

# ASSEMBLER

*a  
ZX Spectrum*

**2.díl**

# Úvodem

Vítám Vás v dalším díle knihy **Assembler a ZX Spectrum**, doufám, že jste na toto setkání čekali netrpělivě nebo alespoň čekali (kdyby ne, asi byste tyto řádky nečetli). Doufám, že Vám první díl alespoň jednou dobře posloužil (musím se přiznat, že několikrát jsem se do knihy podíval). Nyní se zaměříme na grafiku - bude to zřejmě zajímavější, než to, čím jsme se zabývali minule.

Nejprve však musím uvést na pravou míru některé (ty, o kterých jste mi napsali) chyby, které se do minulého dílu dostaly při přepisování, a také doplnit to, co se do prvního dílu mojm nedopatřením dokonce vůbec nedostalo (ač zcela zjevně mělo). Chtěl bych poděkovat panu Pavolu K. z Rožumberoku, který našel (a hlavně mi o nich napsal) nejvíce chyb, nebyl však jediný a proto děkuji i všem ostatním.

Jednotlivé chyby budu uvádět vždy s číslem stránky a řádku, na kterém se vyskytují, případně některými dalšími informacemi, které Vám pomohou chybu nalézt. Pokud se na stránce vyskytují nějaké tabulky, berte při počítání v úvahu jen řádky s textem nebo výpisem strojového kódu, případně prázdné řádky nepočítejte.

**Strana 9, řádek 16** - na konci řádku je napsáno slovo **nesmíte** a správně mělo být **nesmíte zapomenout** nebo **musíte**.

**Strana 16, řádek 9** - místo instrukce **ld hl, 40000** má být napsáno **ld hl,30000**.

**Strana 18, řádek 18** - tady je hned několik chyb, všechny však vyplývají z toho, že jsem si spletl výsledek u operace **bitový XOR**. Vyměňte si v textu předposledního odstavce (popis funkce XOR) číslo 0 za číslo 1 a naopak - operace XOR má výsledek 1 právě tehdy, když jsou oba operandy různé, a hodnotu 0 právě tehdy, když jsou. oba operandy stejné. Opravte si také mezivýsledek v příkladu.

**Strana 24, grafické znázornění funkce RRA** - šipka má ukazovat na opačnou stranu, tedy doprava (podle názvu instrukce Rotate Right Accumulator).

**Strana 25, grafické znázornění funkce RR** - stejná chyba jako v minulém případě, šipka opět směřuje na opačnou stranu než by měla.

**Strana 28, popis skokových instrukcí** - zde by měl být také popis **instrukcí CALL, RST a RET**.

Vezmeme je tedy stručně - instrukce **call NN** je určena pro volání (případně podmíněné) podprogramů (je to jistá obdoba BASICovské instrukce **GO SUB**). Instrukce tedy uloží na zásobník adresu následující instrukce a skočí na adresu, která je uvedena jako operand. Pokud se jedná o podmíněné volání, provádí se pouze v případě, že je splněna příslušná podmínka.

Nepodmíněná instrukce **call NN** je jen jedna a její vykonání trvá **17 T-cyklů**. Podmíněných instrukcí **call cc,NN** je celkem 8 (jsou stejné jako instrukce **JP cc,NN**). Časová náročnost těchto instrukcí je **17 T-cyklů** v případě, že se instrukce provádí, a **10-T cyklů** v případě, že se neprovádí. Délka instrukce je tři byty.

S instrukcemi **call NN** úzce souvisí instrukce **rst N**. Instrukce **rst N** je vlastně jakási kratší a rychlejší varianta instrukce **call NN** - adresa podprogramu, který má být volán je zakódována přímo do kódu instrukce, z toho plyne také omezení adres, které je možno za číslo N dosadit. Instrukce je určena pro volání podprogramů na adresách 0, 8, 16, 24, 32, 40, 48, 56 - tedy (na Spectru) podprogramů v ROMce. Funkce k provedení potřebuje **11 T-cyklů** a je dlouhá jeden byte.

Jakými doplňkem instrukcí **call** jsou instrukce **ret**. Tyto instrukce odebírají ze zásobníku hodnotu a vkládají ji do PC registru. Opět máme podmíněné a nepodmíněné verze instrukce (stejně jako u **call** a **jp**). Nepodmíněná verze instrukce trvá **10 T-cyklů**, podmíněná pak buď **11 T-cyklů** (pokud podmínka platí) nebo **5 T-cyklů** (pokud podmínka neplatí). Instrukce **ret** i **ret cc** jsou dlouhé jeden byte.

Aby byl soupis úplný, musíme u instrukcí **ret** uvést ještě instrukce **retn** (return from non-maskable interrupt) a **reti** (return from interrupt), které se používají pro návrat z nemaskovatelného a maskovatelného přerušení. Oproti instrukci **ret** tyto instrukce navíc ještě generují určitý signál, který může používat připojená periférie - my tyto instrukce používat nebudeme. Obě jsou dlouhé dva byty a trvají **celkem 14 T-cyklů**.

**Strana 33, druhá tabulka** - zde má být zapsána instrukce **out (N),a**.

**Strana 69, řádek 23** - za instrukcí **ld a,(LINE)** chybí instrukce **inc a**.

**Strana 79, řádek 9** - místo instrukce **add hl,hl** má být **add hl,de**.

**Strana 95, řádek 2 odspoda** - v popisu podprogramu **INPCLEAR** je místo registru **de** uveden registr **hl**.

**Strana 101, řádky 4 až 8** - zde je poněkud rozsáhlejší „chyba“ - píše to raději v uvozovkách, protože to je chyba pouze v případě, že si uvedený příklad přeložíte pod hranici 32768. Napíše zde to, co mi o tom napsal pan Pavol K.

Tato chyba je tzv. „chuťovka“, na kterou jsem přišel úplně náhodou. Nachází se na straně 101 v podprogramu označeném jako „ošetření **DELETE**“. Jedná se o část podprogramu mezi návěstími **IP4** a **IP7**. Když jsem celý program psal (vlastně jen

---

podprogram **INPCOM**) poprvé, nijak jsem zmiňované části nerozuměl, ale když mi to fungovalo správně, už jsem se tím nezabýval. Ale jen do chvíle, než jsem program přeložil od nižší adresy (kolem 25000). Program při stisku **DELETE** v případě, že nebyl zadán žádný text, havaroval. Když jsem vzápětí tentýž zdrojový text přeložil na výš (konkrétně na 60000), fungoval bez závad. Po následujícím zkoumání mě v podprogramu (**IP4**) zaujaly dvě instrukce - **dec hl** a **bit 7,(hl)**. Když však není zadán žádný text (**HL** registr ukazuje na první byte jakéhosi minibufferu za návěštím **INLIN2**) a stisknete **DELETE**, tak po instrukci **dec hl** nám registr **HL** ukazuje na vyšší byte adresy **TT2** (před **INLIN** je přeci **call TT2**). A tady je ten problém. Když je adresa **TT2** větší nebo rovna 32768, pak je vše v pořádku (program funguje), protože vyšší byte adresy **TT2** je větší nebo roven 128 a jeho 7. bit je tedy logická 1. V opačném případě se instrukce **call TT2** přepíše a program skončí úplně někde jinde. Jedno z možných řešení je například takovéto:

```

IP4      cp    12           ;test na kód DELETE
        jr    nz,IP8       ;odskok
        ld    bc,INLIN2    ;do BC adresu INLIN2
        or    a           ;vynuluj CARRY flag
        sbc   hl,bc        ;odečti od HL obsah BC (nastaví ZERRO)
        add  hl,bc        ;přičti zpátky (nezmění obsah ZERRO)
        jr    z,IP2       ;odskoč, pokud je prázdný buffer
        ld    (hl),32     ;proved DELETE

IP7

```

Z uvedeného programu vidíme, že vždy před provedením **DELETE** se testuje, jestli se v **HL** registru není hodnota **INLIN2**. V kladném případě je kurzor na začátku textu a **DELETE** není možné provést.

To jsou tedy všechny chyby, o kterých vím.

# *Hýbeme obrazem*

V této kapitole si povíme něco o tom, jak se dělají takové věci, jako je skrolování a rolování obrazovky či její části, na konci si ukážeme nějaký ten rolující text.

Pro ty, co neví, co je to skrolování a rolování, napřed malé vysvětlení: Skrolování je posouvání obsahem obrazovky v nějakém směru (vodorovném nebo svislém). To, co obrazovku opustí, se v ní již neobjeví. Rolování je skoro totéž, co skrolování, liší se tím, že to, co obrazovku na jedné straně opustí, se do ní na druhé straně ihned vrací.

Začneme od nejjednoduššího - budeme skrolovat celou obrazovku všemi hlavními směry (vodorovně a svisle) po bodech a nebudeme hýbat s atributy. Při svislém skrolování

(nahoru a dolů) budeme přesunovat jednotlivé řádky bodů vybraným směrem - vzhledem k organizaci obrazovky tedy budeme přesunovat celé byty. Uvedeme si nejprve skrolování obrazovky směrem nahoru:

```

VYSKA equ 192 ;výška skrolované části v bodech
SIRKA equ 32 ;šířka skrolované části v bytech

SCR_UP ld hl,16384 ;adresa levého horního rohu obrazovky
ld b,VYSKA-1 ;do dej B počet mikrořádků menší o 1
SCU1 push hl ;ulož adresu mikrořádku na zásobník
call DOWNHL ;spočítej adresu dalšího mikrořádku
pop de ;do DE adresu cílového mikrořádku
ld a,b ;uschovej počítadlo řádků do registru A
ld bc,SIRKA ;délka mikrořádku v bytech
push hl ;ulož adresu zdrojového mikrořádku
ldir ;přesuň mikrořádek nahoru
pop hl ;obnov adresu v obrazovce
ld b,a ;vrať do B počet mikrořádků
djnz SCU1 ;zacyklení přes počet řádků
ret ;návrát z podprogramu

DOWNHL inc h ;posun ukazatele o jeden bod dolů
ld a,h
and 7
ret nz
ld a,1
add a,32
ld l,a
ld a,h ;tento podprogram byl již vysvětlen
jr c,DOWNHL2 ;v minulém díle této knihy
sub 8
ld h,a
DOWNHL2 cp 88
ret c
ld h,64
ret

```

Uvedený program tedy začíná s adresou nejvyššího mikrořádku na obrazovce v registru HL, potom ji uloží na zásobník, vypočte adresu následujícího mikrořádku (nyní je to mikrořádek **pod**) a do registru DE obnoví adresu horního mikrořádku. Nyní tedy máme v HL adresu druhého mikrořádku na obrazovce, v DE pak adresu prvního mikrořádku na obrazovce a můžeme tedy pomocí instrukce LDIR provést přesun obsahu. Předtím ovšem uložíme adresu druhého mikrořádku, budeme ji totiž potřebovat. Na konci otestujeme, jestli jsme přesunuli všechny mikrořádky, a pokud ne, pak se vrátíme do cyklu na začátek a přesunujeme další mikrořádek.

Skrolování obrazovky dolů se provádí obdobně jako skrolování nahoru, opět si uvedeme výpis, můžete jej připsat k již napsanému skrolu nahoru, pokud tak neučiníte, opište první dva řádky předchozího výpisu - jsou na nich definovaná návěští **SIRKA** a **VYSKA**, která se v tomto podprogramu také používají:

```

SCR_DOWN ld    hl,22528-32    ;adresa nejnižšího mikrořádku
          ld    b,VYSKA-1     ;počet skrolovaných mikrořádků
SCD1     push  hl             ;ulož adresu mikrořádku na zásobník
          call  UPHL          ;spočítej adresu dalšího mikrořádku
          pop   de            ;do DE adresu cílového mikrořádku
          ld   a,b            ;uschovej počítadlo řádků do registru A
          ld   bc,SIRKA       ;délka mikrořádku v bytech
          push hl             ;ulož adresu zdrojového mikrořádku
          ldir                ;přesuň mikrořádek nahoru
          pop  hl             ;obnov adresu v obrazovce
          ld   b,a            ;vrať do B počet mikrořádků
          djnz SCD1          ;zacyklení přes počet řádků
          ret                 ;návrat z podprogramu

UPHL     ld    a,h            ;posun ukazatele o jeden bod nahoru
          dec  h
          and  7
          ret  nz
          ld  a,1
          sub 32
          ld  l,a
          ld  a,h            ;tento podprogram byl již vysvětlen
          jr  c,UPHL2        ;v minulém díle této knihy
          add a,8
          ld  h,a
UPHL2    cp   64
          ret  nc
          ld  h,87
          ret

```

Tento program se od předchozího liší tím, že prochází mikrořádky místo shora dolů v opačném pořadí - zdola nahoru.

U obou programů můžete měnit velikost skrolované plochy pomocí návěstí **VYSKA** a **SIRKA**. Výšku můžete volit libovolně v rozmezí 2 až 192 bodů, šířku pak v rozmezí 1 až 32 bytů, tedy osminásobků bodů. Pokud změníte v instrukci **ld hl,číslo** uvedenou hodnotu, můžete skrolovat libovolným kusem obrazovky - při nastavování si však dejte pozor, program není nijak ošetřen proti možným chybám v parametrech.

Dále se budeme zabývat skrolováním do stran - nebude to již tak jednoduché jako v předchozím případě, budeme totiž muset pohybovat jednotlivými bity v bytech. Nejjednodušší bude, když si nejdříve vyzkoušíte příklad - připište jej k již napsanému textu a pokud potřebujete jen konkrétní rutinu, opište alespoň definice návěstí **SIRKA** a **VYSKA** a také celý podprogram **DOWNHL**:

```

SCR_RGHT ld    hl,16384      ;adresa začátku prvního mikrořádku
          ld    c,192        ;výška skrolované oblasti
SCR1     push  hl             ;uložíme ukazatel pro pozdější použití
          ld   b,SIRKA       ;počet bytů, které budeme skrolovat
          or   a              ;vynulování přenosu - vstupuje nula
SCR2     rr   (hl)           ;rotace obsahu bytu doprava s přenosem
          inc  l              ;posun na další byte (vpravo)
          djnz SCR2          ;opakuj pro celý řádek

```

```

pop hl ;obnov ukazatel na mikrořádek
call DOWNHL ;posuň se na další mikrořádek
dec c ;počet mikrořádků zmenši o jedničku
jr nz,SCR1 ;a pokud nejsou všechny jdi pro další
ret ;hotovo, vrať se

SCR_LEFT ld hl,16384+31 ;adresa konce prvního mikrořádku
ld c,192 ;výška skrolované oblasti
SCL1 push hl ;uložíme ukazatel pro pozdější použití
ld b,SIRKA ;počet bytů, které budeme skrolovat
or a ;vynulování přenosu - vstupuje nula
SCL2 rl (hl) ;rotace obsahu bytu doleva s přenosem
dec l ;posun na další byte (vlevo)
djnz SCL2 ;opakuji pro celý řádek
pop hl ;obnov ukazatel na mikrořádek
call DOWNHL ;posuň se na další mikrořádek
dec c ;počet mikrořádků zmenši o jedničku
jr nz,SCL1 ;a pokud nejsou všechny jdi pro další
ret ;hotovo, vrať se

```

Program pro skrol doprava pracuje tak, že nastaví HL na začátek prvního mikrořádku a připraví počítadlo mikrořádků. Pak si dočasně uloží adresu mikrořádku na zásobník a provede s každým mikrořádkem posun doprava - nejprve nastaví počítadlo bytů a vynuluje příznak přenosu, potom vždy zarotuje příslušný byte doprava (zleva vstupuje to, co zbylo z minulé rotace - napoprvé je to nula) a posune se na další byte, což opakuje pro celý řádek. Po posunutí jednoho mikrořádku se obnoví ukazatel na jeho začátek, spočítá se adresa následujícího mikrořádku a vše se opakuje pro další mikrořádek.

Další, co byste mohli chtít s obrazovkou dělat, je rolovat ji v jednom ze základních směrů, pro tento účel je potřeba programy pro scroll poněkud upravit - trochu se tím prodlouží a zkomplikují, jsou zde vypsány opět všechny čtyři kombinace - tentokrát je program napsán tak, aby pohyboval jen částí obrazovky:

```

SIRKA equ 20 ;šířka rolované oblasti v bytech
VYSKA equ 64 ;výška rolované oblasti v bodech

ROL_UP ld hl,16452 ;adresa bytu v levém horním rohu
push hl ;ulož adresu
ld de,BUFFER ;adresa oblasti pro úschovu
call LDIR20 ;přesuň řádek do bufferu
pop hl ;obnov ukazatel na začátek
ld b,VYSKA-1 ;do B počet přesunovaných mikrořádků
RLUI push hl ;uschovej ukazatel na začátek řádku
call DOWNHL ;spočítej adresu následujícího řádku
pop de ;v DE je předchozí řádek
ld a,b ;ulož počítadlo mikrořádků do A
ld bc,SIRKA ;do BC šířku rolované oblasti
push hl ;ulož ukazatel na začátek
ldir ;přesuň jeden mikrořádek
pop hl ;obnov ukazatel na začátek

```

```

                ld    b,a                ;vrať do B počet mikrořádků (výšku)
                djnz RLU1                ;konec cyklu přes mikrořádky
RLCM           ex    de,hl                ;do DE adresa posledního mikrořádku
                ld    hl,BUFFER          ;do HL adresu BUFFERu, je v něm 1. řádek
LDIR20        ld    bc,SIRKA            ;do BC počet bytů mikrořádku
                ldird                ;přesuň obsah
                ret                    ;vrať se

ROL_DOWN     ld    hl,20480-28-192      ;adresa bytu v levém dolním rohu
                push hl                ;ulož adresu
                ld    de,BUFFER          ;adresa oblasti pro úschovu
                call LDIR20              ;přesuň řádek do bufferu
                pop  hl                ;obnov ukazatel na začátek
                ld    b,VYSKA-1         ;do B počet přesunovaných mikrořádků
RLD1         push hl                ;uschovej ukazatel na začátek řádku
                call UPHL                ;spočítej adresu následujícího řádku
                pop  de                ;v DE je předchozí řádek
                ld    a,b                ;ulož počítadlo mikrořádků do A
                ld    bc,SIRKA           ;do BC šířku rolované oblasti
                push hl                ;ulož ukazatel na začátek
                ldird                ;přesuň jeden mikrořádek
                pop  hl                ;obnov ukazatel na začátek
                ld    b,a                ;vrať do B počet mikrořádků (výšku)
                djnz RLD1                ;konec cyklu přes mikrořádky
                jr    RLCM               ;skoč do společného konce

```

V těchto dvou podprogramech je jedna nešikovnost, kdyby se odstranila, byl by každý podprogram kratší o 2 byty, jistě na ni přijdete sami ale odstraňte ji až poté, co si program uložíte a bude vám fungovat.

```

ROL_RGHT     ld    hl,16452              ;adresa levého horního rohu
                ld    e,VYSKA           ;do C počet mikrořádků
RLR1         push hl                ;ulož adresu mikrořádku
                push hl                ;a ulož ji ještě jednou
                ld    de,SIRKA-1        ;do DE dej šířku řádku zmenšenou o 1
                add  hl,de              ;posuň se na konec mikrořádku pro
                ld    a,(hl)            ;bit (bod), který má opustit řádek
                rra                    ;vpravo a dej ho do CARRY
                pop  hl                ;obnov ukazatel na začátek mikrořádku
                ld    b,SIRKA           ;do B dej počet bytů na řádku
RLR2         rr    (hl)                ;rotuj doprava s přenosem
                inc  l                  ;posuň se na další byte
                djnz RLR2                ;opakuj pro každý byte
                pop  hl                ;obnov ukazatel na začátek mikrořádku
                call DOWNHL             ;posuň se na další mikrořádek
                dec  c                  ;zmenši počet řádků
                jr    nz,RLR1           ;pokud nejsi na nule skoč na začátek
                ret                    ;vrať se

ROL_LEFT     ld    hl,16451+SIRKA       ;adresa pravého horního rohu
                ld    e,VYSKA           ;do C počet mikrořádků
RLL1         push hl                ;ulož adresu mikrořádku
                push hl                ;a ulož ji ještě jednou
                ld    de,-SIRKA+1       ;záporně šířka řádku zmenšená o 1
                add  hl,de              ;posuň se na konec mikrořádku pro
                ld    a,(hl)            ;bit (bod), který má opustit řádek

```



```

        rla                ;vlevo a dej ho do CARRY
        pop hl             ;obnov ukazatel na začátek mikrořádku
        ld b,SIRKA        ;do B dej počet bytů na řádku
RLL2   rl (hl)            ;rotuj doleva s přenosem
        dec l              ;posuň se na další byte
        djnz RLL2         ;opakuj pro každý byte
        pop hl             ;obnov ukazatel na začátek mikrořádku
        call DOWNHL       ;posuň se na další mikrořádek
        dec c              ;zmenši počet řádků
        jr nz,RLL1        ;pokud nejsi na nule skoč na začátek
        ret               ;vrať se

BUFFER  defs SIRKA       ;pro uložení jednoho mikrořádku

```

Nezapomeňte připsat podprogramy **UPHL** a **DOWNHL**. Všimněte si, že tyto podprogramy se od svých „skrolovacích“ předchůdců liší jen tím, že si uchovávají (horizontální roly) nebo předem zjišťují (vertikální roly) to, co by u skrolů bylo při posunu zničeno nebo přepsáno.

Nakonec části věnované pouze pixelům si ukážeme příklad, jak lze spojit dva podprogramy tak, aby se obraz posunoval v šikmém směru - můžete samozřejmě střídavě volat vodorovný a svislý posun, zkuste to a uvidíte, že to není zrovna nejhezčí, proto raději použijeme speciální podprogram pro tento případ (skrolujeme doprava nahoru):

```

SCR_UPRT ld hl,16384      ;adresa levého horního rohu oblasti
        ld c,VYSKA-1     ;počet mikrořádků
SCR1   push hl            ;ulož ukazatel na začátek mikrořádku
        call DOWNHL       ;spočítej adresu dalšího mikrořádku
        pop de            ;původní ukazatel obnov do DE
        push hl           ;uschovej ukazatel na spodní řádek
        ld b,SIRKA        ;šířka mikrořádku v bytech
        or a              ;vynuluj příznak CARRY
SCR2   ld a,(hl)          ;vyzvedni byte ze spodního mikrořádku
        rra               ;zarotuj jím doprava
        ld (de),a         ;zapiš na horní řádek v posunutém tvaru
        inc l             ;posuň se pro další byte
        inc e             ;na obou řádcích
        djnz SCR2         ;opakuj pro každý byte na řádku
        pop hl            ;obnov ukazatel na spodní řádek
        dec c             ;zmenši počítadlo řádků o jedničku
        jr nz,SCR1        ;a pokud nejsi na nule tak cykli
        ret               ;vrať se

```

Zatím jsme obsah obrazovky posunovali pouze po jednom bodu a nijak jsme se nestarali o atributy. Občas však budeme potřebovat posunovat barevným obrázkem a tady je nutné provádět posun nikoliv o bod ale o celých osm bodů najednou tak, aby se s body mohly posunout také atributy a celý obrázek se nezměnil:

```

SIRKA equ 25 ;šířka rolované oblasti (ve znacích)
VYSKA equ 20 ;výška rolované oblasti (ve znacích)

ROL_RGHT ld hl,16451+SIRKA ;adresa bytu v pravém horním rohu
          ld c,VYSKA-1 ;do C počet řádků
RLR1     push hl ;uložíme adresu konce řádku
          ld b,8 ;8 mikrořádků na znakovém řádku
RLR2     push bc ;ulož počítadla na zásobník
          push hl ;ulož adresu konce řádku
          ld a,(hl) ;vezmi poslední byte na mikrořádku
          push af ;a ulož jej na zásobník
          ld e,1 ;přesuň obsah z registru HL
          ld d,h ;do registru DE
          dec hl ;posuň se pro předchozí byte
          ld bc,SIRKA-1 ;do BC počet přesunovaných bytů
          ldrr ;proved posun doprava
          pop af ;obnov hodnotu posledního bytu na řádku
          ld (de),a ;a zapiš ji na jeho začátek
          pop hl ;obnov adresu konce řádku
          inc h ;posuň se na další mikrořádek
          pop bc ;obnov počítadla řádků
          djnz RLR2 ;uzavři cyklus přes 8 mikrořádků
          pop hl ;obnov ukazatel na konec mikrořádku

          push hl ;a opět jej ulož na zásobník
          call ATTRADR ;vypočítej adresu atributu
          push bc ;uschovej počítadlo řádků
          ld a,(hl) ;vezmi poslední byte na mikrořádku
          push af ;a ulož jej na zásobník
          ld a,1 ;přesuň obsah z registru HL
          ld d,h ;do registru DE
          dec hl ;posuň se pro předchozí byte
          ld bc,SIRKA-1 ;do BC počet přesunovaných bytů
          ldrr ;proved posun doprava
          pop af ;obnov hodnotu posledního bytu na řádku
          ld (de),a ;a zapiš ji na jeho začátek
          pop bc ;obnov počítadlo řádků
          pop hl ;obnov ukazatel na konec řádku
          call DOWNCH ;posuň se na spodní znakovou pozici
          dec c ;zmenši počet řádků o jedničku
          jr nz,RLR1 ;a pokud nejsou na nule, opakuj přesun
          ret ;vrať se

```

Podprogram pro rolování doprava tedy prochází jednotlivé znakové řádky rolované oblasti a provádí s nimi rolování - nejprve zaopatří osm pixelových řádků a nakonec vždy atributový řádek.

```

ROL_LEFT ld hl,16452 ;adresa bytu v levém horním rohu
          ld c,VYSKA-1 ;do C počet řádků
RLR1     push hl ;uložíme adresu počátku řádku
          ld b,8 ;8 mikrořádků na znakovém řádku
RLR2     push bc ;ulož počítadla na zásobník
          push hl ;ulož adresu počátku řádku
          ld a,(hl) ;vezmi první byte na mikrořádku
          push af ;a ulož jej na zásobník

```

```

ld e,l ;přesuň obsah z registru HL
ld d,h ;do registru DE
inc hl ;posuň se pro následující byte
ld bc,SIRKA-1 ;do BC počet přesunovaných bytů
ldir ;proved' posun doleva
pop af ;obnov hodnotu prvního bytu na řádku
ld (de),a ;a zapiš ji na jeho konec
pop hl ;obnov adresu počátku řádku
inc h ;posuň se na další mikrořádek
pop bc ;obnov počítadla řádků
djnz RLL2 ;uzavři cyklus přes 8 mikrořádků
pop hl ;obnov ukazatel na začátek mikrořádku

push hl ;a opět jej ulož na zásobník
call ATTRADR ;vypočítej adresu atributu
push bc ;uschovej počítadlo řádků
ld a,(hl) ;vezmi první byte na mikrořádku
push af ;a ulož jej na zásobník
ld e,l ;přesuň obsah z registru HL
ld d,h ;do registru DE
inc hl ;posuň se pro následující byte
ld bc,SIRKA-1 ;do BC počet přesunovaných bytů
ldir ;proved' posun doleva
pop af ;obnov hodnotu prvního bytu na řádku
ld (de),a ;a zapiš ji na jeho konec
pop bc ;obnov počítadlo řádků
pop hl ;obnov ukazatel na začátek řádku
call DOWNCH ;posuň se na spodní znakovou pozici
dec c ;zmenši počet řádků o jedničku
jr nz,RLL1 ;a pokud nejsi na nule, opakuj přesun
ret ;vrať se

```

Podprogram pro rolování doleva je jen malou modifikací programu pro rolování doprava, rozdíl je pouze ve způsobu zpracování jednoho mikrořádku (atributů).

```

ROL_UP ld hl,16452 ;adresa bytu v levém horním rohu
push hl ;ulož adresu počátku řádku na zásobník
ld de,BUFFER ;adresa pomocné paměti do DE
call LINE_BUF ;ulož celý řádek do pomocné paměti
pop hl ;obnov adresu počátku řádku
ld b,VYSKA-1 ;výška rolované oblasti
RLU1 push hl ;ulož adresu počátku řádku
call DOWNCH ;spočítej dolní znakovou pozici
pop de ;obnov adresu počátku řádku
push bc ;ulož počítadlo řádků
push hl ;ulož adresu počátku řádku
call MOVELINE ;přesuň řádek nahoru
pop hl ;obnov adresu počátku řádku
pop bc ;obnov počítadlo řádků
djnz RLU1 ;konec cyklu přes řádky

RLCM ex de,hl ;přesuň do DE adresu posledního řádku
BUF_LINE ld hl,BUFFER ;do HL adresu počátku pomocné paměti
ld bc,SIRKA ;do BC šířka řádku
push de ;ulož adresu řádku (pro atributy)

```

```

ld a,8 ;řádek se skládá, z 8 mikrořádků
BUF_LIN2 push de ;ulož adresu mikrořádku
push bc ;ulož počet bytů na řádku
ldir ;přesuň řádek z bufferu do obrazovky
pop bc ;obnov délku mikrořádku
pop de ;obnov adresu mikrořádku
inc d ;a posuň se na další mikrořádek
dec a ;zmenši počítadlo mikrořádků
jr nz,BUF_LIN2 ;a pokud nejsi na nule jdi do cyklu
pop de ;obnov adresu počátku řádku
ex de,hl ;přesuň tuto adresu do HL
call ATTRADR ;a spočítej adresu atributu
ex de,hl ;vrať adresu do DE
ldir ;proved přesun atributů
ret ;vrať se

```

Podprogram pro rolování nahoru nejprve uloží celý horní znakový řádek do pomocné paměti, potom přenáší jednotlivé řádky nahoru a nakonec přidá uložený znakový řádek. Pro uložení a vybraní znakového řádku do pomocné paměti slouží podprogramy **LINE\_BUF** a **BUF\_LINE**. Pro přesun znakového řádku na obrazovce z jednoho místa na druhé slouží podprogram **MOVELINE**. Všechny podprogramy pracují nejprve s mikrořádky a nakonec s atributy.

```

ROL_DOWN ld hl,20480+160+4 ;adresa spodního znakového řádku
push hl ;ulož adresu počátku řádku na zásobník
ld de,BUFFER ;adresa pomocné paměti do DE
call LINE_BUF ;ulož celý řádek do pomocné paměti
pop hl ;obnov adresu počátku řádku
ld b,VYSKA-1 ;výška rolované oblasti
RLD1 push hl ;ulož adresu počátku řádku
call UPCH ;spočítej horní znakovou pozici
pop de ;obnov adresu počátku řádku
push bc ;ulož počítadlo řádků
push hl ;ulož adresu počátku řádku
call MOVELINE ;přesuň řádek nahoru
pop hl ;obnov adresu počátku řádku
pop bc ;obnov počítadlo řádků
djnz RLD1 ;konec cyklu přes řádky
jr RLCM ;skoč do přesunu řádku z bufferu

```

Podprogram pro rolování dolů se opět příliš neliší od rolování nahoru.

```

SCR_UPLT ld hl,16452 ;adresa bytu v levém horním rohu
ld b,VYSKA-1 ;výška rolované oblasti
SCRUL1 push hl ;ulož adresu počátku řádku
call DOWNCH ;spočítej dolní znakovou pozici
pop de ;obnov adresu počátku řádku
push bc ;ulož počítadlo řádků
push hl ;ulož adresu počátku řádku
inc hl ;posuň se o jeden znak doleva
ld bc,SIRKA-1 ;do BC délku přesunované části
call MOVELINE2 ;přesuň řádek nahoru a doleva

```

```

pop hl ;obnov adresu počátku řádku
pop bc ;obnov počítadlo řádků
djnz SCRUL1 ;konec cyklu přes řádky
ret ;vrať se

```

Předchozí podprogram provádí skrolování doleva nahoru, je to vlastně jakási kombinace předchozích podprogramů. Možná vás napadlo, že by se skrolování (rolování) šikmými směry dalo udělat tak, že by se zavolal podprogram pro skrolování (rolování) vodorovným směrem a pak totéž pro směr svislý - můžete si to vyzkoušet - vypadá to škubaně a proto je lepší napsat zvláštní program.

```

MOVELINE ld bc,SIRKA ;počet bytů pro přesun
MOVELINE2 push hl ;ulož „odkud“ (druhý vstupní bod)
           push de ;ulož „kam“
           ld a,8 ;počet mikrořádků
MOVELIN2 push hl ;ulož znovu „odkud“
           push de ;ulož znovu „kam“
           push bc ;ulož počet bytů
           ldir ;přesuň jeden mikrořádek
           pop bc ;obnov počet bytů
           pop de ;obnov ukazatel „kam“
           pop hl ;obnov ukazatel „odkud“
           inc h ;posuň se na další mikrořádek
           inc d ;posuň se na další mikrořádek
           dec a ;zmenš počet mikrořádků
           jr nz,MOVELIN2 ;a pokud není nulový jdi pro další
           pop hl ;obnov ukazatel „kam“
           call ATTRADR ;a spočítej odpovídající atributy
           ex de,hl ;tento ukazatel patří do DE
           pop hl ;obnov ukazatel „odkud“
           call ATTRADR ;a spočítej odpovídající atributy
           ldir ;proved přesun atributů
           ret ;a vrať se

ONE_BUF push hl ;ulož ukazatel „odkud“
         ld b,8 ;počet mikrořádků
ONE_BU2 push hl ;ulož ukazatel „odkud“
         ld a,(hl) ;načti obsah adresy v HL (obrazovka)
         ld (de),a ;a zapiš na adresu v DE (paměť)
         inc de ;posuň ukazatel do paměti
         inc h ;posuň ukazatel na obrazovce
         djnz ONE_BU2 ;opakuj osmkrát pro celý znak
         pop hl ;odeber adresu znaku
         call ATTRADR ;spočítej adresu atributů
         ldi ;přesuň atribut do paměti
         ret ;a vrať se

LINE_BUF ld bc,SIRKA ;počet bytů na řádku
          push hl ;ulož ukazatel „odkud“
          ld a,8 ;počet mikrořádků
LINE_BU2 push hl ;ulož ukazatel „odkud“
          push bc ;ulož počet bytů
          ldir ;proved přesun

```

```

pop bc ;obnov počet bytů
pop hl ;obnov ukazatel „odkud“
inc h ;posuň se na další mikrořádek
dec a ;zmenši počet mikrořádků
jr nz,LINE_BU2 ;a pokud nejsi na nule opakuj přesun
pop hl ;obnov ukazatel na začátek řádku
call ATTRADR ;spočítej adresu příslušného atributu
ldir ;přesuň atributy
ret ;a vrať se

ATTRADR ld a,h ;vezmi horní byte adresy
rrca ;a zarotuj ho celkem
rrca ;třikrát doprava,
rrca ;je to vlastně dělení 8
xor %1010000 ;tímto se vytvoří číslo 88,89 nebo 90
ld h,a ;vrať novou hodnotu do horního bytu
ret ;vrať se

```

Uvedený podprogram **ATTRADR** vypočítá z adresy znaku adresu odpovídajícího atributu. V prvním díle je na straně 43 uveden kus programu, který dělá přesně totéž, co tento podprogram (začíná instrukcí **ld a,h**) - je však poněkud delší. Tuto část můžete nahradit uvedeným podprogramem (kromě závěrečné instrukce **ret**) - ušetříte čtyři byty. Pokud chcete lépe pochopit to, co se děje, proveďte si uvedené operace s čísly, která přicházejí v úvahu, jsou to 64, 72 a 80, a to v binárním tvaru (pokud pracujete s **PROMETHEem**, budete to mít jednodušší, stačí totiž jen třikrát protrasovat uvedený podprogram s jednotlivými čísly).

```

DOWNCH ld a,1 ;nejprve posun v rámci třetiny
add a,32 ;šířka řádku je 32 bytů
ld l,a ;hodnotu zpátky do dolního bytu
ret nc ;vrať se když není přechod přes třetinu
ld a,h ;došlo k přechodu přes třetinu a proto
add a,8 ;upravíme také horní byte adresy
ld h,a ;a vrátíme jej do registru H
cp 88 ;ještě budeme testovat, jestli jsme
ret nz ;neopustili obrazovku, vrať se když ne
ld h,64 ;jinak nastavíme první řádek obrazovky
ret ;a vrátíme se zpátky

UPCH ld a,1 ;nejprve posun v rámci třetiny
sub 32 ;šířka řádku je 32 bytů
ld l,a ;hodnotu zpátky do dolního bytu
ret nc ;vrať se když není přechod přes třetinu
ld a,h ;došlo k přechodu přes třetinu a proto
sub 8 ;upravíme také horní byte adresy
ld h,a ;a vrátíme jej do registru H
cp 57 ;ještě budeme testovat, jestli jsme
ret nz ;neopustili obrazovku, vrať se když ne
ld h,80 ;jinak nastavíme poslední řádek
ret ;obrazovky a vrátíme se zpátky

```

Uvedené dva podprogramy počítají adresu znakové pozice **pod** a **nad** znakovou pozicí, na kterou ukazuje registr HL.

```
BUFFER  defs 9*SIRKA           ;místo pro uložení jednoho řádku
```

Tímto delším příkladem jsme probrali ty nejpoužívanější způsoby rolování a skrolování obrazovky. Ještě si povíme něco o tom, jak by se dala uvedená příklady zrychlit. Způsoby, které si ukážeme jsou obecnější a můžete je použít i jinde.

