tabulky na předchozí straně.

Vlastnosti atributů jsou v jednotlivých bajtech zakódovány následujícím způsobem: bajt = 128 * flash + 64 * bright + 8 * paper + ink.

I když uvedený způsob kódování vypadá na první pohled složitě, je vlastně velice jednoduchý a nejlépe je vidět na následujícím schématu. Bajty atributů jednotlivých znaků (je jich celkem 24 * 32 = 768) obsahují:

```
Číslo bitu                        7  6  5  4  3 2 1 0
hodnota bitu, je-li nastaven    128 64 32 16 8 4 2 1

                    bajt atributů  X  X  X  X  X X X X
                                   !  !  !  .  .  !  .  !
                                   .  .  .  .  .  .  .  .
Flash:   1 - bliká ....            .  .  .  .  .  .  .  .
         0 - nebliká ..............  .  .  .  .  .  .  .
Bright:  1 - zvýšený jas ....         .  .  .  .  .  .  .
         0 - normální jas ..........  .  .  .  .  .  .  .
Paper:   000 - černá .....           .  .  .  .  .  .  .
         001 - modrá  .....           .  .  .  .  .  .  .
         010 - červená ....           .  .  .  .  .  .  .
         011 - purpurová ....         .  .  .  .  .  .  .
         100 - zelená .....           .  .  .  .  .  .  .
         101 - bledě modrá ....       .  .  .  .  .  .  .
         110 - žlutá .....            .  .  .  .  .  .  .
         111 - bílá ..................   .  .  .  .  .  .
Ink:     000 - černá ..........          .  .  .  .  .  .
         001 - modrá ..........          .  .  .  .  .  .
         010 - červená ........          .  .  .  .  .  .
         011 - purpurová ......          .  .  .  .  .  .
         100 - zelená .........          .  .  .  .  .  .
         101 - bledě modrá ....          .  .  .  .  .  .
         110 - žlutá ..........          .  .  .  .  .  .
         111 - bílá ..........................  .  .  .  .
```

Nastavením jednotlivých bitů, t.j. vložením odpovídajících hodnot do bajtu atributů můžeme libovolně nastavit atributy kteréhokoliv znaku na obrazovce. Maximální hodnota bajtu atributů může být 128 * 1 + 64 * 1 + 8 * 7 + 7 = 128 + 64 + 56 +7 = 255 (#FF).

Poznámka: Všechny barvy, které může Spectrum generovat na obrazovce jsou kombinace tří základních barev barevného spektra: modré (001), červené (010) a zelené (100). Jejich sečtením vznikne bílá barva (111), sečtením zelené a červené vznikne žlutá (110) atd. V tříbitovém vyjádření je možno použít pouze 8 různých kombinací, což je důvodem, proč Spectrum může generovat pouze osm barev.

Printer buffer

Oblast od adresy 23296 po adresu 23551 (5B00h až 5BFFh) je vyhrazena pro vyrovnávací registr tiskárny. Tato oblast RAM má 256 bajtů a je určena pro dočasné uložení nekompletního řádku určeného pro tisk tiskárnou. Tiskárna totiž musí vytisknout celý řádek naráz, i když je např. příkazy TAB nebo AT rozdělen do několika programových řádků a na obrazovce se vypisuje postupně podle stadií programu.

Pokud není požadováno použití tiskárny a pokud strojní program není delší než 256 bajtů, je možno této části RAM využít pro uložení strojního programu. Je zde totiž chráněn před přepsáním basicovým programem.

Systémové proměnné

Oblast od adresy 23552 po adresu 23733 (5C00h až 5CB5h) je vyhrazena pro uložení tzv. systémových proměnných. Horní hranice, t.j. adresa 23733, ovšem platí pouze pro případ, že ke Spectru není připojen Interface 1, což je přídavný obvod pro řízení Microdrive, je-li připojen a aktivován, posune se horní hranice oblasti systémových proměnných na adresu 23791 (5CEFh). Je to proto, poněvadž Interface 1 potřebuje pro svoji práci některé další systémové proměnné, které jsou pak připojeny za konec normalních systémových proměnných.

Systémové proměnné obsahují informace nutné pro práci počítače a mají přesně vymezený význam. Mimo jiné jsou v nich uloženy adresy začátků pohyblivých oblastí RAM. Všechny systémové proměnné jsou uvedeny v kap. 25 manuálu. Poněvadž některé z nich můžeme výhodně použít ve strojních programech, bude jim věnována samostatná kapitola.

Kanálové informace

Není-li ke Spectru připojen Interface 1, je od adresy 23734 po adresu 23754 (5CB6h až 5CCAh) vyhrazeno 21 bajtů pro uložení tzv. kanálových informací. Do těchto bajtů se během inicializace přenesou z ROM určité hodnoty, sloužící pak k nalezení adresy příslušného kanálu. Pokud není připojen Interface 1, jsou stále otevřené 4 kanály: klávesnice, obrazovka, tiskárna a pracovní prostor. Je-li připojen Interface 1 spolu s Microdrive, jsou vytvořeny další kanály, do kterých se ukládají informace, které mají být přeneseny do jednotlivých sektorů Microdrive. Každý z těchto kanálů pak zabírá 595 bajtů. Pro strojní programy je z kanálů důležité vědět to, že je-li otevřen kanál K, znamená to tisk na spodní část obrazovky. Chceme-li psát na horní část obrazovky, musíme otevřít kanál S. (K - Keyboard = klávesnice, S - Screen = obrazovka). V případě, kdy ke Spectru není připojen Interface 1, neexistuje v RAM oblast Map Microdrive a hned za kanálovými informacemi následuje oblast ve které je uložen basicový program (chyba ve schématu RAM v kap.24 manuálu). Začátek oblasti kanálových informací je v systémové proměnné CHANS. Tato oblast je uzavřena bajtem 80h.

Mapy Microdrive

Je-li ke Spectru připojen Interface 1 s Microdrive, začíná od adresy 23792 (5CF0h) oblast map Microdrive. Její velikost závisí na počtu připojených Microdrive (maximálně 8), poněvadž pro každý aktivovaný Microdrive je vytvořena samostatná mapa o délce 32 bajtů. Do této mapy se ukládají informace o obsazení jednotlivých sektorů Microdrive. Ve schématu RAM v kap.24 je chybně uveden začátek této oblasti na adrese 23734.

Kanálové informace

Při připojeném Interface 1 následuje oblast kanálových informací, která již byla uvedena dříve.

Basicový program

Tato oblast začíná adresou, která je uložena v systémové proměnné PROG, a podle ní se někdy celá oblast označuje jako PROG. Jednotlivé programové řádky jsou v této oblasti uloženy ve tvaru uvedeném v předchozí kapitole. Pokud není připojen Interface 1, začíná oblast PROG na adrese 23755 (5CCBh).

Proměnné

Za oblastí PROG následuje oblast, ve které jsou uloženy hodnoty jednotlivých proměnných. Její začátek je v systémové proměnné VARS a podle toho bývá někdy celá oblast nazývaná VARS. Tato oblast se vytvoří až po spuštění programu a jsou do ní přeneseny hodnoty proměnných z programových řádku v předchozí oblasti, které následují po znaku 14 v programovém řádku.

Hodnoty jednotlivých proměnných jsou uloženy ve tvarech uvedených v předchozí kapitole a ukládají se postupně podle toho, jak následují v Basicovém programu. Příkaz vytvoří potřebný počet bajtů, do nichž jsou pak ukládány jednotlivé hodnoty později. Poslední bajt této oblasti má hodnotu 80h. Hodnoty proměnných zůstávají v této oblasti zachovány i po vymazání všech programových řádků a můžeme s nimi i pak operovat. Vymaže je příkaz NEW, LOAD, CLEAR a RUN.

Basicové řádky před stlačením ENTER

Za oblastí PROG následuje oblast, ve které jsou uloženy příkazy nebo Basicové řádky před stlačením ENTER. Jsou to příkazy nebo řádky, které počítač vypisuje do dolní části obrazovky. Po stlačení ENTER se buď provede příkaz (včetně přenesení hodnot proměnných do oblasti VARS) nebo se programový řádek přenese do oblasti PROG. Do této oblasti se také přenese programový řádek příkazem EDIT a zde je možno provádět změny v programovém řádku, ať už právě vkládaném nebo editovaném. Začátek oblasti je uložen v systémové proměnné E_LINE a poslední bajt v této oblasti má hodnotu 80h. Pokud nemá programový řádek číslo řádku, je po stlačení ENTER vymazán a nikam se nepřenese.

INPUT data

Do další oblasti, začínající na adrese uložené v systémové proměnné WORKSP, se ukládají data vložená z klávesnice po příkazu INPUT. Po stlačení ENTER se přenesou do oblasti VARS.

Pracovní prostor

Na horním konci oblasti INPUT data (po ENTER) je v manuálu část označena jako pracovní prostor (WORKSPACE). V této oblasti provádí počítač operace s řetězci. Obě oblasti se překrývají, poněvadž do prostoru mezi WORKSP a STKBOT buď ukládá INPUT data, která se po ENTER přenesou do dalších oblastí a oblast WORKSP-STKBOT se vynuluje, nebo během provádění programu se v této oblasti provádí operace s řetězci. Nikdy neprobíhají obě činnosti současně.

Zásobník kalkulátoru

Další oblast začínající na adrese uložené v systémové proměnné STKBOT je zásobník kalkulátoru. Sem ukládá kalkulátor jednotlivé hodnoty, se kterými pracuje. Zásobník kalkulátoru narůstá od nižších adres k vyšším, t.j. jeho vrchol je na nejvyšší adrese, dané systémovou proměnnou STKEND.

Rezerva

Dál následuje oblast nazvaná v manuálu jako rezerva. Jsou to volné bajty RAM, na jejichž úkor se rozšiřují ostatní oblasti. Začátek je na adrese uložené v systémové proměnné STKEND a konec je na adrese uložené v registru SP. Chceme-li znát počet volných bajtů, které máme ještě k dispozici, musíme od obsahu registru SP odečíst hodnotu systémové proměnné STKEND.

Teď přeskočím zbytek RAM a začnu od jejího konce.

Konec paměti RAM

Poslední použitelná adresa RAM je uložena v systémové proměnné P_RAMT. Je to 32767 (7FFFh) u verze 16k RAM nebo 65535 (FFFFh) u verze 48k RAM.

Definované grafické znaky

Při inicializaci systému se přenese od poslední adresy v RAM 168 bajtů z tabulky znaků v ROM. Vytvoří se tak oblast pro uložení definovaných grafických znaku označených v manuálu (a) až (u) - kód 144 až 164. Tato oblast začíná na adrese uložené v systémové proměnné UDG. U 16k je to 32000 (7F58h), u 48k je to 65368 (FF58h) a je chráněná před přepsáním basicovým systémem (pokud ovšem nedefinujete vlastní grafické znaky) a před příkazem NEW.

RAMTOP

V systémové proměnné RAMTOP je uložena hodnota o jedničku menší, než hodnota systémové proměnné UDG. Pro 16k verzi je to po zapnutí počítače hodnota 32599 (7F57h) a pro 48k verzi je to hodnota 65367 (FF57h). Je to poslední pamětové místo basicového systému. Nad RAMTOPem nefungují basicové příkazy. Snížením RAMTOPU si můžeme vytvořit oblast pro uložení strojních programů. Spectrum má příkaz CLEAR s možností připojit k tomuto příkazu argument. CLEAR bez argumentu vymaže oblast proměnných. CLEAR s argumentem např. CLEAR 32499, navíc sníží RAMTOP na hodnotu danou argumentem a vytvoří nad RAMTOPem odpovídající počet volných bajtů, které jsou chráněny před přepsáním basicovým programem i před příkazem NEW (CLEAR 32499 vytvoří od adresy 32500 sto volných bajtů pro uložení strojního programu - po 32600 kde je UDG u 16k verze. Začátek volného prostoru je uložen v systémové proměnné RAMTOP (první volný bajt = RAMTOP + 1). Pokud nedojde ke snížení RAMTOPu, je tato oblast tvořena jediným bajtem.