První možností je „rozvinutí cyklů“, znamená to, že instrukci, kterou provádíme v cyklu, rozepíšeme tolikrát, kolikrát se instrukce v cyklu opakuje - samozřejmě, že to nelze použít v případě, že se počet průchodů cyklem mění (ono to jde, ale není zrovna jednoduché). V našem příkladě můžeme například místo sekvence instrukcí:

```

ld  b,32           ;počet opakování
or  a              ;vynulování příznaku CARRY
SCR2  rr  (hl)    ;rotace bytem doprava
inc  l             ;posun na další adresu
djnz SCR2       ;zacyklení
```

Napsat raději tuto sekvenci:

```

srl  (hl)         ;posun doprava (vstupuje nula)
inc  l            ;posun na další adresu
rr   (hl)         ;rotace obsahu bytu doprava
inc  l            ;posun na další adresu
.
.
rr   (hl)         ;poslední rotace doprava (31. instrukce)
```

Uvedená sekvence je sice delší (1 instrukce **srl (hl)** a 31 instrukcí **inc l** a **rr (hl)**) ale také podstatně rychlejší, můžeme si to spočítat, nejprve kratší verzi:

```

ld  b,32           ;                               7 T-cyklů
or  a              ;                               4 T-cykly
SCR2  rr  (hl)    ;                               32 * 15 = 480 T-cyklů
inc  l            ;                               32 * 4 = 128 T-cyklů
djnz SCR2       ;                               31 * 13 + 8 = 411 T-cyklů
```

-----  
celkem tedy: 1030 T-cyklů

A nyní časová náročnost v rychlejší úpravě:

```
sr1 (hl), 31*inc l, 31*rr (hl); 15+31*(4+15) = 604 T-cyklů
```

Jak sami vidíte, je druhá varianta skoro dvakrát tak rychlá jako ta první (přesně 1.7-krát). V některých případech tato skutečnost může znamenat značné zlepšení kvality programu (obraz přestane blikat a trhat se).

Další výhodou tohoto postupu je, že nepotřebujete žádný registr pro uložení počtu průchodů, občas tím odpadne nutnost ukládat něco na zásobník.

Pokud používáte k přenosu instrukci **ldir** (**lddr**), můžete také dosáhnout zrychlení tím, že místo jedné instrukce **ldir** (**lddr**) napíšete tolik instrukcí **ldi** (**ldd**), kolik je číslo v BC v okamžiku provádění instrukce **ldir**. Časový zisk si opět můžeme spočítat, nejprve při použití instrukce **ldir**:

```
ld bc,32 ; 10 T-cyklů
ldir ; 32 * 21 - 5 = 667 T-cyklů
-----
celkem tedy: 677 T-cyklů
```

A nyní při použití instrukcí **ldi**:

```
32 * ldi ; 32*16 = 512 T-cyklů
```

Zde tedy není časová úspora tak velká (pouze 1.3), nicméně i toto řešení občas pomůže zrychlit váš program.

Jiný způsob zrychlení spočívá v tom, že si některé věci spočítáte dopředu. Zde by přicházely v úvahu počáteční adresy jednotlivých mikrořádků. Program by pak pouze odebíral jednotlivé hodnoty z tabulky a prováděl přesuny. Tabulka se tedy bude skládat z dvojbytových hodnot, nejrychlejší způsob, jak taková čísla číst, je pomocí zásobníku. Ukážeme si příklad takového čtení:

```
START ld (SPSTOR+1),sp ;zásobník budeme používat později
      di ;musíme zakázat přerušení
      ld sp,TABEND ;SP ukazuje na konec tabulky
      ld a,191 ;počet přesunů mikrořádků
LOOP pop hl ;odeber adresu „odkud“
      pop de ;odeber adresu „kam“
      ldi ;tolik instrukcí ldi
      .. ;kolik je šířka
      ldi ;skrolované oblasti
```



```

                dec  a                ;počítadlo přesunů zmenši o jedničku
                jr   nz,LOOP          ;dokud není na nule prováděj přesuny
SPSTOR         ld   sp,0              ;sem se zapíše skutečná hodnota SP
                ei                       ;pokud je potřeba
                ret                      ;návrat

TABLE         defw 22240,22496        ;první je „kam“, druhá „odkud“
                defw 21984,22240
                ....
                ....
                defw 16896,17152
                defw 16640,16896
                defw 16384,16640
TABEND

```

Pokud doplníte hodnoty do tabulky (mělo by to být celkem 382 čísel, tedy 191 řádků), bude program dělat skrolování celé obrazovky po pixlech nahoru. Když si program a hlavně tabulku dobře prohlédnete, zjistíte, že by mohla být poloviční - trošku vám poradím, budete muset přidat navíc jednu instrukci **push**. Tento příklad, pokud ho budete chtít vyzkoušet, si raději důkladně projděte trasováním a než jej poprvé spustíte „na ostro“, tak si jej raději uložte. Hodnoty do tabulky si můžete vygenerovat pomocí jednoduchého programu, který bude využívat podprogram **DOWNHL**. Musíte si jej ovšem sami napsat, pokud nevíte jak, tak pro vás tento způsob programování zatím není.

Poslední, čím se tato kapitola bude zabývat, je, jak to nazval Jan Flaška (kterému tímto děkuji za pomoc při výměně několika drátů, byly tuším tři, na mém zadním kole - tedy ne že bych já osobně měl zadní kolo, ale moje kolo je má a ... co je zrovna vám do toho?), **plazící se text**. S pomocí obvyčejného skrolování docílíte celkem zajímavý efekt, nebude sice úplně originální (něco podobného se již objevilo v jednom **MELODY MUSICu**), pro naše účely však postačuje více než dostatečně. Opište si následující příklad:

```

START         im   1                ;nastav první mód přerušení
                ei                       ;povol přerušení
                ld   hl,0              ;vyplníme něčím obrazovku,
                ld   de,16384          ;použijeme na to obsah
                ld   bc,6144           ;paměti ROM, zaplníme však
                ldir                    ;jen pixelovou část

                ld   hl,TEXT           ;nastavíme ukazatele
                ld   (TEXT1+1),hl      ;„do textu“ a „na začátek textu“
                ld   (TEXT2+1),hl      ;na začátek textu

                ld   d,0                ;počítadlo posunů, na začátku nula

TEST         push de                  ;uložíme počítadlo posunů
                ld   a,5                ;nastavíme bledě modrý border po dobu
                out  (254),a            ;kdy program čeká na přerušení
                halt                    ;počkáme na přerušení (synchronizace)
                ld   a,7                ;po dobu, kdy program provádí plazení
                out  (254),a            ;textu, bude border bílý

```

```
ld hl,20480+10 ;adresa levého horního rohu
ld a,22 ;šířka ve znakových pozicích
call SCRLEFT ;zvolenou část obrazovky posuň doleva
```

Adresa levého horního rohu je zadána tak, že první číslo je adresa počátku třetiny obrazovky (16384, 18432, 20480), další číslo (násobek čísla 32 - 32, 64, 96, 128, 160, 192, 224) je posun po řádcích, 32 je první řádek, 64 je druhy řádek,.....řádky jsou číslovány od nuly, poslední číslo (mezi 1 až 31) je číslo sloupce, sloupce jsou číslovány také od nuly. Pokud je číslo řádku nebo sloupce nula, pak tam uvedeno není.

```
ld hl,18432+10 ;adresa levého horního rohu
ld a,9 ;výška ve znakových pozicích
call SCRUP ;zvolenou část obrazovky posuň nahoru
```

```
ld hl,18432+10 ;adresa levého horního rohu
ld a,10 ;šířka ve znakových pozicích
call SCRRIGHT ;zvolenou část obrazovky posuň doprava
ld hl,16384+19+64 ;adresa levého horního rohu
ld a,7 ;výška ve znakových pozicích
call SCRUP ;zvolenou část obrazovky posuň nahoru
```

```
ld hl,16384+64+7 ;adresa levého horního rohu
ld a,13 ;šířka ve znakových pozicích
call SCRLEFT ;zvolenou část obrazovky posuň doleva
```

```
ld hl,16384+64+7 ;adresa levého horního rohu
ld a,17 ;výška ve znakových pozicích
call SCRDOWN ;zvolenou část obrazovky posuň dolů
```

```
ld hl,20480+64+7 ;adresa levého horního rohu
ld a,24 ;šířka ve znakových pozicích
call SCRRIGHT ;zvolenou část obrazovky posuň doprava
```

```
ld hl,20480+64+30 ;adresa levého horního rohu
ld a,4 ;výška ve znakových pozicích
call SCRDOWN ;zvolenou část obrazovky posuň dolů
```

```
ld hl,20480+160+4 ;adresa levého horního rohu
ld a,27 ;šířka ve znakových pozicích
call SCRLEFT ;zvolenou část obrazovky posuň doleva
```

```
ld hl,20480+128+2 ;adresa levého horního rohu
ld a,3 ;šířka ve znakových pozicích
call SCRLEFT ;zvolenou část obrazovky posuň doleva
```

```
ld hl,20480+96+2 ;adresa levého horního rohu
ld a,2 ;šířka ve znakových pozicích
call SCRLEFT ;zvolenou část obrazovky posuň doleva
```

```
ld hl,20480+64+1 ;adresa levého horního rohu
ld a,2 ;šířka ve znakových pozicích
call SCRLEFT ;zvolenou část obrazovky posuň doleva
```

```
ld hl,20480+32 ;adresa levého horního rohu
ld a,2 ;šířka ve znakových pozicích
call SCRLEFT ;zvolenou část obrazovky posuň doleva
```

```
ld hl,20480 ;adresa levého horního rohu
ld a,1 ;šířka ve znakových pozicích
```

```

call SCRLEFT      ;zvolenou část obrazovky posuň doleva

ld hl,20480+128+4 ;adresa levého horního rohu
ld a,3             ;výška ve znakových pozicích
call SCRUP        ;zvolenou část obrazovky posuň nahoru
ld hl,20480+96+3 ;adresa levého horního rohu
ld a,3             ;výška ve znakových pozicích
call SCRUP        ;zvolenou část obrazovky posuň nahoru
ld hl,20480+64+2 ;adresa levého horního rohu
ld a,3             ;výška ve znakových pozicích
call SCRUP        ;zvolenou část obrazovky posuň nahoru
ld hl,20480+32+1 ;adresa levého horního rohu
ld a,3             ;výška ve znakových pozicích
call SCRUP        ;zvolenou část obrazovky posuň nahoru
ld hl,20480       ;adresa levého horního rohu
ld a,3             ;výška ve znakových pozicích
call SCRUP        ;zvolenou část obrazovky posuň nahoru

pop af            ;odeber počítadlo posunů do registru A
inc a             ;zvyš počet posunů o jedničku
cp 10             ;a testuj odsunutí jednoho znaku
ld d,a           ;vrať počítadlo posunů do registru D
jr nz,TEST2      ;přeskoč případně tisk dalšího znaku

TEXT1 ld hl,0      ;ukazatel na další znak do textu
      ld a,(hl)    ;vyzvedni jej
      inc hl       ;a posuň se pro další znak,
      or a         ;nyní testuj konec textu (ukončen nulou)
      jr nz,CHAR3  ;a pokud není, přeskoč nastavení začátku

TEXT2 ld hl,0      ;nastav znovu začátek textu
      ld a,(hl)    ;a jeho vyzvedni první znak
      inc hl       ;posuň se na další znak

CHAR3 ld (TEXT1+1),hl ;zapiš pozici dalšího znaku
      add a,a       ;nyní budeme počítat adresu
      ld l,a       ;grafické předlohy pro znak,
      ld h,15      ;jehož kód je na začátku v registru A
      add hl,hl     ;tento výpočet je podrobněji popsán
      add hl,hl     ;v minulém dílu na straně 47
      ld b,8       ;další část je obyčejný tisk znaku
      ld de,20480+31 ;znak se tiskne pokaždé na stejné místo

CHAR  ld a,(hl)
      rra          ;pokud máte DIDAKTIK GAMA nebo M, tak
      or (hl)     ;tyto dvě instrukce vynechejte
      ld (de),a
      inc hl
      inc d
      djnz CHAR

      ld d,0       ;po vytištění znaku je počet posunů 0

TEST2 call 8020    ;konec cyklu, testuj stisk BREAKu
      jp c,TEST   ;pokud není stisknut, pokračuj dál
      ret        ;pokud je stisknut, vrať se zpět

SCRUP ld b,a       ;výška ve znacích do registru B
SUP0  ld e,1       ;adresu v HL přesuneme
      ld d,h       ;postupně do DE

```

```

SUP4      inc h           ;adresu v HL posuneme o bod dolů
          ld a,h         ;a budeme testovat, jestli nedošlo
          and 7          ;k přechodu na další znakový řádek,
          jr z,SUP1     ;pokud ano, skoč na jeho zpracování
          ld a,(hl)     ;nyní přešů jeden byte
          ld (de),a     ;z adresy v HL na adresu DE
          jr SUP0       ;jdi znovu na začátek

SUP1      ld a,1         ;tato část se velmi podobá
          add a,32      ;podprogramu DOWNHL
          ld l,a        ;a proto ji nebudu komentovat
          ld a,h
          jr c,SUP2
          sub 8
          ld h,a

SUP2      cp 88
          jr c,SUP3
          ld h,66

SUP3      djnz SUP4     ;konec smyčky přes znakové pozice
          ret          ;konec skrolování nahoru

SCRDOWN  ld b,a         ;do B počet znaků - výška
SDOWN0   ld e,(hl)     ;vzvedni nejvyšší byte ve sloupci
          inc h         ;posun na další byte (dolů)
          ld a,(hl)    ;vzvedni byte, který by byl přepsán
          ld (hl),c    ;a musí být posunut dolů, zapiš horní
          ld c,a       ;pro další průchod přepis obsah do C
          ld a,h       ;testuj, zda se nejedná o poslední
          and 7        ;mikrořádek ve znakovém řádku,
          cp 7         ;pokud ne,
          jr nz,SDOWN0 ;můžeš přenášet další byte
          ld a,1       ;další část je jistá modifikace
          add a,32     ;podprogramu DOWNHL a proto
          ld l,a      ;se jí nebudu podrobně zabývat
          ld a,h
          jr c,SDOWN2
          sub 8
          ld h,a

SDOWN2   cp 88
          jr c,SDOWN3
          ld h,66

SDOWN3   djnz SDOWN0  ;ukončení cyklu přes znaky
          ret          ;návrat zpět

SCRLEFT  ld e,a         ;do C dej šířku ve znacích
          dec e        ;a zmenši ji o jedničku
          ld b,0       ;vynuluj B, šířka je nyní v BC
          add hl,bc    ;posuň se na konec skrolované části
          ld d,a      ;zapiš do A šířku ve znacích
          ld e,8      ;znakový řádek má 8 mikrořádků
SLEFT0   push hl       ;ulož adresu konce mikrořádku
          ld b,d      ;zapiš délku mikrořádku do B
          xor a        ;vynuluj příznak CARRY
SLEFT1   rl (hl)      ;zarotuj byte doleva
          dec l        ;posuň se na předchozí byte
          djnz SLEFT1 ;proved' pro všechny byty v řádku

```

```

pop hl ;obnov adresu konce mikrořádku
inc h ;posuň se na další mikrořádek
dec c ;zmenši počítadlo mikrořádků
jr nz,SLEFT0 ;konec cyklu přes mikrořádky
ret ;návrat z podprogramu

SCRRIGHT ld d,a ;do C dej šířku ve znacích
ld c,8 ;znakový řádek má 8 mikrořádků
SRIGHT0 push hl ;ulož adresu začátku mikrořádku
ld b,d ;zapiš délku mikrořádku do B
xor a ;vynuluj příznak CARRY
SRIGHT1 rr (hl) ;zarotuj byte doprava
inc l ;posuň se na následující byte
djnz SRIGHT1 ;proved pro všechny byty v řádku
pop hl ;obnov adresu začátku mikrořádku
inc h ;posuň se na další mikrořádek
dec c ;zmenši počítadlo mikrořádků
jr nz,SRIGHT0 ;konec cyklu přes mikrořádky
ret ;návrat z podprogramu

TEXT defm "Toto je " ;obsah plazícího se textu
defm "plazici se "
defm "text z knihy "
defm "ASSEMBLER a "
defm "ZX SPECTRUM "
defm "(2)..... "
defb 0

```

Uvedený příklad přináší několik novinek - programy pro skrol jsou psány tak, že skrolují vždy pruh, jehož jeden rozměr je 8 bodů, je to vždy ten rozměr, v jehož směru se posun neprovádí. Když srovnáte tyto skroly se skroly pro obecný obdélník, vidíte, že jsou jednodušší a samozřejmě také rychlejší. Nejzajímavější novinka je v podprogramu pro skrol dolů **SCRDOWN**, ve kterém je použit jiný způsob posunu - zatím jsme skrol tímto směrem dělali tak, že jsme **odspoda** procházeli jednotlivé mikrořádky a posunovali je dolů. Nyní začínáme nahoře a přesto dosahujeme stejného výsledku - program si prohlédněte tak, abyste pochopili, jak vlastně pracuje - něco podobného jsme použili v minulém díle na straně 89 pro vložení znaku do textu.

Další novinka je ta, že program dostává informace o začátku a šířce (výšce) obdélníku jako parametry. Podobně si můžete upravit i ostatní podprogramy pro skrolování a rolování a získáte obecně použitelné podprogramy - budete-li je potřebovat, stačí je pouze opsat nebo přihrát do textu a nemusíte je znovu vymýšlet. Mohli jste si všimnout, že volání podprogramů **SCRUP**, **SCRDOWN**, **SCRLEFT**, **SCRRIGHT** je velmi podobné a dalo by se zkrátit, zkuste upravit program tak, abyste mohli podprogramy volat například takto, něco podobného je na straně 54 předchozího dílu:

```

call SCRUP ;volání příslušného podprogramu
defw 16384 ;adresa levého horního rohu
defb 10 ;šířka (výška) obdélníku

```

Když se vám to podaří, ušetříte na každém volání tohoto podprogramu dva byty, natáhne se vám sice vlastní kód podprogramů, ale výsledek by měl být pozitivní - uberete víc, než přidáte.

Celý program je možné zkrátit ještě více, co kdybyste například celý podprogram upravili tak, že by se místo té dlouhé sekvence volání jednotlivých posunů co je mezi návěstími **TEST** a **TEXT1** napsalo například:

```

call SCROLLS          ;zavolání jediného podprogramu
defb 1,22             ;směr skrolování a volitelný rozměr
defw 16384            ;adresa levého horního rohu
defb 2,2              ;směr skrolování a volitelný rozměr
defw 18432+10         ;adresa levého horního rohu
defb 3,12             ;směr skrolování a volitelný rozměr
defw 16384+64+4       ;adresa levého horního rohu
defb 0                ;pokud je místo směru 0, je konec dat

```

Směry by byly kódovány čísla **1 až 4** a číslo **0** by znamenalo, že se máte z podprogramu vrátit. Tímto způsobem by se místo 8 bytů pro jeden posun (call, ld a, ld hl) mohly psát pouze byty čtyři. Konkrétní provedení nechám na vás.

Poslední zajímavost je, že na obrazovce se vytvořilo rozdělení v **BORDERu** na dvě části - první je bílá a druhá (spodní) bledě modrá. Tady můžete vidět, jak je náš příklad rychlý - celé odsunutí textu o jeden bod stihne počítač za méně než padesátinu sekundy - to je totiž doba, po které se opakuje přerušovací signál. To, že se hranice mezi oběma oblastmi neustále pohybuje (poskakuje), je způsobeno tím, že se znak netiskne pokaždé ale jen každých deset posunutí (toto číslo si můžete změnit - je to instrukce **cp 10** před návěstím **TEXT1**). Pokud byste chtěli, aby hranice stála na místě bez pohybu, museli byste zajistit, aby každá větev programu trvala stejně dlouho - zařadit případné čekání. Každý podprogram pro skrol trvá při stejných parametrech stejně dlouho, tuto část tedy upravovat nemusíte, jinak se program větví na dvou místech - jednak při odrolování 10 bodů (to je již zmíněný tisk jednoho znaku) a pak také při tisku znaku v místě, kde se testuje konec textu a případně nastavuje jeho začátek (u návěstí **TEXT2**). Pokud budete program takto upravovat, dejte si při přidávání „zdržovacích“ instrukcí pozor na to, abyste nezničili obsah nějakého registru, který bude potřeba - v našem případě jsou to na daných místech prakticky jen registry **D** a **SP**. Pro hrubé zdržení můžete použít instrukci **djnz** a pro jemné pak **nop** a jiné instrukce, které nic neprovádí (to může být například kombinace instrukcí **scf** a **ret nc**, která pouze nastaví příznak **CARRY** a trvá dohromady **9 T-cyklů**). Někdy se mohou dvě větve programu lišit třeba jen od dva T-cykly, protože však tak rychlá instrukce neexistuje, musíte přidat něco do obou větví - do jedné 4 T-cykly (třeba **nop**) a do druhé 6 T-cyklů (třeba **inc hl**). Opět zdůrazňuji, že tato část je určena pokročilejším programátorům.

Poslední, co bych chtěl k *Hýbeme obrazovkou* připsat, je adresa podprogramu v **ROM**, který provádí atributový **SCROLL** nahoru. Tento podprogram leží na adrese **#DFE** (neboli **3582** dekadicky) a provádí skrolování 23 řádků, pokud budete chtít skrolovat řádků méně, můžete to docílit tím, že podprogram budete volat o dva byty dále - **#E00** (3584) a do registru **B** zapíšete počet řádků pro odskrolování.

---

# *Volba ovládání*

V této kapitole si ukážeme, jak se dá napsat **Volba ovládání**, tedy ta část programu, která vám umožní zvolit si klávesy (nebo joystick), které chcete používat při ovládání programu - obvykle jsou to čtyři klávesy pro směry a jedna pro volbu.

Nejprve si povíme něco o tom, jaké ovládání se nejčastěji používá:

**Klávesnice** - zcela libovolné klávesy, většinou je požadavek, aby CAPS SHIFT, SYMBOL SHIFT, SPACE a ENTER fungovaly jako jakékoliv jiné klávesy a ne tak, jak fungují obvykle. Další požadavek je, aby bylo možno stisknout a testovat i více kláves současně. Pokud není CAPS SHIFT (nebo SYMBOL SHIFT) používán jako funkční klávesa, nemělo by mít jeho případné stisknutí žádný vedlejší účinek.

**Cursor joystick** - u nás tento joystick není příliš rozšířen a tak se tato volba používá jen tehdy, když chcete program ovládat šipkami nebo klávesami s čísly **5, 6, 7, 8** a aktivovat **0**. Z programátorského hlediska je to stejný případ jako testování klávesnice - když má uživatel možnost použít libovolné klávesy, může použít také **Cursor joystick**.

**Sinclair joystick** nebo **Interface II** - podobné jako **Cursor joystick**, pouze se používají jiné klávesy, pokud použijete **Sinclair Left**, tak jsou to klávesy **1, 2, 3, 4, 5** s tímto významy **doleva, doprava, dolů, nahoru a pal**, **Sinclair Right** pak jsou klávesy **6, 7, 8, 9, 0** se stejnými významy, tedy **doleva, doprava, dolů, nahoru a pal**. Opět je to vlastně testování klávesnice, pokud umožníme volbu libovolných kláves, může si uživatel nadefinovat také oba **Sinclair joysticky**.

**Kempston joystick** - tento joystick je připojen na **port 31** a zabírá jeho 0-tý až 4-tý bit. U tohoto joysticku vznikají potíže, protože na stejný port může být připojena také tiskárna a pak jej nelze použít, některé programy však tento joystick testují neustále a pak může docházet k tomu, že je program neovladatelný - na portu totiž zůstala nějaká hodnota poslaná do tiskárny, která je náhodou také přípustnou kombinací pro tento joystick - a pak třeba kurzor v DESKTOPu poskakuje bez dotyku klávesnice... Bity 0 až 4 mají tento význam: **doleva, doprava, dolů, nahoru a pal**.

V našem příkladu si ukážeme podprogram, který umožní nadefinovat si vlastně všechny druhy ovládání - jak klávesnici, tak všechny joysticky. Navíc tu uvidíte další z možných úprav znaků z ROM. Tento program už nejspíš důvěrně znáte, používám jej totiž prakticky ve všech svých programech, opište si následující program:

```

START    im    1                ;nastav mód přerušeni číslo 1
          ei                ;povol přerušeni

          ld    hl,1000        ;následující část provádí testování
          ld    c,0            ;přítomnosti KEMPSTON joysticku
R1        in    a,(31)        ;přečti hodnotu na portu 31
          or    c              ;přidej k němu dosavadní hodnotu
          ld    c,a           ;a vrať vše do registru C
          dec   hl            ;zaznamenej další průchod
          ld    a,h           ;otestuj hodnotu v registru HL
          or    l             ;a pokud to není nula,
          jr    nz,R1        ;jdi znovu testovat port 31
          ld    a,c           ;vezmi to, co jsi získal z portu 31
          cp    32            ;a porovnej s číslem 32
          ld    a,8           ;dej do A osmičku - jen klávesy
          adc   a,0           ;přičti k A hodnotu příznaku CARRY
          ld    (KEYCNT+2),a  ;a zapiš to do testování portů

```

Tato část programu pracuje tak, že nějakou dobu čte obsah portu 31 a nové hodnoty **ORUje** se starými - toto vychází z toho, že pokud není na sběrnici počítače připojeno nic, je na tomto portu takřka stále hodnota 255, tím, že testování provádíme vícekrát, získáme ji určitě, pokud je na sběrnici připojen KEMPSTON interface, je tam obvykle nula a neměli bychom nikdy získat číslo vyšší než 31. Získané číslo se pak porovnává s číslem 32 a pokud je menší, nastaví se příznak CARRY. Tento způsob zjišťování přítomnosti KEMPSTON joysticku se používá například ve hrách PETE COOKA jako jsou ACADEMY, TAU CETI a jiné. Nevím, jak za našimi hranicemi, ale u nás se používají obvody, které tento způsob testování úplně vyřazují z činnosti - třeba interface UR-4 (obvod 8255), používá se hlavně pro připojení tiskárny. U tohoto obvodu záleží, v jakém režimu je nastaven (po zapnutí je vše OK), pokud stisknete přes port A, může po skončení tisku začít program „blbnout“, zůstane tam poslední kód, který byl poslán do tiskárny a ten může znamenat nějaký pohyb joystickem - pokud se naše testování provede po něčem takovém, může se stát, že se testem přítomnost joysticku potvrdí a při definici ovládání se všude nastaví jeden směr joysticku.

Pokud se budete chtít těmto potížím vyhnout stoprocentně, musíte místo tohoto testu zařadit otázku pro uživatele, jestli je připojen KEMPSTON, a v případě, že ano, bude se při definici ovládání testovat 9 různých portů (8 pro klávesnici, 1 pro joystick), v případě, že ne, bude se testovat jen 8 portů (klávesnice).

```

          ld    hl,TEXT1      ;text „Select a key or move joystick to“
          call TEXTOUT       ;vytiskni jej
          ld    hl,TEXT2     ;do HL adresa textu „Right:“
          ld    de,REDEFINE  ;do DE adresa zvoleného ovládání

MAIN      ld    b,5          ;definujeme celkem pět kláves
          push bc            ;ulož počítadlo kláves
          call TEXTOUT       ;vytiskni text, HL se posune za něj
          push hl           ;ulož adresu dalšího textu
          push de           ;ulož adresu pro uložení zvolené klávesy

R3        ld    b,20        ;nyní počkáme
          halt             ;něco kolem

```



```

djnz R3                ;půl sekundy
call KEYRET            ;testuj všechny porty, čekej na stisk
call BEEP              ;oznam nalezení stisku klávesy
pop hl                 ;odeber do HL adresu pro definici
ld (hl),c              ;a postupně do tabulky
inc hl                 ;zapiš port, na kterém
ld (hl),b              ;jsi našel stisk,
inc hl                 ;a také bit,
ld (hl),a              ;který to byl
inc hl                 ;posuň se na volné místo
ld a,e                 ;vezmi do A číslo textu v tabulce
push hl                ;ulož ukazatel do tabulky definice
ld hl,KEYS             ;ukaž na tabulku s názvy kláves
inc e                  ;zvyš číslo textu klávesy
MAIN2 dec e             ;zmenší číslo textu
jr z,MAIN3             ;a při nule skoč na jeho vytištění
MAIN6 inc hl            ;posuň se na další znak
ld a,(hl)              ;a vezmi jeho kód a testuj,
call NUMCHAR           ;zda je to velké písmeno nebo číslo
jr nz,MAIN6            ;pokud ne, jdi pro další znak
jr MAIN2               ;jdi pro další text klávesy

MAIN3 call TEXTOUT2    ;vytiskni text ke klávese
pop de                 ;obnov ukazatel do tabulky definice
pop hl                 ;obnov ukazatel na texty
STOP inc hl            ;a posuň jej na začátek dalšího textu
pop bc                 ;obnov počítadlo kláves
djnz MAIN              ;a případně jdi pro další klávesu
ret                    ;konec definice kláves

TEXTOUT push de         ;ulož obsah registru DE
ld e,(hl)              ;vyzvedni adresu
inc hl                 ;adresu pro umístění
ld d,(hl)              ;textu na obrazovce
inc hl                 ;a nastav se na začátek vlastního textu
ld (PRINTPOS+1),de    ;zapiš adresu pro umístění textu
pop de                 ;nyní už DE nebudeme potřebovat
TEXTOUT2 ld a,(hl)     ;následuje smyčka,
call CHAR              ;která tiskne
inc hl                 ;jednotlivé znaky
ld a,(hl)              ;tak dlouho, dokud další znak nebude
call NUMCHAR           ;velké písmeno nebo číslice
jr nz,TEXTOUT2        ;ret

CHAR exx               ;tisk znaku je tradiční, proto popíši
ld (CH2+1),a          ;jen rozdily - ulož kód znaku pro test
add a,a
ld l,a
ld h,15
add hl,hl
add hl,hl
PRINTPOS ld de,0
push de
call NEXTDE4          ;posuň se o bod a přenes první byte

CH2 ld a,0
push hl               ;vyzvedni znovu kód znaku
;uschovej adresu znakové předlohy

```

```

    ld hl,REPAIR1 ;první tabulka s opravami
    ld b,(hl) ;přečti si velikost tabulky
    inc hl ;a posuň se na vlastní tabulku
L1 cp (hl) ;porovnej kód s tabulkou,
    jr z,L2 ;při rovnosti použij předchozí byte
    jr c,L3 ;tabulka je seřazena podle velikosti
    inc hl ;a pokračuj jen v případě, že jsou
    djnz L1 ;menší, ukončení cyklu
L3 pop hl ;obnov adresu grafické předlohy
    ld a,(hl) ;použij aktuální byte předlohy
    jr OK3 ;zdvojený byte připraven

REPAIR1 defb 22 ;opravu vyžaduje 22 znaků
    defm "#$^acegmnpqr"
    defm "stuvwxy"

L2 pop hl ;touto větví se pokračuje pro znaky
    dec hl ;z tabulky a znamená zdvojení
    ld a,(hl) ;předchozího bytu
    inc hl ;posun zpět na aktuální byte
OK3 call NEXTDE3 ;zapiš do obrazovky a posuň se dál
    call NEXTDE6 ;vypiš celkem 3 byty z grafické předlohy

    ld a,(CH2+1) ;budeme opět zdvojovat a proto si
    push hl ;opět připravíme kód znaku pro
    ld hl,REPAIR2 ;případné opravy
    ld b,(hl) ;další část programu je obdobná jako
    inc hl ;u předchozího zdvojení
L1B cp (hl) ;návěští se liší pouze přidáním B
    jr z,L2B
    jr c,L3B
    inc hl
    djnz L1B
L3B pop hl
    ld a,(hl)
    jr OK4

REPAIR2 defb 9
    defm "#4=WgppyC" ;poslední je znak COPYRIGHT ©

L2B pop hl
    dec hl
    ld a,(hl)
    inc hl
OK4 call NEXTDE3 ;zapiš do obrazovky a posuň se dál
    call NEXTDE6 ;vypiš 3 byty z grafické předlohy
    pop hl ;posun na další pozici, není ošetřena
    inc l ;možnost přechodu mezi třetinami
    ld (PRINTPOS+1),hl
    ld a,l
    and 31
    jr nz,CH3
    ld bc,32
    add hl,bc ;posun o dva standardní znakové řádky
    ld (PRINTPOS+1),hl
CH3 exx
    ret

```

```

NEXTDE6 call NEXTDE2 ;vytištění znaku a posun pozice
NEXTDE4 call NEXTDE2 ;vytištění znaku a posun pozice
NEXTDE2 ld a,(hl) ;přesun jednoho bytu
inc hl
NEXTDE3 ld (de),a
NEXTDE inc d ;následuje posun pozice o bod dolů
ld a,d
and 7
ret nz
ld a,d
sub 8
ld d,a
ld a,e
add a,32
ld e,a
ret nc
ld a,d
add a,8
ld d,a
ret

KEYRET ld hl,TABKEY ;tabulka klávesových portů
KEYCNT ld de,#900 ;do D devítka (osmička), do E nula
KEYLOOP ld c,(hl) ;přečti adresu portu
inc hl
ld b,(hl)
inc hl
in a,(c) ;přečti hodnotu portu
bit 7,c ;test mezi portem 31 a klávesovým portem
jr z,KEY2 ;odskoč při portu 31
cpl ;invertuj obsah portu
KEY2 and 31 ;ponech si jen pět bitů
jr nz,KEYGET ;zaznamenali jsme nějakou klávesu
ld a,e ;přičteme pětku k číslu textu
add a,5 ;abychom mohli vyzvednout
ld e,a ;adresu dalšího portu
dec d ;počítadlo portů zmenši o jedničku
jr nz,KEYLOOP ;a pokud nejsi na nule testuj další port
jr KEYRET ;nic jsi nenašel, hledej tedy znovu

KEYGET push hl ;ulož ukazatel na tabulku portů
push bc ;ulož adresu portu
ld hl,TABBITS ;nastav tabulku bitových masek
ld b,5 ;budeme prohlížet pět bitů
KEY3 cp (hl) ;porovnej získanou hodnotu s povolenými
jr z,KEYOK ;povolená hodnota byla nalezena
inc hl ;posuň se
inc e ;další položka, zvyš číslo textu
djnz KEY3 ;dokud jsi neprošel všechno, pokračuj
pop bc ;obnov adresu portu
pop hl ;obnov adresu do tabulky portů
jr KEYRET ;a skoč znovu na začátek testování

KEYOK pop bc ;obnov adresu portu
pop hl ;obnov adresu do tabulky portů
ret ;vrať se s novými hodnotami

```

Podprogram **KEYRET** postupně testuje jednotlivé porty klávesnice, a pokud je připojen, tak také port Kempston joysticku. Zjišťuje se, jestli na nějakém z těchto portů nedošlo ke stisku jen jedné klávesy, pokud ano, vrací se program s adresou portu v registru BC a s bitovou maskou v registru A, v registru E pak je číslo textu v tabulce, který popisuje danou klávesu. Pokud se nenalezne nic, podprogram začne testovat porty znovu.

```

TABKEY   defw 63486,61438      ;klávesy 1 2 3 4 5 a 6 7 8 9 0
           defw 65278,65022      ;klávesy CS Z X C V a A S D F G
           defw 64510,57342      ;klávesy Q W E R T a P O I U Y
           defw 49150,32766      ;klávesy ENTER L K J H a SPACE SS M N B
           defw 31                ;KEMPSTON joystick

TABBITS  defb 1,2,4,8,16      ;tabulka povolených možností

KEYS     defm "12345"         ;tabulka jmen kláves
           defm "09876"
           defm "CapsZXCV"
           defm "ASDFG"
           defm "QWERT"
           defm "POIUY"
           defm "EnterLKJH"
           defm "SpaceSymbolMNB"
           defm "Kempston right"
           defm "Kempston left"
           defm "Kempston down"
           defm "Kempston up"
           defm "Kempston fire"
           defm "A"

NUMCHAR  cp "0"                ;tento podprogram vrací příznak Z
           ret c                  ;v případě, že se jedná o velké
           cp "9"+1              ;písmeno nebo číslici, jinak
           jr c,OK5              ;vrací příznak NZ
           cp "A"
           ret c
           cp "Z"
           jr z,OK5
           ret nc

OK5      cp a
           ret

TEXT1    defw 16384+160        ;první text
           defm "Select a key "
           defm "or move "
           defm "joystick toA"

TEXT2    defw 18440            ;druhý text - jednotlivé směry
           defm "Right:A"

           defw 18440+64
           defm "Left :A"

```

```

        defw 18440+128
        defm "Down :A"

        defw 18440+192
        defm "Up :A"

        defw 20488
        defm "Fire :A"

BEEP    push hl                ;zvukový signál
        push de
        push bc
        push af
        ld  e,30
        ld  hl,300
        ld  a,16
A2      ld  b,e
        xor 16
        or  7                ;nastavení barvy borderu - bílá
A1      out (254),a
        djnz A1
        dec hl
        inc h                ;tyto dvě instrukce vlastně testují,
        dec h                ;zda není v registru H číslo nula
        jr  nz,A2
        pop af
        pop bc
        pop de
        pop hl
        ret

REDEFINE defs 15                ;3 x 5 bytů pro definované klávesy

```

Tímto jsme si vytvořili program pro definici ovládání, nyní k němu ještě přidáme část, která při zavolání otestuje vybrané klávesy (joystick) a vrátí jednobytovou hodnotu, která ve svých pěti spodních bytech ponese zakódované stavy jednotlivých kláves, kódování je stejné jako u KEMPSTON joysticku.

```

CONTROLS ld  hl,REDEFINE        ;adresa tabulky s navolenými klávesami
        ld  de,5                ;v E je počet kláves, v D bude výsledek
CLP      ld  c,(hl)              ;vyzvedni
        inc hl                  ;adresu portu
        ld  b,(hl)              ;do registru BC
        inc hl                  ;a načti
        in  a,(c)               ;do registru A hodnotu z daného portu
        bit 7,c                 ;zjištění typu portu (kláv. nebo joy.)
        jr  z,NCPL              ;pokud jde o joystick, odskoč pryč
        cpl                    ;jednička nyní znamená stisk klávesy
NCPL     and  (hl)               ;ponech jen testovaný bit
        inc hl                  ;a posuň se na další klávesu
        jr  z,CN                ;pokud stisk nezaznamenán, odskoč
        set 5,d                 ;klávesa zaznamenána
CN       srl  d                  ;zarotuj seznamem kláves
        dec  e                  ;zmenši počítadlo kláves
        jr  nz,CLP              ;a pokud není nulové, testuj další

```

```
ret                ;výsledek testu je v registru D
```

Budete-li si chtít program vyzkoušet, připište k němu na úplný začátek třeba tento krátký program:

```
org 50000          ;program bude na této adrese

call CONTROLS     ;proved' testování definovaných kláves
ld c,d            ;zapiš hodnotu z registru D
ld b,0            ;do registru BC
ret               ;vrať se
ent $             ;od této adresy se program spustí
```

Nyní odešlete příkaz assembleru **RUN** (pokud nepoužíváte **PROMETHEa**, musíte zajistit, aby se program nejdříve přeložil a pak spustil) a nadefinujte si ovládání. Program se vrátí do Assembleru, pak vyskočte do basicu a napište tento program:

```
10 PRINT AT 0,0;USR 50000,, :GO TO 10
```

Ten zajistí neustálé testování nadefinovaných kláves a vypisování zjištěného stavu. Tento způsob samozřejmě není určen pro programy v BASICu a proto si ukážeme nějaký složitější příklad - bude to následující kapitola.

# Šipka

Za tímto nepřilíš jasným názvem se skrývá něco, co se používá v mnoha programech (ART STUDIO, ORFEUS, ACADEMY,.....Jméno Růže a HEROES). Je to šipka, se kterou můžete pohybovat po obrazovce a vybírat z nabídek programu.

Program, který si napíšeme, bude pracovat tak, že po zavolání vám nejprve umožní nadefinovat si ovládání (použijeme celý program z předchozí kapitoly), pak nakreslí na obrazovku šipku a umožní vám s ní po obrazovce pohybovat tak dlouho, dokud nestisknete klávesu, kterou jste si definovali jako **fire**. Souřadnice, kam ukazovala šipka při stisku **fire**, bude uložena v registru HL - v H bude y-ová souřadnice, v L pak bude souřadnice x-ová. Po stisku **fire** šipka z obrazovky zmizí. Následující program připište za příklad z předchozí kapitoly (začíná návěštím **START**, smažte případný předprogram) a místo instrukce **ret** za návěštím **STOP** napište instrukci **jp SIPKA** a na začátek napište **ent \$**.

```
SIPKA ld bc,#2464      ;souřadnice šipky v bodech
      push bc         ;uschováme je na zásobník
      ld ix,MATRIX   ;registrem IX si ukážeme na grafiku
```

```

exx                                ;budeme používat i alternativní registry
ld hl, FREE32                       ;do HL' si uložíme adresu volného místa
exx                                ;a vrátíme se k původním registrům
ld a, b                               ;přeneseme X-ovou souřadnici do A
call #22B1                          ;a spočítáme adresu bytu v obrazovce
ld (RESSIP+1), hl                   ;budeme ji dále potřebovat později
ld (POCROT+1), a                   ;a uložíme si také polohu v bytu

```

Adresa špičky šipky je v bodech a levý horní roh má souřadnici (0,0), tedy jinak než v BASICu, osa Y je otočená - čísla rostou směrem dolů. Nejprve vypočteme, na jaké adrese je byte, kde bude nakreslena špička šipky a také o kolik bodů doprava se musí šipka posunout v tomto bytu (číslo od 0 do 7), obě čísla se zapíší pro pozdější použití.

```

SLQP      ld b, 16                       ;šipka je vysoká 16 bodů
          push bc                         ;ulož počet bytů pro vykreslení
          push hl                         ;ulož adresu kam se bude šipka kreslit
          push hl                         ;a to dokonce dvakrát (ji ulož)
          ld h, 0                          ;nyní připravíme předlohu a masku šipky
          ld l, (ix+0)                    ;vyzvedni první byte předlohy
          ld d, h                          ;příprava pro masku
          ld e, (ix+16)                   ;přečti první byte masky
          inc ix                            ;posuň ukazatel na další část grafiky
          ld a, 8                          ;odečti od osmi posun vzhledem
POCROT    sub 0                            ;k bytu pro kreslení šipky - posun zleva
          ld b, a                          ;a dej ho do registru B
SLQP2     add hl, hl                      ;posuň doleva předlohu
          ex de, hl                      ;prohoď vzájemně registry HL a DE
          add hl, hl                      ;posuň doleva masku
          ex de, hl                      ;prohoď registry zpátky
          djnz SLQP2                     ;opakuj tolikrát, kolikrát je potřeba

```

Posunutí jednoho bytu obrázku i masky se provádí pomocí instrukce **add hl,hl**, ta provádí posuny doleva, my však potřebujeme posun doprava, proto například místo tří posunů doprava provedeme pět posunů doleva.

```

ex (sp), hl                          ;do HL obnovíme adresu v obrazovce
pop bc                                  ;a do BC tak přijde obsah předlohy
ld a, (hl)                             ;vyzvedni původní obsah bytu
exx                                       ;a ulož jej
ld (hl), a                             ;do ukládací
inc hl                                    ;paměti
exx                                       ;a posuňte se na další adresu
ld a, d                                   ;vezmi první byte daného řádku masky
cpl                                       ;a převrať v něm nuly a jedničky
and (hl)                                 ;ponech z obrazovky to, co je mimo šipku
or b                                     ;připoj předlohu šipky
ld (hl), a                             ;a vše zapíš do obrazovky
call RIGHTL                             ;posuň se na další byte v řádku
ld a, (hl)                             ;vyzvedni původní obsah bytu
exx                                       ;a ulož jej
ld (hl), a                             ;do ukládací
inc hl                                    ;paměti

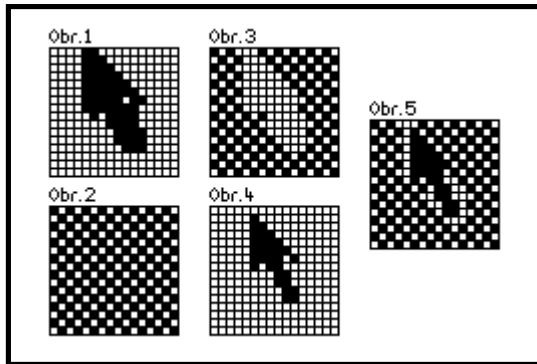
```

```

exx                ;a posuňte se na další adresu
ld a,e             ;vezmi druhý byte daného řádku masky
cpl                ;a převrať v něm nuly a jedničky
and (hl)          ;ponech z obrazovky to, co je mimo šipku
or c               ;připoj předlohu šipky
ld (hl),a         ;a vše zapiš do obrazovky
pop hl            ;obnov adresu do obrazovky
call DOWNHL      ;posuň se o mikrořádek dolů
pop bc            ;obnov počítadlo mikrořádků
djnz SLQP        ;opakuj dokud nevykreslíš celou šipku

```

Vlastní vykreslení šipky probíhá tak, že se vždy z obrazovky ponechá to, čemu v masce odpovídají nulové bity, k tomu se připojí předloha a vše se zase zapiše zpátky do obrazovky. Současně s kreslením šipky do obrazovky se ukládá do pomocné paměti původní obsah obrazovky aby mohl být vrácen při odkreslení šipky. Více vám snad napoví názorný obrázek, který ukazuje, co se děje s obrazovkou při jednotlivých logických operacích:



Na obrázku číslo jedna vidíte masku, je posunuta o čtyři body doprava, na obrázku číslo dvě je nějaký obrázek na obrazovce (jemná šachovnice), na obrázku číslo tři je vidět, co se stane s obrázkem na obrazovce, když se spojí s maskou (instrukce **ld a,d cpl and (hl)**). Na obrázku číslo čtyři je vidět předloha, stejně jako maska je také posunuta o čtyři body doprava. Konečně na obrázku číslo pět je vidět výsledek celého kreslení šipky na obrazovku (instrukce **or b ld (hl),a**).

```

halt                ;počkej na přerušení

```

Obrázek šipky se stihl vykreslit dříve, než se paprsek vykreslující obraz na monitor dostal do té části, kam můžeme na Spectru kreslit. Nyní program čeká až se na obrazovce zobrazí znovu celý obraz.

```

RESSIP ld hl,0          ;zde je nyní adresa do obrazovky
ld b,16             ;šipka je vysoká šestnáct bodů
ld de,FREE32       ;budeme vracet původní obsah obrazovky
RLOOP ld a,(de)     ;vzvedni byte z pomocné paměti
ld (hl),a          ;a zapiš jej zpět do obrazovky

```



```

inc de           ;posuň ukazatel do pomocné paměti
push hl         ;ulož ukazatel na obrazovku
call RIGHTL    ;posuň se na další v rámci mikrořádku
ld a,(de)      ;a znovu přesuň
ld (hl),a      ;jeden byte
inc de         ;zpátky do obrazovky
pop hl         ;obnov ukazatel na první byte řádků
call DOWNHL    ;posuň se dolů o mikrořádek
djnz RLOOP    ;smaž postupně celou šipku

```

Mazání šipky se provádí v době, kdy se vykreslování obrazu z paměti počítače na obrazovku televizoru teprve začíná - paprsek je v té době teprve v borderu a proto smazání šipky není vlastně vidět, bylo by, kdybychom počkali déle, my však stihneme šipku znovu vykreslit (možná na jiném místě).

```

call CONTROLS   ;do D si přečteme stav zvolených kláves

pop hl         ;obnov bodové souřadnice šipky
bit 4,d        ;testuj, zda není stisknuto FIRE
ret nz         ;a vrať se, když ano

ld c,3         ;nastav do C krok posunu šipky
bit 3,d        ;testuj směr nahoru
jr z,AA1      ;a odskoč, když není zvolen
ld a,h        ;dej Y-ovou souřadnici do A,
sub c         ;odečti krok posunu
ld h,a        ;a vrať Y-ovou souřadnici do H
jr nc,AA1     ;pokud nepodlezla nulu, odskoč
ld h,183      ;šipka vyjela nahoře a objeví se dole

AA1 bit 2,d    ;testuj směr dolů
jr z,AA2      ;a odskoč, když není zvolen
ld a,h        ;do A dej Y-ovou souřadnici,
add a,c       ;přičti krok posunu
ld h,a        ;a vrať zpět do H
cp 186       ;testuj spodní hranici obrazovky
jr c,AA2     ;a pokud není překročena, odskoč
ld h,0       ;šipka se objeví nahoře

AA2 bit 1,d    ;testuj směr doleva
jr z,AA3      ;odskoč, když není zvolen
ld a,l        ;vyzvedni X-ovou souřadnici,
sub c         ;odečti krok posunu
ld l,a        ;a vrať zpátky do L

AA3 bit 0,d    ;testuj směr doprava
jr z,AA4      ;a odskoč, když není zvolen
ld a,l        ;vyzvedni X-ovou souřadnici,
add a,c       ;přičti krok posunu
ld l,a        ;a vrať zpátky do L

AA4 ld (SIPKA+1),hl ;zapiš nové souřadnice
jp SIPKA      ;a skoč na nové vykreslení šipky

```

Šipka se tedy neustále vykresluje a maže, přesto neblíká - je to způsobeno tím, že se znovuykreslení provádí v době, kdy se zrovna nezobrazuje ta část obrazovky, do které můžete kreslit a psát. Uvědomte si, že se šipka takto vykreslí celkem 50krát za vteřinu. Do programu nemůžete libovolně vkládat cokoliv - zkuste si třeba před instrukcí **jp SIPKA** přidat nějaký čekací cyklus (ne pomocí **halt!**) - třeba **WAIT djnz WAIT**, jehož délku můžete ovlivnit nastavením registru B.

```

RIGHTL  ld  a,1           ;tento podprogram zajišťuje posun
        inc a           ;doprava s tím, že když dojde
        xor 1           ;k opuštění řádku na pravé straně,
        and 31          ;nastaví se adresa znovu na začátek
        xor 1           ;řádku vlevo
        ld  l,a
        ret

```

Tímto podprogramem se vlastně inkrementuje (zvětšuje o jedničku) spodních pět bitů v registru L. Program pracuje takto - přenesení obsah z registru L do registru A, tam jej zvětší o jedničku, pomocí XOR vytvoří jedničky na místech, kde se při inkrementaci něco změnilo oproti původnímu stavu, ponechá si z toho jen změny ve spodních pěti bitech, znovu pomocí XOR změní původní stav a vše vrátí do registru L - tím zůstane zachován původní obsah horních tří bitů a změní se pouze dolních pět bitů.

```

MATRIX  defb 0,64,96,112 ;předloha šipky v DEFB
        defb 120,124,122,88
        defb 12,12,6,6,0,0
        defb 0,0,192,224,240 ;od třetího čísla začíná maska
        defb 248,252,254,255
        defb 250,94,30,15,15
        defb 7,0,0,0

FREE32  defs 32         ;místo pro uložení povodního obsahu
        ;obrazovky

DOWNHL  ....          ;tento podprogram si můžete najít
        ;v předchozích kapitolách

```

Pokud vám program funguje tak, že se po definici ovládání okamžitě vrátí zpátky, přidejte nějakou pauzu po jejím dokončení (před instrukcí **jp SIPKA** po definici kláves).

Naše šipka se tedy může pohybovat po celé obrazovce. V případě, že dojde k nějakému okraji, objeví se na opačné straně - to je výhodné pro rychlejší přesun z jedné strany na druhou.

Budete-li chtít rychlost šipky snížit nebo zpřesnit nastavení šipky, musíte změnit krok - instrukce **ld c,3**. Šipku můžete také upravit tak, že se při delším stisku nějakého směru zrychlí její pohyb (podobně jako v ART STUDIU).

Budete-li chtít omezit možnost pohybu šipky, můžete upravit tu část programu, která se stará o změnu souřadnic šipky - následuje ihned po nastavení kroku posuvu - návěští **AA1** až **AA4** v programu.

# Jemná grafika

Pod tímto názvem se skrývá to, co lze v BASICu dělat pomocí příkazů **PLOT** a **DRAW**, tedy body a čáry. Použití ROM jsem již popsal (stručně, ale přece) v prvním díle a proto se jím již zabývat nebudu.

Nakreslit bod vlastně znamená nastavit jedničku ve vybraném bitu na vybrané adrese. Způsob, jak zjistit adresu bytu a bit, který se má nastavit, ze souřadnic, je tedy celé umění. Nebudeme se zabývat atributy - ty se nastavují obdobně jako při tisku znaků - do podprogramu **ATTRADR** přidejte za první instrukci navíc instrukci **and %11111000**. Případné nastavení atributů proveďte po zapsání bodu na obrazovku - v době, kdy máte registr HL nastavený na příslušný byte.

Budeme pracovat v jiné soustavě souřadnic, než na jakou jste zvyklí z BASICu. Souřadnice 0,0 bude mít levý horní roh obrazovky, X-ová souřadnice bude stejná, Y-ová souřadnice bude růst opačným směrem, maximální dosažitelný bod bude Y-ovou souřadnicí 191 (úplně nejspodnější mikrořádek na obrazovce), nebudeme tedy mít žádné „nedostupné“ oblasti obrazovky jako v BASICu.

Raději si opět ukážeme nějaké příklady:

```

PLOT1  ld  a,b           ;do A dej Y-ovou souřadnici
        call #22B1      ;vypočítej adresu a bit pro tento bod
        ld  b,a         ;číslo bitu dej do registru B
        inc b           ;změň rozsah z 0-7 na 1-8
        ld  a,1        ;zatím "-1" bit v bytu
PLOT1A rrca            ;zarotuj bytem doprava
        djnz PLOT1A    ;opakuj rotaci až do požadovaného bitu
        xor (hl)       ;xoruj původní obsah bytu s registrem A
        ld  (hl),a     ;a vrať vše zpátky
        ret            ;vrať se z podprogramu

```

U tohoto podprogramu je číslování bitů chápáno opačně - ne podle váhy (významu) bitu ale podle jeho polohy - nultý bit je tedy nejvíce vlevo. Výpočet adresy bytu a polohy bitu si popíšeme v dalším příkladu. Podprogram se volá se souřadnicemi v registrech B (Y-ová) a C (X-ová). Rutina nijak netestuje překročení Y-ové meze, pokud bude větší než 191, dojde k zapsání bitu do paměti mezi adresami 22528 až 24575. Pokud budete chtít, můžete si potřebné testování překročení rozsahu nebo změnu souřadnicové soustavy doplnit, stačí hodnotu v B odečíst od 192 (nebo 176 pokud budete chtít používat stejné souřadnice jako BASIC). Pokud dojde k přetečení, byla překročena povolená mez.

Ještě si spočítáme časovou náročnost tohoto podprogramu. Nebudu tu rozepisovat výpočet - počítal jsem to pomocí PROMETHEA a vyšlo mi toto - v nejrychlejším případě trvá vykreslení bodu **179 T-cyklů**, v nejpomalejším pak **298 T-cyklů**. Rozdílnost je způsobená různým počtem rotací - nejméně 1, nejvýše 8.

V dalším podprogramu odstraníme závislost rychlosti vykreslení bodu na poloze bodu vzhledem k bytu. Podprogram bude rychlejší než předchozí, také si tu objasníme jak pracuje již několikrát použitý podprogram **PIXEL-ADD** (na adrese #22B0):

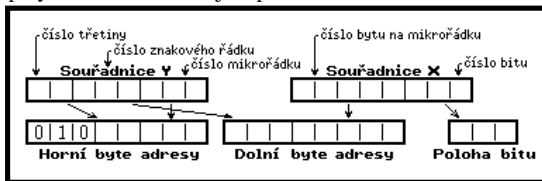
```

PLOT2   ld    a,b           ;přesuň Y-ovou souřadnici do A
        and  a           ;vynuluj příznak CARRY
        rra           ;zarotuj 0 zleva
        scf           ;nastav příznak CARRY
        rra           ;zarotuj 1 zleva
        and  a           ;vynuluj příznak CARRY
        rra           ;zarotuj 0 zleva
        xor  b           ;nyní do A přidáme spodní tři bity
        and  #F8        ;z původní Y-ové souřadnice
        xor  b           ;v A je nyní 64+8*INT(b/64)+(b mod 8)
        ld  h,a         ;vyšší byte adresy je tedy připraven
        ld  a,c         ;nyní budeme počítat nižší byte adresy
        rlca          ;zarotuj třikrát doleva
        rlca          ;pro správné umístění části Y-ové
        rlca          ;souřadnice mající vliv na nižší byte
        xor  b           ;přidáme bity 5,4,3 z Y-ové souřadnice
        and  %11000111 ;místo původních bitů - ty určují polohu
        xor  b           ;uvnitř bitu a pro adresu nemají význam
        rlca          ;nyní rotuj ještě dvakrát doleva,
        rlca          ;a takto vytvořený spodní byte
        ld  l,a         ;dej do registru L, v HL je nyní adresa
        ld  a,c         ;do A dáme ještě polohu bitu uvnitř bytu
        and  7          ;zde podprogram z ROM končí

        add  a,a         ;vynásob získané číslo 8 add a,a add a,a
        ld  b,a         ;dej číslo do registru B
        ld  a,254       ;kód instrukce set 7,(hl) do registru A
        sub  b           ;odečti osminásobek polohy a takto
        ld  (CODE+1),a ;získaný kód zapiš do instrukce,
        set  0,(hl)     ;kterou okamžitě proved'
CODE     ret            ;vrať se z podprogramu

```

Abyste lépe pochopili, co vlastně podprogram **PIXEL-ADD** provádí, ukážeme si to názorně na obrázku - v souřadnicích X a Y je vlastně zakódována celá adresa, stačí jen správně zpřeházet jednotlivé skupiny bitů a získáme jak požadovanou adresu, tak také polohu bitu v bytu. Číslo třetiny určuje, ve které třetině se bude bod nacházet, číslo znakového řádku určuje, v kterém řádku třetiny se bude bod nacházet, číslo mikrořádku určuje, na kterém z osmi mikrořádků příslušného znakového řádku se bude bod nacházet, číslo bytu na mikrořádku udává relativní adresu bytu vzhledem k počátku mikrořádku, číslo bitu určuje polohu bodu. vzhledem k bytu (0 je bit nejvíce vlevo, tedy vlastně sedmý bit). Způsob, jakým se přenáší skupinka bitů z jednoho registru do druhého si prohlédněte podrobněji - je velice zajímavý.



Ještě se krátce zastavím u způsobu, jakým je počítán kód instrukce **set ?,(hl)**. Jednotlivé instrukce **set 0,(hl)**, **set 1,(hl)**, ... **set 7,(hl)** mají operační kódy 198, 206, ... 254. Stačí tedy číslo bitu vynásobit osmi a odečíst od 254 - dostaneme operační kód požadované instrukce, pak stačí už jen zapsat tento kód na správné místo (instrukce má prefix #CB a tak je to až druhý byte) a vykonat právě vzniklou instrukci.

Pokud jste přemýšleli, jakým způsobem se vlastně bod na obrazovku vykreslí, mám zde na mysli to, jak se zachová v případě, že už tam bude jednička, pak jste mohli zjistit, že podprogram **PLOT1** kreslí body vlastně způsobem **OVER 1** v BASICu. Pokud je bit nulový, pak jej změní na jedničku, pokud v něm již jednička je, pak jej vynuluje. Naproti tomu podprogram **PLOT2** pracuje tak, že **vždy** daný bit nastaví na jedničku. Kdybyste chtěli první podprogram změnit, stačí instrukci **XOR** nahradit instrukcí **OR**, pak bude pracovat stejně jako **PLOT2**. Podprogram **PLOT2** však upravit tak, aby pracoval jako **PLOT1** nejde, můžete jej pouze upravit tak, aby daný plot smazal - místo kódu instrukce **set 7,(hl)**, který je 254, dáte kód instrukce **res 7,(hl)**, který je 190. Kdybyste chtěli, aby podprogram **PLOT1** bod pouze mazal, museli byste místo instrukce **xor (hl)** dát instrukce **cpl a and (hl)**.

Jistým opakem příkazu **PLOT** v BASICu je funkce **POINT**, její realizace je poměrně jednoduchá. Budete-li chtít, aby podprogram **PLOT1** prováděl funkci **POINT**, stačí, když místo instrukcí **xor (hl)** a **ld (hl),a** použijete instrukci, **and (hl)**. Podprogram se pak bude vracet s příznakem **nz** v případě, že tam bod bude, a s příznakem **z** v případě, že tam bod nebude. Můžete také testovat hodnotu v registru A, pokud je nulový, bod tam nebyl, pokud je nenulový, bod tam byl.

U podprogramu **PLOT2** stačí, když místo kódu instrukce **set 7,(hl)**, který je 254, použijete kód instrukce **bit 7,(hl)**, který je 126. Podprogram se pak bude vracet s nastaveným příznakem **ZERRO** (platí tedy podmínka **z**) v případě, že tam bod nebude, a s vynulovaným příznakem **ZERRO** (platí podmínka **nz**) v případě, že tam bod bude.

Sami byste nyní měli být schopni napsat program, který bude přenášet vybraný kus obrazovky na jiné místo - dva cykly v sobě, test bodu a jeho zápis na novou pozici. Program můžete různě upravovat, můžete třeba vytvořit zrcadlovou kopii a podobně.

Nyní si ukážeme, jak se dá kreslení bodu ještě zrychlit, také uvidíte několik efektů a v textu naleznete užitečný podprogram, který generuje náhodná čísla v rozsahu 0 až 65535 - je to obdoba funkce **RND** z BASICu. Dejte se do opisování, při psaní si o některých zajímavých částech příkladu řekneme více:

```

START    call MAKETAB      ;vytvoř tabulku adres mikrořádků
          ei              ;povol přerušení
          ld bc,0         ;začínáme v levém horním rohu
LOOP1    push bc          ;ulož souřadnice - Smyčka přes řádky
LOOP2    push bc          ;ulož souřadnice - smyčka přes sloupce
          call PLOT3      ;nakresli bod na souřadnicích z BC
          pop bc          ;obnov souřadnice
          inc c           ;zvyš X-ovou souřadnici o jedničku
          jr nz,LOOP2     ;pokud není nulová, kresli body

```

```

pop af ;do A dej Y-ovou souřadnici
inc a ;zvětši ji o jedničku
cp 192 ;testuj spodní konec obrazovky
ld b,a ;ulož zpátky do B a pokud nejsi na
jr c,LOOP1 ;konci, pokračuj dalším řádkem

```

Právě napsaná část programu tedy vlastně vyplní celou obrazovku body - při spuštění by měla být prázdná, jinak ji zinvrtuje. Pokud používáte assembler, který před spuštěním programu nesmaže obrazovku, pak musíte na začátek programu tuto operaci přidat.

```

CLEAR ld hl,0 ;celkem 65536 bodů
CLEAR2 push hl ;ulož počítadlo bodů
call RANDOM ;dej do HL náhodné číslo
ld a,h ;nyní otestuj, jestli se v H
cp 192 ;nenachází číslo větší než 192
jr c,CLEAR3 ;pokud ne, odskoč, pokud ano,
sub 128 ;odečti od něj 128
CLEAR3 ld c,l ;nyní přeneseme souřadnice bodu
ld b,a ;do registrů B a C
call PLOT3 ;a vykreslíme bod (typ OVER 1)
pop hl ;obnov počítadlo bodů
dec hl ;zmenši počet bodů o jedničku
ld a,h ;otestuj, jestli
or l ;již nejsou všechny
jr nz,CLEAR2 ;a pokud ne, zpracuj další

```

To, co jste právě napsali, je vlastně jakési efektní invertování obrazovky, kdy je každému bodu změněn obsah - ještě se k němu vrátíme.

```

NO_STARS equ 25 ;celkem 25 hvězd se bude pohybovat

STARS ld b,NO_STARS ;do B dej počet hvězd
ld ix,SPACE ;a do IX ukazatel na tabulku parametrů
STARS2 call RANDOM ;do HL dej náhodné číslo
ld a,h ;a uprav hodnotu v registru H
and 63 ;na rozsah 0 až 63
add a,64 ;a přičti 64, rozsah je 64 až 127
ld (ix+0),l ;zapiš X-ovou souřadnici hvězdičky
ld (ix+1),a ;zapiš Y-ovou souřadnici hvězdičky
call RANDOM ;spočítej další náhodné číslo
ld a,l ;a uprav jeho rozsah
and 7 ;nejprve na 0 až 7
inc a ;a pak na 1 až 8
ld (ix+2),a ;zapiš rychlost hvězdičky
ld de,3 ;posuň se na další
add ix,de ;hvězdičku
djnz STARS2 ;a totéž proved s dalšími hvězdičkami
jr STARS7 ;skoč do vykreslení hvězdiček

STARS5 ld ix,SPACE ;nastav do IX tabulku parametrů
ld b,NO_STARS ;a do B dej počet hvězdiček
STARS4 push bc ;ulož počet hvězdiček
ld c,(ix+0) ;do C dej X-ovou souřadnici

```

```

ld b,(ix+1) ;do B dej Y-ovou souřadnici
call PLOT3 ;smaž hvězdičku
ld de,3 ;a posuň se na další
add ix,de ;hvězdičku
pop bc ;obnov počítadlo hvězdiček
djnz STARS4 ;opakuj pro všechny hvězdičky

STARS7 ld ix,SPACE ;nastav tabulku parametrů do IX
ld b,NO_STARS ;nastav počet hvězdiček do B
STARS3 push bc ;ulož počítadlo hvězdiček na zásobník
ld a,(ix+2) ;vzvedni rychlost hvězdičky
ld c,(ix+0) ;vzvedni X-ovou souřadnici
add a,c ;změň X-ovou souřadnici o rychlost
ld c,a ;vrať novou X-ovou souřadnici do C
ld (ix+0),a ;a zapiš ji zpátky do tabulky
ld b,(ix+1) ;do B dej Y-ovou souřadnici hvězdičky
call PLOT3 ;vykresli hvězdičku
ld de,3 ;a posuň se na
add ix,de ;další hvězdičku
pop bc ;obnov počítadlo hvězdiček
djnz STARS3 ;opakuj pro všechny hvězdičky
ld a,0 ;nastav černý
out (254),a ;border
halt ;počkej na přerušení
ld a,4 ;nastav zelený
out (254),a ;border
call 8020 ;testuj stisk klávesy BREAK
ret nc ;při jejím stisku se vrať
jr STARS5 ;skoč na mazání hvězdiček

```

Právě jste napsali něco, co bude na obrazovce v prostřední třetině kreslit letící hvězdy. Program pracuje tak, že po přerušení smaže a ihned zase vykreslí všechny hvězdičky. Doba, po kterou se hvězdičky překreslují, je na borderu signalizována černou barvou, doba, po kterou program čeká na přerušení je v borderu signalizována zelenou barvou. Hvězdičky se na obrazovce nevyskytují (jsou smazány a nejsou nakresleny) pouze v době, kdy obrazovka není zrovna vykreslována - paprsek běží borderem.