Stack (zásobník)

Pod RAMTOPem je poslední oblast RAM, tzv. Stack. Někdy se označuje jako zásobníková paměť, uživatelský zásobník, zásobník návratových adres, nebo pouze zásobník. V dalším budeme tuto oblast označovat pouze jako zásobník. Do zásobníku ukládá Z80 jednak návratové adresy při provádění instrukce CALL a RST, jednak obsahy registrů při provádění instrukce PUSH. Naopak ze zásobníku Z80 vybírá návratové adresy při provádění instrukce RET a obsahy registrů při provádění instrukce POP. Poslední informace se ukládá na vrchol zásobníku. Označení vrchol zásobníku není příliš vhodné, poněvadž zásobník narůstá odshora dolů, t.j. od vyšších adres k nižším adresám. Vrchol zásobníku je tudíž nejnižší adresa. Další zvláštnost je v tom, že adresa vrcholu zásobníku není uložena v žádné systémové proměnné, ale přímo v registru SP Z80 (SP znamená Stack pointer = ukazatel zásobníku). Vybírání informací ze zásobníku se děje v opačném pořadí než ukládání, t.j. informace se vybírá z vrcholu zásobníku. Jako první se vybere poslední vložená informace, jako druhá pak předposlední atd. Pokud se do zásobníku ukládají dvoubajtová čísla, děje se ukládání v obvyklém pořadí nižší bajt, vyšší bajt, vyšší bajt je na vrcholu zásobníku. Vybírání se děje v pořadí vyšší bajt, nižší bajt. Ve schématu na str.8 je zásobník rozdělen na zásobník a zásobník GOSUB. Je to proto, že do zásobníku se též ukládají návratové adresy příkazů GOSUB.

Tím jsme se dostali na druhý konec oblasti volné paměti (rezervy) a vyčerpali tak celý rozsah RAM.

4. Uložení strojních programů v RAM a zaváděcí programy

Když nyní víme, jak je organizována RAM, můžeme se rozhodnout, kam náš strojní program uložíme, aby byl chráněn před přepsáním nebo vymazáním. Pak ještě zbývá otázka, jakým způsobem strojní program vložíme do paměti.

4.1 Uložení str.programů v RAM

U Spectra existuje řada možností pro uložení strojního programu v RAM. První a nejpohodlnější je uveden již v manuálu. Pomocí příkazu CLEAR + argument se v paměti vytvoří pod oblastí UDG oblast pro uložení našich strojních programů. Poněvadž příkaz CLEAR + argument současně sníží RAMTOP na danou hodnotu, je v dané oblasti strojní program chráněn před vymazáním nebo přepsáním basicovým programem nebo příkazem. Argument příkazu CLEAR získáme tak, že od adresy uložené v systémové proměnné UDG (t.j. 32600 u 16k nebo 65368 u 48k verse) odečteme počet bajtů strojního programu + 1 a tuto hodnotu použijeme jako argument pro CLEAR. Máme-li např. strojní program o délce 120 bajtů, pak to bude 23600 - 121 = 23479. Příkaz CLEAR 23479 pak přesune RAMTOP na adresu 23479 a od adresy 23480 začíná první bajt pro uložení strojního programu.

Další možností je uložení strojního programu do vyrovnávacího registru tiskárny. Předpokladem je, že během provádění programu nebude potřeba tisk tiskárnou, a že program je kratší, než 256 bajtů. Pak je možno uložit strojní program od adresy 23296 (5B00h).

Další možností je uložení strojního programu do příkazu REM. U této alternativy je však několik zvláštností. Použijete-li bajt 0Eh, pak tento bajt a následujících pět bajtů zmizí z řádku za REM. V paměti ale zůstávají a počítač je považuje za pětibajtově vyjádřené číslo typu floating point. Také bajt 10 způsobí, že při editování zbytek řádku zmizí.

Další možností je uložení strojního programu do nulového řetězcového pole. Pro toto řetězcové pole musíme na začátku programu vytvořit potřebný počet bajtů na začátku oblasti VARS. Dosáhneme toho příkazem DIM X$ N, kde N je potřebný počet bajtů pro strojní program. Příkaz DIM vytvoří na začátku oblasti proměnných 6 + N bajtů. V prvních šesti bajtech je hlavička, t.j. název a další údaje, do dalších bajtů můžeme uložit strojní program. První bajt strojního programu je na adrese VARS + 6 = PEEK 23627 + 256 * PEEK 23628.

4.2 Zaváděcí programy

Zaváděcí programy slouží k uložení jednotlivých bytů strojního programu do paměti na zvolené místo. Tyto zaváděcí programy mohou být jednoduché a spartánské, nebo dlouhé a komfortní.

Nejjednodušší a nejpracnější je postupné POKE adresa, hodnota. Je vhodné pro použití u krátkých programů.

A naopak - pro delší programy doporučujeme některý assembler, například EDIT/ASSEMBLER, LASER GENIUS, MRS, GENS 3 a pod.

4.3 Vyvolávání str. programů

Strojní programy vyvoláváme (spouštíme) RAND USR n, PRINT USR n nebo POKE USR n, kde n je adresa začátku strojního programu v decimálním tvaru. Můžeme také zvolit např. LET Z = n a pak strojní program spustit RAND USR Z, příp. PRINT USR Z. Ne všechny strojní programy spouštíme od jejich prvního bajtu (viz např. programy na str. 33), takže slovo začátek v předchozí větě nemusí vždy znamenat první byte strojního programu, na což je také třeba dát pozor při spouštění.

5. Systémové proměnné

Systémové proměnné obsahují informace o okamžitém stavu systému a řadu z nich můžeme, nebo dokonce musíme, při sestavování strojních programů použít. Proto jim je věnována samostatná kapitola.

Přehled systémových proměnných je uveden v kap. 25 manuálu. Přesto je ještě jednou uvedeme v příloze spolu s dodatečnými systémovými proměnnými generovanými po připojení Interface 1. Jak již bylo uvedeno, systémové proměnné zaujímají jeden nebo dva bajty, výjimečně i více bajtů (KSTATE, FRAMES a MEMBOT). Některé z nich mohou být měněny pomocí POKE, u jiných se to nedoporučuje, ze všech však mohou být pomocí PEEK získávány informace. Místo POKE a PEEK je možno použít str. instrukcí LD.

Chceme-li získat decimálně vyjádřený obsah jednobajtové systémové proměnné, stačí PRINT PEEK adresa proměnné. U dvoubajtových proměnných je třeba použít PRINT PEEK adresa proměnné + 256 * PEEK (adresa proměnné + 1). Chceme-li změnit obsah jednobajtové systémové proměnné, postačí POKE adresa proměnné, nová decimální hodnota. Při měnění hodnoty dvoubajtové systémové proměnné usíme použít POKE adresa proměnné, nová hodnota - 256 * integer (nová hodnota / 256) a POKE adresa proměnné + 1, integer (nová hodnota / 256). (Viz kap.25 manuálu). Nyní k jednotlivým systémovým proměnným (pro lepší přehlednost je uvedu v abecedním pořadí).

ATTR_P - adresa 23693 (5C8Dh) - 1 bajt.

8 bitů této systémové proměnné obsahuje stále běžné barevné informace. Používá je např. CLS a podobné příkazy. Pro strojní program nejsou využitelné.

ATTR_T - adresa 23695 (5C8Fh) - 1 bajt.

Barevná informace pro právě vydávaný znak, např. při RST #10.

BORDCR - adresa 23624 (5C48h) - 1 bajt.

Barevná informace pro spodní část obrazovky. Pomocí POKE (příp. strojní instrukcí) můžeme docílit, aby vkládaný řádek ve spodní části obrazovky měl zvýšený jas (BRIGHT) nebo blikal (FLASH), pro což Spectrum normálně příkazy nemá. Také můžeme dosáhnout toho, aby vkládaný řádek nebyl vidět, t.j. aby byla stejná barva PAPER a INK, což normálně nemůže nastat. Použijeme-li POKE, pak musí následovat CLS. Nový stav pak trvá až do dalšího příkazu BORDER nebo NEW.

B_REG - adresa 23655 (5C67h) - 1 bajt.

Tuto systémovou proměnnou používá kalkulátor jako počitadlo. Bližší viz kapitola o RM, část RST #28.

CHANS - adresa 23631 a 23632 (5C4Fh, 5C50h) - 2 bajty.

V systémové proměnné je uložena adresa začátku oblasti kanálových informací. Bez připojeného Interface 1 je v systémové proměnné adresa 23734.

CHARS - adresa 23606 a 23607 (5C36h, 5C37h) - 2 bajty.

Obsahuje adresu začátku tabulky znaku minus 256 (0100h) na konci ROM, t.j. normálně je v ní hodnota 15360 (3000h). Této systémové proměnné můžeme použít, potřebujeme-li více, než 21 definovaných grafických znaků. Můžeme si totiž vytvořit ve volné části RAM novou tabulku všech znaků a změnou hodnoty v systémové proměnné CHARS získat přístup k této nové tabulce (v CHARS musí být hodnota o 256 nižší, než je začáteční adresa nové tabulky). Zpět do původní tabulky ROM se vrátíme pomocí:

```
POKE 23606,0:POKE 23607,60.
```

CH_ADD - adresa 23645 a 23646 (5C5Dh, 5C5Eh) - 2 bajty.

Systémová proměnná obsahuje adresu znaků (v oblasti INPUT, v Basicové oblasti nebo oblasti proměnných), který bude jako příští přenesen do Display file a jejím prostřednictvím vypsán na obrazovku. Můžeme použít ve strojních programech.

COORDS - adresa 23677 a 23678 (5C7Dh, 5C7Eh) - 2 bajty.

Obsahuje souřadnice X (v 23677) a Y (v 23678) posledního bodu zobrazeného příkazem PLOT nebo DRAW. Můžeme měnit, čímž měníme polohu bodu na obrazovce. Zvolíme-li např. LET X0 = 23677 : LET Y0 = 23678, pak můžeme použít DRAW A - PEEK X0, B - PEEK Y0 a dostaneme přímku z bodu, jehož souřadnice jsou v COORDS, do bodu o souřadnicích A,B.

CHURCHL - adresa 23633 a 23634 (5C51h, 5C52h) - 2 bajty.

Obsahuje adresu z oblasti kanálových dat, příslušnou pro právě otevřený kanál. Nemá použití.

DATARD - adresa 23639 a 23640 (5C57h, 5C58h) - 2 bajty.

Obsahuje adresu čárky za poslední položkou v řádku DATA, která byla přečtena příkazem READ v programu.

DEFADD - adresa 23563 a 23564 (5C0Bh, 5C0Ch) - 2 bajty.

Obsahuje adresu argumentu definované funkce DEF FN.

DEST - adresa 23629 a 23630 (5C4Dh, 5C4Eh) - 2 bajty.

Adresa proměnné při jejím vyhodnocování, jinak 0.

DEF_CC - adresa 23684 a 23685 (5C84h, 5C85h) - 2 bajty.

Obsahuje adresu prvního bajtu vypisovaného na obrazovku v Display file - viz tab. na str. 11. Vzhledem k systému uložení jednotlivých bajtů znaků v display file se nedoporučuje měnit.