Budete-li chtít zvětšit počet hvězd, měli byste program upravit - tak, že se každá hvězdička smaže a vzápětí znovu nakreslí na novém místě (zatím se všechny najednou smažou a pak se najednou znovu nakreslí).

```

PLOT3 ld l,b ;do L dej Y-ovou souřadnici
ld h,SCRNADRS/512 ;polovina horního bytu tabulky adres
add hl,hl ;vynásob dvěma
ld a,(hl) ;vzvedni
inc l ;do HL
ld h,(hl) ;adresu
ld l,a ;počátku mikrořádku
ld a,c ;nyní přičteme
rrca ;posun vzhledem
rrca ;k počátku mikrořádku,
rrca ;který je osminou X-ové souřadnice
and 31 ;ponech pouze spodních pět bitů
add a,l ;a přičti
ld l,a ;nyní máme adresu v registru HL

```

```

ld a,c ;nyní ještě bitovou masku
and 7 ;tu získáš tak, že ponecháš 3 bity
ld c,a ;vrať zpátky do C
ld b,TABLE/256 ;nyní je v BC adresa bitové masky
ld a,(bc) ;vzvedni bitovou masku do A
xor (hl) ;naXORuj původní obsah obrazovky
ld (hl),a ;a vrať výsledek do obrazovky
ret ;návrat z podprogramu

MAKETAB ld b,192 ;obrazovka má 192 mikrořádků
ld de,16384 ;adresa prvního mikrořádku na obrazovce
ld hl,SCRNADRS ;adresa tabulky
MAKETAB2 ld (hl),e ;zapiš spodní byte do tabulky
inc hl ;posuň se na další byte
ld (hl),d ;zapiš horní byte do tabulky
inc hl ;posuň se
ex de,hl ;prohoď HL a DE
call DOWNHL ;spočítej adresu spodního bytu
ex de,hl ;prohoď HL a DE zpátky
djnz MAKETAB2 ;opakuj pro každý mikrořádek
ret

```

Tento podprogram vytvoří tabulku adres začátků mikrořádků. Mohli byste jej naprogramovat i bez použití podprogramu DOWNHL - museli byste použít podprogram PIXEL-ADD, který je v paměti ROM.

```

RANDOM ld de,0 ;zde je zapsáno poslední náhodné číslo
ld h,e ;dále se spočítá nové číslo v sérii
ld l,253 ;„náhodných“ čísel, způsob, jak se
ld a,d ;to dělá vám přesně nepopíši, protože
or a ;to nevím, tento program pochází
sbc hl,de ;z celočíselného kompilátoru
sbc a,0 ;firmy HISOFT, máte-li zájem, můžete
sbc hl,de ;si způsob výpočtu zjistit sami,
sbc a,0 ;důležité je vědět, že sekvence čísel,
ld e,a ;která je tímto podprogramem generována
ld d,0 ;obsahuje všechna čísla v rozsahu
sbc hl,de ;0 až 65535 a každé pouze jednou,
jr nc,AB ;čísla se samozřejmě opakují
inc hl ;s periodou 65536
AB ld (RANDOM+1),hl
ret

```

Jak možná víte, nejsou „náhodná“ čísla v počítači skutečně náhodná, proto se jim také říká „pseudonáhodná“. Náhodná jsou pro nás jen tehdy, pokud nevíme jak se vytvářejí a z jakého čísla se vychází. Proto tento podprogram používejte tak, že na začátku zapíšete nějaké náhodné číslo na adresu **RANDOM+1** a **RANDOM+2** - můžete použít hodnotu systémové proměnné **FRAMES** (ta se mění s časem a ten je obvykle závislý také na uživateli) a obsah registru **R** (ten se také s časem mění). Pokud potřebujete (podobně jako zde) pouze čísla z určitého rozsahu hodnot, a nevíte, že se sekvence bude po každém spuštění opakovat stejně, nemusíte žádnou inicializaci provádět).



```

LAST          ;návěštlí pro zjištění konce programu
AOLEN        equ  $-START          ;návěštlí bude obsahovat délku programu
              org  LAST/512+1*512 ;takto se nastaví ORG na první další
              ;volnou adresu, jejíž horní byte je
              ;dělitelný čtyřmi (podmínka pro PLOT3)

SCRNADRS     defs 512              ;vynech 512 bytů na tabulku adres

TABLE        defb 128,64,32,16     ;tabulka bitových masek
              defb 8,4,2,1

SPACE        defs 3*NO_STARS       ;vynech místo pro tabulku hvězdiček

```

Už jste asi měli možnost si program prohlédnout v chodu, přemýšleli jste o tom, proč se vlastně při bodovém invertování prostřední pruh takřka smazal (rychleji než oba dva krajní) a pak najednou zase úplně vybarvil? Řekli jsme si, že podprogram RANDOM vrací všechna čísla v rozmezí 0 až 65535 právě jednou, nicméně v programu se čísla z rozsahu 192-255 upravují na rozsah 64-127, do tohoto rozsahu pak tedy vlastně padne dvakrát tolik čísel, než jinač - proto se pruh nejdříve smazal a pak zase zaplnil.

Pokud chcete, aby program invertoval celou obrazovku stejně, stačí, když místo instrukce **jr c,CLEAR3** vložíte instrukci **jr nc,CLEAR3**, přemístíte návěštlí **CLEAR3** z instrukce **ld c,l** na instrukci **pop hl** a smažete instrukci **sub 128**.

Pokud budete chtít tento program použít na efektní mazání obrazovky, použijte nějaký kratší podprogram pro **PLOT**, tenhle je sice rychlý ale zbytečně dlouhý a upravte jej tak, aby body bud doplňovat nebo mazal (nikoliv měnil).

U každého podprogramu pro kreslení bodu jsme si psali časovou náročnost, tento trvá celkem 128 T-cyklů. Program se dá ještě zrychlit (na 113 T-cyklů) tím, že tabulku TABLE zopakujete 32 krát za sebou (samozřejmě programově) - nebude potom potřeba upravovat registr C. Také můžete vhodným přeorganizováním tabulky adres docílit zrychlení až na 102 T-cykly, napovím vám, že musíte odstranit instrukci **add hl,hl** a místo instrukce **inc l** použijete nyní instrukci **inc h**, není to však všechno. Další zrychlení lze provést tím, že trochu promícháte čtení adresy z tabulky a přičtení části X-ové souřadnice k dolnímu bytu adresy - tak se dostanete až na 94 T-cyklů (dokonce 90 T-cyklů v případě, že bude Y-ová souřadnice rovnou v registru L). Jestli někdo vymyslíte ještě rychlejší **PLOT**, pak mi napište (na adresu firmy PROXIMA), vaše řešení by mě opravdu zajímalo.

Nyní se budeme věnovat čárám, tedy ne černé magii ale tomu, co v BASICu dokážete dělat pomocí příkazu **DRAW**. Nejprve si ukážeme mírně upravený podprogram z ROM, kterým se kreslí čáry. Algoritmus, podle kterého tento podprogram kreslí je nejrychlejší a nejjednodušší algoritmus, který znám a pravděpodobně ani lepší neexistuje.

```

...
ld  bc,0          ;nejprve nakreslíme bod na 0,0
ld  (LASTXY+1),bc ;tento bod také bude výchozí pro čáru
call PLOT3        ;bod nakreslíme, DRAW to nedělá

```

```

ld de,257 ;do D a E dej jedničky (volba směru)
ld bc,191*256*255 ;do B dej 191, do C dej 255
call DRAW ;nakreslí čáru 255 doprava a 191 dolů
...

DRAW ld a,c ;porovnej X a Y
cp b ;a pokud je X větší,
jr nc,DLXGTY ;skoč dopředu
ld l,c ;do L dej menší rozměr (nyní směr X)
push de ;ulož volbu směrů
xor a ;vlož vertikální krok, protože čára
ld e,a ;půjde rychleji nahoru nebo dolů
jr DLLARGER ;než do stran, skoč do společné části
DLXGTY or c ;tímto se zjistí, jestli nejsou oba
ret z ;rozměry nulové, pak není co kreslit
ld l,b ;do L dej menší rozměr (nyní směr Y)
ld b,c ;do B dej větší rozměr
push de ;uschovej volbu směrů
ld d,0 ;nyní nastav horizontální krok
DLLARGER ld h,b ;dej větší rozměr také do H
ld a,b ;do A dej polovinu
rra ;většího rozměru čáry
DLLOOP add a,l ;přičti menší rozměr a pokud je větší
jr c,DLDIAG ;než 255, skoč na diagonální krok
cp h ;nyní porovnej s větším rozměrem a pokud
jr c,DLHRVT ;je menší, skoč na svislý (vodor.) krok
DLDIAG sub h ;A je větší než H, zmenší jej o H
ld c,a ;a zapiš do C (pro uschování)
exx ;přepni na alternativní registry
pop bc ;naplň alternativní BC volbou směru
push bc ;a opět hodnotu vrať na zásobník
jr LASTXY ;skoč na provedení kroku

DLHRVT ld c,a ;A zapiš do C (pro uschování)
push de ;ulož nastavený horizontální nebo
exx ;diagonální krok na zásobník a potom
pop bc ;z něj do alternativního BC
LASTXY ld hl,0 ;do HL souřadnice minulého bodu
ld a,b ;nejprve zpracujeme Y-ovou
add a,h ;souřadnici, přičteme k ní
ld b,a ;nastavený krok
ld a,c ;potom zpracujeme X-ovou
inc a ;souřadnici, nejprve ji zvětšíme
add a,l ;o jedničku a přičteme k ní krok,
jr c,DLRANGE ;při přetečení skoč na test
jr z,DLERR ;při nule (=256) skoč na chybu
DLPLOT dec a ;jedničku zase odečti
ld c,a ;a vrať hodnotu do C
ld (LASTXY+1),bc ;zapiš nové souřadnice pro příště
call PLOT3 ;vykreslí bod
exx ;přepni registry
ld a,c ;obnov hodnotu v A (pro volbu kroku)
djnz DLLOOP ;opakuji pro všechny body delšího rozměru
pop de ;odeber zbytečnou hodnotu
ret ;a vrať se z kreslení čáry

DLRANGE jr z,DLPLOT ;pokud platí, jde o 255 a to je ještě OK

```

```
DLERR    pop    af          ;odeber hodnotu - zde by mohlo být
         ret              ;vypsání chyby - a vrať se
```

Podprogram střídá diagonální (posun v obou směrech) a vertikální nebo horizontální kroky (posun pouze v jednom směru). Ke zjišťování, jaký typ kroku se má použít, se na začátku do registru A nastaví polovina delšího rozměru a pak se přičítá rozměr kratší. Pokud při přičtení kratšího rozměru dojde k překročení rozměru většího, provede se odečtení většího rozměru a diagonální krok, pokud ne, provede se horizontální nebo vertikální krok. Tento způsob kreslení vychází z parametrického vyjádření úsečky, pokud je znáte, můžete přemýšlet, jak spolu souvisí - toto je vlastně jakési rozložení problému.

Dále se budeme zabývat některými speciálními případy přímk - vodorovné a svislé nebo diagonální. Začneme třeba tou svislou - pokusíme se zrychlit kreslení přímky a použijeme k tomu již notoricky známé podprogramy DOWNHL a UPHL. Podprogram bude mít na vstupu v registrech B a C souřadnice (stejně jako PLOT) a v registru A délku čáry.

```
DRAWDOWN ld    e,a          ;uschovej délku čáry do E
         ld    l,b          ;Y-ovou souřadnici do registru L
         ld    h,SCRNADRS/512 ;polovina horního bytu tabulky adres
         add   hl,hl        ;vynásob dvěma
         ld    a,(hl)       ;vyzvedni
         inc  l             ;do HL
         ld    h,(hl)       ;adresu
         ld    l,a          ;počátku mikrořádku
         ld    a,c          ;nyní přičteme
         rrca             ;posun vzhledem
         rrca             ;k počátku mikrořádku,
         rrca             ;který je osminou X-ové souřadnice
         and  31           ;ponech pouze spodních pět bitů
         add   a,l          ;a přičti
         ld    l,a          ;nyní máme adresu v registru HL
         ld    a,c          ;nyní ještě bitovou masku
         and  7            ;tu získáš tak, že ponecháš 3 bity
         ld    c,a          ;vrať zpátky do C
         ld    b,TABLE/256 ;nyní je v BC adresa bitové masky
         ld    a,(bc)       ;vyzvedni bitovou masku do A
         ld    c,a          ;dej bitovou masku do C
         ld    b,e          ;délku čáry do B
DRAWD2   ld    a,(hl)       ;vykresli bod
         xor  c             ;na zadaných
         ld    (hl),a        ;souřadnicích
         call DOWNHL        ;a posuň adresu o byte dolů
         djnz DRAWD2        ;a pokud není čára hotova, opakuj
         ret              ;konec kreslení
```

Můžete si všimnout, že tento způsob kreslení nepočítá stále znovu adresu každého bodu čáry, využívá toho, že bod, který je o jeden mikrořádek níž než bod předcházející, má stejnou bitovou masku a liší se pouze v adrese.

Pokud budete chtít kreslit čáru opačným směrem, tedy nahoru, musíte nahradit posun adresy dolů (volání podprogramu **DOWNHL**) posunem adresy nahoru (volání podprogramu

**UPHL**). Budete-li chtít kreslení ještě urychlit, musíte podprogram **DOWNHL** rozepsat místo jeho volání a instrukce **ret** samozřejmě nahradit instrukcemi **jr**.

Těď si ukážeme první způsob, jak kreslit vodorovné čáry. Podprogram se bude velice podobat předchozímu a bude kreslit čáry doprava.

```

DRAWRGHT ld e,a ;uschovej délku čáry do E
          ld l,b ;Y-ovou souřadnici do registru L
          ld h,SCRNADRS/512 ;polovina horního bytu tabulky adres
          add hl,hl ;vynásob dvěma
          ld a,(hl) ;vzvedni
          inc l ;do HL
          ld h,(hl) ;adresu
          ld l,a ;počátku mikrořádku
          ld a,c ;nyní přičteme
          rrca ;posun vzhledem
          rrca ;k počátku mikrořádku,
          rrca ;který je osminou X-ové souřadnice
          and 31 ;ponech pouze spodních pět bitů
          add a,l ;a přičti
          ld l,a ;nyní máme adresu v registru HL
          ld a,c ;nyní ježte bitovou masku
          and 7 ;tu získáš tak, že ponecháš 3 bity
          ld c,a ;vrať zpátky do C
          ld b,TABLE/256 ;nyní je v BC adresa bitové masky
          ld a,(bc) ;vzvedni bitovou masku do A
          ld c,a ;dej bitovou masku do C
          ld b,e ;délku čáry do B
DRAWR2 ld a,(hl) ;vykreslí bod
        xor c ;na zadaných
        ld (hl),a ;souřadnicích
        rrc c ;zarotuj masku doprava a pokud
        jr nc,DRAWR3 ;došlo k přesunu jedničky přes okraj
        inc l ;posuň také adresu
DRAWR3 djnz DRAWR2 ;a pokud není čára hotova, opakuj
        ret ;konec kreslení

```

Chcete-li čáru kreslit doleva, nahraďte rotaci doprava rotací doléva (instrukci **rrc c** nahraďte instrukcí **rlc c**) a posun adresy doprava posunem adresy (instrukci **inc l** nahraďte instrukcí **dec l**).

Kreslení čáry ve vodorovném směru by bylo možno ještě zrychlit, ptáte se jak? (jednoduše a navíc samozřejmě bez namáčení!). Nápad je takový, že by se dalo osm vedle sebe ležících bodů kreslit najednou (pokud jsou v jednom bytu). Menší problémy budeme mít se začátkem a ukončením čáry, ty však vyřešíme pomocí tabulky). Program je vytvořen tak, aby se čáry XORovaly a bylo možno je druhým vykreslením smazat.

```

DRAWRPST ld d,a ;délku čáry přemísti do registru D
          ld l,b ;Y-ovou souřadnici do registru L
          ld h,SCRNADRS/512 ;polovina horního bytu tabulky adres
          add hl,hl ;vynásob dvěma
          ld a,(hl) ;vzvedni

```

```

inc 1 ;do HL
ld h,(hl) ;adresu
ld l,a ;počátku mikrořádku
ld a,c ;nyní přičteme
rrca ;posun vzhledem
rrca ;k počátku mikrořádku,
rrca ;který je osminou X-ové souřadnice
and 31 ;ponech pouze spodních pět bitů
add a,l ;a přičti
ld l,a ;nyní máme adresu v registru HL
ld a,c ;testuj, zde jsou X-ová souřadnice
or d ;a délka nulové
ld e,255 ;dej do E binárně samé jedničky
ld b,32 ;nastav počet bytů na 32
jr z,DRAWRF3 ;a odskoč na vlastní kreslení
ld a,c ;nyní ještě bitovou masku
and 7 ;tu získáš tak, že ponecháš 3 bity
add a,TABLE2-TABLE ;počátek tabulky TABLE2 není na nule
ld c,a ;vrať zpátky do C
ld b,TABLE2/256 ;nyní je v BC adresa bitové masky
ld a,(bc) ;vyzvedni masku pro začátek čáry
ld e,a ;a ulož ji do E

ld a,c ;zjistí, kolik bodů ze začátku čáry
and 7 ;se v prvním bytu vynechává, přičti toto
add a,d ;číslo k délce a odečti osmičku, je to
sub 8 ;vlastně délka čáry, která by zbyla po
ld d,a ;zaplnění části prvního bytu, dej do D
jr c,DRAWRF2 ;pokud došlo k přetečení, odskoč dál

ld a,(hl) ;nyní můžeš úspěšně nakreslit část
xor e ;čáry, která padne
ld (hl),a ;do prvního bytu,
inc l ;posuň se na další byte

ld a,d ;nyní zjistíme, kolik celých bytů čára
rra ;zabírá, vydělíme tedy
rra ;obsah registru A
rra ;celkově osmi,
and 31 ;ponech číslo od nuly do třiceti jedné
ld e,255 ;do E dej číslo 255 - maska
jr z,DRAWRF2 ;odskoč pokud není žádný celý byte
ld b,a ;dej do registru B počet bytů

DRAWRF3 ld a,(hl) ;a jednotlivé byty
xor e ;čáry postupně
ld (hl),a ;vykreslí do
inc l ;obrazovky,
djnz DRAWRF3 ;opakuj pro všechny byty

DRAWRF2 ld a,d ;do A dej délku čáry
and 7 ;a ponech z ní jen délku v jednom bytu,
ret ;pokud je nulová, čára je hotová, konec
add a,TABLE3-TABLE ;přičti relativní počátek třetí tabulky
ld c,a ;a dej ho do registru C,
ld b,TABLE3/256 ;do registru B dej horní byte adresy
ld a,(bc) ;dej do A masku pro byte konce čáry
and e ;spoj předchozí masku s právě získanou

```

```

xor (hl)           ;a vše zapiš do obrazovky
ld (hl),a         ;stejně jako v předchozích
ret              ;případech

TABLE   defb #80,#40,#20,#10 ;tuto tabulku už byste měli mít
        defb #08,#04,#02,#01 ;napsanou, jen ji zkontrolujte

TABLE2  defb %11111111      ;tabulka začátků čáry
        defb %01111111      ;bitové masky pro XOR
        defb %00111111
        defb %00011111
        defb %00001111
        defb %00000111
        defb %00000011
        defb %00000001

TABLE3  defb %00000000      ;tabulka konců čáry
        defb %10000000      ;bitové masky pro XOR
        defb %11000000
        defb %11100000
        defb %11110000
        defb %11111000
        defb %11111100
        defb %11111110

```

Pokud je čára krátká a vejde se do jednoho bytu, získá se její výsledná maska spojením příslušné počáteční a koncové masky pomocí logické bitové funkce AND. Rychlost, s jakou se čára kreslí je několikanásobně vyšší než u předchozího příkladu. Tento podprogram použijeme při kreslení kruhu - vyplněné kružnice.

Abychom nezůstali jen u bodů a čar, ukážeme si, jak se dá nakreslit třeba kružnice. Náš podprogram bude „poněkud“ rychlejší než ten, který používá ROM. V registrech D a E vstupují souřadnice středu a v registru A vstupuje poloměr kružnice. Napřed ještě upravte podprogram **PLOT3** tak, aby zachovával hodnoty registrů HL a DE.

```

CIRCLE  call SQUAREA        ;spočítej druhou mocninu čísla v A
        push hl            ;uschovej výsledek na zásobník
        srl h              ;nyní proved dělení
        rr l               ;obsahu HL dvěmi
        call SQRROOT       ;a spočítej odmocninu - souřadnice X
        ld a,c             ;a do A dej celou část výsledku

        pop hl             ;obnov druhou mocninu poloměru
        push af            ;uschovej relativní souřadnici X
        inc a              ;zvětši o jedničku
        push af            ;také uschovej
        dec a              ;nyní zase vrať na původní hodnotu

        push hl            ;ulož druhou mocninu poloměru
        call SQUAREA       ;druhá mocnina X-ové souřadnice
        ld c,l             ;a výsledek dej do

```

```

ld    b,h          ;registru BC
pop   hl          ;obnov druhou mocninu poloměru
or    a           ;vynuluj příznak CARRY
push  hl         ;ulož opět druhou mocninu poloměru
sbc   hl,b       ;a odečti od ní druhou mocninu X
call  SQROOT    ;výsledek odmocni a máš Y souřadnici
pop   hl         ;obnov druhou mocninu poloměru

pop   bc         ;obnov původní číslo do B
ld    c,b       ;a přesuň ho také do C
sub   c         ;nyní odečti od A
dec   a         ;a odečti jedničku, pokud nedojdeš
jr    nz,CIRCLE0 ;k nule, odskoč dál

push  bc         ;ulož relativní souřadnice vůči středu
ld    a,d       ;nyní přičti k Y-ové souřadnici středu
add   a,b       ;relativní Y-ovou souřadnici a dostaneš
ld    b,a       ;skutečnou Y-ovou souřadnici bodu
ld    a,e       ;přičti k X-ové souřadnici středu
add   a,c       ;relativní X-ovou souřadnici a dostaneš
ld    c,a       ;tak skutečnou X-ovou souřadnici bodu
call  PLOT3     ;vykresli bod na těchto souřadnicích
pop   bc         ;obnov relativní souřadnice vůči středu

push  bc         ;ulož relativní souřadnice
ld    a,d       ;od Y-ové souřadnice středu odečti
sub   b         ;relativní Y-ovou souřadnici a máš
ld    b,a       ;skutečnou Y-ovou souřadnici bodu
ld    a,e       ;od X-ové souřadnice středu odečti
sub   c         ;relativní X-ovou souřadnici a máš
ld    c,a       ;skutečnou X-ovou souřadnici bodu
call  PLOT3     ;vykresli bod
pop   bc         ;obnov relativní souřadnice

push  bc         ;ulož relativní souřadnice
ld    a,d       ;k Y-ové souřadnici středu přičti
add   a,b       ;relativní Y-ovou souřadnici a máš
ld    b,a       ;skutečnou Y-ovou souřadnici bodu
ld    a,e       ;od X-ové souřadnice středu odečti
sub   c         ;relativní X-ovou souřadnici a máš
ld    c,a       ;skutečnou X-ovou souřadnici bodu
call  PLOT3     ;vykresli bod
pop   bc         ;obnov relativní souřadnice

push  bc         ;ulož relativní souřadnice
ld    a,d       ;od Y-ové souřadnice středu odečti
sub   b         ;relativní Y-ovou souřadnici a máš
ld    b,a       ;skutečnou Y-ovou souřadnici bodu
ld    a,e       ;k X-ové souřadnici středu přičti
add   a,c       ;relativní X-ovou souřadnici a máš
ld    c,a       ;skutečnou X-ovou souřadnici bodu
call  PLOT3     ;vykresli bod
pop   bc         ;obnov relativní souřadnice

CIRCLE0 pop af    ;obnov horní hranici pro X

CIRCLE2 push af   ;ulož relativní X-ovou souřadnici
push  af         ;a ulož ještě jednu

```

```

push hl ;ulož druhou mocninu poloměru
call SQUARE ;druhá mocnina X-ové souřadnice
ld c,l ;a výsledek dej do
ld b,h ;registru BC
pop hl ;obnov druhou mocninu poloměru
or a ;vynuluj příznak CARRY
push hl ;ulož opět druhou mocninu poloměru
sbc hl,bc ;a odečti od ní druhou mocninu X
call SQRROOT ;výsledek odmocni a máš Y souřadnici
ld b,a ;tu dej do B a máme relativní souřadnice
pop hl ;obnov druhou mocninu poloměru
pop af ;a obnov relativní X-ovou souřadnici
ld c,a ;a přesuň ji do C

push bc ;ulož relativní souřadnice
ld a,d ;k Y-ové souřadnici středu přičti
add a,c ;relativní X-ovou souřadnici a výsledek
ex af,af' ;zatím ulož do záložního A
ld a,e ;k X-ové souřadnici středu přičti
add a,b ;relativní Y-ovou souřadnici a máš
ld c,a ;skutečnou X-ovou souřadnici
ex af,af' ;nyní obnov původní obsah registru A
ld b,a ;a v B je nyní skutečná souřadnice
call PLOT3 ;vykresli bod
pop bc ;obnov relativní souřadnice

push bc ;ulož relativní souřadnice vůči středu
ld a,d ;nyní přičti k Y-ové souřadnici středu
add a,b ;relativní Y-ovou souřadnici a dostaneš
ld b,a ;skutečnou Y-ovou souřadnici bodu
ld a,e ;přičti k X-ové souřadnici středu
add a,c ;relativní X-ovou souřadnici a dostaneš
ld c,a ;tak skutečnou X-ovou souřadnici bodu
call PLOT3 ;vykresli bod na těchto souřadnicích
pop bc ;obnov relativní souřadnice vůči středu

push bc ;ulož relativní souřadnice
ld a,d ;k Y-ové souřadnici středu přičti
add a,c ;relativní X-ovou souřadnici a výsledek
ex af,af' ;zatím ulož do záložního A
ld a,e ;od X-ové souřadnice středu odečti
sub b ;relativní Y-ovou souřadnici a máš
ld c,a ;skutečnou X-ovou souřadnici
ex af,af' ;nyní obnov původní obsah registru A
ld b,a ;a v B je nyní skutečná souřadnice
call PLOT3 ;vykresli bod
pop bc ;obnov relativní souřadnice

push bc ;ulož relativní souřadnice
ld a,d ;od Y-ové souřadnice středu odečti
sub b ;relativní Y-ovou souřadnici a máš
ld b,a ;skutečnou Y-ovou souřadnici bodu
ld a,e ;od X-ové souřadnice středu odečti
sub c ;relativní X-ovou souřadnici a máš
ld c,a ;skutečnou X-ovou souřadnici bodu
call PLOT3 ;vykresli bod
pop bc ;obnov relativní souřadnice

```



```

pop af ;obnov relativní X-ovou souřadnici
push af ;a opět ji ulož zpátky na zásobník
or a ;testuj, zda je nulová a pokud ano
jr z,CIRCLE3 ;přeskoč následující část programu

push bc ;ulož relativní souřadnice
ld a,d ;k Y-ové souřadnici středu přičti
add a,b ;relativní Y-ovou souřadnici a máš
ld b,a ;skutečnou Y-ovou souřadnici bodu
ld a,e ;od X-ové souřadnice středu odečti
sub c ;relativní X-ovou souřadnici a máš
ld c,a ;skutečnou X-ovou souřadnici bodu
call PLOT3 ;vykreslí bod
pop bc ;obnov relativní souřadnice

push bc ;ulož relativní souřadnice
ld a,d ;od Y-ové souřadnice středu odečti
sub b ;relativní Y-ovou souřadnici a máš
ld b,a ;skutečnou Y-ovou souřadnici bodu
ld a,e ;k X-ové souřadnici středu přičti
add a,c ;relativní X-ovou souřadnici a máš
ld c,a ;skutečnou X-ovou souřadnici bodu
call PLOT3 ;vykreslí bod
pop bc ;obnov relativní souřadnice

push bc ;ulož relativní souřadnice
ld a,d ;od Y-ové souřadnice středu odečti
sub c ;relativní X-ovou souřadnici a výsledek
ex af,af' ;zatím ulož do záložního A
ld a,e ;k X-ové souřadnici středu přičti
add a,b ;relativní Y-ovou souřadnici a máš
ld c,a ;skutečnou X-ovou souřadnici
ex af,af' ;nyní obnov původní obsah registru A
ld b,a ;a v B je nyní skutečná souřadnice
call PLOT3 ;vykreslí bod
pop bc ;obnov relativní souřadnice

push bc ;ulož relativní souřadnice
ld a,d ;od Y-ové souřadnice středu odečti
sub c ;relativní X-ovou souřadnici a výsledek
ex af,af' ;zatím ulož do záložního A
ld a,e ;od X-ové souřadnice středu odečti
sub b ;relativní Y-ovou souřadnici a máš
ld c,a ;skutečnou X-ovou souřadnici
ex af,af' ;nyní obnov původní obsah registru A
ld b,a ;a v B je nyní skutečná souřadnice
call PLOT3 ;vykreslí bod
pop bc ;obnov relativní souřadnice

CIRCLE3 pop af ;obnov relativní X-ovou souřadnici
sub l ;a odečti do ní jedničku
jp nc,CIRCLE2 ;pokud jsi nepodlezl nulu, kresli dál
ret ;vrať se

```

Nezapomeňte na úpravu podprogramu **PLOT3** - musí zachovávat obsahy registrů HL a DE, je to velice důležité.

Uvedený program pro kreslení kružnice je založen na analytickém vyjádření rovnice kružnice, které vypadá takto  $x^2 + y^2 = r^2$ . Program počítá souřadnice pro polovinu jednoho kvadrantu a kreslí podle těchto souřadnic celkem osm bodů. Na začátku programu se občas (záleží to na poloměru kružnice) vykreslí čtyři body, které by se použitým algoritmem nevykreslily. Uprostřed kreslení je test, zda není X-ová souřadnice nulová a odskok pokud ano. Tímto způsobem se zajišťuje, aby se body kružnice na osách procházejících jejím středem nekreslily dvakrát - to je důležité proto, aby se kružnice mohla kreslit způsobem „OVER 1“ a tedy mohla druhým nakreslením případně smazat.

Ještě si napíšeme dva podprogramy, které jsou pro kreslení kružnice nezbytné, jsou to **SQUAREA** a **SQROOT**, které počítají druhou mocninu a druhou odmocninu.

```

SQROOT  xor  a                ;vynuluj CARRY příznak
        push de              ;ulož hodnoty
        push hl              ;registrů HL a DE
SQR2    ld   de,64            ;naplň fiktivní DE0 číslem 16384 (1282)
        ld   a,l              ;nyní vytvoříme jakýsi fiktivní
        ld   l,h              ;registr HLA, který bude na začátku
        ld   h,d              ;obsahovat číslo, které odmocňujeme
        ld   b,8              ;celý proces se bude opakovat 8 krát
SQR4    sbc  hl,de            ;nyní se testuje, jestli je daný řád
        jr   nc,SQR3          ;v odmocnině zastoupen, zkusmo se odečte
        add  hl,de            ;když ne, tak se zase přičte
SQR3    ccf                  ;ještě se převrátí příznak CARRY
        rl   d                ;a zleva se rotuje do registru D
        add  a,a              ;nyní se náš fiktivní registr HLA
        adc  hl,hl            ;vynásobí dvěma a posléze
        add  a,a              ;ještě jednou, takže je vlastně
        adc  hl,hl            ;celkově vynásoben čtyřmi
        djnz SQR4            ;a pokud to není poosmé, jdeme znovu
        xor  a                ;v tomto okamžiku je v registru D druhá
        sub  h                ;odmocnina (celé číslo z ní), naše
        ld   a,0              ;varianta ji však zaokrouhlí nahoru
        adc  a,d              ;vždy, když zbyl nějaký zbytek
        ld   c,d              ;celou část odmocniny pak uloží do C,
        pop  hl                ;mohla by se ještě hodit
        pop  de                ;obnov registry HL a DE
        ret                   ;a vrať se

SQUAREA ld   b,8              ;výpočet druhé mocniny
        push de              ;je vlastně jen
        ld   hl,0             ;modifikovaný program
        ld   d,l              ;pro násobení
        ld   e,a              ;dvou dvoubytových čísel
SQ2     add  hl,hl            ;nebudu ho tedy vysvětlovat,
        rla                  ;můžete se podívat
        jr   nc,SQ3           ;na stranu 79
        add  hl,de            ;v prvním dílu
SQ3     djnz SQ2              ;této knihy, je
        pop  de                ;tam popsán
        ret                   ;podrobně

```

Kreslení kružnice by se dalo ještě zrychlit. Prohlédnete-li si pozorně výpis, zjistíte, že tam vlastně zbytečně ukládá relativní X-ová souřadnice na zásobník a zbytečně složité se testuje její rovnost nule. Druhou možností, jak celý program zrychlit, je upravit podprogram **SQUAREA** tak, aby druhou mocninu nepočítal ale vybíral z tabulky (dlouhé 512 bytů). Obdobnou úpravu pro **SQROOT** použít nelze (tolik místa ve SPECTRU není), tady můžete maximálně rozepsat hlavní cyklus osmkrát. Volání podprogramů **PLOT3**, **SQUAREA** a **SQROOT** také můžete nahradit přímo tímto podprogramem - budete muset vytvořit nová návěští místo těch, která se nacházejí uvnitř těchto podprogramů.

Ke kružnicím patří také kruhy, tedy vyplněné kružnice, ukážeme si program, na jejich kreslení - je to modifikace programu pro kreslení kružnic, je však mnohem kratší.

```

RING      push af          ;uschovej poloměr
          call SQUAREA     ;spočítej jeho druhou mocninu
          pop af          ;vrať poloměr do registru A

RING2     push hl          ;ulož druhou mocninu poloměru
          push af          ;ulož relativní X-ovou souřadnici
          push af          ;a ulož ještě jednou
          push hl          ;ulož druhou mocninu poloměru
          call SQUAREA     ;druhá mocnina X-ové souřadnice
          ld c,l          ;a výsledek dej do
          ld b,h          ;registru BC
          pop hl          ;obnov druhou mocninu poloměru
          or a            ;vynuluj příznak CARRY
          push hl          ;ulož opět druhou mocninu poloměru
          sbc hl,bc       ;a odečti od ní druhou mocninu X
          call SQROOT     ;výsledek odmocni a máš Y souřadnici
          ld c,a          ;tu dej do C a máme relativní souřadnice
          pop hl          ;obnov druhou mocninu poloměru
          pop af          ;a obnov relativní X-ovou souřadnici
          ld b,a          ;a přesuň ji do B

          push bc         ;ulož relativní souřadnice
          ld a,c          ;spočítej délku čáry, je to
          add a,a         ;dvojnásobek relativní X-ové souřadnice
          ld l,a          ;a ulož je do registru L
          ld a,d          ;od Y-ové souřadnice středu odečti
          sub b           ;relativní Y-ovou souřadnici a máš
          ld b,a          ;skutečnou Y-ovou souřadnici bodu
          ld a,e          ;od X-ové souřadnice středu odečti
          sub c           ;relativní X-ovou souřadnici a máš
          ld c,a          ;skutečnou X-ovou souřadnici bodu
          push de         ;ulož souřadnice středu kruhu
          ld a,l          ;dej do registru A délku čáry (v L)
          push hl         ;ulož druhou mocninu poloměru
          call DRAWRFST   ;vykresli co nejrychleji čáru
          pop hl          ;obnov druhou mocninu poloměru
          pop de          ;obnov souřadnice středu kruhu
          pop bc          ;obnov relativní souřadnice

          pop af          ;dej do A Y-ovou souřadnici
          push af         ;a opět ji ulož na zásobník

```

```

or a ;nyní testuj její nulovost
jr z,RING3 ;a pokud nastává, přeskoč následující

push bc ;ulož relativní souřadnice
ld a,d ;k Y-ové souřadnici středu přičti
add a,b ;relativní Y-ovou souřadnici a máš
ld b,a ;skutečnou Y-ovou souřadnici bodu
ld a,e ;od X-ové souřadnice středu odečti
sub c ;relativní X-ovou souřadnici a máš
ld c,a ;skutečnou X-ovou souřadnici bodu
push de ;ulož souřadnice středu kruhu
ld a,l ;dej do registru A délku čáry (v L)
call DRAWRFST ;vykresli co nejrychleji čáru
pop de ;obnov souřadnice středu kruhu
pop bc ;obnov relativní souřadnice

RING3 pop af ;obnov X-ovou souřadnici
pop hl ;obnov druhou mocninu poloměru
sub l ;odečti jedničku od poloměru
jp nc,RING2 ;a dokud nejsi pod nulou, opakuj
ret ;kruh je nakreslen, konec

```

V samotném závěru kapitoly o jemné grafice si uděláme jednoduchý prográmek, který nám ukáže většinu toho, co naše podprogramy dokáží:

```

START im l ;raději nastavíme první mód přerušení
call MAKETAB ;vytvoř tabulku obrazkových adres
ei ;povol přerušení

ld a,4 ;zelený
out (254),a ;border

ld b,10 ;další akci
LOOP0 push bc ;budeme dělat desetkrát

ld bc,0 ;souřadnice levého horního rohu
LOOP1 push bc ;ulož
ld a,0 ;čára dlouhá 256 bodů
call DRAWRFST ;nakresli čáru - plný mikrořádek
pop bc ;obnov
inc b ;zvyš Y-ovou souřadnici
ld a,b ;a testuj, zda
cp 192 ;se nedosáhlo spodního okraje obrazovky
jr c,LOOP1 ;pokud ne, opakuj

pop bc ;obnov původní obsah B
djnz LOOP0 ;a pokud není konec, opakuj

ld bc,0 ;levý horní roh
LOOP3 push bc ;ulož
ld a,0 ;čára o délce 256 bodů
call DRAWRGHT ;kreslí čáru doprava
pop bc ;obnov
inc b ;zvyš Y-ovou souřadnici

```

```

ld a,b ;a testuj
cp 192 ;konec obrazovky
jr c,LOOP3 ;cykli

LOOP4 ld bc,0 ;levý horní roh
push bc ;ulož
ld a,192 ;čára dlouhá 192 bodů
call DRAWDOWN ;kreslí dolů
pop bc ;obnov
inc c ;posuň X-ovou souřadnici doprava
jr nz,LOOP4 ;opakuj 256 krát

STEP equ 5 ;krok bude 5 bodů

LOOP ld a,0 ;začneme poloměrem nula
push af ;ulož poloměr
ld de,96*256+128 ;střed na 128,96
call CIRCLE ;vykresli kružnici
pop af ;obnov poloměr
add a,STEP ;přičti krok
cp 96 ;a porovnej s maximálním poloměrem
jr c,LOOP ;a případně kresli dál

LOOP2 ld a,0 ;začneme poloměrem nula
push af ;ulož poloměr
ld de,96*256+128 ;střed na 128,96
call CIRCLE ;vykresli kružnici
pop af ;obnov poloměr
add a,STEP ;přičti krok
cp 96 ;a porovnej s maximálním poloměrem
jr c,LOOP2 ;a případně kresli dál

call RANDRAW ;kresli „náhodné“ čáry

LOOP5 ld a,1 ;začínáme s poloměrem 1
ld de,96*256+50 ;a středem na 50,96
push af ;ulož poloměr
push de ;ulož střed
call RING ;vykresli kruh
pop de ;obnov střed
inc e ;posuň střed o bod doprava
pop af ;obnov poloměr
add a,1 ;přičti k poloměru jedničku
cp 96 ;porovnej s maximálním poloměrem
jr c,LOOP5 ;a případně opakuj

LOOP8 dec e ;posuň střed o bod doleva (kompenzace)
ld a,95 ;poloměr na 95
push af ;ulož poloměr
push de ;ulož střed
call RING ;vykresli kruh
pop de ;obnov střed
dec e ;posuň střed o bod doleva
pop af ;obnov poloměr
sub 1 ;a odečti od něj jedničku
jr nc,LOOP8 ;opakuj dokud nejsi na -1

RANDRAW ld bc,0 ;nejprve uděláme bod na souřadnicích 0,0

```

```

ld (LASTXY+1),bc ;ulož jako „poslední“ souřadnice
call PLOT3 ;a bod vykresli

ld bc,191*256+255 ;nyní udělej čáru
ld de,257 ;255,191, tedy
call DRAW ;do pravého dolního rohu

ld b,20 ;budeme kreslit dalších 20 čar
ld hl,0 ;zařídíme, aby náhodná čísla byla při
ld (RANDOM+1),hl ;každém volání stejná
LOOP6 push bc ;ulož počítadlo čar
LOOP7 call RANDOM ;spočítej náhodné souřadnice
ld a,h ;a pokud by Y-ová vyšla větší
cp 192 ;než 192, spočítej
jr nc,LOOP7 ;souřadnice nové
ld b,h ;přesuň souřadnice
ld c,l ;do registrů B a C
call DRAWTO ;nakresli čáru do bodu (C,B)
pop bc ;obnov počítadlo čar
djnz LOOP6 ;opakuji

ret ;návrat

DRAWTO ld hl,(LASTXY+1) ;vzvedni souřadnice posledního bodu
ld de,257 ;nastav oba směry jako kladné
ld a,b ;nyní od současné souřadnice Y
sub h ;odečti minulou souřadnici Y
jr nc,DRAWTO2 ;a pokud to vyjde kladné, odskoč dále
cpl ;jinak přehod u čísla znaménko
inc a ;což udělají právě tyto dvě instrukce
ld d,-1 ;vertikální směr je záporný
DRAWTO2 ld b,a ;a získané číslo zapiš zpět do B
ld a,c ;od současné souřadnice X
sub l ;odečti minulou souřadnici X
jr nc,DRAWTO3 ;a pokud to vyjde kladné, odskoč dále
cpl ;jinak přehod znaménko u čísla
inc a ;(invertování a přičtení jedničky)
ld e,-1 ;horizontální směr je záporný
DRAWTO3 ld c,a ;a výsledné číslo dej zpět do C
;pokračuj dále do rutiny DRAW

DRAW ld a,c ;tohle už byste měli mít z dřívějšíka

```

# Proporční tisk

Proporční nebo také proporcionální tisk je takový tisk, při kterém má každé písmeno svou vlastní šířku. Písmena v jednotlivých řádcích tedy nejsou nad sebou jako u všech

předchozích typů tisku. Proporční tisk je obvykle hezčí a také úspěšnější (na řádek se vejde více textu) než tisk obyčejný. Je však také náročnější na programátora (ale my už toho umíme dost) a na čas. U proporčního tisku lze také umístit text na zcela libovolnou pozici na obrazovce, jak v ose X, tak v ose Y. Náš příklad navíc umí základy tzv. "**word processingu**" neboli zpracování textu, rutina, kterou si zachvíli ukážeme, totiž umí rozšiřováním mezer zarovnávat slova k pravému okraji.

Příklad je vytvořen tak, aby jej bylo možno používat z BASICu pomocí tisku přes kanál číslo 7. Jedná se kompletní tiskový podprogram, který je obdobou známého **rst 16**. Rutina dokáže tisknout standardní ASCII znaky (kódy 32 až 127), malá a velká česká písmena (kódy 128 až 157), vypisovat klíčová slova BASICu (kódy 165 až 255) a zpracovávat řídicí kódy AT, TAB a OVER, které však mají poněkud odlišný význam:

**AT číslo řádku, číslo sloupce** - pozice je v bodech, levý horní roh obrazovky má souřadnici 0,0. Číslo sloupce je současně také levým okrajem textu při zarovnávání.

**TAB číslo sloupce** - pozice pravého okraje v bodech. Pokud budete chtít, aby se například listing BASICu tiskl mezi 30 a 220 pixelovým sloupcem obrazovky, dejte tuto sekvenci příkazů **PRINT #7;AT 0,30;TAB 220;; LIST #7**.

**OVER výška mezery mezi řádky** - opět v bodech, můžete si nastavit své vlastní řádkování, po přeložení je zde hodnota 12 mikrořádků.

Tisk je prováděn pomocí instrukcí **OR** - přidává se tedy k podkladu a nemaže jej, před tiskem si tedy případně musíte vyčistit tu část obrazovky, kam se bude tisknout. Nyní vlastní výpis - znakový soubor na konci má naprosto stejný formát jaký používá DESKTOP, jste-li tedy uživateli (legálními a řádně registrovanými!) DESKTOPu, nemusíte znakový soubor opisovat, stačí, když návštěvím WIDTHS ukážete na jeho začátek a návštěvím FONT bude mít hodnotu o 126 vyšší než návštěvím WIDTHS - v příkladu je použit font COBRA (druhý standardní font DESKTOPu). Tisková rutina však bude pracovat s libovolným z mnoha DESKTOPových fontů.

```

org 50000 ;program uložíme od adresy 50000
ent $ ;zde bude začínat inicializace

START
INICIALE ld hl,CHADR-23733 ;takto je potřeba upravit adresu, na
ld (23588),hl ;které je adresa výkonné rutiny pro
ret ;tisk přes kanál číslo 7

CHADR defw CHAROUT ;adresa vlastní výkonné rutiny

CHAROUT ;sem bude skákat podprogram z adresy 16
STATUS ld c,0 ;zde jsou informace o tom, jestli
inc c ;další kód nemají být data
dec c ;testuj, zda zde není nula
jr z,CODES ;pokud ano, nejsou to data a odskoč
bit 0,c ;testuj, zda nejde o první souřadnici AT
jr nz,AT1 ;odskoč na její zpracování
bit 1,c ;testuj, zda nejde o druhou souřadnici
jr nz,AT2 ;AT a odskoč na její zpracování
bit 2,c ;testuj první část parametru TAB

```

```

      jr  nz,TAB1          ;a odskoč na její zpracování
      bit 3,c              ;testuj druhou část parametru TAB
      jr  nz,TAB2          ;a odskoč na její zpracování
;cokoliv jiného znamená OVER
OVER  ld  (OVERL+1),a      ;zapiš novou hodnotu meziřádkové mezery
END2  xor  a                ;a vynuluj STATUS, další kód již bude
      ld  (STATUS+1),a    ;opět normální písmeno
      ret                  ;návrat ze zpracování znaku

TAB1  ld  c,8              ;nastav druhou část TAB
      ld  (TAB2+1),a      ;zapiš první část parametru pro TAB
      jr  SETING          ;skoč na společný konec

TAB2  ld  a,0              ;první část parametru (nižší byte) zapiš
      ld  (REDGE+1),a     ;jako nový pravý okraj textu,
      call INITIAL        ;zavolej inicializaci
      jr  END2            ;a skoč na vynulování STATUSu a návrat

AT1   ld  c,2              ;nastav druhou část AT
      ld  (AT2+1),a       ;zapiš první parametr pro AT
      jr  SETING          ;a skoč pro nastavení STATUSu a konec

AT2   ld  h,0              ;zapiš do H první parametr AT (řádek)
      ld  l,a              ;zapiš do L druhý parametr AT (sloupec)
      ld  (POSITION+1),hl ;zapiš nové souřadnice
      ld  (LEDGE+1),a     ;sloupec je současně také levým okrajem
      call INITIAL        ;zavolej inicializaci
      jr  END2            ;a skoč na vynulování STATUSu a návrat

CODES cp  " "              ;zjistí, jestli se jedná o tisknutelné
      jr  nc,CHARS        ;znaky nebo řídicí kódy, odskoč se znaky
      cp  13              ;testuj znak ENTER
      jr  z,ENTEROUT      ;a pokud to je on, vytiskni obsah buferu
      sub 21              ;odečti kód OVER
      ld  c,16            ;nastav 4 bit - řídicí kód OVER
      jr  z,SETTING       ;a pokud to je on, odskoč
      dec  a               ;nyní test na AT
      ld  c,1              ;nastav 0 bit - řídicí kód AT
      jr  z,SETTING       ;odskoč pokud je to on
      dec  a               ;testuj kód TAB
      ld  c,4              ;nastav 2 bit - řídicí kód TAB
      jr  z,SETTING       ;pokud je to on, odskoč
      ld  c,0              ;jakýkoliv jiný kód ignoruj
SETTING ld a,c             ;zapiš nastavený kód
      ld  (STATUS+1),a    ;do proměnné STATUS
      ret                  ;návrat ze zpracování znaku

ENTEROUT ld a,0           ;proved test, jestli je něco
      dec  a               ;v textovém zásobníku
      jp  nc,NEXTLN       ;pokud ne, pouze odřádkuj
      ld  (ENTEROUT+1),a  ;nastav „prázdný“ textový zásobník
      jr  OVEROUT        ;skoč na vytištění textu ze zásobníku

CHARS  ld  hl,ENTEROUT+1  ;nastav signál,
      ld  (hl),1          ;že byl přijat znak
      sub 165             ;odečti kód prvního klíčového slova,
      jp  nc,#C10         ;a odskoč do ROM na tisk klíčového slova
      add  a,165          ;přičti zpátky odečtenou hodnotu

```



```

cp 158 ;testuj, zde není překročen povolený
jr c,CHRUT ;rozsah,
ld a,32 ;znaky od 158 do 164 nahrad' mezerou
CHRUT ld hl,BUFFER ;zapiš získaný znak
ld (hl),a ;do znakového zásobníku,
inc hl ;posuň se na další byte
ld (hl),255 ;a zapiš hned znak konce textu
ld (CHRUT+1),hl ;a novou adresu si zapamatuj pro příště
ld c,a ;nyní spočítej
ld b,0 ;adresu, kde je uložena
ld hl,WIDTHS-32 ;šířka právě
add hl,bc ;přijatého znaku
POS2 ld a,0 ;na tomto místě je zapsána šířka zatím
ld b,a ;přijatého textu, zapiš ji do B
REDGE ld e,254 ;nastavený pravý okraj tisku
inc e ;testuj jej
dec e ;na nulu
jr nz,CONT2 ;a v kladném případě odskoč
add a,(hl) ;k dosavadní šířce textu přičti novou
jr nc,CONT ;testuj pouze přetečení přes 256
jr OVERLOAD ;a případně skoč na vytištění zásobníku

CONT2 add a,(hl) ;přičti k dosavadní novou šířku
jr c,OVERLOAD ;a odskoč při překročení 256
cp e ;a navíc testuj také pravý okraj
jr nc,OVERLOAD ;a při překročení také tiskni
CONT ld (POS2+1),a ;zapiš nově získanou šířku
ld a,c ;nyní testuj, zda přijatý znak
cp " " ;není právě mezera
call z,STARTS2 ;a pokud ano, zapiš si jeho polohu
ret ;návrat ze zpracování znaku

STARTS2 ld hl,STARTS ;adresa tabulky poloh mezer
inc hl ;posuň se na další byte
ld (hl),b ;a zapiš tam současnou polohu
ld (STARTS2+1),hl ;zapamatuj si ukazatel do tabulky poloh
ld hl,SPACES+1 ;a zvětši o jedničku
inc (hl) ;počet mezer, které jsou v zásobníku ret
ret

OVERLOAD ld a,c ;testuj, zda znak, který překročil
cp " " ;pravou mez není mezera
call z,STARTS2 ;a případně si zapiš její polohu
SPACES ld a,0 ;zde se ukládá počet přijatých mezer
or a ;testuj, zda není nulový
jr z,OVEROUT ;a pokud ano, nezarovnávej text doprava
ld b,a ;zapiš počet mezer do B
ld hl,STARTS+1 ;nastav tabulku poloh mezer
dec b ;uber jednu mezeru
jr z,OVEROUT ;a pokud nezbyla žádná, nezarovnávej
OVER3 inc (hl) ;posuň polohu o jeden bod doprava
push bc ;ulož počet mezer
push hl ;ulož adresu zpracovávané polohy
OVER7 inc hl ;posuň se na další polohu a zvětši ji
inc (hl) ;o jedničku, tímto se vlastně všechna
djnz OVER7 ;další slova posunou doprava
ld hl,(STARTS2+1) ;do HL adresu konce řádku
ld a,(REDGE+1) ;a do A pravý okraj, nyní testuj,

```

```

cp      (hl)           ;zda již byl dosažen
pop     hl             ;obnov adresu zpracovávané polohy
pop     bc             ;obnov počet mezer
inc     hl             ;a posuň se na další polohu
jr      z,OVEROUT     ;odskoč při dosažení pravého okraje
djnz   OVER3          ;rozšiřuj každou mezeru
jr      SPACES        ;když rozšíříš všechny, dělej to znovu

OVEROUT ld  a,(REDGE+1) ;vyzvedni pozici pravého okraje
        ld  (REDGE2+1),a ;a zapiš ji do pracovního místa
        ld  hl,BUFFER    ;do HL adresu textového zásobníku
        ld  de,STARTS    ;do DE adresu poloh jednotlivých mezer
        ld  a,(hl)       ;vyzvedni první znak ze zásobníku
        cp  32           ;a pokud to není mezera,
        jr  nz,OVER4     ;tak odskoč, jinak
        inc de           ;přeskoč její polohu

OVER4   ld  a,(de)       ;vyzvedni polohu slova
REDGE2  cp  0           ;a porovnej ji s pravým okrajem
        jr  z,OVER6     ;pokud je stejná, tisk řádku skončil
        inc de           ;posuň se na další pozici
        ld  (POSITION+1),a ;zapiš pozici do X-ové souřadnice
        ld  a,(hl)       ;vyzvedni znak

OVER5   inc hl          ;a posuň ukazatel do zásobníku textu
        call PRINT      ;vytiskni znak na nastavenou pozici
        ld  a,(hl)       ;testuj konec textu
        cp  255          ;v zásobníku
        jr  z,OVER62    ;a při jeho dosažení odskoč
        cp  32           ;testuj mezeru
        jr  nz,OVER5    ;a pokud to není ona, tiskni dále
        jr  OVER4       ;jinak předtím ještě nastav pozici X

NEXTLN  ld  hl,POSITION+2 ;adresa Y-ové pozice do HL
        ld  a,(hl)       ;vezmi její hodnotu do A

OVERL   add a,12         ;a přičti šířku řádku
        sub 192         ;odečti výšku obrazovky
        ld  (hl),a      ;a zapiš hodnotu zpátky
        ret nc          ;vrať se pokud nedošlo k přetečení
        add a,192       ;přičti k souřadnici 192
        ld  (hl),a      ;a zapiš zpět do paměti
        ret             ;konec podprogramu

OVER6   inc hl          ;posuň se na další znak
OVER62  ex  de,hl       ;přesuň ukazatel do registru DE
        call NEXTLN    ;proved' posun Y-ové pozice
        ld  hl,BUFFER-1 ;do HL adresu textového zásobníku

OVER2   ld  a,(de)       ;a nyní přesuneme to, co zbylo po
        inc hl          ;tisku jednoho řádku na začátek
        ld  (hl),a      ;textového zásobníku,
        inc de          ;musíme si to totiž nechat
        inc a           ;pro další řádek
        jr  nz,OVER2    ;přenášej až do kódu 255 včetně
        ld  (CHRT+1),hl ;nastav ukazatel do textového zásobníku

        ld  hl,BUFFER    ;do HL adresu textového zásobníku
        ld  a,(STARTS)   ;do A dej pozici levého okraje
OVER1   ld  bc,WIDTHS-32 ;do BC adresu tabulky šířek znaků
        ld  e,(hl)       ;do E kód znaku
        inc e           ;test na koncovou zarážku (255)

```

```

jr      z,OVER0B      ;a pokud to je ona, odskoč
dec     e              ;vrať zpátky na původní hodnotu kódu
ld      d,0           ;nyní je kód v registru DE
ex      de,hl         ;přehod registry HL a DE
add     hl,bc         ;nyní přičti k HL (vlastně DE) obsah BC
add     a,(hl)        ;přičti k pozici šířky znaku
ex      de,hl         ;vrať zpátky HL a DE
inc     hl            ;posuň se na další znak
jr      OVER1         ;opakuji dokud nenalezneš zarážku

OVER0B  push af        ;ulož postupně registry
        push bc       ;protože budeš
        push de       ;skákat doprostřed podprogramu,
        push hl       ;který je na konci obnovuje
        jr      OVER0 ;a nyní tam skoč

INITIAL push af        ;ulož registry na zásobník
        push bc       ;je dobré, když si zvyknete
        push de       ;ukládat registry v nějakém pevném
        push hl       ;pořadí, snížíte tím riziko chyb
        ld      hl,BUFFER ;nastav ukazatel do textového
        ld      (CHRUT+1),hl ;zásobníku na jeho začátek
LEDGE   ld      a,10    ;nastavení levého okraje (počáteční)
        ld      (STARTS),a ;první znak v bufferu začíná úplně vlevo
OVER0   ld      (POS2+1),a ;zde se zapisuje, kam až sahá text
        ld      hl,STARTS ;tabulka začátků je v každém
        ld      (STARTS2+1),hl ;případě prázdná
        xor     a       ;vynuluj také počítadlo mezer, které
        ld      (SPACES+1),a ;jsou v textovém zásobníku
        pop     hl      ;obnov registry
        pop     de
        pop     bc
        pop     af
        ret            ;a vrať se

```

Tento podprogram slouží současně také pro ukončení zpracování znaku v případě, že tento znak byl buď ENTER nebo se jeho přijetím způsobilo překročení pravého okraje, což znamená, že se zarovná k pravému okraji a vytiskne to, co se na řádek vejde a zbytek se ponechá v bufferu.

```

PRINT   push de        ;ulož na zásobník registry DE a HL
        push hl       ;jsou v nich totiž důležité ukazatele

        ld      hl,WIDTHS-32 ;spočítej adresu, kde je uložena
        ld      d,0       ;šířka znaku, jehož kód
        ld      e,a       ;je nyní v registru A,
        add     hl,de     ;hodnotu z této adresy
        ld      c,(hl)   ;pak dej do registru C

        push bc         ;uschovej šířku znaku
POSITION ld bc,0        ;do BC dej X-ovou a Y-ovou souřadnici
        ld      a,b     ;Y-ovou souřadnici přesuň do A
        call   #22B0    ;a zavolej už důvěrně známý výpočet

        ex      de,hl   ;adresu v obrazovce dej do DE

```

```

add hl,hl ;v HL je nyní kód znaku
add hl,hl ;tento kód budeme
ld b,h ;celkově násobit
ld c,l ;dvanácti, protože
add hl,hl ;právě taková, je
add hl,bc ;je výška znakové předlohy
ld bc,-32*12+FONT ;do BC dej adresu znakových předloh
add hl,bc ;a nyní máme v HL adresu předlohy
pop bc ;obnov šířku znaku v C
ex de,hl ;předlohu do DE, adresu pozice do HL
push bc ;šířku znaku si opět uschovej
ld b,12 ;budeme tisknout celkem dvanáct bytů

LOOP ld a,(de) ;vyzvedni byte předlohy
inc de ;a posuň ukazatel na další
push bc ;ulož počítadlo bytů předlohy
push de ;ulož ukazatel do předlohy
or a ;testuj, zda byte předlohy není nula
jr z,LOOP2 ;a pokud ano, přeskoč vykreslení
ld e,a ;zapiš byte předlohy do E a do D dej
ld d,0 ;nulu, registrem DE budeme posunovat
ld a,(POSITION+1) ;vyzvedni X-ovou souřadnici do A
and 7 ;a ponech z ní jen bitový posun,
jr z,NOROTAT ;pokud je posun nulový, přeskoč jej

ROTAT srl e ;posuň doprava obsah registru E
rr d ;a také registru D
dec a ;zmenš početadlo posuvů
jr nz,ROTAT ;a pokud není nulové, posouvej dál

NOROTAT ld a,(hl) ;vyzvedni byte z obrazové paměti
or e ;naORuj část posunuté předlohy
ld (hl),a ;a zapiš výsledek zpátky
inc hl ;posuň se na další byte
ld a,(hl) ;a proved s ním
or d ;totéž, co s tím
ld (hl),a ;předchozím
dec hl ;vrať se zpátky

LOOP2 call DOWNHL ;notoricky známý podprogram
pop de ;obnov ukazatel do předlohy
pop bc ;obnov počítadlo bytů předlohy
djnz LOOP ;opakuj pro každý byte předlohy

pop bc ;obnov v C šířku tištěného písmene
ld hl,POSITION+1 ;adresa X-ové souřadnice
ld a,(hl) ;vyzvedni X-ovou souřadnici do A
add a,c ;přičti k ní šířku písmene
ld (hl),a ;a zapiš ji zpátky

pop hl ;obnov ukazatele
pop de
ret ;návrat z tisku jednoho znaku

BUFFER defs 100 ;sem se ukládá řádek před vytištěním

STARTS defs 40 ;zde jsou pozice jednotlivých slov

WIDTHS defb 3,2,6,6,6,6,7,3,4 ;šířky jednotlivých znaků v bodech
defb 4,6,6,3,5,3,6,7,5
defb 7,6,7,7,7,6,7,7,3

```

```

defb 3,4,5,4,6,7,8,7,7
defb 7,6,6,8,8,4,7,8,6
defb 9,8,7,7,7,7,6,6,8
defb 8,9,8,8,7,4,6,4,6
defb 8,6,6,6,5,6,5,5,6
defb 7,4,3,7,4,9,7,5,6
defb 6,6,5,5,7,8,9,6,8
defb 5,5,2,5,5,8,5,6,5
defb 6,5,6,5,7,6,7,5,7
defb 8,8,6,8,4,5,6,7,7
defb 8,5,7,6,6,7,8,7

FONT defw 0,0,0,0,0,0,32768 ;grafické předlohy - nic horšího
      defw 32896,32896,128 ;vás nemohlo potkat
      defw 128,0,18432,37008
      defw 216,0,0,0,20560
      defw 20728,20728,80,0

      defw 8192,43120,28832 ;5% - no comment
      defw 43048,8304,0
      defw 51200,53448,8224
      defw 39000,152,0,8192
      defw 20560,21536,34964

      defw 116,0,0,16384,128 ;10% - už máte prvních 12 znaků
      defw 0,0,0,8192,32832
      defw 32896,32896,8256
      defw 0,32768,8256
      defw 8224,8224,32832

      defw 0,0,0,8272,8440 ;80% - to byla legrace, 15%
      defw 80,0,0,8192,63520
      defw 8224,0,0,0,0,0
      defw 49152,16576,128,0
      defw 0,61440,0,0,0,0,0

      defw 0,49152,192,0 ;20% - no comment
      defw 2056,4112,8224
      defw 16448,32896,0
      defw 12288,33864,33924
      defw 18564,48,0,8192

      defw 41056,8224,8224 ;25% - máte už čtvrtinu
      defw 240,0,14336,17476
      defw 4104,29728,140,0
      defw 63488,8336,2160
      defw 36872,96,0,2048

      defw 6152,18472,2300 ;30% - skoro třetina
      defw 28,0,15360,28708
      defw 1096,18564,48,0
      defw 14336,45124,33992
      defw 18564,48,0,51200

      defw 4280,24600,16416 ;35% - více než třetina
      defw 64,0,12288,18504
      defw 18480,18564,48,0
      defw 12288,33864,19588

```

```
defw 34868,112,0,0,0

defw 49344,49152,192,0 ;40% - no comment
defw 0,0,49344,49152
defw 16576,128,0,8192
defw 32832,8256,0,0,0
defw 0,240,240,0,0,0

defw 32768,8256,32832 ;45% - odporně malé číslo
defw 0,0,24576,34960
defw 4104,32,32,0,0
defw 30720,44180,32956
defw 120,0,4096,10256

defw 14376,17476,238,0 ;50% - polovina, teď už to půjde dolu
defw 61440,18504,18544
defw 18500,240,0,13312
defw 33868,32896,18564
defw 48,0,61440,17480

defw 17476,18500,240,0 ;55% - moc rychle dolů to nejde
defw 63488,16456,16496
defw 18496,248,0,63488
defw 16456,16496,16448
defw 224,0,13312,33868

defw 36480,19588,52,0 ;60% - no comment
defw 60928,17476,17532
defw 17476,238,0,57344
defw 16448,16448,16448
defw 224,0,7168,2056

defw 2056,37000,96,0 ;65% - skoro dvě třetiny
defw 60416,20552,20576
defw 17480,238,0,57344
defw 16448,16448,18496
defw 248,0,16640,25409

defw 21859,18773,235,0 ;70% - no comment
defw 52736,25700,21588
defw 19532,228,0,12288
defw 33864,33924,18564
defw 48,0,61440,17480

defw 28744,16448,224,0 ;75% - tři čtvrtiny
defw 12288,33864,33924
defw 18580,52,0,61440
defw 17480,28744,18504
defw 236,0,26624,34968

defw 6240,51336,176,0 ;80% - čtyři pětiny
defw 63488,8360,8224
defw 8224,112,0,60928
defw 17476,17476,17476
defw 56,0,60928,17476

defw 10280,4136,16,0 ;85% - no comment
defw 60160,18761,21833
```

```
defw 8758,34,0,60928
defw 10308,4112,17448
defw 238,0,60928,10308

defw 4112,4112,56,0 ;90% - chybí už jen desetina
defw 64512,2180,8208
defw 33856,252,0,57344
defw 32896,32896,32896
defw 224,0,32896,16448

defw 8224,4112,2056,0 ;%95 - co k tomu dodat
defw 57344,8224,8224
defw 8224,224,0,0
defw 28704,8360,8224
defw 32,0,0,0,0,0

defw 65280,0,0,18480 ;100% - musíte však dnešní
defw 57408,16448,248,0 ;plán splnit na 200%,
defw 0,24576,28688
defw 37008,104,0,49152
defw 28736,18504,18504

defw 176,0,0,24576 ;105% - takhle plnili i černí baroni
defw 32912,36992,96,0
defw 6144,28688,37008
defw 37008,104,0,0
defw 24576,61584,36992

defw 96,0,8192,16464 ;110% - takhle také plnili
defw 16608,16448,224,0 ;i černí baroni
defw 0,45056,18504
defw 16432,34928,112
defw 49152,28736,18504

defw 18504,236,0,16384 ;115% - dokonce i takhle plnili
defw 49152,16448,16448 ;černí baroni
defw 224,0,16384,49152
defw 16448,16448,32832
defw 0,49152,22592

defw 24656,18512,204,0 ;120% - no comment
defw 49152,16448,16448
defw 16448,224,0,0
defw 46592,18761,18761
defw 219,0,0,45056

defw 18504,18504,236,0 ;125% - no comment
defw 0,24576,37008
defw 37008,96,0,0
defw 45056,18504,18504
defw 16496,224,0,26624

defw 37008,37008,4208 ;130% - to je na BSP
defw 56,0,45056,16456
defw 16448,224,0,0
defw 24576,24704,36880
defw 96,0,16384,57408
```

```
defw 16448,20560,32,0 ;135% - no comment
defw 0,55296,18504
defw 18504,52,0,0
defw 60928,10308,4136
defw 16,0,0,60160

defw 18761,8789,34,0,0 ;140% - černí baroni ....
defw 55296,8272,20512
defw 216,0,0,60928
defw 9284,6184,20496
defw 32,0,61440,8336

defw 36928,240,0,12288 ;145% - Stachanov hadr
defw 16448,32832,16448
defw 48,0,32896,32896
defw 32896,32896,32896
defw 32896,49152,8224

defw 4128,8224,192,0,0 ;150% - Jak se kalila ocel?
defw 20480,160,0,0,0
defw 14336,37444,41642
defw 37546,14404,0
defw 8216,24576,61584

defw 36992,96,0,8216 ;155% - GEROJ !!!
defw 18680,28736,18496
defw 248,0,8272,24576
defw 61584,36992,96,0
defw 8272,18680,28736

defw 18496,248,0,8272 ;160% - no comment
defw 24576,24704,36880
defw 96,0,8272,34928
defw 6240,51336,176,0
defw 8272,24576,32912

defw 36992,96,0,4136 ;165% - doufám, že provádíte etickou
defw 18480,32900,18564 ;samoregulaci výroků, které při
defw 48,0,8272,45056 ;opisování pronášíte na mou adresu!
defw 16456,16448,224,0
defw 8272,18672,28744

defw 18504,236,0,8272 ;170% - no comment
defw 61440,8336,36928
defw 240,0,4136,34044
defw 4104,17440,252,0
defw 4108,60928,9284

defw 6184,20496,32 ;175% - vydrž, už brzy bude konec
defw 4108,17646,4136
defw 4112,56,0,8216
defw 24576,28688,37008
defw 104,0,12,4112

defw 14376,17476,238,0 ;180% - zas tak brzy ten konec nebude
defw 16432,49152,16448
defw 16448,224,0,16432
defw 16608,16448,16448
```



```

defw 224,0,22688,4112

defw 36976,37008,104,0 ;185% - že se Vám to chce psát?!
defw 8272,18672,17476
defw 18500,240,0,8272
defw 45056,18504,18504
defw 220,0,4136,25806

defw 21604,19532,228,0 ;190% - už je to tady!
defw 16432,24576,37008
defw 37008,96,0,4108
defw 18480,33924,18564
defw 48,0,20520,57408

defw 16448,20560,32,0 ;195% - končíme pánové!
defw 8272,43256,8224
defw 8224,112,0,12336
defw 55296,18504,18504
defw 52,0,14392,17646

defw 17476,17476,56,0 ;200% - jste opravdový hrdina
defw 8216,55296,18504
defw 18504,52,0,4108
defw 17646,17476,17476
defw 56,0

```

Pro vyzkoušení tiskového podprogramu proveďte spuštění - tím se připojí tisk na kanál číslo 7. Potom se vraťte z assembleru do BASICu a napište tento krátký program:

```

10 FOR i=32 TO 255
20 LET a$=CHR$ i
30 PRINT #7;a$;" ";
40 NEXT i
50 PRINT #7 ;toto je nutné pro vytištění zbytku
;textu z textového zásobníku

```

Příklad spusťte příkazem RUN, můžete také zkusit LIST #7, bohužel CAT #7 se provést nedá (alespoň s disketovými jednotkami Didaktik 40 a Didaktik 80 ne).

# *Plníme obrazovku*

Pod tímto názvem se skrývá to, co se v angličtině nazývá FILL a používá se v grafických programech (ART STUDIO, ARTIST) nebo v některých textových hrách při vykreslování obrázků. Je to tedy program, který dostane jako parametr souřadnice nějakého bodu a vyplní všechny prázdné body, ke kterým existuje nějaká cesta z vybraného bodu,

kteřá se skládá pouze z vodorovných a svislých úseček a vede pouze přes prázdné body. Pokud se vám tato definice zdá poněkud těžkopádná, máte pravdu, zkuste si však vymyslet lepší definici. Tímto způsobem však FILL provádět nebudeme, napíšeme si ještě jednu definici, podle které budeme program realizovat:

Nadefinujeme si množinu všech bodů, které vyplníme - množina FILL, do této množiny patří všechny tyto body:

- samotný vybraný bod pokud je prázdný.
- pokud jsi prázdný a tvým sousedem (vlevo, vpravo, dole, nahoře) je nějaký bod z množiny FILL, pak také patříš do množiny FILL.

Kdyby to snad ještě někomu nebylo jasné, uvedu zde raději příslušný algoritmus slovně a pak jej napíšu jako program. Předchází text je ukázka toho, co vás čeká, když se vydáte na Matematicko-fyzikální fakultu University Karlovy (případně jinou VŠ podobného zaměření). Dovolím si malou definici: *Matematik je člověk, který 90% času věnuje tomu, že se snaží pochopit definice jiných matematiků a zbytek tomu, že sám vymýšlí definice, jejichž pochopení zase jiní matematikové věnují 90% svého času...* ono to ale bohužel jinak nejde!

Nyní už zmíněný algoritmus:

- 1) Ulož na zásobník souřadnice vybraného bodu
- 2) Testuj, zda je zásobník prázdný, pokud ano, skonči.

3) Odeber ze zásobníku souřadnice bodu a testuj, zda je bod prázdný, pokud ano, vyplň ho a ulož na zásobník souřadnice těch, z jeho čtyř sousedů, kteří jsou také nevybarvení. Jdi na bod 2.