DFCCL - adresa 23686 a 23687 (5C86h, 5C87h) - 2 bajty.

Totéž jako DEF_CC, ale pro spodní část obrazovky (DEF_CC pro horní část obrazovky).

DF_SZ - adresa 23659 (5C6Bh) - 1 bajt.

Obsahuje počet řádků ve spodní části obrazovky - normálně dva. Může být měněna hodnotami od 0 do 24, přičemž ale POKE 23659,0 znamená, že počítač nemá kam vypisovat sdělení a při první potřebě vypsat sdělení (např. SCROLL?) vypadne ze systému.

Podobně při INPUT nebo SAVE a při stlačení klávesy BREAK (po BREAK následuje sdělení), čehož je možno použít při ochraně programu s autostartem proti zastavení pomocí BREAK. Pokud se do takového programu vloží POKE 23659,0, pak první BREAK vede ke zhroucení systému.

Je-li obsah DF_SZ roven jedné, je možné použít PRINT AT 22,xx, je-li však obsah DF_SZ roven nule, nefunguje PRINT AT 23,xx. Chceme-li přesto psát na poslední řádek, je lépe použít PRINT AT 22,31 ' nebo PRINT # 0.

ECHO_E - adresa 23682 a 23683 (5C82h, 5C83h) - 2 bajty.

Obsahuje adresu ve spodní části obrazovky, za kterou nejde dál posunout kurzor směrem doprava, číslo sloupce a číslo řadku; číslo konce bafru pro vstup (INPUT z kanálu "K").

ERR_NR - adresa 23610 (5C3Ah) - 1 bajt.

Obsahuje kód sdělení minus 1. Je možno použít buď k zastavení programu bez vypsání sdělení nebo vypsání libovolného sdělení tím, že se do systémové proměnné vloží hodnota požadovaného sdělení minus 1. Hodnoty sdělení jsou v příloze B manuálu. G až H pokračují hodnotami 16 až 27. (při vypsání sdělení dojde vždy k zastavení basicového programu.

ERR_SP - adresa 23613 a 23614 (5C3Dh, 5C3Eh) - 2 bajty.

Obsahuje návratovou adresu na kterou se vrátí program po návratu z podprogramu RST 08, který vyvolává sdělení, t.j. např. adresu, kterou počítač pokračuje po zadpovězení dotazu SCROLL?

E_LINE - adresa 23641 a 23642 (5C39h, 5C3Ah) - 2 bajty.

Obsahuje adresu editovací oblasti v RAM.

E_PPC - adresa 23625 a 23626 (5C49h, 5C4Ah) - 2 bajty.

Obsahuje adresu běžného řádku, t.j. řádku u kterého stojí programový kursor.

FLAGS - adresa 23611 (5C3Bh) - 1 bajt.

Různá návěští. Je-li nastaven bit 0, před klíčovým slovem se netiskne pauza , je-li nastaven bit 1, znamená to, že může být použita tiskárna, je-li nastaven bit 2, znamená to kurzor L nebo C, je-li nulován pak kurzor K, je-li nastaven bit 3, znamená to nastavení na vstup L nebo C, je-li nulován pak vstup K, je-li nastaven bit 5, znamená to, že byla stisknuta klávesa bit 6 určuje typ vstupních dat - 0 znaky a 1 čísla, bit 7 nastaven znamená vykonávání programu a nulován kontrolu syntaxe.

FLAGS2 - adresa 23658 (5C6Ah) -1 bajt.

Další návěští. Bit 1 je nastaven po mazání obrazovky, bit 2 je nulován, je-li prázdný bafr obrazovky, bit 2 je nastaven, probíhá-li operace uvnitř uvozovek, bit 3 je nastaven při CAPS LOCK a bit 4 je nastaven, je-li kanál K aktuálním kanálem.

FLAGX - adresa 23665 (5C71h) - 1 bajt.

Další návěští. Bit 1 je nastaven po odvolání do neexistující proměnné, bit 5 nastaven v režimu INPUT a nulován při EDIT, bit 7 nastaven při INPUT LINE.

FRAMES - adresa 23672 až 23674 (5C78h - 5C7Ah) - 3 bajty.

Při každém přerušení, což se děje 50 krát za sekundu, je o jedničku zvýšena hodnota systémové proměnné FRAMES. Počáteční hodnota při zapnutí počítače je 0. FRAMES tak funguje jako počitadlo délky provozu počítače a může být použito k měření času (buď průběžně nebo stopky). Hodnotu FRAMES zjistíme pomocí:

PRINT PEEK 23672 + 256 * PEEK 23536 * PEEK 23674.

Max. hodnota, kterou může počítač uložit do 3 bytů FRAMES je 2 na 24 -1 = 16 777 215, což odpovídá době 3 dny, 21 hod., 12 min. a 24.3 sec. Během provádění příkazu BEEP, operací s kazetovým magnetofonem, tiskárnou a při všech nemaskovaných přerušeních nedochází k inkrementaci hodnoty FRAMES, což je třeba brát v úvahu, chceme-li FRAMES používat k přesnému měření času. Nepřesnost činí 0,01 %, t.j. cca 9 sec/den.

KSTATE - adresa 23552 až 23559 (5C00h - 5CC7h) - 8 bajtů.

Systémovou proměnnou používá počítač při vyhodnocování stlačené klávesy, resp. při vyhodnocování dvou stlačených kláves a při opakování. Osm bajtů je rozděleno do dvou čtveřic pro dvě stlačené klávesy. V prvních bajtech je FFh, není-li stlačena žádná klávesa, nebo C0h, dojde-li ke stlačení klávesy. Druhé a třetí bajty jsou nezajímavé, ve čtvrtých bajtech je kód stlačené klávesy bez ohledu na současně stlačenou SHIFT klávesu. Znamená to, že v těchto bajtech je vždy kód bílého znaku v levé horní části klávesy (písmene A až Z a číslice 1 až 0). Má to použití např. při zjišťování právě stlačené klávesy, že nemusíme brát ohled, byla-li stlačena klávesa "A" nebo "a".

K_SUR - adresa 23643 a 23644 (5C5Bh, 5C5Ch) - 2 bajty.

Adresa kursoru v opravovaném či vkládáném řádku. Je možno použít při různých programech.

K_DATA - adresa 23656 (5C08h) - 1 bajt.

Do systémové proměnné se ukládá barevná informace vložena klávesnicí před jejím dalším zpracováním.

LAST_K - adresa 23560 (5C08h) - 1 bajt.

Obsahuje kód poslední stlačené klávesy s ohledem na současné stlačení SHIFT klávesy (t.j. rozlišuje např mezi "A" a "a"). V příloze A manuálu je u kódu 15 poznámka nepoužito, ale v LAST_K se objeví kód 15 při současném stlačení CAPS SHIFT a 9 (GRAPHICS). Obdobně je kód TRUE VIDEO (CAPS SHIFT a 3) 4 a kód INV VIDEO (CAPS SHIFT a 4) 5. V manuálu taktéž označeno jako nepoužito.

LISTSP - adresa 23615 a 23616 (5C3Fh, 5C40h) - 2 bajty.

Adresa návratu z automatického listingu.

MASK_P - adresa 23604 (5C8Eh) - 1 bajt.

Obsahuje masku, kterou se přenáší barevná informace ze syst. proměnné ATTR_P (bit MASK_P AND odpovídající bit ATTR_P. Je-li bit v MASK_P nastaven, použije se hodnota odpovídajícího bitu v ATTR_P, jinak 0). Můžeme použít např. chceme-li omezit počet barev na obrazovce, nebo chceme-li např. vyloučit BRIGHT nebo FLASH.

MASK_T - adresa 23696 (5C90h) - 1 bajt.

Totéž jako MASK_P, ale pro ATTR_T.

MEM - adresa 23656 a 23657 (5C68h, 5C69h) - 2 bajty.

Adresa začátku zásobníkové paměti kalkulátoru (obvykle MEMBOT, ale ne vždy).

MEMBOT - adresa 23698 až 23727 (5C92h - 5CAFh) - 30 bajty.

Používá kalkulátor jako zásobníkovou paměť pro uložení hodnot, se kterými pracuje (blíže viz kap. ROM, RST #28).

MODE - adresa 23617 (5C41h) - 1 bajt.

Obsahuje hodnoty 0 až 2 podle typu kurzoru na obrazovce. 0 = K, L a C, 1 = E a 2 = G. Je možno použít ke změně režimu klávesnice. Např. POKE 23617,2 vyvolá režim G.

NTWPPC - adresa 23618 a 23619 (5C42h, 5C43h) - 2 bajty.

Obsahuje číslo řádku, na který má skočit příkaz GO TO nebo GO SUB. Změnou můžeme vynutit skok na danou řádku.

NSPPC - adresa 23620 (5C44h) - 1 bajt.

Obsahuje číslo příkazu v řádku, na který má skočit GO TO nebo GO SUB. Chceme-li využít skok na n-tý příkaz v řádku L, musíme provést :

```
POKE 23619,L-256*INT(L/256):POKE 23619,INT(L/256):POKE 23620,n
```

Po POKE 23618 a POKE 23619, musí vždy následovat POKE 23620.

NXTLIN - adresa 23637 a 23638 (5C55h, 5C56h) - 2 bajty.

Začáteční adresa dalšího programového řádku, který má být programem proveden.

OLDPPC - adresa 23662 a 23663 (5C6Eh, 5C6Fh) - 2 bajty.

Proměnná obsahuje číslo řádku, kterým program pokračuje po CONTINUE.

OSPPC - adresa 23664 (5C70h) - 1 bajt.

Proměnná obsahuje číslo příkazu v řádku, kterým pokračuje CONTINUE.

PIP - adresa 23609 (5C39h) - 1 bajt.

Délka trvání zvukového signálu při stlačení klávesy. Po zapojení obsahuje nulu, což odpovídá délce signálu cca 1/3 sec. Je možné měnit hodnoty od 0 do 255, čímž se délka signálu prodlužuje.

PPC - adresa 23621 a 23622 (5C45h, 5C46h) - 2 bajty.

Obsahuje číslo právě prováděného řádku.

PROG - adresa 23635 a 23636 (5C53h, 5C54h) - 2 bajty.

Adresa začátku oblasti pro uložení basicového programu. Bez připojeného Interface 1 je to 23755. Je možno použít k zabránění editování řádku např. se strojním programem v REM. Po vložení programu do REM provedeme :

```
POKE 23755, 0 : POKE 23756, 0
```

čímž jsme změnili číslo řádku REM na 00 a není možné ho normálně editovat.

PR_CC - adresa 23680 (5C80h) - 1 bajt.

Méně významný bajt příští pozice tisku při LPRINT (ve vyrovnávacím registru tiskárny - významný bajt je 5Bh).

P_FLAG - adresa 23697 (5C91h) - 1 bajt.

Návěští. Bit 0 - stav OVER dočasný, bit 1 - stav OVER stálý, bit 2 stav INVERSE dočasný, bit 3 stav INVERSE stálý, bit 4 INK 9 dočasný, bit 5 INK 9 stálý, bit 6 PAPER 9 dočasný, bit 7 PAPER 9 stálý.

P_POSN - adresa 23679 (5C7Fh) - 1 bajt.

Obsahuje číslo sloupce pozice LPRINT.

P_RAMT - adresa 23732 a 23733 (5CB4h, 5CB5h) - 2 bajty.

Adresa posledního fyzického bajtu RAM. Pro 16k RAM je to 32767 (7FFFh) a pro 48k RAM je to 65535 (FFFFh).

RAMTOP - adresa 23730 a 23731 (5CB2h, 5CB3h) - 2 bajty.

Adresa posledního bajtu basicového systému - viz kap. 3. o členění RAM.

RASP - adresa 23608 (5C38h) - 1 bajt.

Délka varovného tónu při vkládání znaků do řádku delšího, než 2 spodní řádky. Normálně obsahuje hodnotu 64 a může být měněn od 0 do 255. Pro urychlení vkládání dlouhých řádků můžete zkusit POKE 23608, 0.

REPDEL - adresa 23561 (5C09h) - 1 bajt.

Obsahuje dobu, po kterou musí být stlačena klávesa, aby došlo k opakování (v 1/50 s). Normální hodnota je 35, ale může být měněná od 0 do 255 (0 = 256).

REPPER - adresa 23562 (5C0Ah) - 1 bajt.

Délka intervalu opakování při stlačené klávese.

SCR_CT - adresa 23692 (5C8Ch) - 1 bajt.

Obsahuje počet řádků + 1, které budou scrolovány na obrazovce nahoru, než se objeví sdělení SCROLL? Pro eliminování sdělení SCROLL? můžeme použít POKE 23692,255. Pro scroll o jeden řádek můžeme použít následující program:

```
POKE 23692, 255 : PRINT AT 21, 21 '' AT 21, 0;
```

SEED - adresa 23671 (5C76h, 5C77h) - 2 bajty.

Pomocí systémové proměnné SEED generuje počítač náhodná čísla. Pokaždé při vyvolání funkce RND se hodnota v SEED změní podle následujícího programu:

```
LET SEED = 75 * (SEED + 1): LET SEED = SED - 65537 * INT
(SEED/65537) - 1
```

Funkce RANDOMIZE přenese do SEED hodnoty ze dvou nižších bajtů systémové proměnné FRAMES, která pak tvoří nový základ pro RND. Vložíme-li RANDOMIZE N, pak je v SEED hodnota N v HEX tvaru, čehož se dá použít ve strojních programech.

STKBOT - adresa 23651 a 23652 (5C63h, 5C64h) - 2 bajty.

Adresa začátku oblasti zásobníku kalkulátoru - viz rozdělení RAM a kapitola o ROM.

STKEND - adresa 23653 a 23654 (5C65h, 5C66h) - 2 bajty.

Adresa vrcholu zásobníku kalkulátoru a začátek volné paměti.

STRLEN - adresa 23666 a 23667 (5C72h, 5C73h) - 2 bajty.

Délka právě vyhodnocovaného řetězce.

STRMS - adresa 23568 až 23605 (5C10h - 5C35h) - 38 bajtů.

Adresy kanálů připojených k jednotlivým proudům. Během inicializace systému se do prvních 14 bajtů přenesou hodnoty z ROM pro stále otevřené kanály K, S, P a R. Použití při připojení Interface 1 a dalších vlastních periférií.

adresa	proud	kanál
23568	253	K
23570	254	S
23572	255	R
23574	0	K
23576	1	K
23578	2	S
23580	3	P
23582	4	
.	.	
.	.	
23604	15	

SUBPPC - adresa 23623 (5C47h) - 1 bajt.

Obsahuje číslo právě vykonávaného příkazu v basicovém řádku. Můžeme použít ve spojení s ERR_NR, chceme-li do sdělení vložit číslo řádku a příkazu v řádku, u kterého došlo k zastavení programu. Je-li l číslo řádku a n číslo příkazu v řádku, bude:

```
POKE 23621, l-256*INT(l/256)  ; číslo řádku do syst.prom.
POKE 23622, INT(L/256)        ; PPC
POKE 23623, n                 ; číslo příkazu do SUBPPC
```

S_POSN - adresa 23688 a 23689 (5C88h, 5C89h) - 2 bajty.

Obsahuje číslo sloupce (v 23688) a číslo řádku (v 23689) běžné pozice PRINT (t.j. právě vypisovaného znaku) v interním číslování podle následující tabulky. Používá se ve strojních programech - viz kap. o ROM.

Ale pozor! interně Spectrum používá jiného způsobu číslování řádků než v basicu. Toto číslování je zřejmé z následující tabulky:

Basic ----->	0	1	2	3	4	5	6	.	.	.	26	27	28	29	30	31
Interně ->	33	32	31	30	29	28	27	.	.	.	7	6	5	4	3	2
0	24															
1	23															
2	22															
V 3	21 V															
4	20															
5	19															
.	.															
.	.															
21	3															
22	2															
23	1															

SPOSNL - adresa 23690 a 23691 (5C8Ah, 5C8Bh) - 2 bajty.

Totéž jako S_POSN, ale pro spodní část obrazovky.

S_TOP - adresa 23660 a 23661 (5C6Ch, 5C6Dh) - 2 bajty.

Obsahuje číslo řádku, který bude zobrazen jako první na obrazovce při automatickém listingu.

TVDATA - adresa 23566 a 23567 (5C0Eh, 5C0Fh) - 2 bajty.

Obsahuje řídicí znaky pro AT a TAB a informaci o barvě.

TVFLAG - adresa 23612 (5C3Ch) - 1 bajt.

Návěští pro obrazovku. Je-li bit 0 nastaven, znamená to spodní část obrazovky, je-li nulován, znamená to horní část obrazu. Použití ve strojních programech při tisku na horní část obrazovky - viz kap. o ROM - RST 10. Bit 3 nastaven znamená režim EDIT, bit 4 nastaven pak automatický listing, bit 4 je-li nastaven bude po stisku klávesy dolní část obrazovky vymazána.

T_ADDR - Adresa 23668 a 23669 (5C74h, 5C75h) - 2 bajty.

Adresa příští položky v syntaktické tabulce. Pro strojní programy nepoužitelné.

UDG - adresa 23675 a 23676 (5C7Bh, 5C7Ch) - 2 bajty.

Adresa začátku oblasti pro definované grafické znaky. Normálně obsahuje 32600 (7F58) nebo 65368 (FF58) pro 16/48k RAM.

VARS - adresa 23627 a 23628 (5C4Bh, 5C4Ch) - 2 bajty.

První adresa oblasti, ve které jsou uloženy hodnoty proměnných. Délku basicového programu dostaneme odečtením hodnoty v syst. proměnné PROG od hodnoty v syst. proměnné VARS.

WORKSP - adresa 23649 a 23650 (5C61h, 5C62h) - 2 bajty.

Obsahuje adresu oblasti pracovního prostoru.

Z_PTR - adresa 23647 a 23648 (5C5Fh, 5C60h) - 2 bajty.

Adresa prvního znaku, ve kterém je syntaktická chyba, t.j. znaku následujícího po "?".

NMIREG - adresa 23728 a 23729 (5CB0h, 5CB1h) - 2 bajty.

Jsou určeny pro vektor nemaskovaného přerušení. Normálně obsahují 0000 (podle manuálu nepoužity).

Následují systémové proměnné pro Interface 1.

BAUD - adresa 23647 a 23648 (5CC3h, 5CC4h) - 2 bajty.

Rychlost přenosu informací pro RS 232 dle následujícího vztahu. Lze užít pro nastavení nestandardních rychlostí přenosu.

BAUD = (3500000 / (26 * rychlost)) - 2

CHADD - adresa 23755 a 23756 (5CCBh, 5CCCh) - 2 bajty.

Dočasná paměť pro CH_ADD.

COPIES - adresa 23791 (5CEFh) - 1 bajt.

Počet kopií tvořených instrukcí SAVE.

D STR 1 - adresa 23766 a 23767 (5CD6h, 5CD7h) - 2 bajty.

Začátek osmibajtového pole hlavičky, druhý bajt číslo mikrodrive (1 - 8) nebo číslo přijímající stanice v síti nebo rychlost přenosu seriového kanálu.

D STR 2 - adresa 23775 až 23781 (5CDEh, 5CE5h) - 8 bajtů.

Dalších osm bajtů hlavičky souboru v instrukci MOVE nebo LOAD.

FLAGS 3 - adresa 23734 (5CB6h) - 1 bajt.

Návěští: bit 0 nastaven při realizaci rozšířeného souboru instrukcí
bit 1 CLEAR
bit 2 změna ERR SP
bit 3 práce v síti
bit 4 LOAD
bit 5 SAVE
bit 6 MERGE
bit 7 VERIFY

HD 00 - adresa 23782 (5CE6h) - 1 bajt.

Kód typu dat 0 program, 1 číselná data, 2 znaková data, 3 bajty.

HD 0B - adresa 23783 a 23784 (5CE7h, 5CE8h) - 2 bajty.

Délka dat (0 - 65536)

HD 0D - adresa 23785 a 23786 (5CE9h, 5CEAh) - 2 bajty.

Adresa začátku dat (0 - 65536)

HD 0F - adresa 23787 a 23788 (5CEBh, 5CECh) - 2 bajty.

Délka programu nebo název datového pole.

HD 11 - adresa 23789 a 23790 (5CEDh, 5CEEh) - 2 bajty.

Pro číslo řádku automatického startu programu.

IOBORD - adresa 23750 (5CC6h) - 1 bajt.

Barva borderu během vstupně - výstupních operací.

L STR 1 - adresa 23769 (5CD9h) - 1 bajt.

Typ zařízení "m", "n", "t" nebo "b".

N STR 1 - adresa 23770 a 23771 (5CDAh, 5CDBh) - 2 bajty.

Délka názvu souboru.

NTDCS - adresa 23764 (5CD4h) - 1 bajt.

Kontrolní bajt bloku přenášeného v síti.

NTDEST - adresa 23758 (5CCEh) - 1 bajt.

Paměť pro číslo přijímající stanice v síti.

NTHCS - adresa 23765 (5CD5h) - 1 bajt.

Kontrolní bajt hlavičky bloku v síti.

NTLEN - adresa 23763 (5CD3h) - 1 bajt.

Délka bloku přenášeného v síti 0 - 255.

NTNUMB - adresa 23760 a 23761 (5CD0h, 5CD1h) - 2 bajty.

Počet bloků informací přenášených v síti.

NTRESP - adresa 23757 (5CCDh) - 1 bajt.

Pro kód odpovědi stanice ze sítě.

NTSTAT - adresa 23749 (5CC5h) - 1 bajt.

Vlastní číslo stanice v síti (1 - 64).

NTSRCE - adresa 23759 (5CCFh) - 1 bajt.

Paměť pro číslo vysílající stanice v síti.

NTTYPE - adresa 23762 (5CD2h) - 1 bajt.

Hlavička typu dat v síti: 0 data 1 EOF.

S STR 1 - adresa 23768 (5CD8h) - 1 bajt.

Pro čísla proudů (0 - 15)

SBTR - adresa 23737 až 23746 (5CB9h - 5CC2h) - 10 bajtů.

Podrogram vyvolání procedury z ROM Spectra:

```
LD HL,hodnota
CALL adresa
LD (23738),HL
RET
```

SECTOR - adresa 23753 a 23754 (5CC9h, 5CCAh) - 2 bajty.

Pracovní oblast pro Microdrive, obvykle slouží pro počítání sektorů.

SER_FL - adresa 23751 a 23752 (5CC7h, 5CC8h) - 2 bajty.

Pracovní oblast pro RS 232, první bajt návěští, druhý přenášený znak.

T STR 1 - adresa 23773 a 23774 (5CDCh, 5CCDh) - 2 bajty.

Adresa začátku názvu souboru.

VECTOR - adresa 23735 a 23736 (5CB7h a 5CB8h) - 2 bajty.

Adresa použita pro rozšíření interpreteru Basicu, normálně 01F0h.

To je všechno, co se podařilo o systémových proměnných zjistit.