Teď si tento FILL naprogramujeme (všimněte si, že tu je nejrychlejší PLOT):

```

ent $ ;tady se to spouští

START ld (SPSTOR+1),sp ;ulož současnou hodnotu SP registru
di ;zákaz přerušení
call PREPIS ;popiš první třetinu obrazovky textem
ld sp,0 ;nastav SP registr na konec paměti
call MAKETAB ;vytvoř tabulky pro kreslení bodu

ld bc,30*256+128 ;plnit budeme od bodu na (128,30)

FILL ld hl,-1 ;souřadnice (255,255) pro označení
push hl ;konce ulož na zásobník
push bc ;a ulož také souřadnice výchozího bodu

FILL2 pop bc ;odeber souřadnice bodu ze zásobníku
ld a,b ;testuj, zda se nejedná

```

```

and c ;o bod se souřadnicemi (255,255),
inc a ;v takovém případě
jr z,FILLEND ;odskoč - není co vyplňovat
call POINT ;zavolej test bodu o souřadnicích (C,B)
jr nz,FILL2 ;pokud tam již bod je, zpracuj další
push bc ;ulož si souřadnice bodu
call PLOT3 ;nakresli bod na uvedené souřadnice
pop bc ;obnov souřadnice bodu
ld a,b ;testuj Y-ovou souřadnicí na nulu,
or a ;v takovém případě by šlo o nejvyšší
jr z,NOUP ;řádek a nad ním už žádný není
dec b ;posuň se o řádek nahoru a testuj
call POINT ;tento bod na vyplnění,
jr nz,NOUP2 ;když je již vyplněn, odskoč
push bc ;bod není vyplněn, ulož jeho souřadnice.
NOUP2 inc b ;vrať se zpátky na původní řádek

NOUP ld a,b ;nyní budeme testovat bod dole, nejprve
cp 64 ;zjistíme, jestli už nejsme na spodním
jr z,NODOWN ;řádku, když ano, tak odskočíme
inc b ;když ne, posuneme se na dolní bod
call POINT ;a otestujeme, jestli už je nakreslen,
jr nz,NODOWN2 ;když ano, tak odskočíme, když ne
push bc ;tak jeho souřadnice uložíme na zásobník
NODOWN2 dec b ;vraťme se na původní souřadnice

NODOWN ld a,c ;dále zpracujeme boční sousedy, test na
or a ;rovnost nule u X-ové souřadnice a když
jr z,NOLEFT ;ano, tak odskoč (levý okraj obrazovky)
dec c ;zmenši X-ovou souřadnicí o jedničku
call POINT ;testuj bod na těchto souřadnicích
jr nz,NOLEFT2 ;a pokud tu již je, odskoč
push bc ;když ne, ulož jeho souřadnice
NOLEFT2 inc c ;vraťme se zpět

NOLEFT ld a,c ;poslední souseď, kterého jsme zatím
cp 255 ;neprohlédli, je vpravo, test, zda jsme
jr z,FILL2 ;na pravém okraji, když ano, odskoč
inc c ;posuň se doprava
call POINT ;a testuj, zda je bod již vyplněn,
jr nz,NORIGHT2 ;když ano, odskoč
push bc ;když ne, ulož souřadnice na zásobník
NORIGHT2 dec c ;vrať se zpátky
jr FILL2 ;skoč pro další bod

FILLEND ld a,4 ;konec vyplňování, signalizujeme
out (254),a ;to nastavením zeleného borderu
SPSTOR ld sp,0 ;obnov původní hodnotu zásobníku
ret ;a vrátí se

POINT push bc ;ulož souřadnice na zásobník
ld l,b ;najdi v tabulce adresu mikrořádku
ld h,SCRNADRS/256 ;(tabulka je jinak organizována!)
ld a,c ;nyní spočítej,
rrca ;který byte
rrca ;na mikrořádku
rrca ;je ten,
and 31 ;ve kterém

```

```

add a,(hl) ;je hledaný bod
inc h ;nyní posuň ukazatel do tabulky adres
ld h,(hl) ;na vyšší byte a dej ho do H a dej
ld l,a ;do L nižší byte - v HL je adresa bytu
ld b,TABLE/256 ;nyní dej do B horní byte tabulky masek
ld a,(bc) ;vzvedni masku do registru A
and (hl) ;ponech z obrazovky pouze testovaný
pop bc ;bod (bit), obnov souřadnice bodu
ret ;a vrať se

PLOT3 ld l,b ;najdi v tabulce adresu mikrořádku
ld h,SCRNADRS/256 ;(tabulka je jinak organizována!)
ld a,c ;nyní spočítej,
rrca ;který byte
rrca ;na mikrořádku
rrca ;je ten,
and 31 ;ve kterém
add a,(hl) ;je hledaný bod
inc h ;nyní posuň ukazatel do tabulky adres
ld h,(hl) ;na vyšší byte a dej ho do H a dej
ld l,a ;do L nižší byte - v HL je adresa bytu
ld b,TABLE/256 ;nyní dej do B horní byte tabulky masek
ld a,(bc) ;vzvedni masku do registru A,
xor (hl) ;připoj tento bit
ld (hl),a ;do vybraného bytu
ret ;a vrať se

MAKETAB ld b,192 ;obrazovka má 192 mikrořádků
ld de,16384 ;adresa prvního mikrořádku
ld hl,SCRNADRS ;adresa prvního bytu tabulky
MAKETAB2 ld (hl),e ;ulož
inc h ;adresu
ld (hl),d ;mikrořádku
dec h ;do tabulky
inc hl ;a spočítej
ex de,hl ;adresu
call DOWNHL ;následujícího mikrořádku
ex de,hl ;toto proved
djnz MAKETAB2 ;pro všechny mikrořádky
ld b,32 ;do B dej 32 (32*8=256)
ld hl,TABLE ;do HL adresu tabulky bitových masek
MAKETAB3 ld (hl),128 ;vyrob jednotlivé masky
inc l
ld (hl),64
inc l
ld (hl),32
inc l
ld (hl),16
inc l
ld (hl),8
inc l
ld (hl),4
inc l
ld (hl),2
inc l
ld (hl),1

```

```

inc 1
djnz MAKETAB3
ret

PREPIS  ld  a,2          ;otevři kanál
        call #1601     ;číslo 2
        ld  a,22       ;nastav
        ret 16         ;tiskovou
        xor a          ;pozici
        rst 16         ;na levý
        xor a          ;horní roh
        rst 16         ;obrazovky
        ld  b,0        ;budeme tisknout 256 znaků
PREPIS2 ld  a,r         ;vezmi hodnotu z R
        and 63         ;uprav na rozsah 0-63
        add a,32       ;posuň na rozsah 32-95
        rst 16         ;tiskni znak
        djnz PREPIS2   ;opakuji
        ret           ;návrat

LAST

        org  LAST/256+1*256 ;org na adresu se spodním bytem 0

SCRNADRS defs 512        ;tabulky začínají vždy na adrese #XX00
TABLE     defs 256       ;tabulky bitových masek (32x za sebou)

```

Předtím, než program spustíte, zkontrolujte, jestli je přeložen nejvýše na adresu 43000 a má za sebou volno až do konce paměti (pokud pracujete s PROMÉTHEEM a máte jej nainstalován na adrese 24000, je vše v pořádku). Tento program totiž potřebuje velké množství paměti k tomu, aby pracoval (čím větší je vyplňovaná plocha, tím hůř). Použitelnost tohoto programu je tedy omezena na případy, kdy chceme vyplňovat malé plochy a máme poměrně dosti volné paměti k dispozici. Pokud si budete chtít zjistit, kam se až zásobník dostal, vyčistěte si před spuštěním programu paměť nulami a po skončení se monitorem podívejte, co se všechno zaplnilo.

Další program, který si ukážeme se od předchozího bude lišit jen v jediném detailu, bude však podstatný: Místo abychom použili ukládání dat do zásobníku, použijeme ukládání dat do fronty - anglicky FIFO - to, co jsme uložili jako první také jako první vyjmeme - při našem vyplňování obrazovky budeme tedy nejprve zpracovávat body, které jsou nejbliž zvolenému bodu. Program nejjednodušěji napíšete tak, že upravíte předchozí program a připsáte některé podprogramy:

```

ent $

START  im  1          ;nastav mód přerušování 1 a zakaž je
        di           ;to je kvůli práci s D40 (pokud ji máte)
        call PREPIS   ;popiš horní třetinu obrazovky
        call MAKETAB  ;vytvoř tabulky

```

```

ld bc,32*256+128 ;souřadnice výchozího bodu

FILL ld hl,SPACE ;inicializuj ukazatele
ld (PUSHPTR+1),hl ;v podprogramech PUSHBC
ld (POPPTR+1),hl ;a POPBC
call PUSHBC ;ulož souřadnice výchozího bodu

FILL2 call POPBC ;odeber souřadnice bodu z fronty
ld a,b ;testuj, zda se nejedná
and c ;o bod se souřadnicemi (255,255),
inc a ;v takovém případě
jr z,FILLEND ;odskoč - není co vyplňovat
call POINT ;zavolej test bodu o souřadnicích (C,B)
jr nz,FILL2 ;pokud tam již bod je, zpracuj další
push bc ;ulož si souřadnice bodu
call PLOT3 ;nakresli bod na uvedené souřadnice
pop bc ;obnov souřadnice bodu
ld a,b ;testuj Y-ovou souřadnicí na nulu,
or a ;v takovém případě by šlo o nejvyšší
jr z,NOUP ;řádek a nad ním už žádný není
dec b ;posuň se o řádek nahoru a testuj
call POINT ;tento bod na vyplnění,
jr nz,NOUP2 ;když je již vyplněn, odskoč
call PUSHBC ;bod není vyplněn, ulož jeho souřadnice

NOUP2 inc b ;vrať se zpátky na původní řádek

NOUP ld a,b ;nyní budeme testovat bod dole, nejprve
cp 191 ;zjistíme, jestli už nejsme na spodním
jr z,NODOWN ;řádku, když ano, tak odskočíme
inc b ;když ne, posuneme se na dolní bod
call POINT ;a otestujeme, jestli už je nakreslen,
jr nz,NODOWN2 ;když ano, tak odskočíme, když ne
call PUSHBC ;tak jeho souřadnice uložíme na zásobník

NODOWN2 dec b ;vrátíme se na původní souřadnice

NODOWN ld a,c ;dále zpracujeme boční sousedy, test na
or a ;rovnost nule u X-ové souřadnice a když
jr z,NOLEFT ;ano, tak odskok (levý okraj obrazovky)
dec c ;zmenší X-ovou souřadnicí o jedničku
call POINT ;testuj bod na těchto souřadnicích
jr nz,NOLEFT2 ;a pokud tu již je, odskoč
call PUSHBC ;když ne, ulož jeho souřadnice

NOLEFT2 inc c ;vrátíme se zpět

NOLEFT ld a,c ;poslední souseď, kterého jsme zatím
cp 255 ;neprohlédli, je vpravo, test, zda jsme
jr z,FILL2 ;na pravém okraji, když ano, odskok
inc c ;posuň se doprava
call POINT ;a testuj, zda je bod již vyplněn,
jr nz,NORIGHT2 ;když ano, odskoč
call PUSHBC ;když ne, ulož souřadnice na zásobník

NORIGHT2 dec c ;vrať se zpátky
jr FILL2 ;skoč pro další bod

FILLEND ld a,4 ;konec vyplňování, signalizujeme
out (254),a ;to nastavením zeleného borderu
ret ;a vrať se

```

```

PUSHBC  push de          ;budeme pracovat s registry
        push hl         ;HL a DE a proto je ulož
PUSHPTR  ld hl,0         ;ukazatel ukládání,
        ld (hl),c       ;zapiš X-ovou souřadnici
        inc hl          ;posuň se,
        ld (hl),b       ;zapiš Y-ovou souřadnici
        inc hl          ;posuň se,
        ld de,SPACEEND ;nyní budeme testovat, zda
        or a            ;se ukazatel do fronty
        sbc hl,de       ;dostal na konec vyhrazené
        add hl,de       ;paměti, pokud
        jr nz,PUSHBC2   ;ne, odskoč,
PUSHBC2  ld hl,SPACE     ;pokud ne, nastav opět začátek
        ld (PUSHPTR+1),hl ;ulož novou hodnotu zpět
        pop hl          ;obnov obsahy registrů
        pop de         ;HL a DE
        ret            ;a vrať se

POPBC    push de         ;ulož registry
        push hl        ;DE a HL
POPPTR  ld hl,0         ;ukazatel odebírání
        ld de,(PUSHPTR+1) ;testuj, zda se již
        or a           ;nedostal na stejnou
        sbc hl,de      ;adresu jako ukazatel
        add hl,de      ;ukládání, pokud ano, není ve frontě
        ld bc,-1      ;nic a to signalizuj číslem 65535,
        jr z,POPBC3   ;a případně skoč na konec podprogramu
        ld c,(hl)     ;vyzvedni X-ovou souřadnici
        inc hl        ;a posuň se,
        ld b,(hl)     ;vyzvedni Y-ovou souřadnici
        inc hl        ;a posuň se,
        ld de,SPACEEND ;nyní testuj, zda
        or a          ;ses nedostal na
        sbc hl,de     ;konec vyhrazené
        add hl,de     ;datové oblasti
        jr nz,POPBC2  ;a pokud ne, odskoč,
        ld hl,SPACE   ;nastav ukazatel na začátek oblasti
POPBC2  ld (POPPTR+1),hl ;a ulož zpátky
POPBC3  pop hl        ;obnov registry
        pop de        ;HL a DE
        ret          ;vrať se

POINT   ...           ;tohle opište z předchozího programu

PLOT3   ...           ;opište z předchozího programu

MAKETAB ...          ;opište z předchozího programu

PREPIS  ...           ;opište z předchozího programu

LAST    ...           ;adresa prvního volného bytu
        org LAST/256+1*256 ;nastav ORG na adresu #XX00

SCRNADRS defs 512     ;tabulka adres jednotlivých mikrořádků
TABLE   defs 256     ;tabulka bitových masek

SPACE   defs 2048     ;prostor vyhrazený pro frontu
SPACEEND nop         ;a první volný byte za daty

```

Prohlédněte si pořádně způsob, jakým se s **frontou** pracuje, je to tradiční technika. Fronta je cirkulující - začátek i konec se neustále pohybují. Data se do fronty zapisují od začátku a v okamžiku, kdy se dojde na konec vyhrazené oblasti, začne se zapisovat znovu od začátku, současně (střídavě) se provádí odebírání dat také od začátku, a v případě, že se dojde na konec, začne se znovu od začátku. V době, kdy se nová data zapisují znovu na stejné místo, jsou už stará data přečtena a proto to nevádí. Pokud by došlo k více zápisům než čtením, může dojít k přetečení naší datové struktury a rozpozná se to při čtení jako případ prázdné **fronty**. Pokud dojde ke čtení z místa, kam se má právě zapisovat - fronta je prázdná - vrátí se číslo 65535. Naše implementace nedokáže rozpoznat, kdy došlo k podečením (ve frontě nejsou data) nebo přetečení (fronta je přeplněná). Jak to rozlišit, si ukážeme v dalším příkladu.

Při spuštění si můžete všimnout, že se oblast vyplňuje jako by čtvercem pootočeným o 45 stupňů (Kozákův magický obrazec). Výsledky tohoto podprogramu jsou mnohem lepší než u předchozího - se stejnou velikostí paměti vybarví celou obrazovku a hlavně je ošetřen proti možnosti sebezpřepsání zásobníkem (tu první verzi můžete také upravit, stačí když budete testovat, kam ukazuje registr SP a pokud by to bylo příliš nízké, FILL ukončit tak, že byste pouze odebírali body ze zásobníku dokud byste nenarazili na zarážku - číslo 65535). Také byste mohli projít daty na zásobníku a vyhazovat z něj ty body, které jsou již vybarveny (znamenalo by to ovšem ty, co vybarveny nejsou posunovat - pokud půjdete shora dolů, nebude to činit příliš velké potíže), po redukci zásobníku (pokud k ní dojde) můžete pokračovat dál. Stejně můžete naložit s frontou v případě přeplnění fronty (detekce viz další příklad), také ji lze „zredukovat“.

Poslední z pŮlících programů bude založen na této úvaze: *Když mám bod, mohu ho chápat také jako celou vodorovnou čáru, na které leží. Při hledání sousedních bodů si budu prohlížet všechny body nad čarou a pod čarou, budu si však z každé pamatovat jen jeden bod. Při vlastním plnění si vždy nejprve naleznu počátek čáry - buď plný bod nebo levý okraj, pak si zjistím délku čáry a vybarvím ji, dále projdu všechny body nad čarou a zapamatuji si ty, které v každém souvislém úseku prázdných bodů nejvíce vlevo (body nad a pod čarou procházím zleva doprava). Totéž udělám pro body pod čarou. Všechna data uložím do fronty, ze začátku fronty odeberu souřadnice bodu a jdu na začátek, pokud ve frontě nic není, končím. Místo do fronty můžete samozřejmě data ukládat na zásobník, bude to stejné. Při práci si nebudeme body pamatovat jako souřadnice ale jako adresu a bitovou masku - zrychlí se tím práce (nebudu muset pro každý PLOT a POINT počítat znovu adresu příslušného bytu a masku - vím, že se jedná o bod **nad** nebo **pod**, což je stejná maska a byte **nad** nebo **pod** bytem, případně o bod **vlevo** nebo **vpravo**, což je rotace masky na správnou stranu s případným posunem adresy o jedničku. Také si budeme všimnat bytů, které jsou celé prázdné (0) nebo celé plné (255), u těch zrychlíme kreslení čáry i prohlížení.*

```

ent      $
BUFSIZE  equ  1024
START    im    1          ;pokud používáte D40 a tlačítko SNAP
         di
         call PREPIS     ;popíšeme horní třetinu obrazovky

```



```

ld bc,120*256+128 ;do B a C souřadnice výchozího bodu

FILL ld hl,SPACE ;inicializuj podprogramy
ld (PUSHPTR+1),hl ;PUSH a POP, které
ld (POPPTR+1),hl ;zajišťují realizaci FRONTY
ld hl,0 ;vynuluj také počítadlo záznamů,
ld (PUSHCNT+1),hl ;které jsou uloženy ve FRONTĚ

ld a,b ;vlož Y-ovou souřadnici počátku do A
call #2280 ;a spočítej adresu bytu a polohu bitu
ld b,a ;v něm,
inc b ;nyní si
ld a,1 ;příprav
FILL9 rrca ;masku
djnz FILL9 ;s vybraným bitem
ld c,a ;masku ulož do C
call PUSH ;a masku spolu s adresou ulož do FRONTY

FILL0 call POP ;vyzvedni z FRONTY adresu a masku bodu
ld a,c ;testuj, zda maska není 255, to by
inc a ;znamenal prázdnu frontu a tedy
jp z,FILLEND ;také konec celého algoritmu
ld a,(hl) ;testuj, zda čára, která je tímto bodem
and c ;reprezentována, není již vybarvena,
jr nz,FILL0 ;pokud ano, jdi pro další bod

GOLEFT ld a,(hl) ;vyzvedni obsah bytu
or a ;a testuj, zda je to nula,
jr nz,GOLEFT2 ;pokud ne, odskoč a ponech původní masku
ld c,128 ;nastav masku s bitem úplně vlevo
GOLEFT2 ld b,c ;ulož současnou masku
ld e,1 ;a současný spodní byte adresy,
rlc c ;rotuj maskou doleva
jr nc,GOLEFT3 ;a pokud nedošlo k přetečení, odskoč
ld a,1 ;testuj, zda nejsi na
and 31 ;na začátku mikrořádku,
jr z,GOLEFT4 ;pokud ano, odskoč - jsi na začátku
dec l ;posuň adresu bytu doleva
GOLEFT3 ld a,(hl) ;vyzvedni obsah bytu
and c ;a ponech pouze vybraný bit,
jr z,GOLEFT ;pokud je to nula, jdi dále doleva
GOLEFT4 ld c,b ;vrať masku a spodní byte adresy
ld l,e ;předchozího bodu (poslední prázdny)

push hl ;ulož adresu bytu
push bc ;ulož bitovou masku
ld e,0 ;vynuluj počítadlo délky čáry
FILLRGHT ld a,(hl) ;vyzvedni byte pro testování
and c ;a ponech z něj vybraný bit, pokud
jr nz,FREND ;je nenulový, odskoč - čára je hotová
ld a,(hl) ;nakresli bod
or c ;na vybrané
ld (hl),a ;souřadnice
inc e ;zvyš počítadlo bodů o jedničku,
rrc c ;zarotuj maskou doprava, a pokud došlo
jr nc,FILLRGHT ;k přesunu bitu přes okraj, odskoč
FR2 inc l ;posun se na další byte
ld a,1 ;nyní otestuj, zda ses

```

```

and 31 ;posunem nedostal za konec řádku,
jr z,FREND ;pokud ano, odskoč - čára je hotova
ld a,(hl) ;vzvedni obsah získané adresy
or a ;a testuj ho na rovnost nule
jr nz,FILLRGHT ;pokud není nula, pokračuj bitově,
ld (hl),255 ;pokud je nula, vyplň tento byte
ld a,e ;a zvyš
add a,8 ;délku
ld e,a ;čáry o osm,
jr FR2 ;odskoč na další posun

FREND pop bc ;obnov bitovou masku
pop hl ;obnov adresu bytu

push hl ;ulož adresu bytu
ld b,e ;do registru B dej délku čáry
push bc ;uschovej bitovou masku a délku čáry

call UPHL ;spočítej adresu bytu nad tímto bytem
ld a,h ;a otestuj, zda ses
cp 64 ;nedostal mimo rozsah platných řádků,
jr c,DOWN ;pokud ano, přeskoč další testy

ld a,(hl) ;testuj, zda je zde plno nebo volno,
and c ;pokud je zde plno,
jr nz,TUR1B ;odskoč na hledání prázdného bodu
TUR7 call PUSH ;ulož souřadnice bodu na čáře
TUR2 ld a,(hl) ;testuj, zda je tento
and c ;bod prázdný, pokud
jr nz,TUR1B ;není, odskoč na hledání prázdného
rrc c ;zarotuj doprava masku a pokud nedošlo
jr nc,TUR3 ;k přechodu přes okraj, odskoč na další
TUR8 inc l ;posuň se na další byte na řádku
ld a,(hl) ;testuj, zda tento byte není
or a ;nulový, pokud ne,
jr nz,TUR3 ;tak odskoč na další bod,
ld a,b ;byte je nulový, odečti od zbývajících
sub 8 ;délky řádky 8 (šířka bytu) a pokud
jr c,TUR3 ;je to moc, odskoč na další bod
jr z,TUR9 ;pokud jsi dorazil na nulu, skonči
ld b,a ;vrat upravenou délku do B
jr TUR8 ;a testuj další byte na nulu

TUR3 djnz TUR2 ;opakuj pro všechny body čáry
jr TUR9 ;konec čáry

TUR1 ld a,(hl) ;vzvedni byte
and c ;a testuj hodnotu vybraného bitu,
jr z,TUR7 ;pokud je nulový, našel jsi další,
TUR1B rrc c ;jinak rotuj maskou doprava a pokud
jr nc,TUR4 ;nedošlo k přesunu, odskoč na další bod
TUR8B inc l ;posuň se na další byte
ld a,(hl) ;a testuj, zda je v něm
inc a ;hodnota 255 (plný),
jr nz,TUR4 ;pokud není, odskoč na další
ld a,b ;jinak od délky čáry
sub 8 ;odečti osmičku (šířka bytu)
jr c,TUR4 ;a pokud je to moc, jdi na další byte

```

```

jr z,TUR9 ;pokud jsi dorazil na nulu, skonči
ld b,a ;vrať zmenšenou délku do registru B
jr TUR8B ;a odskoč na test dalšího bytu

TUR4 djnz TUR1 ;opakuj pro další body čáry
TUR9 ;body nad čarou jsou otestovány

DOWN pop bc ;obnov bitovou masku a délku čáry
pop hl ;obnov adresu bytu
call DOWNHL ;posuň se o byte dolů
ld a,h ;nyní testuj, zda nejsi
cp 88 ;mimo obrazovku
jp nc,FILL0 ;pokud ano, odskoč na konec FILLu

ld a,(hl) ;testuj, zda je zde plno nebo volno,
and c ;pokud je zde plno,
jr nz,TDR1B ;odskoč na hledání prázdného bodu
TDR7 call PUSH ;ulož souřadnice bodu na čáře
TDR2 ld a,(hl) ;testuj, zda je tento
and c ;bod prázdný, pokud
jr nz,TDR1B ;není, odskoč na hledání prázdného
rrc c ;zarotuj doprava masku a pokud nedošlo
jr nc,TDR3 ;k přechodu přes okraj, odskoč na další
TDR8 inc l ;posuň se na další byte na řádku
ld a,(hl) ;testuj, zda tento byte není
or a ;nulový, pokud ne,
jr nz,TDR3 ;tak odskoč na další bod,
ld a,b ;byte je nulový, odečti od zbývajících
sub 8 ;délky řádky 8 (šířka bytu) a pokud
jr c,TDR3 ;je to moc, odskoč na další bod
jr z,TDR9 ;pokud jsi dorazil na nulu, skonči
ld b,a ;vrať upravenou délku do B
jr TDR8 ;a testuj další byte na nulu

TDR3 djnz TDR2 ;opakuj pro všechny body čáry
jr TDR9 ;konec čáry

TDR1 ld a,(hl) ;vzvedni byte
and c ;a testuj hodnotu vybraného bitu,
jr z,TDR7 ;pokud je nulový, našel jsi další,
TDR1B rrc c ;jinak rotuj maskou doprava a pokud
jr nc,TDR4 ;nedošlo k přesunu, odskoč na další bod
TDR8B inc l ;posuň se na další byte
ld a,(hl) ;a testuj, zda je v něm
inc a ;hodnota 255 (plný),
jr nz,TDR4 ;pokud není, odskoč na další
ld a,b ;jinak od délky čáry
sub 8 ;odečti osmičku (šířka bytu)
jr c,TDR4 ;a pokud je to moc, jdi na další byte
jr z,TDR9 ;pokud jsi dorazil na nulu, skonči
ld b,a ;vrať zmenšenou délku do registru B
jr TDR8B ;a odskoč na test dalšího bytu

TDR4 djnz TDR1 ;opakuj pro další body čáry
TDR9 ;body pod čarou jsou otestovány
jp FILL0 ;odskoč na zpracování další čáry

```

---

```

FILLEND  ld  a,4                ;nastav zelený border,
          out (254),a          ;což je signál, že program skončil
          ret                  ;návrat z podprogramu

UPHL     ld  a,h                ;tento UPHL se od tradičního liší tím,
          dec h                ;že netestuje přetečení horního okraje
          and 7
          ret nz
          ld  a,1
          sub 32
          ld  l,a
          ld  a,h
          ret c
          add a,8
          ld  h,a
          ret

DOWNHL   inc  h                ;pro DOWNHL platí totéž, co pro UPHL
          ld  a,h
          and 7
          ret nz
          ld  a,1
          add a,32
          ld  l,a
          ld  a,h
          ret c
          sub 8
          ld  h,a
          ret

PUSH     push de                ;ulož registr DE
          push hl               ;a adresu bytu
          push hl               ;(tu dvakrát, bude potřeba)

PUSHCNT  ld  hl,0              ;zvyš počet
          inc hl                ;uložených záznamů
          ld  (PUSHCNT+1),hl    ;o jedničku,
          ld  de,BUFSIZE+1     ;testuj, zda jich
          or  a                 ;není tolik, kolik
          sbc hl,de             ;je velikost FRONTY,
          jp  z,24000           ;pokud ano, skoč zpět do ASSEMBLERU
          pop de                ;do DE vezmi adresu bytu

PUSHPTR  ld  hl,0              ;nyní do HL adresu pro ukládání,
          ld  (hl),e            ;ulož
          inc hl                ;nejprve
          ld  (hl),d            ;adresu
          inc hl                ;bytu
          ld  (hl),c            ;a pak
          inc hl                ;bitovou masku,
          ld  de,SPACEEND     ;do DE dej adresu konce
          or  a                 ;oblasti pro uložení
          sbc hl,de             ;fronty, testuj, zda
          add hl,de             ;jsi ji ukazatelem dosáhl
          jr  nz,PUSH2         ;a pokud ne, odskoč
          ld  hl,SPACE         ;do HL dej adresu počátku oblasti
PUSH2    ld  (PUSHPTR+1),hl    ;posunutý ukazatel zapiš

```

---

```

pop hl ;obnov adresu bytu (registr HL)
pop de ;obnov také registr DE
ret ;a vrať se

POP push de ;ulož hodnotu v DE na zásobník
ld hl,(PUSHCNT+1) ;do HL dej počet záznamů
dec hl ;uložených ve FRONTĚ,
ld (PUSHCNT+1),hl ;zmenši o jedničku a opět ulož
ld a,h ;testuj, zda v registru HL
and l ;není hodnota 65535,
ld c,a ;pokud ano, je v C nyní 255,
inc a ;zvětši obsah A o jedničku,
jr z,POP3 ;a pokud se dostaneš na nulu, odskoč

POPPTR ld hl,0 ;do HL odebírací ukazatel
ld e,(hl) ;do registru DE
inc hl ;postupně
ld d,(hl) ;odeber adresu
inc hl ;bytu a do
ld c,(hl) ;registru C
inc hl ;odeber bitovou masku,
push de ;uschovej registr DE
ld de,SPACEEND ;do DE zapiš adresu
or a ;konce oblasti, pro
sbc hl,de ;ukládání záznamů
add hl,de ;fronty, testuj, zda ji ukazatel dosáhl,
pop de ;obnov registr DE,
jr nz,POP2 ;odskoč, když konec dosažen není
ld hl,SPACE ;do HL dej adresu počátku oblasti
POP2 ld (POPTR+1),hl ;posunutý ukazatel si ulož
POP3 ex de,hl ;hodnotu z DE dej do HL - adresa bytu
pop de ;obnov původní hodnotu DE
ret ;vrať se zpátky

PREPIS .... ;podprogram si opište z předminulého

SPACE defs 3*BUFSIZE ;vynechej místo na uložení FRONTY
SPACEEND ;adresa konce oblasti pro uložení FRONTY

```

To je tedy poslední příklad na program, který vyplní ohraničenou plochu. Oproti předchozím má tu výhodu, že nepotřebuje zdaleka tolik paměti, jako ty předchozí - většinou vystačí s necelým kilobytem - zkoušejte nastavovat **BUFSIZE** na stále menší hodnoty - když se program vrátí do assembleru teplým startem, došlo k přeplnění fronty - musíte zvětšit vyhrazené místo.

Reakci programu na přeplnění můžete samozřejmě změnit - obvykle asi nebudete chtít, aby se program vrátil někam jinam. Po zjištění přeplnění můžete udělat to, že odstraníte ty body, které jsou již vyplněny - což znamená, že jsou vyplněny i čáry, které reprezentují. Nejjednodušší bude, když budete do počtu bodů vždy odebírat (POP) z fronty bod, otestujete, jestli je již vyplněn nebo ne, pokud nebude, vrátíte jej (PUSH) zase do fronty. Teprve v případě, že ani po této redukci dat se jejich počet nezmenší, nelze obrazec s touto velikostí paměti tímto algoritmem vyplnit. Toto řešení samozřejmě můžete použít i v předchozím případě (vyplňování pootočeným čtvercem), které je vhodné pro případy, kdy chcete mít vyplnění efektivní.

Ještě na závěr něco k použitému způsobu ukládání dat, tedy k ZÁSObNÍKU a FRONTĚ. **Zásobník** se používá tam, kde chcete mít přístup k datům, která jste uložili naposledy, **fronta** pak tam, kde chcete nejprve zpracovávat data, která jste uložili nejdříve. Při použití zásobníku se obvykle něco (v našem případě vybarvováno provádí do hloubky (vyplňování vyrazí jedním směrem tak daleko, jak daleko to jen jde), v případě použití fronty se to provádí do šířky (vybarvují se nejdříve body, které jsou nejbližší výchozímu bodu).

# Spritová grafika

Když budete chtít napsat nějakou akční hru, budete v 90 % případů potřebovat sprity - pohyblivé obrázky. S jedním jsme se již setkali - ano, byla to šipka. Nyní si ukážeme další příklady a povíme víc o tom, jak se dá se sprity pracovat.