6. Podprogramy ROM

ROM znamená "Read Only Memory" t.j. paměť, ze které je možno pouze vybírat informace. Tyto informace jsou do ROM vloženy při výrobním procesu a není možno je později měnit. Znamená to, že pro kteroukoliv adresu v ROM můžeme použít PEEK, ale POKE nejde.

Celkem má ROM Spectra 16384 bytů, proto se označuje jako 16k ROM a je pro obě verze (t.j. 16k RAM a 48k RAM) stejná.

V ROM je uložen tzv. provozní systém Spectra, t.j. všechny nutné informace pro procesor Z80 A. Bez těchto informací v ROM by byl počítač sbírkou jednotlivých součástek, neschopnou jakékoliv činnosti (některé počítače mají v ROM pouze zaváděcí program a hlavní program se musí před zahájením práce vložit buď z pásku, nebo z pružného disku. Má to tu výhodu, že můžeme používat různé programovací jazyky, pokud jsou ovšem v daném provozním systému k dispozici. Spectrum má vše co potřebuje ke své práci v ROM. Má to zase tu nevýhodu, že bez dalších kouzel může pracovat pouze v Basicu.

Teprve ten, kdo detailně ovládá ROM, je absolutním pánem Spectra a může plně využít všech výhod strojního kódu. Ale právě detailní znalost ROM a jejich jednotlivých podprogramů je nejnáročnější a nutně vyžaduje detailní prostudování některého komentovaného výpisu ROM. V této stručné příručce můžu dát jen nejzákladnější informace o ROM a jejich podprogramech.

ROM začíná na adrese 0000 a končí na adrese 16383 (3FFFh). Celou ROM můžeme rozdělit do několika hlavních částí (toto rozdělení je pouze orientační, poněvadž programy z jedné části vyvolávají podprogramy z jiné části.

0000h - 0094h restarty a návazné malé podprogramy

0095h - 028Dh tabulky všech symbolů uvedených na klávesnici

028Eh - 03B4h podprogramy pro test klávesnice a vyhodnocení stlačené klávesy

03B5h - 04A9h podprogramy pro tónový generátor

04AAh - 04C1h krátký nevyužitý podprogram ze ZX81

04C2h - 09F3h podprogramy pro obsluhu kazetového magnetofonu

09F4h - 0F2Bh podprogramy pro výpis znaku na obrazovce

0F2Ch - 11B6h podprogramy pro editování basicového řádku nebo příkazu, t.j. jeho vkládání a změny včetně INPUT

11B7h - 12A1h inicializační podprogram, který po zapojení počítače provede test RAM, nastaví veškeré počáteční hodnoty systémových proměnných a vypíše na obrazovku sdělení "1982 Sinclair Research Ltd."

12A2h - 1A47h hlavní provaděcí podprogram, který přenese vložené řádky do oblasti PROG a vypíše jejich listing na obrazovce

1A48h - 1B16h tabulky pro interpretaci basicových příkazů a jejich parametrů pro syntaktickou kontrolu

1B17h - 24FAh hlavní interpretační podprogram jednotlivých basicových příkazů a provádění syntaktického testu

24FBh - 32C4h podprogramy pro vyhodnocení výrazu v jednotlivých příkazech a funkcích a podprogramy pro matematické operace a přípravu dat pro kalkulátor

32C5h - 386Dh podprogram kalkulátoru včetně tabulky odskoku po RST 28

386Eh - 3CFFh pouze hodnoty FFh

3D00h - 3FFFh tabulka bajtů jednotlivých ASCII znaků, které mohou být vypsány na obrazovku

A teď trochu podrobněji k jednotlivým částem ROM.

0000h - 0094h Restarty

Jsou to nejdůležitější a nejčastěji používané podprogramy v ROM. Vyvolávají se instrukcemi RST. Jednotlivé restarty jsou:

0000h - RST 00 Start systému po jeho zapojení. Bezprostředně vyvolá inicializační podprogram START/NEW na adrese 11CBh.

0008h - RST 08 Podprogram ERROR-1. Je vyvolán vždy, je-li důvod k vypsání sdělení ve spodní části obrazovky (přehled sdělení je v příloze B manuálu). RST 08 můžeme použít při návratu ze strojního programu do basicu místo instrukce RET. Po instrukci RST 08 musí ale následovat bajt FFh, čímž se ukončí strojní program a přejde na pokračování basicového programu. Je možné i vynucené sdělení, když po RST 08 následuje bajt příslušného sdělení (tab. od adresy 1391h v ROM).

0010h - RST #10 Podprogram PRINT-A-1. Tento podprogram vypíše na obrazovku znak, jehož kód je v registru A (do místa běžné pozice PRINT). Je to jeden z nejuniverzálnějších podprogramů ROM. Lze ho použít ke vložení řídicích znaků obrazovky nebo ke změně kanálu. Vzhledem k jeho univerzálnosti se u něj zdržím trochu déle a předběhnu ostatní části ROM.

Napíšeme-li následující strojní program, vložíme do počítače a spustíme, objeví se na prvním řádku spodní části obrazovky písmeno A:

```
3E 41     LD A,65        ; Kód "A" do reg.A
D7        RST #10        ; vypiš znak
C9        RET            ; zpět do basicu
```

Proč znak A je snad jasné. Ale proč ve spodní části obrazovky? Je to proto, že normálně má Spectrum po zahájení činnosti otevřen kanál K, který je určený pro spodní část obrazovky. Chceme-li náš znak vypsat do horní části, musíme nejdříve otevřít kanál S, příslušný pro tuto část obrazovky (kanál P je pak pro tiskárnu). Provedeme to následujícím strojním programem:

```
3E 02     LD A,2         ; číslo proudu do A
CD 01 16  CALL #1601     ; otevři kanál S
```

CALL #1601 vyvolá podprogram na adrese 1601h, který otevře kanál S. Stejného účinku dosáhneme následujícím programem:

```
AF        XOR A          ; nuluje A
32 3C 5C  LD (#5C3C),A   ; 0 do syst.prom. TVFLAG
```

což je vlastně výsledek předešlého programu.

Napíšeme-li nyní:

```
3E 02     LD A,2          ;
CD 01 16  CALL #1601      ; kanál S
3E 41     LD A,65         ; znak A
D7        RST #10         ; výpis
C9        RET             ; zpět Basic
```

objeví se A v levém horním rohu obrazovky. Levý horní roh proto, že je to v tomto případě běžná pozice PRINT (při prázdné obrazovce)

Rychlost strojního kódu můžete vyzkoušet následujícím doplněním:

```
3E 02     LD A,2          ;
CD 01 16  CALL #1601      ; kanál S
3E 41     LD A,65         ; znak A
D7        RST #10         ; výpis
18 FB     JR $-3          ; opakování od LD A,65
C9        RET             ; zpět Basic
```

a porovnáním tohoto programu s Basicovým:

```
1 PRINT "A";
2 GO TO 1
```

Většinou však nechceme psát pouze do levého horního rohu obrazovky, ale na určité místo (např. na řádek 10 a sloupec 15). Máme pak víc možností. Jednou z nich je využít řídicích znaků obrazovky (viz manuál, kap.14 a příl.A). Použijeme řídicí znak AT, jehož kód je 22. Napíšeme-li pak program:

```
3E 02     LD A,2
CD 01 16  CALL #1601      ; kanál S
3E 16     LD A,22         ; řídicí znak AT
D7        RST #10         ; proveď
3E 0A     LD A,#0A        ; řádek 10
D7        RST #10         ; proveď
3E 0F     LD A,#0F        ; sloupec 15
D7        RST #10         ; proveď
3E 41     LD A,#41        ; znak A
D7        RST #10         ; proveď
C9        RET             ; Basic
```

objeví se znak A na požadovaném místě obrazovky.

Máme však i další možnost. Můžeme použít podprogram ROM na adrese 0DD9h, který na obrazovku nastaví pozici PRINT podle parametru v reg. páru BC, přičemž v B je číslo řádku a v C číslo sloupce (interní číslování - řádek 0Eh a sloupec 12h). Pak náš program pro výpis na obrazovku dostane předřazeny 3 řádky:

```
06 0E     LD B,#0E        ; basic řádek 10
0E 12     LD C,#12        ; basic sloupec 15
CD D9 0D  CALL #0DD9      ; PRINT AT
```

a pokračuje normálně:

```
3E 41     LD A,#41      ; znak A
D7        RST #10       ; proved
C9        RET           ; Basic
```

Jako první musí samozřejmě být:

```
3E 02     LD A,2
CD 01 16  CALL #1601    ; kanál S
```

Místo dvou instrukcí LD B,#0E a LD C,#12 můžeme napsat

```
01 12 0E  LD BC,#0E12
```

Pokud napíšeme program v následujícím pořadí:

```
          LD A,2
          CALL #1601
          LD A,#41
          LD BC,#0E12
          CALL #0DD9
          RST #10
          RET
```

nebude fungovat, resp. na obrazovce se objeví něco jiného. Je to proto, poněvadž podprogram CALL #0DD9 přepíše hodnotu v registru A. Musíme proto před vyvoláním CALL #0DD9 původní hodnotu v reg. A uchovat. Nejjednodušší způsob je PUSH AF a pak POP AF. Program pak bude vypadat následovně:

```
          LD A,2
          CALL #1601    ; kanál S
          LD A,#41      ; znak A
          LD BC,#0E12   ; řádek a sloupec
F5        PUSH AF       ; uchování hodnot v A
          CALL #0DD9    ; PRINT AT
F1        POP AF        ; původní hodnoty do A
          RST #10       ; výpis
          RET           ; Basic
```

Protože však člověk je náročný a požaduje vypsání více znaků na obrazovce najednou, můžeme pokračovat osvědčeným způsobem, poněvadž RST #10 vypíše nový znak na následující pozici PRINT, t.j. za předchozí znak. Vypadalo by to následovně:

```
          LD A,2
          CALL #1601
          LD A,#41      ; znak A
          RST #10
          LD A,#48      ; znak H
          RST #10
          LD A,#4F      ; znak O
          RST #10
          LD A,#4A      ; znak J
          RST #10
          RET
```

Je to ale zdlouhavé a "nemotorné" a zabírá zbytečně mnoho místa. Nic nám ale nebrání použít podprogramu PR-STRING na adrese 203Ch, určeného pro výpis řetězců. Před vyvoláním podprogramu musí být v DE adresa prvního znaku řetězce a v BC počet znaků v řetězci, a samozřejmě od dané adresy jednotlivé kódy znaků v řetězci. Pak nám pro vypsání "AHOJ" na obrazovku postačí následující program. (zvolil jsem jeho uložení do vyrovnávacího registru tiskárny):

```
5B00 11 06 10 02  DEFB #11,#06,#10,#02 ; PAPER 6, INK 2,
   4 12 01        DEFB #12,#01         ; FLASH 1
   6 41 48 4F 4A  DEFM "AHOJ"
   A 3E 02        LD A,02
   C CD 01 16     CALL #1601           ; kanál S
   F 11 00 5B     LD DE,#5B00          ; adresa 1. znaku
  12 01 0A 00     LD BC,#000A          ; 10 znaků
  15 CD 3C 20     CALL #203C           ; podprg. PR-STRING
5B18 C9           RET                  ; basic
```

Podprogram CALL #203C není nic jiného než smyčka, která za Vás provede opakovaně LD A,n , RST #10 z předchozího příkladu, postupně pro všechny hodnoty uložené v bajtech od adresy v registrovém páru DE.

Pomocí CALL #0DD9 je možno určit pozici tisku na obrazovce, musíme dávat ale pozor na přepsání registrů a použít PUSH/POP. Není snad třeba zdůrazňovat, že řetězec se znaky může být na zcela jiném místě paměti, než vlastní prováděcí program, který začíná obvyklým LD A,02. Znamená to, že uvedený program musíme odstartovat pomocí RAND USR 23306, nikoliv RAND USR 23296. Naše znaky, které chceme vypsat na obrazovku, včetně řídících znaků obrazovky, můžeme dokonce uložit do řetězcové proměnné (např B$). Pak ovšem musíme před vyvoláním programu zjistit adresu začátku naší řetězcové proměnné v oblasti VARS, zvýšit tuto hodnotu o 3 a přenést do DE. V bajtech VARS + 1 a VARS + 2 je délka řetězce, kterou přeneseme do BC (viz kap.2).

V předchozím podprogramu jsme v jednotlivých bajtech na začátku programu (5B00h) definovali barvu pro PAPER a INK a blikání nápisu. Můžeme si to zvolit proto, že pomocí RST #10 můžeme realizovat veškeré řídící znaky na obrazovku, nikoliv pouze AT, jak již bylo uvedeno. Zásada je, že po řídícím znaku musí následovat RST #10 a pak příslušné parametry řídícího znaku, následované vždy RST #10.

Je třeba ještě upozornit, že RST #10, podobně jako CALL #0DD9, přepíše všechny registry. Používáme-li BC jako počítadlo při opakování, musíme před každým vyvoláním RST #10 pomocí PUSH BC obsah BC uložit do zásobníku a pak vrátit pomocí POP BC. Pochopitelné, že pro všechny uvedené řídící znaky obrazovky můžeme použít podprogramu PR-STRING podle příkladu na předchozí straně.

0018h - RST #18 Podprogram GET-CHAR (přisuň znak) uloží do A kód aktuálního znaku, t.j. znaku adresovaného syst. proměnnou CH_ADD

0020h - RST #20 Podprogram NEXT-CHAR (přisuň následující znak) uloží do A kód znaku následujícího po znaku, který by do A uložil RST #18.

Co je to ale aktuální a následující znak? Nejlépe je to vidět, vložíme-li následující krátké programy:

```
RST #18      a    RST #20
RST #10           RST #10
RET               RET
```

a odstartujeme je PRINT USR XXXX,A (kde XXXX je adresa začátku strojního programu). První program vypíše na obrazovku "," a druhý vypíše "A". Je to proto, že prvním znakem na který počítač po provedení PRINT USR XXXX narazí, je právě čárka a dalším znakem je A.

Pomocí smyčky tak na obrazovku můžeme vypsat i více znaků, následujících po příkazu PRINT USR XXXX. Je přitom ale důležité mít na paměti, že RST #20 zanedbává mezery (a samozřejmě přepisuje registry). Napíšeme-li smyčku ve tvaru:

```
ZAC     RST #20         ; přisuň následující znak
        RST #10         ; vypiš znak
        JR ZAC          ; znovu
```

a odstartujeme PRINT USR XXXX, ABCDE, pak program sice vypíše na obrazovku požadované "ABCDE", ale pak si dělá co chce, poněvadž mu chybí RET pro návrat do basicu. Můžeme tomu odpomoci tím, že si zvolíme návratový znak, při němž se program vrátí. Může to vypadat třeba následovně:

```
ZAC     RST #20         ; znak do A
        CP #30          ; je v A znak "0" ?
        RET Z           ; ano, zpět do basicu
        RST #10         ; ne, vypíše znak
        JR ZAC          ; opakování
```

Když pak program spustíme PRINT USR XXXX, ABCDE0, vypíše nám na obrazovku "ABCDE". Znak 0 pak znamená konec a návrat do basicu pomocí instrukce RET Z. Zkusíme-li tento program vyvolat PRINT USR XXXX, AB CD E0, pak se na obrazovce objeví stejně jako předtím "ABCDE" bez mezer.

Chceme-li nápis umístit do jiného místa obrazovky, musíme podobně jako v předchozí části, nastavit kanál a pozici PRINT na obrazovce.

Znaky, které chceme zobrazit, můžeme také umístit do jiného místa v paměti a změnit adresu aktuálního znaku v syst. proměnné CH_ADD.

0028h - RST #28 Podprogram FP-CALC. Tento podprogram je sice sám o sobě velice jednoduchý, má pouze 3 bajty, ale vzápětí vyvolá podprogram CALCULATE na adrese 335Bh, což je kalkulátor. Jeho pomocí je možno provádět veškeré funkce včetně rozhodování a skoku. Poněvadž se jedná o velice užitečný podprogram, zase přeskočím a uvedu ho zde celý.

Podprogram kalkulátoru vyvoláváme instrukcí RST #28, po níž následuje kód požadované operace kalkulátoru. Jednotlivé kódy a jimi vyvolané operace jsou:

kód	příslušná činnost
00	je-li výrok pravdivý, relativní skok daný hodnotou následujícího bajtu (jako JR, n)
01	výměna poslední a předposlední hodnoty na vrcholu zásobníku
02	vynuluje vrchol zásobníku
03	odečte hodnotu uloženou pod vrcholem zásobníku od hodnoty na vrcholu zásobníku a výsledek uloží na vrchol zásobníku (-)
04	vynásobí dvě hodnoty uložené na vrcholu zásobníku a pod ním a výsledek uloží na vrchol zásobníku (*)
05	vydělí vrchol hodnotou pod vrcholem a výsledek uloží na vrchol zásobníku (:)
06	umocní vrchol hodnotou pod vrcholem a výsledek uloží na vrchol zásobníku (X na Y)
07	provede funkci X OR Y, výsledek je na vrcholu zásobníku a je X je-li Y=0, jinak 1 (OR)
08	provede X AND Y, výsledek na vrcholu je X, je-li Y<>0, jinak 0 (AND)
09	provede <= pro čísla, výsledek na vrcholu je 0-nepravda 1-pravda (<=)
0Ah	provede >= pro čísla, výsledek jako 09 (>=)
0Bh	provede <> pro čísla, výsledek jako 09 (<>)
0Ch	provede > pro čísla, výsledek viz 09 (>)
0Dh	provede < pro čísla, výsledek viz 09 (<)
0Eh	provede = pro čísla, výsledek viz 09 (=)
0Fh	sečte vrchol a hodnotu pod vrcholem a výsledek uloží na vrchol zásobníku (+)
10h	provede $ AND číslo, výsledek na vrcholu je X$, je-li Y<>0, jinak prázdný řetězec (X$ AND Y)
11h	provede $ <= , výsledek viz 09
12h	provede $ >= , výsledek viz 09
13h	provede $ <> , výsledek viz 09
14h	provede $ > , výsledek viz 09
15h	provede $ < , výsledek viz 09
16h	provede $ = , výsledek viz 09
17h	provede X$ + Y$, výsledek uloží na vrchol
18h	provede VAL$, výsledek uloží na vrchol
19h	provede USR$, výsledek uloží na vrchol
1Ah	provede READ-IN, t.j. přenese číslo z vrcholu do registru A, a současně změní na číslo typu integer
1Bh	provede změnu znaménka hodnoty na vrcholu (NEG)
1Ch	provede CODE

kód	příslušná činnost
1Dh	provede VAL
1Eh	provede LEN
1Fh	provede SIN
20h	COS
21h	TAN
22h	ARC SIN = ASN
23h	ARC COS = ACS
24h	ARC TAN = ATN
25h	LN
26h	EXP = E^X
27h	INT
28h	SQR
29h	SGN
2Ah	ABS
2Bh	PEEK
2Ch	IN
2Dh	provede USR N, vyvolá strojní program na adrese dané hodnotou na vrcholu zásobníku, po ukončení tohoto podprogramu se na vrchol uloží obsah BC
2Eh	STR$
2Fh	CHR$
30h	provede NOT, uloží na vrchol 1, byla-li předchozí hodnota na vrcholu 0, jinak dá na vrchol 0
31h	provede DUP, okopíruje vrchol na nový vrchol zásobníku
32h	provede dělení N modulo M, t.j. vydělí číslo N číslem M a výsledek uloží následovně: INT(N/M) na vrchol a zbytek pod vrchol, před vyvoláním je M na vrcholu a N pod
33h	provede nepodmíněný relativní skok o velikosti dané následujícím bajtem (JR n)
34h	uloží na vrchol číslo typu floating point, jehož hodnota je dána následujícími pěti bajty
35h	provede relativní skok daný následujícím bajtem a sníží hodnotu v syst.proměnné BREG o 1, pokračuje, dokud není BREG = 0 (jako DJNZ),do BREG dáme předem počet opakování
36h	provede < 0 vrcholu zásobníku, výsledek na novém vrcholu bude 1, je-li splněno, jinak 0
37h	provede > 0, ostatní viz 36h
38h	konec výpočtu kalkulátoru
39h	provede redukci argumentu SIN a COS v rozmezí -0.5 a 0.5
3Ah	provede oddělení celočíselné části se zaokrouhlením
3Bh	FP-CALC-2, je to podprogram pro provedení jediné aritm. operace, jejíž kód je v syst. proměnné BREG
3Ch	převede číslo ve tvaru xEm na floating point (x na vrcholu, m je v reg.A)
3Dh	převede číslo z vrcholu typu integer na floating point
86h	generuje polynomický rozvoj pro výpočet trigonometrických funkcí, následuje 6 konstant float.p.
88h	viz 86h, ale 8 konstant
8Ch	viz 86h, ale 12 konstant
A0h	uloží na vrchol 0
A1h	uloží na vrchol 1
A2h	uloží na vrchol 0.5

kód	příslušná činnost
A3h	uloží na vrchol PI/2
A4h	uloží na vrchol 10
C0h	přenese číslo typu floating point z vrcholu do zásobníkové paměti č.0
C1h	viz C0h, ale do paměti 1
C2h	viz C0h, ale do 2
C3h	viz C0h, ale do 3
C4h	viz C0h, ale do 4
C5h	viz C0h, ale do 5
E0h	přenese číslo ze zásobníkové paměti č.0 na vrchol
E1h	viz E0h, ale z paměti 1
E2h	viz E0h, ale z 2
E3h	viz E0h, ale z 3
E4h	viz E0h, ale z 4
E5h	viz E0h, ale z 5

Jako zásobníkovou paměť používá kalkulátor 30 bajtů systémové proměnné MEMBOT. Při přenosech ze zásobníku do zásobníkové paměti zůstává hodnota na vrcholu zásobníku zachována, při přenosu ze zásobníkové paměti na vrchol zásobníku se přenesena hodnota umistí na vrchol zásobníku a původní hodnota na vrcholu zásobníku se přesune pod vrchol.

Tím jsem vyčerpal všechny funkce kalkulátoru. Zůstává otevřená otázka, jak přenést počáteční hodnoty na vrchol zásobníku, poněvadž instrukce RST #28 předpokládá, že na vrcholu zásobníku je již dána hodnota, se kterou se bude provádět předepsaná funkce.