Začneme nejdříve programem, který nám z obrázku vytvoří sprite - vybere část obrazu a uloží ho do paměti tak, aby se nám s ním dobře pracovalo.

```

org 42000 ;začátek programu na 42000 - budeme jej
          ;volat také z BASICu
ent $ ;vstupní bod

DATA equ 50000 ;sprity se budou ukládat od 50000

START im 1 ;nastav mód přerušeni 1
      ld a,4 ;a také zelený
      out (254),a ;border

MAIN2 xor a ;vynuluj informaci
      ld (LAST+1),a ;o poslední stisknuté klávese

MAIN ei ;povol přerušeni
WAIT ld b,1 ;doba, po kterou bude program čekat
      ld a,1 ;a nyní ji
      ld (WAIT+1),a ;nastav na jedničku

WAIT2 push bc ;ulož délku čekání
      halt ;počkej na přerušeni
      call DRAWBOX ;vykreslí vybírací okénko
      halt ;počkej na přerušeni
      call DBAWBOX ;smaž vybírací okénko (kreslí se OVER 1)
      pop bc ;obnov délku čekání
      djnz WAIT2 ;a pokud není hotovo, opakuj čekání
      ld hl,(ENTER+1) ;vzvedni adresu, kde sprity končí
      ld bc,DATA ;a odečti od nich adresu, kde začínají,
      or a ;získáš tak současnou délku spritů,
      sbc hl,bc ;bude potřeba při návratu, tuto
      ld b,h ;délku zapiš také do registru BC, to je
      ld c,1 ;potřeba proto, aby mohla být čtena
      call 8020 ;BASICem - testuj stisk BREAKu
      ret nc ;a pokud je stisknutý, vrať se
      ld a,191 ;do A dej horní byte adresu portu,

```

```

in    a,(254)           ;na kterém je klávesa ENTER, přečti
rra                   ;jeho hodnotu a zarotuj 0-tý bit
jp    nc,ENTER        ;doprava, pokud je 0, je stisknut ENTER
call  CONTROLS       ;volej testování pěti zvolených kláves
ld    a,d             ;pokud je byte se stisknutými klávesami
or    a               ;nulový (nic není stisknuto),
jr    z,MAIN2        ;skoč na začátek

LAST   cp    0           ;zde je zapsána stará „klávesa“
ld    (LAST+1),a      ;novou hodnotu ulož jako starou
jr    z,NOWAIT       ;jsou-li nová i stará stejně, odskoč
cp    16             ;testuj, zda nová není jenom FIRE,
jr    z,NOWAIT       ;a pokud ano, odskoč
ld    a,5            ;nastav dobu čekání na 10 přerušení
ld    (WAIT+1),a     ;(vždy při změně kláves)

NOWAIT bit    4,d         ;testuj stisk FIRE
jr    nz,FIRE        ;a pokud je stisknuto, odskoč

bit    0,d           ;testuj klávesu pro směr VPRAVO
jr    z,LEFT        ;pokud není stisknuta odskoč
ld    hl,(DRAWBOX+1) ;vyzvedni adresu levého horního rohu
ld    a,(WIDTH+1)   ;vyzvedni šířku, přičti šířku
add   a,1           ;ke spodnímu bytu adresy a zjisti,
and   31            ;jestli nejsi na začátku dalšího
jr    z,LEFT        ;mikrořádku, pokud ano, odskoč
inc   l             ;jen nyní můžeš posunout rámeček vpravo
ld    (DRAWBOX+1),hl ;zapiš novou adresu levého horního rohu

LEFT   bit    1,d         ;testuj klávesu pro směr DOLU
jr    z,DOWN        ;a pokud není stisknuta, odskoč
ld    hl,(DRAWBOX+1) ;vyzvedni adresu levého horního rohu
ld    a,1           ;a testuj, zda nejsi na začátku
and   31            ;mikrořádku, pokud tomu tak je,
jr    z,DOWN        ;nelze se již posunout doleva a odskoč,
dec   l             ;posuň adresu rohu doleva
ld    (DRAWBOX+1),hl ;a zapiš ji zpátky

DOWN   bit    2,d         ;testuj, zda není zvolen směr DOLŮ
jr    z,UP          ;pokud není, odskoč
DOWNTEST ld hl,0        ;sem se při vykreslení zapsala adresa
ld    a,h           ;počátku spodního okraje rámečku,
sub   87            ;zjisti, zda se nejedná o spodní
ld    c,a           ;mikrořádek - tento, test vlastně
ld    a,1           ;zjišťuje, jestli není v horním bytu
and   %11100000    ;adresy číslo 87 a jestli nejvyšší tři
sub   %11100000    ;bity dolního bytu nejsou samé jedničky,
or    c             ;pokud to nastává, platí podmínka z,
ld    hl,(DRAWBOX+1) ;přečti si adresu levého horního
call  nz,DOWNHL     ;rohu a pokud můžeš, posuň se dolů
ld    (DRAWBOX+1),hl ;adresu zase zapiš zpátky

UP     bit    3,d         ;testuj, zda není zvolen směr NAHORU
jr    z,END         ;když ne, odskoč
ld    hl,(DRAWBOX+1) ;vyzvedni adresu levého horního rohu
ld    a,1           ;a testuj, zda nejsi na nejvyšším
and   %11100000    ;mikrořádku - tento test zjišťuje, zda
ld    c,a           ;jsou horní tři bity dolního bytu nulové

```

```

ld a,h ;a zda je v horním bytu číslo 64, pokud
sub 64 ;testovaná podmínka nastává, platí
or c ;podmínka z,
call nz,UPHL ;pokud můžeš, posuň se nahoru
ld (DRAWBOX+1),hl ;zapiš výsledek zpátky

END jp MAIN ;a skoč na začátek programu

FIRED bit 0,d ;testuj směr DOPRAVA
jr z,FLEFT ;a pokud není zvolen, odskoč
ld hl,(DRAWBOX+1) ;vzvedni adresu levého horního rohu
ld a,(WIDTH+1) ;a šířku rámečku, ke spodnímu bytu
add a,1 ;adresy přičti šířku a testuj, jestli
and 31 ;nejsi na začátku dalšího mikrořádku,
jr z,FLEFT ;pokud ano, odskoč
ld a,(WIDTH+1) ;nejsi a proto můžeš zvětšit
inc a ;šířku vybíracího okénka
ld (WIDTH+1),a ;o jedničku

FLEFT bit 1,d ;testuj směr DOLEVA
jr z,FDOWN ;a pokud není zvolen, odskoč
ld a,(WIDTH+1) ;vzvedni šířku rámečku,
dec a ;zmenši ji o jedničku,
jr z,FDOWN ;a pokud jsi na nule, odskoč
ld (WIDTH+1),a ;jinak novou šířku zapiš zpět

FDOWN bit 2,d ;testuj směr DOLŮ
jr z,FUP ;a pokud není zvolen, odskoč
ld hl,(DOWNTTEST+1) ;vzvedni adresu spodního okraje
ld a,h ;a zjisti, jestli neleží na spodním
sub 87 ;mikrořádku - to se dělá stejně
ld c,a ;jako v případě posunu dolů
ld a,1
and %11100000
sub %11100000
or c
jr z,FUP ;pokud je úplně dole, odskoč
ld a,(HEIGHT+1) ;vzvedni výšku
inc a ;rámečku a zvyš
ld (HEIGHT+1),a ;ji o jedničku

FUP bit 3,d ;testuj směr NAHORU
jr z,FEND ;a pokud není zvolen, odskoč
ld a,(HEIGHT+1) ;vzvedni výšku rámečku,
dec a ;zmenši ji o jedničku
cp 3 ;a zjisti, jestli je alespoň tři,
jr c,FEND ;pokud ne, odskoč na konec
ld (HEIGHT+1),a ;zapiš novou šířku rámečku

FEND jp MAIN ;skoč na začátek programu

ENTER ld de,DATA ;adresa, kam se budou sprity ukládat
ld hl,(DRAWBOX+1) ;adrese levého horního rohu okénka
ld a,(HEIGHT+1) ;výška okénka
ENTER2 push af ;ulož výšku
ld a,(WIDTH+1) ;šířka okénka

```



```

ENTER3  ld  b,a           ;zapiš ji do B
        push hl         ;a uschovej adresu začátku mikrořádku
        ld  a,(hl)      ;přesuň jeden byte z obrazovky
        ld  (de),a      ;do paměti
        inc hl          ;a posuň
        inc de          ;ukazatele
        djnz ENTER3     ;opakuj pro všechny byty na řádku
        pop hl         ;obnov adresu počátku mikrořádku
        call DOWNHL     ;spočítej adresu následujícího řádku
        pop af          ;obnov počítadlo mikrořádků
        dec a           ;a zmenši jej o jedničku
        jr  nz,ENTER2   ;pokud nejsi na nule, skoč pro další
        ld  (ENTER+1),de ;ulož si adresu volného místa
        call BEEP       ;oznam provedení zvukovým signálem
WAIT3   ld  b,30        ;počkej asi půl vteřiny
        halt           ;aby se stačila uvolnit
        djnz WAIT3     ;klávesa ENTER a nedošlo k opakování
        jp  MAIN        ;skoč na začátek programu

DRAWBOX ld  hl,16384    ;adresa levého horního rohu
WIDTH   ld  b,4        ;šířka okénka v bytech
        push bc         ;bojží ulož,
        push hl         ;budeme to ještě potřebovat
DB1     ld  a,(hl)     ;nyní
        cpl            ;nakresli
        ld  (hl),a     ;vodorovnou
        inc l          ;čáru pomocí
        djnz DB1       ;invertování
        dec l          ;vrať se o byte zpátky,
HEIGHT  ld  b,52       ;protože budeme kreslit
        dec b          ;svislou čáru - pravý okraj,
        dec b          ;odečti od výšky horní a dolní čáru
        push bc        ;a ulož na zásobník
DB2     call DOWNHL    ;posuň se o bod dolů
        ld  a,(hl)     ;a nakresli bod
        xor l          ;do nejpravějšího bitu
        ld  (hl),a    ;ve vybraném bytu
        djnz DB2       ;opakuj
        pop bc         ;obnov délku svislé čáry,
        pop hl         ;obnov adresu levého horního rohu,
DB3     call DOWNHL    ;posuň se o bod dolů
        ld  a,(hl)     ;a nakresli bod
        xor 128        ;do nejlevějšího bitu
        ld  (hl),a    ;ve vybraném bytu,
        djnz DB3       ;opakuj
        call DOWNHL    ;posuň se ještě o bod níž
        ld  (DOWNTTEST+1),hl ;a ulož dosaženou adresu pro testování
        pop bc         ;obnov délku vodorovné čáry
DB4     ld  a,(hl)     ;a nakresli ji
        cpl            ;pomocí invertování,
        ld  (hl),a     ;toto je
        inc l          ;spodní čára,
        djnz DB4       ;opakuj, dokud máš
        ret            ;a pak se vrať

UPHL    ....          ;standardní UPHL

```

```

DOWNHL    ....                ;standardní DOWNHL

BEEP      push af              ;uložíme registry,
          push bc              ;které budeme při
          push de              ;vytváření zvuku potřebovat
          ld b,0               ;hlavní smyčka
BEEP9     ld a,4               ;border bude zelený
BEEP4     add a,8              ;přepni EAR
          out (254),a          ;a pošli výsledek na port 254
          ld c,10              ;čekací smyčka,
BEEP3     dec c                ;která ovlivňuje dobu mezi jednotlivými
          jr nz,BEEP3          ;přepnutími a tím také výšku tónu
          djnz BEEP4           ;konec hlavní smyčky
          pop de                ;obnovení
          pop bc                ;všech
          pop af                ;registrů
          ret                   ;a návrat

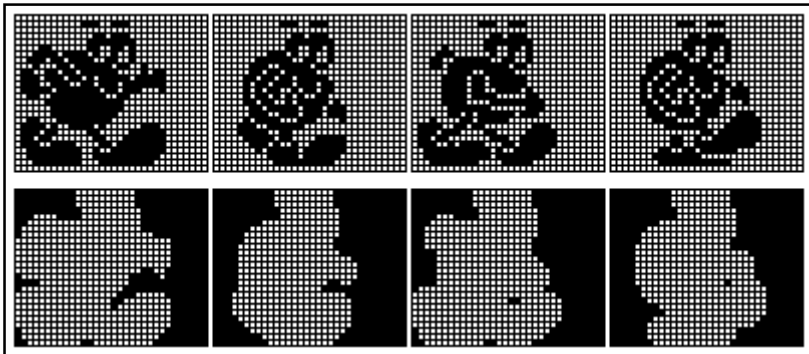
CONTROLS  ....                ;opište si z kapitoly VOLBA OVLÁDÁNÍ

REDEFINE  defb 254,223,1      ;klávesa P
          defb 254,223,2      ;klávesa O
          defb 254,253,1      ;klávesa A
          defb 254,251,1      ;klávesa Q
          defb 254,127,4      ;klávesa M

```

Program se ovládá pomocí kláves **O**, **P**, **Q**, **A**, **M**, **Enter** a **Break**. První čtyři jsou určeny pro posunování okénkem (O=doleva, P=doprava, Q=nahoru, A=dolů), pokud stisknete pátou (M) a nějaký směr, bude se měnit velikost okénka, stiskem Enteru odešlete obsah vybraného okénka do paměti jako sprite a konečně poslední klávesa slouží k návratu z tohoto podprogramu.

Program není ani zdaleka napsán nejúspornějším způsobem, značné zkrácení můžete docílit tím, že některé často používané hodnoty - adresu levého horního rohu, šířku a výšku rámečku budete číst jen na začátku a ukládat jen na konci a ne stále.



Program si můžete vylepšit třeba tím, že přidáte definici ovládacích kláves, magnetofonové (diskové) operace (nahrání obrázku a uložení spritů) a budete někde

ukazovat, kolik paměti již sprity zabraly, kde se právě nachází rámeček a jakou má zrovna velikost - to již nechám na vás.

Nyní si celý zdrojový text uložte a nahrajte ART STUDIO. Zvolte si funkci zvětšení (magnify) a překreslete do levého horního rohu obrázek z předchozí stránky. Jednotlivé předlohy a masky nakreslete hned vedle sebe - výsledný obrázek by měl mít velikost 128 bodů na šířku a 52 bodů na výšku.

Obrázek si uložte na kazetu nebo disketu, nahrajte si assembler, do něj zdrojový text, přeložte a vyskočte do BASICu. Z basicu nahrajte obrázek do obrazovky (LOAD "jméno" SCREENS - případně s hvězdičkou pro D40), spusťte přeložený program (LET a=USR 42000, do proměnné a se nám uloží délka vytvořených spritů) a vyberte všechny čtyři dvojice předloha-masky. Pokud jste vše opsali správně, měla by být první dvojice po spuštění nastavena a stačí, když stisknete klávesu ENTER. Po vybrání první dvojice se přesuňte na další (čtyři posuny doprava) a tak pro všechny čtyři dvojice. Po vybrání všech dvojic se vraťte stiskem BREAKu. Když si nyní prohlédnete obsah proměnné a, mělo by v ní být číslo 832 (=4\*4\*52). Vytvořené sprity si opět uložte (SAVE "jméno"CODE 50000,a).

Abyste věděli, co jste vlastně udělali - pomocí programu jste postupně převedli do paměti čtyři části obrazu - jednotlivé mikrořádky vybraného okénka jsou uloženy postupně, první je nejvyšší mikrořádek a poslední nejspodnější mikrořádek. Jednotlivé sprity tedy začínají na relativních adresách 0, 208,416, 624 - předloha je vždy na začátku a maska o 104 byty dál.

Sprity tedy máme vytvořené, můžeme je rozhábat (oživit, animovat). Smažte zdrojový text a napište tento:

```

ent $ ;zde se program naprosto spustí

SPRITES equ 64000 ;tady jsou uloženy sprity
WIDTH equ 4 ;šířka spritu v bytech (v bodech x 8)
HEIGHT equ 26 ;výška spritu v bodech
SPRLEN equ WIDTH*HEIGHT*2 ;délka spritu - předloha + maska

START im 1 ;nastav mód přerušování 1
ei ;a povol přerušování
ld hl,16384 ;vyplň
ld de,16385 ;pixelovou
ld bc,6144 ;část obrazovky
ld (hl),%1010101 ;tímto číslem
ldir ;(vzniknou svislé pruhy)
ld bc,767 ;nastav v atributové části obrazovky
ld (hl),7 ;bílý papír a černý inkoust
ldir ;(sprite je vlastně inverzní)

LOOP1 ld de,SPRITES ;do DE dej adresu prvního atributu
ld b,4 ;do B počet fází (máme čtyři)
LOOP2 push de ;ulož na zásobník adresu spritu
push bc ;a počítadlo fází spritu
XPOS ld bc,140*256+8 ;souřadnice, kde se sprite vykreslí
call DRAWSPR ;nakresli sprite
xor a ;nastav
out (254),a ;černý border

```

```

WAIT      ld  b,9                ;a počkej celkem
          halt                 ;devětkrát na
          djnz WAIT            ;přerušení - rychlost pohybu
          ld  a,4              ;nastav
          out (254),a          ;zelený border
          call REDRAWSP        ;vrať zpátky původní obsah obrazovky
          ld  hl,XPOS+1        ;do HL adresu, kde
          ld  a,(hl)           ;je uložena X-ová
          add a,8              ;pozice spritu,
          ld  (hl),a           ;posuň ji o osm doprava
          pop bc               ;obnov počítadlo fází
          pop de               ;obnov adresu počátku fáze
          ld  hl,SPLEN         ;přičti délku spritu
          add hl,de            ;jeho počáteční adrese,
          ex  de,hl            ;získáš tak adresu následníka
          djnz LOOP2           ;opakuj pro všechny fáze

          call 8020            ;testuj BREAK
          jr  c,LOOP1          ;a pokud není stisknut, kreslí znovu
          ret                  ;vrať se

DRAWSPR   ld  a,b              ;dej Y-ovou souřadnici také do A
          call #22B0           ;spočítej adresu bytu a polohu bitu
          ld  (REDRAWSP+1),hl ;ulož adresu pro smazání spritu

          ex  de,hl            ;prohoď adresu spritu a adresu v obr.
          push hl              ;ulož na zásobník adresu spritu
          exx                  ;přehoď na alternativní registry
          pop hl               ;do HL' dej adresu spritu
          ld  bc,SPLEN/2       ;do BC' dej polovinu délky spritu
          add hl,bc            ;sečti oba registry - máš adresu masky
          ld  de,SPACE         ;do DE' dej adresu úschovného prostoru
          exx                  ;vrať zpátky původní registry

DRS1      ld  c,HEIGHT         ;do C dej výšku spritu
          ld  b,WIDTH          ;do B dej počet bytů na mikrořádek
          push de              ;ulož adresu mikrořádku v obrazovce
DRS2      ld  a,(de)           ;vzvedni byte
          exx                  ;přepni alternativní sadu registrů
          ld  (de),a           ;uschovej původní obsah bytu
          inc de               ;a posuň se na volné místo
          and (hl)             ;ponech z obsahu jen to, co je v masce
          inc hl               ;jedničkové, posuň se na další byte
          exx                  ;přepni registry zpátky
          or  (hl)             ;přidej byte předlohy
          ld  (de),a           ;a to vše zapiš zpět do obrazovky
          inc hl               ;posuň ukazatel do předlohy
          inc e                 ;a adresu na mikrořádku
          djnz DRS2           ;opakuj pro každý byte řádku
          pop de               ;obnov adresu počátku mikrořádku
          call DOWNDE          ;posuň se o bod na obrazovce dolů
          dec c                 ;další mikrořádek
          jr  nz,DRS1          ;kreslí jen pokud ještě nejsou všechny
          ret                  ;vrať se zpátky

REDRAWSP  ld  de,0             ;sem se při kreslení ukládá adresa,
          ld  hl,SPACE         ;kam se má původní obsah vracet,
          ld  c,HEIGHT         ;do C dej výšku spritu

RDRS1     ld  b,WIDTH          ;do B dej počet bytů na mikrořádku
          push de              ;ulož adresu KAM

```

```

RDRS2  ld  a,(hl)           ;přesuň jeden byte
        ld  (de),a         ;z paměti na obrazovku
        inc hl             ;a posuň oba
        inc e              ;ukazatele,
        djnz RDRS2        ;opakuj do počtu bytů na řádku
        pop de             ;obnov adresu počátku mikrořádku
        call DOWNDE       ;a spočítej adresu bytu o bod níž
        dec c              ;pokud jsou ještě
        jr  nz,RDRS1      ;nějaké mikrořádky, opakuj
        ret                ;vrať se

DOWNDE  inc  d              ;tradiční podprogram DOWNHL
        ld  a,d            ;upravený pro registr DE
        and 7
        ret nz
        ld  a,e
        add a,32
        ld  e,a
        ld  a,d
        jr  c,DOWNDE2
        sub 8
        ld  d,a
DOWNDE2 cp  88
        ret  c
        ld  d,64
        ret

SPACE   defs  SPLEN           ;místo pro původní obsah obrazovky

```

Předtím, než program spustíte, nesmíte zapomenout na adresu 64000 nahrát vytvořené sprity. Při spouštění pozor, program není upraven pro volání z BASICu (používá HL' a nevrací tam hodnotu 10072, pokud byste chtěli program volat z BASICu, musíte před návrat přidat instrukce **ld hl,10072** a **exx**.

Doufám, že vám Olli (ze hry Olli & Lissa) pěkně rázuje zleva doprava, pokud ne, museli jste někde udělat chybu.

Možná jste si už při psaní všimli, že se nijak nevyužívá informace o tom, jaký bit v bytu odpovídá nastaveným souřadnicím - tento program umožňuje kreslit sprity jen na bytové pozice. Jeho výhoda spočívá v tom, že je rychlý - nemusí totiž provádět žádné bitové posuny. Sprite, který jsem použil je nakreslen tak, že jednotlivé fáze se mají kreslit vždy o byte dál, tak si musíte kreslit i vlastní sprity. Pokud chcete použít tento program a kreslit sprity i na jiné, než bytové pozice, musíte si nakreslit více předloh a masek, které se budou lišit jen posunem vůči bytu (třeba o čtyři body) a můžete pak kreslit sprity s přesností na čtyři body. Informaci o poloze bitu v bytu pak využijete pro volbu správné předlohy. Pokud budete vytvářet program, kde budete používat více typů spritů (avšak najednou jen několik, můžete si potřebné posunuté předlohy a masky vyrobit programově až při běhu - ušetříte paměť, kterou budete jistě potřebovat na jiné věci).

Časovou náročnost kreslení sprity si můžete odhadnout podle blikajícího zeleného pruhu v borderu - čas, po který se obraz (a tedy i border) vykresluje vždy znovu trvá 1/50 sekundy (čas mezi dvěma přerušeními). Dobu, kterou z této jedné padesátiny zabírá

vykreslení spritu pak svítí border zeleně, ostatní dobu svítí border černě - vykreslení našeho spritu tedy trvá asi šestinu až sedminu času, který uběhne mezi dvěma přerušeni. Rychlost, jakou se Olli pohybuje, můžete změnit tím, že změníte číslo, které se dává do registru B před návěštím **WAIT**.

Asi vás už napadlo, co dělat v případě, že budete chtít mít víc spritů - budete si muset připravit tolik úložného prostoru, kolik spritů budete kreslit, upravit program pro kreslení spritu tak, aby bylo možno volit, kam se bude obraz ukládat a také program pro obnovení původního stavu obrazovky tak, aby mohl číst z několika míst - stačí, když uvedené parametry do programu předáte v registrech, můžete si ovšem vytvořit komplexnější ukládání, které si bude pro každou část pamatovat **odkud** a **kolik** (šířka a výška) bytů bylo uloženo a samozřejmě **kde** jsou byty uschovány a **počet** těchto oblastí. Program REDRAWSP pak podle dat vrátí vše zpátky do obrazovky.

```

ent $ ;místo, kde se program může spustit

SPRITES equ 64000 ;adresa, kde jsou sprity uloženy
WIDTH equ 2 ;počet bytů na mikrořádku spritu
HEIGHT equ 16 ;výška spritu v bodech
SPRLEN equ WIDTH*HEIGHT*2 ;délka jednoho spritu

START im 1 ;nastav mód přerušeni jedna
ei ;a povol přerušeni
ld hl,16384 ;vyčisti
ld de,16385 ;obrazovku
ld bc,6144
ld (hl),0
ldir
ld bc,767 ;a nastav
ld (hl),7 ;černý papír
ldir ;a bílý inkoust

LOOP1 ld de,SPRITES ;do DE dej adresu spritů
ld b,4 ;máme celkem čtyři fáze,
LOOP2 push de ;ulož adresu spritu
push bc ;ulož číslo fáze
XPOS ld bc,80*256+8 ;souřadnice pro nakreslení
call DRAWSPR ;vykresli sprite
xor a ;nastav
out (254),a ;černý border
ld b,7 ;počkej
WAIT halt ;celkem
djnz WAIT ;7x na přerušeni
ld a,4 ;nastav
out (254),a ;zelený border
call REDRAWSP ;vrať původní obsah obrazovky
ld hl,XPOS+1 ;posuň
ld a,(hl) ;X-ovou
add a,1 ;souřadnici
ld (hl),a ;doprava
pop bc ;obnov počítadlo mikrořádků
pop de ;obnov ukazatel na sprite
ld hl,SPRLEN ;do HL dej délku spritu
add hl,de ;a přičti k ukazateli,
ex de,hl ;ukazatel na nový sprite dej do DE

```

```

ld a,b ;testuj, zde se nejedná o poslední
cp 2 ;fázi, ta se totiž kreslí jako
jr nz,LOOP3 ;ta druhá, pokud ne, odskoč,
ld de,SPRITES+SPRLEN ;nastav ukazatel na druhou fázi
LOOP3 djnz LOOP2 ;opakuji celkem čtyřikrát

call 8020 ;testuj klávesu BREAK
jr c,LOOP1 ;a pokud není stisknuta, skoč
ret ;na začátek, jinak se vrať

DRAWSPR ld a,b ;do A dej Y-ovou souřadnici spritu
call #22B0 ;vypočítej adresu bytu a polohu bitu
ld (REDRAWSP+1),hl ;zapiš adresu obrazovky pro smazání
ex de,hl ;prohod adresy obrazovky a předlohy
ld (SHIFT+1),a ;uschovej počet posunů doprava
exx ;přepni na alternativní registry
ld de,SPACE ;do DE' dej adresu úložného prostoru
exx ;vrať zpátky původní registry

DRS1 ld b,HEIGHT ;do B dej výšku spritu
push bc ;uschovej počítadlo mikrořádků
ld (SCRADR+1),de ;uschovej adresu v obrazovce
ld a,(hl) ;vyzvedni nejprve
inc hl ;první dva byty
ld d,(hl) ;masky
inc hl ;a potom hned
ld e,(hl) ;dva byty
inc hl ;předlohy,
ld e,(hl) ;posunuj stále
inc hl ;adresu do spritu,
push hl ;ulož adresu v předloze,
ld hl,255*256 ;do H dej 255 (maska) a do L dej nulu,
SHIFT ld b,0 ;nyní budeme posunovat
inc b ;řádek předlohy i masky doprava,
dec b ;testuj, zda je vůbec
jr z,DRS2 ;potřeba posunovat, pokud ne, odskoč

DRS3 scf ;do masky musí zleva vstupovat jednička
rra ;rotuj doprava
rr d ;všechny tři
rr h ;byty masky (třetí je na začátku 255)
srl c ;do předlohy vstupuje, zprava nula, rotuj
rr e ;stejně jako u masky všechny tři byty,
rr l ;které k ní patří (třetí je zpočátku 0)
djnz DRS3 ;opakuji tolikrát, kolikrát je potřeba

DRS2 ld b,a ;dej část masky z registru A do B
push hl ;uschovej třetí byty masky a předlohy
SCRADR ld hl,0 ;do HL adresu v obrazovce
ld a,(hl) ;vyzvedni původní obsah obrazovky,
exx ;přepni na alternativní registry
ld (de),a ;a zapiš ho do úschovné paměti,
inc de ;posuň ukazatel do ní,
exx ;vrať zpátky původní registry,
and b ;ponech to, čemu v masce odpovídají
or c ;jedničky, připoj předlohu
ld (hl),a ;a vše zapiš zase zpátky do obrazovky
inc hl ;posuň ukazatel na další byte
ld a,(hl) ;vyzvedni původní obsah bytu
exx ;přepni na alternativní registry

```

```

ld (de),a ;a zapiš původní obsah do úložné paměti,
inc de ;posuň ukazatel do úschovné paměti
exx ;a vrať zpátky původní registry
and d ;ponech jen bity odpovídající masce,
or e ;připoj předlohu
ld (hl),a ;a vše zapiš zpět do obrazovky,
inc hl ;posuň se na další adresu
pop de ;do DE vezmi poslední byty řádku
ld a,(hl) ;vzvedni původní obsah bytu,
exx ;přepni na alternativní registry
ld (de),a ;a zapiš původní obsah do úložné paměti,
inc de ;posuň ukazatel do úschovné paměti
exx ;a vrať zpátky původní registry
and d ;ponech jen bity odpovídající masce,
or e ;připoj předlohu
ld (hl),a ;a vše zapiš zpět do obrazovky,
ld de,(SCRADR+1) ;vzvedni počáteční adresu tohoto řádku
call DOWNDE ;posuň se o řádek dolů
pop hl ;obnov ukazatel do spritu
pop bc ;obnov počítadlo mikrořádků spritu
djnz DRS1 ;a opakuj pokud není všechno vykresleno
ret ;vrať se zpět

REDRAWSP ld de,0 ;adresa, kam data na obrazovku patří
ld hl,SPACE ;do HL adresu úložného prostoru
ld c,HEIGHT ;do C počet mikrořádků
RDRS1 ld b,WIDTH+1 ;do B počet bytů na mikrořádku
push de ;ulož adresu počátku mikrořádku
RDRS2 ld a,(hl) ;přenes
ld (de),a ;jeden
inc hl ;mikrořádek
inc e ;z paměti
djnz RDRS2 ;na obrazovku,
pop de ;obnov adresu počátku mikrořádku
call DOWNDE ;spočítej adresu následujícího řádku
dec c ;zmenši počet mikrořádků
jr nz,RDRS1 ;a pokud nejsi na nule, opakuj
ret ;vrať se

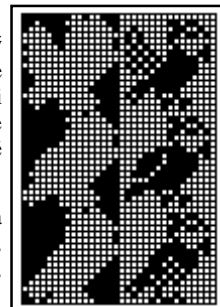
DOWNDE .... ;podprogram je stejný jako minule

SPACE defs WIDTH+1*HEIGHT*2 ;místo pro uložení původního obsahu
;obrazovky, pozor na vyčíslení výrazu,
;provádí se zleva doprava

```

K programu také patří sprity, jsou zobrazeny na obrázku, který vidíte vpravo. Až je překreslíte (třeba v ART STUDIU), použijte opět spritovací program - celý obrázek vyberte najednou, při ukládání by vám měla vyjít délka 96 bytů. Všimněte si, že se tentokrát mikrořádky masky a mikrořádky předlohy pravidelně střídají - je to dáno programem.

Pokud budete chtít pracovat s většími sprity, tak zvětšení na výšku problémy činit nebude, horší to bude se zvětšením do šířky, tam narazíme na problémy s malým počtem registrů pro rotace,





budete muset nejspíš odděleně rotovat masku a předlohu a používat k ukládání dat paměť nebo zásobník, také máte k dispozici registry IX a IY (jejich poloviny LX a HX a LY a HY) a alternativní sadu registrů - práce s registry je skoro vždy rychlejší než práce s pamětí, záleží však na případu.

Nakreslit jeden sprite, to umíme, potíže nastanou v okamžiku, kdy budeme chtít vykreslit spritů více, narazíme na problém s překrýváním spritů. Když si rozmyslíte, v jakém pořadí se mají sprity kreslit a hlavně odkreslovat, zjistíte, že se prodlužuje doba, kdy na obrazovce nic není. Sprity se musí odkreslovat v opačném pořadí, než v jakém se vykreslily, nelze tedy v každém přerušení zpracovávat jeden sprite. Čas, po který nejsou všechny sprity vykresleny se prodlužuje s jejich počtem, při vyšším počtu zjistíte, že se to tradičním způsobem nedá stihnout vykreslit v době, kdy je paprsek mimo obrazovku - kdy kreslí border pod a nad obrazem. Potom musíte použít nějaké finty - například si dopředu rotovat spritem a propočítat adresy, kam se bude vracet původní obsah obrazovky a kde se bude vykreslovat sprite - vše si můžete uložit na zásobník a pak jenom jednoduchým programem číst data ze zásobníku a zapisovat je do obrazovky. Pokud je spritů skutečně mnoho, nemusí ani tento způsob kreslení vést k cíli, pak je potřeba použít něco, co se nazývá „vnitřní“ nebo „pracovní“ obrazovka - o tom však pojednává jedna z dalších kapitol.

O všech těchto problémech by se dalo psát dlouho a dlouho, dá se o tom napsat třeba i diplomová práce (ověřeno z vlastní zkušenosti). Některé další věci se můžete dozvědět v seriálu **George K.'s Animace**, který vychází v časopise **ZX Magazin** a který se některými detaily zabývá podstatně podrobněji a pomaleji. My nyní na nějaký čas sprity opustíme ale ještě se s nimi nerozloučíme.

# Přerušení

Ačkoliv k tomu název této kapitoly navádí, nejedná s o přerušení textu v této knize nebo o přerušení vaší pozornosti. Budeme se tu věnovat tomu, o čem se tu již delší dobu mluví aniž by se přesně řeklo, co to znamená.

V hardware počítače je obvod, který dokáže každou padesátinu sekundy (to je mimo jiné také kmitočet v naší síti (220V, 50Hz) posílat procesoru nějaký signál. Podle toho, v jakém stavu se procesor zrovna nachází, s ním něco dělá. Jsou dvě základní možnosti - buď je přerušení zakázáno, pak jej procesor prostě ignoruje, nebo je povoleno, a pak se zpracuje podle toho, jaký mód přerušení je zrovna nastaven.

Pokud je nastaven mód jedna (instrukcí **im 1**), provede se podprogram na adrese 56 (#38), kde se v ROM Spectra nachází testování klávesnice a hodiny.

Pokud je nastaven mód přerušení dva (instrukcí **im 2**), je to složitější. Procesor si vyzvedne číslo uložené v registru I, to bude horní byte adresy, vyzvedne číslo z datové sběrnice a udělá z něj dolní byte adresy - toto číslo by mělo obstarávat připojené zařízení, na Spectru však o žádném nevíme. Proto nám nezbyvá než předpokládat, že je tam naprosto

cokoliv (samořejmě v rozsahu 0 až 255). Procesor má tedy adresu, nyní se předpokládá, že toto číslo ukazuje do tabulky adres, z ní se vyzvedne adresa a zavolá se podprogram, který na ní začíná. Podrobněji opět v nějakém příkladu.

V obou případech se při volání podprogramu zakáže přerušení (to proto, aby program pro obsluhu přerušení nemohl být znovu přerušen). Při návratu z programu (na zásobníku je adresa instrukce, která se měla v okamžiku přijetí přerušení začít provádět, můžete ji občas potřebovat) obsluhy přerušení tedy nesmíte zapomenout přerušení povolit.

V basicu se pomocí přerušení tedy realizuje testování klávesnice, možná se vám už stalo, že se vám program ve strojovém kódu vrátil zpátky do basicu, vypsal OK 0,1 a dál na nic nereagoval - pravděpodobně jste při návratu zapoměli povolit přerušení.

V programech psaných ve strojovém kódu se pomocí přerušení (hlavně jeho druhého módu) zajišťuje všechno možné.

Signál přerušení má jednu důležitou vlastnost, která souvisí se zobrazováním obrazu na obrazovce - nastává totiž právě v okamžiku, kdy začíná paprsek vykreslovat znovu celou obrazovku. Velice potřebný je tento signál proto, aby sprity na obrazovce neblikaly. Blikání spritů si můžete všimnout u některých starších her (Galaxians, Lunar Jetman), tam se totiž na nějakou synchronizaci s kreslením obrazu ještě nebral zřetel.

Přerušení je vhodné pro činnosti, které se mají pravidelně opakovat a pro vytváření různých efektů, případně pro činnosti, které se mají provádět nezávisle na tom, co dělá hlavní program. Je to vlastně jakýsi zárodek více úloh prováděných najednou (multitasking), o tom si však ještě povíme. Uvedeme si opět příklad. Program vychází z druhého příkladu na spritovou grafiku a proto uděláte nejlépe, když do něj dopíšete to, co v něm není, a přepíšete to, co je v něm jinak.

```

ent      $                ;místo, kde se program může spustit

SPRITES equ 64000        ;adresa, kde jsou sprity uloženy
WIDTH   equ 2            ;počet bytů na mikrořádku spritu
HEIGHT  equ 16           ;výška spritu v bodech
SPRLEN  equ WIDTH*HEIGHT*2 ;délka jednoho spritu

START    di              ;zákaz přerušení
         im 2            ;nastav mód přerušení 2
         ld a,#FD        ;nastav do registru I
         ld i,a          ;číslo #FD, tabulka začíná na #FDOO
         ld hl,#FD00     ;vyplníme tabulku přerušení
         ld de,#FD01     ;číslem #FE, všechny adresy v tabulce
         ld bc,256       ;tedy budou mít hodnotu #FEFE, takto
         ld (hl),#FE     ;si zařídíme, že nám nezáleží na
         ldir            ;hodnotě spodního bytu (tu neovlivníme)
         ld hl,#FEFE    ;do HL dej adresu, kde má být program
         ld (hl),195    ;pro obsluhu přerušení, nám se však
         inc hl         ;tato adresa zrovna nehodí a proto
         ld (hl),INTRPT?256 ;si tam dáme skok na nějakou
         inc hl         ;vhodnější adresu, nemusíme se tedy
         ld (hl),INTRPT/256 ;starat, kde program pro přerušení bude

         ld hl,16384    ;vyplň

```

```

ld de,16385 ;obrazovku
ld bc,6144 ;číslem 255
ld (hl),255
ldir
ld bc,767 ;a nastav
ld (hl),56 ;bílý papír
ldir ;a černý inkoust
ei ;povol přerušení

LOOP1 ld de,SPRITES ;do DE dej adresu spritů
ld b,4 ;máme celkem čtyři fáze,
LOOP2 push de ;ulož adresu spritu
push bc ;ulož číslo fáze
XPOS ld bc,20*256+8 ;souřadnice pro nakreslení
call DRAWSPR ;vykresli sprite
xor a ;nastav
out (254),a ;černý border
ld b,5 ;počkej
WAIT halt ;celkem
djnz WAIT ;5x na přerušení
ld a,4 ;nastav
out (254),a ;zelený border
call REDRAWSP ;vrať původní obsah obrazovky
ld hl,XPOS+1 ;posuň
ld a,(hl) ;X-ovou
add a,1 ;souřadnici
ld (hl),a ;doprava
pop bc ;obnov počítadlo mikrořádků
pop de ;obnov ukazatel na sprite
ld hl,SPRLEN ;do HL dej délku spritu
add hl,de ;a přičti k ukazateli,
ex de,hl ;ukazatel na nový sprite dej do DE
ld a,b ;testuj, zde se nejedná o poslední
cp 2 ;fázi, ta se totiž kreslí jako
jr nz,LOOP3 ;ta druhá, pokud ne, odskoč,
ld de,SPRITES+SPRLEN ;nastav ukazatel na druhou fázi
LOOP3 djnz LOOP2 ;opakuje celkem čtyřikrát

call 8020 ;testuj klávesu BREAK
jr c,LOOP1 ;a pokud není stisknuta, skoč
ret ;na začátek, jinak se vrať

DRAWSPR .... ;to si opište, není tu žádná změna

REDRAWSP .... ;tady platí naprosto totéž

DOWNDE .... ;a tady to jiné není

INTRPT push af ;ulož všechny registry, které
push bc ;budeš v programu, který
push de ;provádí obsluhu přerušení
push hl ;potřebovat (raději více než méně)

ld a,1 ;nastav
out (254),a ;modrý border
ld hl,757 ;a čekej
WAIT3 dec hl ;nějakou dobu,
inc h ;testuj, jestli

```

```

dec h ;je v H konečně
jr nz, WAIT3 ;nula, když ne, čekej dál
ld a, 7 ;nastav
out (254), a ;bílý border,
ld b, 33 ;čekej než se vykreslí
WAIT2 djnz WAIT2 ;dva mikrořádky

ld a, 2 ;nastav
out (254), a ;červený border
nop ;čekej
nop ;(jemné ladění)
ld hl, 312 ;čekej
WAIT4 dec hl ;delší
inc h ;dobu
dec h ;s červeným
jr nz, WAIT4 ;borderem

ld a, r ;nyní vytvoříme
and 7 ;podle registru R
ld c, a ;atribut, který
rlca ;bude mít stejný
rlca ;inkoust
rlca ;i papír,
or c ;s tímto
ld hl, 22528+32 ;atributem
ld b, 32 ;vybarvíme
FILL1 ld (hl), a ;druhý
inc hl ;atributový řádek
djnz FILL1 ;obrazovky,
out (254), a ;na stejnou barvu nastavíme border
ld b, 135 ;a budeme čekat, dokud se
WAIT5 djnz WAIT5 ;nevykreslí osm mikrořádků

ld a, 1 ;nastav
out (254), a ;modrý border
ld b, 0 ;a opět
WAIT6 djnz WAIT6 ;si nějakou
nop ;dobu
nop ;počkej

ld a, 7 ;nastav
out (254), a ;bílý border
ld b, 33 ;a počkej až se vykreslí
WAIT7 djnz WAIT7 ;dva mikrořádky

ld a, 0 ;nastav
out (254), a ;černý border
ld hl, 100 ;a teď po
WAIT8 ld a, r ;nějakou
out (254), a ;dobu
dec hl ;posílej
nop ;na border
nop ;různé barvy,
nop ;navíc se tu
ld a, h ;také generuje
or l ;zvuk
jr nz, WAIT8 ;(dost nepřijemný)

```

```

ld    a,0           ;nastav
out   (254),a       ;černý border

pop   hl           ;obnov
pop   de           ;původní
pop   bc           ;hodnoty
pop   af           ;registrů,
ei    ei           ;povol přerušeni
ret   ret          ;a vrať se

SPACE  defs WIDTH+1*HEIGHT*2 ;úložný prostor

```

Majitelé Didaktiků, hlavně těch, co nesou hrdé označení M upozorňuji, že jsem časové konstanty (délky čekacích smyček) empiricky zjistil na ZX Spectru, může se stát, že vzhledem k odlišné rychlosti počítačů dojde k poškození pruhů - budete muset prodloužit čekací smyčky - délky zvyšujte pomalu (po jedničkách), jinak nic nezjistíte. Pokud to bude nutné, použijte instrukce NOP pro jemnější doladění.