V ROM Spectra je pro danou úlohu několik podprogramů:

CALL 2D28h - STACK-A přenese hodnotu z registru A na vrchol zásobníku a převede ho na číslo typu floating point
CALL 2D2Bh - STACK-BC přenese BC viz CALL #2D28
CALL 2AB1h - STK-ST-0 přenese B na vrchol a pak CDEA pod vrchol
CALL 24FBh - SCANNING vyhodnotí výraz a výsledek uloží na vrchol

Pro opačný postup, t.j. přesunutí hodnot z vrcholu zásobníku do registru jsou v ROM následující podprogramy:

CALL 2DD5h - FP-TO-A přenese hodnotu z vrcholu zásobníku do registru A
CALL 2314h - STK-TO-A stejné, jako CALL #2DD5, navíc v C 1, je-li A kladné, nebo FFh je-li záporné
CALL 2DA2h - FP-TO-BC přenese z vrcholu hodnotu do BC a současně uloží do A nižší bajt výsledku
CALL 2307h - STK-TO-BC provede totéž, jako CALL #2DA2, navíc do DE uloží znaménko výsledku (1, FFh viz výše)
CALL 2BF1h - STK-FETCH uloží 5 bajtů z vrcholu do registru BCDEA
CALL 2DE3h - vypíše na obrazovku poslední hodnotu t.j. vrchol

Můžeme také použít kód 34h, jehož pomocí lze na vrchol zásobníku uložit číslo v pětibajtovém vyjádření. Nejlépe ale přiblížíme činnost kalkulátoru na příkladech. Chceme-li vypočítat odmocninu (sin x^2 + cos x^2), můžeme postupovat následovně:

```
   LD A,x      ;                      hodnota x do A
   CALL #2D28  ; x                    přenese hodnotu z A na vrchol
   RST #28     ; x                    zásobníku a začne výpočet
   DEFB #31    ; x,x                  zdvojí vrchol zas. (DUP)
   DEFB #1F    ; x,sin x              vypočte sin x
   DEFB #31    ; x,sinx,sinx          zdvojí sin x
   DEFB #04    ; x,sin x^2            vypočte sin x * sin x
   DEFB #01    ; sin x^2,x            přehodí x a sin x^2
   DEFB #20    ; sin x^2,cos x        vypočte cos x
   DEFB #31    ; sinx^2,cosx,cosx     zdvojí cos x
   DEFB #04    ; sin x^2,cos x^2      vypočte cos x * cos x
   DEFB #0F    ; sin x^2+cos x^2      sečte sin x^2 a cos x^2
   DEFB #28    ; odm.sinx^2+cosx^2    vypočte odmocninu
   DEFB #38    ; -------!!-------     konec výpočtu
   CALL #2DA2  ;                      přenese poslední hodnotu do BC
   RET         ;                      zpět do Basicu
narůstání zásobníku ----------->
```

Pro výpočet můžeme použít i např. zásobníkové paměti kalkulátoru. Zde pozor, protože při výpočtu trigonometrických a jiných funkcí používá kalkulátor zásobníkové paměti č. 0 až 2.

První řádek je sice jednoduchý, ale málo univerzální. Můžeme použít např.: RST #20 ; přenese hodnotu do registru A. Zde musíme program spustit PRINT USR XXXX,x, kde x je naše číslo.

Prostřednictvím kalkulátoru můžeme provádět podmíněné i nepodmíněné skoky, vyvolávat další strojní podprogramy, manipulovat s řetězci, provádět logické operace apod.

0030h - RST #30. Podprogram vyvolá další podprogram, jehož prostřednictvím vytvoří BC volných bajtů v oblasti pracovního prostoru (workspace). BC = obsah registrového páru BC.

0038h - RST #38. Podprogram MASK-INT. 50 krát za sekundu vyvolá ULA obvod maskované přerušení, během něhož provádí Z80 test klávesnice a vyhodnocuje stlačenou klávesu. Přerušení vyvolá RST #38. Každé přerušení inkrementuje počítadlo v systémové proměnné FRAMES a vyvolá podprogram testující stav klávesnice, tj. byla-li stlačena klávesa a která.

Je třeba se ještě zmínit o způsobu, jak je prováděn test klávesnice. Celá klávesnice je rozdělena do osmi bran, jejichž adresy jsou v následující tabulce:

adr.segmentu	klávesnice	adr.segmentu
F7FEh <63486>	1 2 3 4 5 6 7 8 9 0	EFFEh <61438>
FBFEh <64510>	Q W E R T Y U I O P	DFFEh <57342>
FDFEh <65022>	A S D F G H J K L en	BFFEh <49150>
FEFEh <65278>	cs Z X C V B N M ssp	7FFEh <32766>

Podprogram na adrese 028Eh testuje postupně jednotlivé segmenty pomocí instrukce IN A,(C), přičemž adresu segmentu uloží do BC. Výsledkem tohoto podprogramu jsou hodnoty v DE. Není-li stisknuta žádná klávesa, DE = FFFFh, v opačném případě obsahuje registr E hodnotu mezi 00 a 27h, které jsou přiřazeny jednotlivým klávesám.

Tabulka hodnot:

1-5...	24h	1Ch	14h	0Ch	04h	03h	0Bh	13h	1Bh	23h ...6-0
Q-T...	25h	1Dh	15h	0Dh	05h	02h	0Ah	12h	1Ah	22h ...Y-P
A-G...	26h	1Eh	16h	0Eh	06h	01h	09h	11h	19h	21h ...G-en
cs-V...	27h	1Fh	17h	0Fh	07h	00h	08h	10h	18h	20h ...B-sp

Dojde-li ke stlačení současně s jednou "shift" klávesou, obsahuje registr D hodnotu "shift" klávesy. Dojde-li ke stlačení obou "shift" kláves, obsahuje D hodnotu Caps Shift a E hodnotu Symbol Shift. Je-li stlačena klávesa "A", je v DE hodnota FF26h, jsou-li současně stlačeny SS a "A", pak je v DE 1826h. Caps Shift a Symbol Shift dají 2718h. Někdy je možno tohoto podprogramu a hodnot v DE použít ve strojních programech pro testování stlačené klávesy.

Po ukončení popsaného podprogramu pokračuje provádění dalšího podprogramu od adresy 02C3h, jehož výsledkem je vypočtení kódu stlačené klávesy (včetně všech "shiftu") a jeho uložení do systémové proměnné LAST_K. Tím vlastně končí činnost RST #38.

Adresou 0038h končí podprogramy tzv. nulté stránky vyvolávané instrukcí RST a začínají podprogramy vyvolávané instrukcí CALL.

0066h - zde začíná obsluha nemaskovaného přerušení. Spectrum při své normální činnosti NMI nepoužívá, ale je možné jej tímto podprogramem obsloužit.

0095h - tabulka všech "tokens" a znaků.

028Eh - podprogram pro test klávesnice (popsaný již u RST #38), vyvolaný podprogramem KEYBOARD.

02BFh - podprogram KEYBOARD, vyvolaný RST #38.

03B5h - od této adresy začínají podprogramy tónového generátoru. Zabudovaný reproduktor je aktivován bitem 4 portu FEh. Následuje-li po OUT (#FE) bit 4=0, je reproduktor aktivován, je-li bit 4=1, je reprodutor vypnut. Rytmem zapínání a vypínání reproduktoru je daná frekvence tónu.

Podprogram na adrese 03B5h můžeme využít pro generování tónu následujícím programem:

```
LD DE,délka tónu    ;např. 0105h pro 1 sec.
LD HL,výška tónu    ;např. 066Ah pro střední "C"
CALL #03B5          ;tón
RET                 ;Basic
```

Frekvence středního "C" je 261,63Hz. Pro jeho dosažení musí být reproduktor 261 krát aktivován a vypnut, t.j.261,63 x 2 = 523,26 strojních instrukcí. Doba mezi jednotlivými strojními instrukcemi 1/523,26 = 1,911ms. Při kmitočtu 3,5MHz je T = 285,7ns. Pro dosažení požadovaného intervalu je třeba 1,911 / 0,0002857 = 6689 strojních cyklů. V našem případě je pak 6689 / 4 - 30,125 = 1642D = 066Ah, (30,125 je korekce), vloženo do registru HL. Hodnota 0105h v DE je frekvence tónu * doba trvání v sekundách. Tedy 261,63 * 1.

Obdobně lze vypočítat hodnoty pro jiné tóny, i když rychlejší asi bude metoda zkusmého dosažení hodnot.

Stejného výsledku dosáhneme použitím podprogramu pro funkci BEEP na adrese 03F8h. Před voláním musí být hodnoty pro dobu i výšku tónu na vrcholu zásobníku. Tyto hodnoty se však od předchozích liší.

Je zřejmé, že podprogram na adrese 03B5h přepisuje všechny registry, které je tudíž nutné uložit instrukcí PUSH. Během provádění též nejdou hodiny.

04C2h - zde začínají podprogramy pro obsluhu magnetofonu, tj. funkce SAVE, LOAD, VERIFY a MERGE.

Hlavní podprogram začíná na adrese 0605h a podle hodnoty systémové proměnné TADDR vyvolá příslušnou funkci.

```
TADDR-E0h = 0   -  SAVE
            1   -  LOAD
            2   -  VERIFY
            3   -  MERGE
```

Jednotlivé funkce vyvoláme pomocí:

```
POP  AF
LD   A,0 až 3  ;podle požadované funkce
CALL #060B
```

Podprogramy pro jednotlivé funkce začínají na adresách:

```
07CBh - VERIFY
0808h - LOAD
08B6h - MERGE
0970h - SAVE
```

Vlastní nahrávání pro LOAD, VERIFY a MERGE na adrese 0556h a při SAVE na adrese 04C2h.

Těchto podprogramů se při strojním programování bude používat pravděpodobně málo, nebot se většinou nahrávání provádí již v basicovém programu.

09F4h - na této adrese začíná program pro vypsání znaku uloženého v registru A buď na obrazovku nebo tiskárnou. Tento podprogram je vyvolán podprogramem RST #10 poté, co je do HL přenesena ze systémové proměnné CURCHL velikost odskoku pro výpis na obrazovku. Tento program končí tím, že přenese na PRINT-pozici do registru obrazovky (Display file) jednotlivé bajty znaku, který má být vypsán. Tyto bajty jsou uloženy v tabulce znaků, začínající na adrese 3D00h. Z registru obrazovky jsou znaky vydány na obrazovku prostřednictvím ULA obvodu, tj. hardwarově.

PRINT-pozici určuje podprogram na adrese 0B03h v závislosti na obsahu systémových proměnných SPOSN a DF CC a podle nastavení bitu 1 v systémové proměnné FLAGS (nastavení bitu znamená tisk tiskárnou) a dále podle nastavení bitu 0 v systémové proměnné TVFLAG (je-li nulován je tisk v horní části obrazovky, je-li nastaven, pak v dolní části - v tom případě se používají hodnoty ze systémových proměnných SPOSNL a DFCCL). Hodnota ze SPOSN je v BC a hodnota z DF CC je v HL.

Pro výpis na horní část obrazovky můžeme tento podprogram upravit následovně:

```
LD  BC,číslo řádku a sloupce ;řádek v B,sloupec v C  )*
LD  HL,adr.znaku v registru obrazovky ;první bajt
XOR A
LD  (#5C3C),A  ;nastaví kanál S pro horní část obrazovky
CALL #0B10
```

/* poznámka: číslování řádků a sloupců podle tabulky v kapitole o systémových proměnných

Z této části ROM lze použít ještě několik dalších podprogramů:

0D6Bh - podprogram CLS vymaže obrazovku.

0E44h - podprogram pro vymazání B spodních řádků obrazovky. B je hodnota v registru B.

0E00h - podprogram pro scrollování obrazovky. Počet scrollovaných řádku v registru B.

0E38h - podprogram scrolluje atributy.

0E88h - podprogram vypočte k dané adrese bajtu v Display file adresu příslušného atributu v registru atributů. Adresa bajtu je v HL, výsledek v BC.

0EACh - příkaz COPY.

0ECDh - tiskárna vytiskne 256 bajtů uložených ve vyrovnávacím registru tiskárny (adresa 5B00h až 5BFFh).

0EDFh - vynuluje vyrovnávací registr tiskárny.

0F2Ch - zde začíná podprogram pro editování řádku buď při jeho vkládání z klávesnice, nebo při editaci řádku z obrazovky. Též obsluhuje vkládání dat příkazem INPUT.

0FF3h - příkaz EDIT. Přenese do spodní části obrazovky kurzorem označený řádek.

0FF3h - posune kurzor dolů.

1007h - posune kurzor doleva.

100Ch - posune kurzor doprava.

1015h - posune kurzor doleva a vymaže znak před ním.

1059h - posune kurzor nahoru.

10A8h - podprogram vyhodnotí kód stlačené klávesy a uloží do systémové proměnné LAST K.

11B7h - od této adresy začíná inicializace systému po příkazu NEW (NEW = CALL 11B7h).

11CBh - od této adresy inicializace systému po zapnutí počítače.

11DAh - začatek testu RAM.

1219h - základní nastavení systémových proměnných. Inicializace končí vypsáním "Copyrightu" na obrazovce.

12A9h - následuje hlavní prováděcí podprogram, řídící editování, provádění přímých příkazů a vydávání sdělení. V této části jsou podprogramy o kterých již byla zmínka, např. podprogram na adrese 15F2h pro výdej znaku, na adrese 1601h pro otevření kanálu aj. Z dalších je možno použít:

15D4h - podprogram čeká na stlačení libovolné klávesy a pak pokračuje v provádění následující části programu. Používá se např. u LOAD

15EFh - podprogram pro konečnou fázi vypsání znaků na obrazovku, tj. přenesení příslušného bajtu z tabulky znaků umístěné od adresy 3D00h do Display file, odkud jsou již znaky dále přenášeny hardwarově pomocí ULA.

1601h - podprogram pro otevření kanálu. V registru A musí být číslo odpovídajícího proudu. Je to FDh, 00 nebo 01 pro kanál K (spodní část obrazovky), FEh nebo 02 pro kanál S (horní část obrazovky), FFh pro kanál R (pracovní prostor) a 03 pro kanál P (tiskárna). Výsledkem otevření jednotlivých kanálů je pak následující nastavení bitů v systémových proměnných:

```
kanál K: SET 0, TVFLAG ;spodní část obrazovky
         RES 5, FLAGS  ;je možno klávesu
         SET 4, FLAGS2 ;kanál K
         RES 1, FLAGS  ;ne tiskárna
kanál S: RES 0, TVFLAG ;celá obrazovka
         RES 1, FLAGS  ;ne tiskárna
kanál P: SET 1, FLAGS  ;tiskárna použita
```

1655h - podprogram vytvoří potřebný počet volných bajtů. Počet bajtů v BC. HL ukazuje na první pozici nad nově vytvořenou zónou volných bajtů. Po skončení podprogramu je v HL adresa bajtu před začátkem nově vytvořené zóny, v DE poslední adresa této zóny, čili volné bajty jsou od HL + 1 do DE.

160Ch - podprogram k prohledávání tabulek v ROM. V HL je adresa začátku tabulky a v C je kód hledaného znaku. Nalezení znaku ohlásí nastavení indikátoru přenosu (SCF).

17F9h - podprogram pro LIST.

17F5h - podprogram pro LLIST.

1855h - vypíše na obrazovku basicový řádek, jehož počáteční adresa (tj.adresa 1. bajtu čísla řádku) je v HL.

196Eh - vyhledá adresu počátku basicového řádku, jehož číslo je v HL. Po skončení podprogramu je v HL adresa začátku hledaného řádku, případně adresa následujícího (není-li v programu zadaný), v DE je začátek předchozího řádku.

1A1Bh - vypíše na obrazovku obsah registrového páru BC (v hexa tvaru).

1B17h - začátek hlavního podprogramu překladače basicových příkazů a syntaktického textu.

Některé basicovské příkazy:

1B8Ah - RUN k provedení přímého příkazu (řádku ve spodní části obrazovky).

1BB2h - REM

1CEEh - STOP

1CF0h - příkaz IF. Před vyvoláním podprogramu musí být na vrcholu zásobníku kalkulátoru hodnota výrazu mezi IF a THEN.

1D03h - FOR. Před vyvoláním musí být na vrcholu zásobníku hodnoty řídící proměnné a její koncové hodnoty.

1DA8h - NEXT

1DECh - READ

1E27h - DATA

1E42h - RESTORE

1E4Fh - RANDOMIZE

1E5Fh - CONTINUE

1E67h - GO TO

1E7Ah - OUT. Parametry musí mít na zásobníku.

1E80h - POKE. Parametry na zásobníku.

1EA1h - RUN

1EACh - CLEAR

1EEDh - GO SUB

1F1Ah - vypočte adresu prvního volného místa v paměti. Celkový rozsah volné paměti pak získáme pomocí PRINT 65536 - USR 7962 ;1F1Ah = 7962. (PRINT 32768 - USR 7962 pro 16k.)

1F23h - RETURN

1F3Ah - PAUSE. Hodnota na zásobníku.

1F54h - test klávesy BREAK.

1F60h - DEF FN

1FC9h - LPRINT

1FCDh - PRINT

2089h - INPUT

2294h - BORDER

22AAh - vypočte adresu bodu jemné grafiky na obrazovce. V BC je adresa bodu (x,y) (u PLOT, DRAW a CIRCLE se souřadnice ve strojním kódu shodují s basicovými, tj. levý dolní roh (0,0), pravý horní (255,175)). Po skončení podprogramu je v HL adresa příslušného bajtu v Display file a v A je pozice bajtu.

22CBh - POINT. Před vyvoláním musí být na zásobníku kalkulátoru souřadnice bodu (x,y), po skončení je poslední hodnota na zásobníku 1, je-li pixel barvou ink, nebo 0, je-li pixel barvou paper.

22DCh - PLOT. Před vyvoláním na zásobníku (x,y) - poslední hodnota.

22E5h - totéž jako 22DCh, rozdíl je v zadání souřadnic bodu: v B je y a v C je x. Bod o souřadnicích x = 100d y = 50d zobrazíme pomocí

```
LD   A,02
CALL #1601        ;kanál S
LD   BC,#3264     ;y=50d, x=100d
CALL #22E5        ;podpr. PLOT
RET               ;zpět Basic
```

2307h - STK-TO-BC (viz odstavec kalkulátor).

2314h - STK-TO-A (viz kalkulátor).

2320h - začátek podprogramu CIRCLE. Vlastní podprogram začíná na adrese 232Dh. Před vyvoláním musí být na zásobníku kalkulátoru hodnoty x, y a r (poslední hodnota). Basicovému CIRCLE 100,100,50 bude odpovídat strojní program:

```
LA   A,02
CALL #1601        ;kanál S
LD   A,#64        ;x=100d do A
CALL #2D28        ;A na zásobník
LD   A,#64        ;y=100d do A
CALL #2D28        ;A na zásobník
LA   A,#32        ;r=50d do A
CALL #2D2B        ;A na zásobník
CALL #232D        ;CIRCLE
RET
```

Pokud tento program nebude správně fungovat, vlož před LD A,#64 ještě: EXX / PUSH HL / EXX a za CALL #232D: EXX / POP HL / EXX. Je to uchování obsahu registrového páru HL.

2382h - začátek DRAW.

24B7h - rovnou přímku z bodu x0,y0 do bodu x,y zobrazíme buď pomocí CALL #24B7, když jsme před tím uložili na zásobník kalkulátoru hodnoty x,y a do systémové proměnné COORDS hodnoty x0,y0.

24BAh - totéž jako 24B7h, jen jiné zadání souřadnic: v B je ABS y, v C je ABS x, v D je SGN y, v E je SGN x, COORDS jako v předchozím podprogramu. Přímku z bodu 50,50 do bodu 100,100 vykreslíme pomocí:

```
LD   A,02
 CALL #1601        ;kanál S
 EXX
 PUSH HL
  EXX               ;uchování HL
 LD   A,#32        ;x0=50d do A
 LD   (#5C7D),A    ;A do COORDS-x       (pokud je x<>y
 LD   (#5C7E),A    ;A do COORDS-y        je třeba LD A,y)
  LD   BC,#6464     ;x=y=100d do BC
 LD   DE,#0101     ;x i y kladné hodnoty
 CALL #24BA        ;podprogram DRAW
 EXX
 POP HL
 EXX               ;do HL původní hodnota
 RET               ; zpět Basic
```

2394h - zobrazení části kruhu (basicky DRAW x,y,a). Hodnoty x,y,a (poslední hodnota) jsou na zásobníku kalkulátoru, x0 a y0 v COORDS.

24FBh - od této adresy začínají podprogramy vyhodnocení výrazu. Sám podprogram 24FBh je velmi důležitý (tzv. Scanning podprogram). Vyhodnotí výraz a výsledek uloží na vrchol zásobníku kalkulátoru, odkud si pak vyhodnocené hodnoty vybírají další podprogramy pro příkazy. K tomuto podprogramu malý příklad: Chceme vyhodnotit (vypočíst) hodnotu vyrazu PI * 3 + 2. Mohli bychom použít kalkulátoru, ale následujícím strojovým programem to jde rychleji.

```
  RST #20     ;přisuň následující znak
  CALL #24FB ;SCANN-vypočte hodnotu výrazu a uloží na SP
  CALL #2DE3 ;vypíše obsah vrcholu SP na obrazovku
  RET         ;Basic
```

Tento podprogram musíme vyvolat: PRINT USR xxxxx,PI*3+2 Obdobně můžeme vyhodnotit pro Spectrum přípustné výrazy včetně výrazu se řetězci a pod. Jedná-li se o výrazy s řetězci, pak podprogram CALL #24FB uloží na zásobník pět bajtů, v nichž jsou následující hodnoty:

1. bajt není definován
2. a 3. bajt je adresa začátku řetězce
4. a 5. bajt je délka řetězce

2AFFh - LET

2C02h - DIM

2D28h - STACK-A (viz odstavec kalkulátor)

2D2Bh - STACK-BC dtto

2D3Bh - podprogram Integer to floating point. Změní číslo typu integer uložené na SP na tvar floating point a uloží ho tamtéž.

2D4Fh - začátek aritmetických podprogramů používaných kalkulátorem. Sám podprogram na adrese 2D4Fh změní číslo typu x E n, tj. desetinné exponenciální číslo na typ float point, které uloží na SP.

335Bh - na této adrese začíná podprogram kalkulátoru a dál následují jednotlivé operace kalkulátoru vyvolané pomocí kódu za RST #28.

3D00h - na této adrese začíná tabulka znaků, které může Spectrum zobrazit na obrazovce (nejedná se o tzv. definované znaky). Každý znak sestává z 8 po sobě následujících bajtů. Adresa prvního bajtu znaku je (kód znaku - 32) * 8 + 3D00h. Např. kód "a" je 61h, první bajt znaku "a" v tabulce je na adrese (61h - 32) * 8 + 3D00h = 3F08h. Znak z tabulky do Display file (a tím také na obrazovku) přeneseme následujícím strojním programem:

```
        LD HL,nn        ; cílová adresa v display file
        LD DE,mn        ; adresa počátku znaku v tab. =
                        ; (kód - 32) * 8 + 3D00h
        LD B,#08        ; počitadlo - 8 bajtů
ZNOVU   LD A,(DE)       ; bajt do A
        LD (HL),A       ; a pak do displ.file
        INC H           ; adresa dalšího bajtu display file
        INC DE          ; další bajt v tabulce
        DJNZ ZNOVU      ; skok na opakování
```

Pro stejný účel můžeme samozřejmě použít i příslušný program z ROM.

Tím je vyčerpaná stručná (a upřímně řečeno velice kusá) informace o ROM a jejich podprogramech.

---===***===---

Opravil a doplnil : Bohuslav Labaj s kolegy, Třinec r. 1988.

Natištěno pro ZO SVAZARM Karolinka

březen 1989