To je k přerušeni asi všechno, další příklad, k čemu se dá přerušeni použít, naleznete v následující kapitole.

# Multitasking

V této kapitole si ukážeme, jakým způsobem je možno spustit několik programů „naráz“. Programy nebudou pracovat současně ale budou se střídat každou padesátinu sekundy, tedy pro uživatele prakticky nepozorovatelně. Tomuto způsobu zpracování programů se říká **multitasking** (multi = více, task = úloha). Nejprve si tedy ukážeme příklad a pak si něco řekneme k zajímavým detailům vlastního programu.

```

ent $

START di           ;inicializace se zakázaným přerušeni
ld    (RETSP+1),sp ;ulož původní SP pro návrat
im    2           ;nastav mód přerušeni 2
xor   a          ;na začátku nemáme
ld    (NUMPROC+1),a ;žádný proces
ld    a,#FD      ;horní byte tabulky
ld    i,a        ;do registru I
ld    hl,#FD00   ;vytvořime
ld    de,#FD01   ;tabulku
ld    bc,256     ;adres
ld    (hl),#FE   ;pro
ldir  ;přerušeni
ld    hl,#FEFE   ;na adrese, kam
ld    (hl),195   ;ukazují všechny

```

```

inc hl ;položky tabulky
ld (hl),INTRPT?256 ;vytvoříme
inc hl ;instrukci jp INTRPT
ld (hl),INTRPT/256

ld hl,STACK1 ;první adresa za místem pro zásobník
ld de,RUTINA1 ;první rutiny, startovní adresa pro
call INSPROC ;první rutinu, vlož je do systému

ld hl,STACK2 ;první adresa za místem pro zásobník
ld de,RUTINA2 ;druhé rutiny, startovní adresa pro
call INSPROC ;druhou rutinu, vlož je do systému

ld hl,STACK3 ;první adresa za místem pro zásobník
ld de,RUTINA3 ;třetí rutiny, startovní adresa pro
call INSPROC ;třetí rutinu, vlož je do systému

ld hl,STACK4 ;první adresa za místem pro zásobník
ld de,RUTINA4 ;čtvrté rutiny, startovní adresa pro
call INSPROC ;čtvrtou rutinu, vlož je do systému

ld hl,TEXT1 ;do HL' dej adresu prvního textu
ld de,20480+255 ;do DE' dej adresu v obrazovce, kde
exx ;se bude vypisovat znak, přepni registry
ld hl,STACK5 ;první adresa za místem pro zásobník
ld de,RUTINA5 ;páté rutiny, startovní adresa pro
call INSPROC ;pátou rutinu, vlož je do systému

ld hl,TEXT2 ;do HL' dej adresu prvního textu
ld de,20480+31 ;do DE' dej adresu 0 obrazovce, kde
exx ;se bude vypisovat znak, přepni registry
ld hl,STACK6 ;první adresa za místem pro zásobník
ld de,RUTINA5 ;šesté rutiny, startovní adresa pro
call INSPROC ;pátou rutinu, vlož je do systému

ld a,-1 ;proces -1 jako aktuální
ld (PROCESS+1),a ;pro začátek

PROCESS ld a,0 ;číslo aktuálního procesu
inc a ;zvyš o jedničku
NUMPROC cp 0 ;a porovnej s počtem procesů,
jr c,LOOP ;pokud jsi v povoleném rozsahu, odskoč
xor a ;jinak prováděj procesy znovu od začátku
LOOP ld (PROCESS+1),a ;a nový aktuální proces ulož pro příště
call 8020 ;testuj BREAK
jr c,LOOP2 ;a odskoč, když není stisknut
RETSP ld sp,0 ;do SP původní hodnotu
im 1 ;mód přerušení jedna,
ei ;povol přerušení
ret ;a vrať se zpátky

LOOP2 ld a,(PROCESS+1) ;vyzvedni číslo aktuálního procesu,
add a,a ;spočítej
ld c,a ;adresu
ld b,0 ;v tabulce
ld hl,PROCTAB ;procesů, kde
add hl,bc ;se nachází
ld a,(hl) ;jeho hodnota

```

```

inc hl ;SP registru,
ld h,(hl) ;vyzvedni
ld l,a ;hodnotu do HL,
ld a,0 ;nastav
out (254),a ;černý border
ld sp,hl ;nastav registr SP
pop iy ;a obnov hodnoty
pop ix ;všech registrů
pop hl
pop de
pop bc
pop af
ex af,af'
exx
pop hl
pop de
pop bc
pop af
ei ;povol přerušení
ret ;a vrať se do vybrané rutiny

INTRPT push af ;program pro obsluhu přerušení
push bc ;začíná tím, že se na zásobník
push de ;uloží všechny registry
push hl
exx
ex af,af'
push af
push bc
push de
push hl
push ix
push iy
ld a,(PROCESS+1) ;adresa, kam ukazuje registr SP
add a,a ;se musí zapsat na správné
ld c,a ;místo do tabulky procesů,
ld b,0 ;místo se spočítá
ld hl,PROCTAB ;podle čísla procesu,
add hl,bc ;získaná adresa
ld (ADR+2),hl ;se zapíše do instrukce
ADR ld (0),sp ;konečně můžeš adresu uložit
jp PROCESS ;a skočit pro další proces

INSPROC push hl ;uschovej adresu konce zásobníku,
ld hl,NUMPROC+1 ;do HL dej adresu počtu procesů (úloh),
ld a,(hl) ;vyzvedni počet procesů,
inc (hl) ;zvyš počet procesů o jedničku
add a,a ;původní počet vynásob dvěma
ld c,a ;a dej do C,
ld b,0 ;do B dej 0, v BC je relativní poloha
ld hl,PROCTAB ;v tabulce procesů, do HL adresa tabulky
add hl,bc ;sečti relativní a absolutní adresu
ld (ADR2+2),hl ;a získané místo zapiš do instrukce,
pop hl ;obnov adresu konce zásobníku
ld (SPSTOR+1),sp ;ulož současný ukazatel na zásobník
ld sp,hl ;použij zásobník procesu,
push de ;ulož na něj nejprve startovací adresu

```

```

push af ;a potom postupně všechny registry
push bc ;včetně alternativních a indexregistrů
push de
push hl
exx
ex af,af'
push af
push bc
push de
push hl
push ix
ld iy,23610 ;v programu používáme služby ROM
push iy
ADR2 ld (0),sp ;do instrukce se zapisuje adresa
SPSTOR ld sp,0 ;do tabulky, obnov původní hodnotu
ret ;SP registru a vrať se

PROCTAB defs 20 ;místo pro 10 úloh (můžete zvětšit)

RUTINA1 ld a,2 ;první rutina
call #1601 ;tiskne
RUT1A ld a,22 ;do horní
rst 16 ;třetiny
xor a ;obrazovky
rst 16 ;náhodně
xor a ;obarvené
rst 16 ;znaky
RUT1B ld b,0
push bc
ld a,r
res 7,a
ld (23695),a
ld a,r
and 63
add a,32
rst 16
pop bc
djnz RUT1B
jr RUT1A

RUTINA2 ld hl,18432 ;tento
ld bc,2048 ;podprogram
RUT2A ld a,r ;vyplňuje
ld (hl),a ;pixely
inc hl ;v prostřední
dec bc ;třetině
ld a,b ;náhodnými
or c ;hodnotami,
jr nz,RUT2A ;po jednom vyplnění
halt ;se počká na přerušení, bohužel se
jr RUTINA2 ;to nestihne před příchodem paprsku

RUTINA3 ld a,r ;tento program
ld hl,22528+512+32 ;vyplňuje atributy

```



```

RUT3A    ld    b,192           ;ve spodní třetině,
         ld    (hl),a         ;náhodným číslem
         inc  l               ;vynechává přitom
         djnz RUT3A          ;oba krajní atributové
         jr   RUTINA3        ;řádky, to vše dělá do zblbnutí

RUTINA4  ld    a,r           ;tato rutina vytváří zvuk,
         and  24              ;dělá to tak, že bere
         out (254),a         ;hodnotu z registru R
         jr   RUTINA4        ;a posílá ji neustále na port

RUTINA5  exx                ;parametry v alternativních registrech
RUT5G    push hl             ;ulož adresu počátku textu
RUT5F    push hl             ;ulož adresu tisknutého znaku
         push de             ;ulož adresu pro tisk do obrazovky
RUT5A    ld    a,(hl)        ;vyzvedni kód znaku,
         add  a,a             ;a spočítej
         ld    l,a           ;adresu
         ld    h,15          ;znakové
         add  hl,hl          ;předlohy
         add  hl,hl          ;do registru HL
         ld    b,8           ;znak má celkem osm bytů,
RUT5B    ld    a,(hl)        ;přenes
         ld    (de),a        ;postupně
         inc  hl             ;celý
         inc  d               ;jeden
         djnz RUT5B          ;znak,
         pop  hl             ;dej do HL adresu v obrazovce
         push hl             ;a opět ji vrať na zásobník
RUT5I    xor  a              ;testuj
         in   a,(254)        ;stisk
         cpl                ;libovolně
         and  31             ;klávesy
         jr   nz,RUT5I       ;a v kladném případě čekej
         ld    e,8           ;proved osmkrát posun řádku doprava,
RUT5E    ld    c,8
         push hl
RUT5D    push hl
         ld    b,32
         xor  a
RUT5C    rl  (hl)
         dec  l
         djnz RUT5C
         pop  hl
         inc  h
         dec  c
         jr   nz,RUT5D
         pop  hl
         ld    a,e
         and  3
         jr   nz,RUT5H
         halt                ;po čtyřech posunech počkej na přerušení
RUT5H    dec  e
         jr   nz,RUT5E
         halt                ;po skončení posunu počkej na přerušení
         pop  de             ;obnov adresu pro tisk do obrazovky

```

```

pop hl ;obnov adresu znaku
bit 7,(hl) ;testuj, zda nejde o invertovaný znak
inc hl ;posuň ukazatel,
jr z,RUT5F ;když není konec textu, tiskni dál
pop hl ;obnov adresu počátku textu
jr RUT5G ;a skoč na začátek rutiny

TEXT1 defm "Toto je text " ;text, který
defm "1, tiskne ho " ;tiskne pátá
defm "rutina " ;úloha
defm 'RUTINA5 '

TEXT2 defm "Toto je text " ;text, který
defm "2, tiskne ho " ;tiskne šestá
defm "taky rutina " ;úloha
defm 'RUTINA5 '

STACKS defs 100 ;místo pro zásobník první úlohy
STACK1 defs 100 ;místo pro zásobník druhé úlohy
STACK2 defs 100 ;místo pro zásobník třetí úlohy
STACK3 defs 100 ;místo pro zásobník čtvrté úlohy
STACK4 defs 100 ;místo pro zásobník páté úlohy
STACK5 defs 100 ;místo pro zásobník šesté úlohy
STACK6 ;

```

Na začátku programu se vytvoří tabulka přerušení a připraví vše pro přerušovací mód dva. Potom se do našeho systému začlení celkem šest úloh - tady je zajímavě, že úlohy 5 a 6 mají společný kód. Při začleňování úloh do systému se připraví zásobník každé úlohy do takového stavu, v jakém by byl po přerušení. Potom se začnou provádět jednotlivé úlohy tak, že se pravidelně střídají, každá dostane vyhrazen čas mezi dvěma přerušeními. V každém přerušení se také testuje BREAK a pokud je stíšen, program se ukončí.

Každý proces (úloha) je zastupován adresou, na kterou ukazuje zásobník po přijetí přerušení, obsahy jednotlivých registrů má každý proces v době kdy nepracuje uloženy na svém zásobníku. Důležité je, aby si procesy vzájemně nepřepisovaly zásobníky a kódy.

Pokud použijete jeden kód pro více procesů, nesmí se tento kód sám modifikovat ani nesmí brát data z pevných adres - může mít data jen v registrech nebo na vyhrazeném místě a přistupovat k nim pouze přes nějaký registr (například pomocí IX). Při vložení procesu do systému mu můžete předat inicializační hodnoty pomocí alternativních registrů a pomocí IX - pokud chcete vložit dat více, můžete je zapsat někam do paměti a ukázat na ně pomocí nějakého registru (nejlépe IX). Procesy tedy mohou sdílet kód ale nikoliv data!

Pokud chcete, aby spolu procesy nějak komunikovaly - třeba že jeden bude dělat nějaké činnosti v závislosti na tom, co mu nějaký jiný vybere, musíte si zajistit předávání zpráv mezi procesy - nejlépe pomocí vyhrazeného místa v paměti, kam budou přistupovat právě jen ty procesy, které mají. Dejte si však pozor, aby procesy na sebe nečekaly vzájemně, tedy první by čekal, až něco udělá druhý, a ten by zase čekal na nějaký signál prvního procesu, v tomto případě by se nedělo nic - takovému stavu (kdy procesy čekají vzájemně) se říká uváznutí (deadlock). Něco podobného se Vám určitě alespoň jednou povedlo když se váš program vrátil do BASICu se zakázaným přerušením - čekal pak na signál, o stisku klávesy jenže klávesnice se netestovala a tak jej dostat nemohl.

Na větších počítačích (jak rozměrově, tak cenově i počtem bitů) se multitasking často používá - například pro obsluhu periférií (na PC tisk na pozadí) nebo proto, aby na jednom počítači mohlo pracovat více lidí najednou - tam má ovšem každý svůj terminál (klávesnici a monitor). Výhoda multitaskingu spočívá v tom, že se toho udělá víc, vypadá to absurdně ale je to tak. Počítač je totiž obvykle rychlejší než člověk a tak obvykle dost velkou část času čeká na uživatele (třeba když přišete text do textového editoru, čeká počítač na každé další stisknutí klávesy) nebo na periférii (když se tiskne, není tiskárna tak rychlá jako počítač a počítač musí čekat, než může posílat další znaky). Stačilo by, kdyby se čekání věnovala jen malá chvilka (testovat klávesnici jen několikrát za vteřinu a ne stále, stejně tak připravenost tiskárny se nemusí pořád testovat) a ve zbytku se dělala nějaká další užitečná činnost. Na Spectru byste však třeba s tiskem na pozadí asi moc nepochodili (mohli byste sice v době tisku psát další text, jenže byste nesměli měnit ten, který se tiskne), Spectrum má totiž moc málo paměti a nemá obvykle ani disketu ani pevný disk.

Něco, co by se dalo nazvat multitaskingem se na Spectru používá pro testování klávesnice, hraní hudby (Manic Miner a některé další hry) a vytváření zvuků (většina těch lepších) při vlastním hraní. Některé efekty v borderu (třeba to, že u hry Academy je horní část obrazovky modrá, pak je bílá linka a zbytek je černý) se také dělají v přerušení. Pokud má hra nějaké hodiny, pak jsou také skoro vždy obsluhovány v přerušení, také některé rolující texty ve hrách při hraní jsou kresleny v přerušení, František Fuka (tak jsem ho sem přeci jen propašoval) ve hře **TETRIS 2** určitě používá pro oba hráče něco podobného.

Tak jak je náš příklad napsán, je to nejobecnější možná varianta na Spectru, přece však klade na jednotlivé podprogramy některá omezení - nesmíte měnit mód přerušení (to není tak úplně pravda, rozmyslete si to), můžete je na čas zakázat (úloha si tak присvojí veškerý čas procesoru - třeba při ukládání na kazetu nebo disketu je to však nezbytné), ale měli by jste je opět po nějaké době povolit, aby si mohly škrtnout i další úlohy. Dejte si pozor na přepsání kódu nebo dat jiného procesu (na lepších počítačích se to dělá hardwarově, tady to však nejde), skoro jistě by se celý systém zhroutil.

V příkladu se všechny procesy střídají pravidelně a není tam žádná možnost, jak nějaký proces zrušit (mohl by to třeba provést dokonce sám, případně nějaký hodný z jeho kamarádů by to udělal za něj - mimochodem funkce, která toto provádí v UNIXu se jmenuje KILL). Pokud budete chtít, můžete si tam něco podobného dopsat sami. Můžete také program upravit tak, aby prováděl jen jeden vybraný proces a pomocí nějaké klávesy bylo možno mezi procesy přepínat.

Další možnosti práce s procesy jsou, že každému procesu přiřadíte také informaci o tom, jak velký má přiděl čas procesoru - například jeden dostane 8 padesátin a druhý jen jednu, pak se tedy bude ten první volat 8 a teprve pak se zavolá ten druhý a tak stále dokola.

Ještě další varianta by byla, kdyby procesy samy mohly vytvářet (tedy samozřejmě nemyslím, že by je samy programovaly, pouze by vybíraly ty hotové a zadávaly jim potřebné parametry - například, chcete v programu smazat obrazovku, nic snazšího, už z dřívějšíka máte napsaný proces, který to umí, spustíte jej a dál se věnujete nějaké své činnosti, když „mazací“ proces skončí, sám se smaže - pozor, to si ale musíte nejprve dopsat) a spouštět jiné procesy (to je parádní guláš...), v praxi se to sice moc nepoužívá ale až se dostanete (pokud jste tam už nebyli) na MFF UK (obor Informatika), jako byste to našli. Některé zkoušky (konkrétně Operační systém UNIX a jazyk C) mají i část, kdy dostanete nějaké dva procesy, které spolu vzájemně komunikují, vy máte určit, co to vlastně bude dohromady dělat, vy na

něco přijdete (někdy), nadnesete to a pak se dozvíte, že to sice také ale ještě navíc tohle a muselo by se to také takhle opravit - prostě paráda. Abych to jen nezlehčoval, určitě se takhle dají dělat zajímavé grafické efekty, chce to však dobrý nápad!

O multitaskingu bychom si mohli povídat ještě dlouho, pokud vás to zajímá, pak existuje spousta literatury (nejmenuje se sice zrovna "**Multitasking snadno a rychle**" nebo "**Vybrané kapitoly z multitaskingu**", ale v každé příručce o UNIXu něco najdete).

# Vnitřní, pracovní obrazovka

Tak vás vítám v poslední kapitole, uteklo to? Ukážeme si tu, co dělat, když chceme kreslit něco, co se prostě v jednom přerušení stihnout nedá. Nejčastěji se používá u 3D her a simulátorů (Elite, Tau Ceti, Mikronaut, Driller....., Tomahawk, Fighter Bomber), ve kterých trvá vykreslení jednoho pohledu mnohem déle než jedno přerušení.

Práce s vnitřní obrazovkou se příliš neliší od práce s obrazovkou obvyklou, rozdílů jsou v tom orientaci na ní - vypočet adresy bodu a bitu, posun o řádek nahoru nebo dolů, atd. Uspořádání vnitřní obrazovky si můžete vymyslet sami, má to tu výhodu, že je můžete přizpůsobit požadavkům řešeného problému - docela určitě změníte rozložení mikrořádků a budete je ukládat v přirozeném pořadí. Vnitřní obrazovka může mít (a většinou má) menší rozměry, než má obvyklá obrazovka. Atributy, pokud nějaké použijete, obvykle umístíte až za pixely, není však vyloučeno, že jimi budete pixelové řádky prokládat. Můžete tedy použít naprosto všechno, co jsme si ukázali pro obrazovku normální - upravíte jen některé detaily, algoritmy zůstanou beze změny.

To, co jsme si zatím neukázali, je způsob, jak rychle přenést vnitřní obrazovku na obrazovku skutečnou a také způsob, jak se dají všechny byty obrazovky velice rychle vyplnit libovolnou hodnotou (nejrychlejší CLS). Ukážeme si zase příklad:

```

ent    $                ;zde se spustíme

SPRITES equ 64000      ;sprity jsou na 64000
WIDTH  equ 32          ;sprite je široký 32 bodů
HEIGHT equ 26          ;a vysoký 26 bodů
SPRLEN equ WIDTH*HEIGHT*2 ;délka spritu (maska+předloha)
SPREND equ 4*SPRLEN+SPRITES ;adresa konce spritů

START  im 1            ;přerušení v módu 1
       ei              ;a povol jej

       ld hl,22528     ;nastavíme
       ld de,23529     ;si

```

```

ld bc,767 ;černý papír
ld (hl),7 ;a bílý
ldir ;inkoust

MAIN ld h,%1101101 ;do registru H dej číslo,
ld l,%1001001 ;do registru L dej číslo - pozadí
MAIN2 push hl ;obě čísla ulož, budeme je rotovat
call CLRSCR ;vyčisti těmito čísly obrazovku
ld hl,SCREEN+1376 ;vyplň
ld bc,1024-256-128 ;spodní
FILL ld (hl),255 ;část
inc hl ;vnitřní
dec bc ;obrazovky
ld a,b ;číslem
or c ;255,
jr nz,FILL ;kreslíme podlahu

ld ix,TEXT ;nakreslíme
ld de,SCREEN+512+8 ;do vnitřní
ld c,TEXTLEN ;obrazovky
TT ld a,(ix+0) ;nějaký
inc ix ;nápís
add a,a
ld l,a
ld h,15
add hl,hl
add hl,hl
push de
TT2 ld b,16
ld a,(hl)
ld (de),a
inc hl
bit 0,b
jr nz,TT3
dec hl
TT3 ld a,e
add a,32
ld e,a
ld a,d
adc a,0
ld d,a
djnz TT2
pop de
inc de
dec c
jr nz,TT

SELSPR ld de,SPRITES ;do DE dej adresu předloh
ld hl,SCREEN+778 ;do HL adresu do vnitřní obrazovky
RELPOS ld a,0 ;a přičti k ní
add a,1 ;relativní adresu
ld l,a ;kam se mají
ld a,h ;sprity
adc a,0 ;vykreslit
ld h,a

call DRAWSPR ;vykresli sprite
dec hl ;posuň tiskovou pozici

```

```

dec hl ;o 16 bodů doleva
ld bc,64 ;a o dva body
add hl,bc ;dolů,
call DRAWSPR ;vykresli sprite
dec hl ;posuň tiskovou pozici
dec hl ;o 16 bodů doleva
ld bc,64 ;a o dva body
add hl,bc ;dolů,
call DRAWSPR ;vykresli sprite
dec hl ;posuň tiskovou pozici
dec hl ;o 16 bodů doleva
ld bc,64 ;a o dva body
add hl,bc ;dolů,
call DRAWSPR ;vykresli sprite
dec hl ;posuň tiskovou pozici
dec hl ;o 16 bodů doleva
ld bc,64 ;a o dva body
add hl,bc ;dolů,
dec hl ;posuň tiskovou pozici
dec hl ;o 16 bodů doleva
ld bc,64 ;a o dva body
add hl,bc ;dolů,
call DRAWSPR ;vykresli sprite

PAUSE ld a,0 ;zde
inc a ;se
cp 4 ;ovlivňuje
jr c,MAIN4 ;rychlost,
xor a ;jakou
MAIN4 ld (PAUSE+1),a ;sprity
jr nz,MAIN5 ;chodí

ex de,hl ;přehod adresy (s HL se lépe pracuje)
ld ix,RELPOS ;zde se
inc (ix+1) ;zvyšuje X-ová souřadnice spritů
ld a,(ix+1) ;a pokud
cp 31 ;přeleze
jr c,MAIN6 ;číslo 31,
ld (ix+1),0 ;zapiš tam nulu
MAIN6 ld de,SPRELN ;přičti
add hl,de ;k adrese
ld de,SPREND ;spritu
or a ;délku jednoho
sbc hl,de ;spritu a testuj
add hl,de ;zda jsi se
jr nz,MAIN3 ;nedostal už
ld hl,SPRITES ;za sprity, pokud ano, nastav začátek
MAIN3 ld (SELSPR+1),hl ;a vše zapiš zpět
ex de,hl ;vrať adresy

MAIN5 halt ;počkej na přerušění,
ld a,1 ;nastav
out (254),a ;modrý border
ld de,20480 ;a vykresli vnitřní obrazovku
call MOVESCR ;do spodní třetiny té skutečné
YPOS ld a,0 ;dále vykresli vnitřní obrazovku
ADD1 add a,1 ;na vybrané
and 63 ;místo

```

```

        ld  (YPOS+1),a      ;v horních dvou
        push af            ;třetinách
        jr  nz,MAIN8A     ;zajištění posunování je celkem
        ld  a,1           ;jednoduché a tak si ho rozeberte sami
        ld  (ADD1+1),a
MAIN8A  cp  63
        jr  nz,MAIN8
        ld  a,-1
MAIN8   ld  (ADD1+1),a
        ld  c,0
        pop af
        call #22B0        ;spočítej adresu do HL
        ex  de,hl        ;a přehod ji do DE
        call MOVESCR     ;vykresli obrazovku
        ld  a,0          ;nastav
        out (254),a      ;černý border
        pop hl           ;obnov byty s pozadím
        rlc h            ;a proved potřebné
        rrc l            ;rotace
        xor a            ;testuj
        in  a,(254)      ;stisk
        cpl              ;libovolné
        and 31           ;klávesy
        jp  z,MAIN2      ;a případně pokračuj v kreslení
        ret              ;vrať se

TEXT    defm ">> PROXIMA " ;ne aby vás napadlo
        defm "software <<" ;tento text nějak měnit!
TEXTLEN equ $-TEXT       ;délka textu do TEXTLEN

DRAWSPR push hl          ;ulož adresu spritu v pracovní obrazovce
        push de          ;ulož adresu předlohy spritu
        ex  de,hl        ;přehod obě adresy
        push hl          ;ulož adresu spritu
        exx              ;přepni na alternativní registry
        pop hl           ;spočítej adresu, kde
        ld  bc,SPRLBN/2  ;se nachází
        add hl,bc        ;maska spritu do HL'
        exx              ;vrať původní registry

        ld  c,HEIGHT     ;do C dej výšku spritu
DRS1    ld  b,WIDTH      ;do B dej počet bytů na řádku předlohy
        push de          ;uschovej adresu do vnitřní obrazovky
DRS2    ld  a,(de)       ;vzvedni původní byte,
        exx              ;přepni na alternativní registry,
        and (hl)         ;ponech bity podle masky
        inc hl           ;a posuň ukazatel na masku
        exx              ;vrať původní registry
        or  (hl)         ;přidej byte předlohy
        ld  (de),a       ;a zapiš vše do vnitřní obrazovky,
        inc hl           ;posuň ukazatel na předlohu
        inc de           ;i ukazatel do vnitřní obrazovky
        djnz DRS2       ;opakuj pro všechny byty řádku
        pop de           ;obnov adresu počátku mikrořádku
        ld  a,e          ;nyní posuneme
        add a,32         ;adresu ve vnitřní obrazovce tím,
        ld  e,a          ;že k ní přičteme

```

```

ld    a,d          ;číslo 32, zde vidíte, jak se
adc   a,0          ;zjednoduší DOWNDE v případě,
ld    d,a          ;že používáte vnitřní obrazovku,
dec   c            ;zmenší počítadlo řádku spritu
jr    nz,DRS1     ;a dokud není nulové, pokračuj
pop   de           ;obnov obě
pop   hl           ;adresy
ret                    ;a vrať se

DOWNDE    ....    ;DOWNHL upravený pro registr DE

MOVESCR  ld    b,64          ;budeme přenášet celkem 64 mikrořádků
ld    hl,SCREEN            ;z paměti na adresu udanou registrem DE
MOVESCR2 push de            ;uschovej počátek mikrořádku v obrazovce
ld    c,255                ;do C dej takové číslo, aby se B při
ldi                    ;instrukcích LDI nezměnilo, první
ldi                    ;druhá
ldi                    ;třetí
ldi                    ;čtvrtá
ldi                    ;pátá
ldi                    ;šestá
ldi                    ;sedmá
ldi                    ;osmá
ldi                    ;devátá
ldi                    ;desátá
ldi                    ;jedenáctá
ldi                    ;dvanáctá
ldi                    ;třináctá
ldi                    ;čtrnáctá
ldi                    ;patnáctá
ldi                    ;šestnáctá
ldi                    ;sedmnáctá
ldi                    ;osmnáctá
ldi                    ;devatenáctá
ldi                    ;dvacátá
ldi                    ;dvacátá první
ldi                    ;dvacátá druhá
ldi                    ;dvacátá třetí
ldi                    ;dvacátá čtvrtá
ldi                    ;dvacátá pátá
ldi                    ;dvacátá šestá
ldi                    ;dvacátá sedmá
ldi                    ;dvacátá osmá
ldi                    ;dvacátá devátá
ldi                    ;třicátá
ldi                    ;třicátá první
ldi                    ;třicátá druhá
pop   de            ;obnov adresu mikrořádku v obrazovce
call  DOWNDE       ;posuň se o mikrořádek dolů
djnz  MOVESCR2     ;opakuji pro všechny mikrořádky
ret                    ;a vrať se

CLRSCR   ld    (SPSTORE+1),sp ;uschovej hodnotu registru SP
di                    ;zákaz přerušení protože budeš pomoci
ld    sp,ENDSCR     ;zásobníku čistit obrazovku

```



```

                ld    b,0                ;cyklus proběhneš celkem 256 krát
CLRSCR2        push hl                 ;a v cyklu uložíš 2
                push hl                 ;4
                push hl                 ;6
                push hl                 ;8 bytů, tedy celkem 8*256=2048 bytů
                djnz CLRSCR2           ;opakuje
SPSTORE        ld    sp,0              ;sem se zapíše původní hodnota SP
                ei                       ;povol přerušení
                ret                      ;a vrať se zpátky

SCREEN         defs 32*61              ;celkem 2048 bytů pro vnitřní obrazovku
ENDSCR

```

Při psaní vám asi došlo, že budete potřebovat sprity, jsou to sprity z prvního příkladu v kapitole spritová grafika (tam naleznete podrobnosti).

Přesun obrazovky se dá ještě zrychlit tím, že si nebude adresy v obrazovce počítat ale budete je číst z nějaké tabulky. Opakované použití instrukcí **ldi** místo instrukce **ldir** je výhodné pro vyšší rychlost, kterou takto dosáhnete. Snažte se zajistit, aby se vnitřní obrazovka vykreslila v době, kdy není příslušná část obrazovky vykreslována - to je důležité hlavně proto, aby se obraz netrhal - pokud netušíte, jak to vypadá, přehodte vykreslení ze třetí třetiny do první a druhé vykreslení vnitřní obrazovky (to, co se hýbá) dočasně vypusťte. Ze zkušeností vím, že se nedá stihnout vykreslit úplně celá obrazovka (tak aby nedocházelo k trhání, pokud to budete chtít udělat, nečekejte na přerušení - chyba se sice projeví ale pokaždé na jiném místě a nebude tak příliš patrná).

Já jsem vnitřní obrazovku použil takřka ve všech svých výtvorech (ORFEUS, BAD DREAM, MAH JONGG a svým způsobem také DESKTOP - obrazovku tvoří jeden řádek textu, ten se pak už jen zobrazí).

Někdy vnitřní obrazovku použít musíte - typický příklad je program SCREEN TOP (ten jsem nepsal já ale Jiří Vondráček blahé paměti, ode mne je tam jen grafický editor z programu WLEZLEY 7 a ještě mírně upravený), který pracuje s obrazovkou o rozměrech 512x384 bodů, tu jinak než vnitřní neuděláte. Podobně je na tom také program WIRE STUDIO, o jiných nevíme.

Další zmínka bude věnována některým námětům na zrychlení vykreslení vnitřní obrazovky. V některých případech není třeba kreslit obrazovku celou, stačí, když vykreslíte a zobrazíte jen ty části, které se od posledního vykreslení změnilly. Ovšem zde si dejte pozor, aby vykreslení nebylo rychlejší než testování a zjišťování, co se má vlastně vykreslit, také kreslení pouze částí obrazu může být znatelně pomalejší.

Pokud netušíte, jak se kreslí hry, kde se sprity navzájem překrývají a navíc jsou případně překryty ještě kulisami (tak si nazveme to, co bývá ještě blíže než sprity a nepohybuje se - alespoň ne všemi směry), pak je to zcela jednoduché. Kreslit se začíná od toho, co je úplně vzadu a končí se tím, co je úplně vpředu - neboli napřed nakreslíte pozadí, pak sprity a nakonec kulisy (typické pro některé střílečky - ZYNAPS nebo DAN DARE)

nebo také sprity a kulisy současně (všechny „rohovky“ - třeba BATMAN, KNIGHT LORE či ALIEN 8). Zcela v tomto duchu se kreslí figurky v našem příkladě, nemáme tam však žádné kulisy, budete-li nějaké chtít, stačí, když přemístíte vykreslení nápisu až za vykreslení spritů - část zdrojového textu od instrukce **ld ix,TEXT** až po **jr nz,TT** přesuňte před návěští **MAIN5** a toto návěští ještě přesuňte k instrukci **ld ix,TEXT**, nebo ho nejprve nechte kde je a podívejte se, co to udělá, pak ho dejte tam, kam patří.

Příklad je symbolický - znázorňuje totiž davy programátorů, kteří po přečtení druhého dílu mé knihy napsali nějakou kvalitní hru a jdou nám ji nabídnout k distribuci. Neváhejte proto a až něco vytvoříte, přijďte (napište).

# Pomluva

Tentokrát jsem si název pro poslední kapitolku, ve které vás obšťastním nějakými radami, vymyslel sám. Chtěl bych zde uvést na pravou míru jistou dezinformaci, která se dostala na toto místo do prvního dílu: *Není pravda, že pan Petr Koudelka se jmenuje Petr Koudelka, nýbrž je pravda, že pan Petr Koudelku se jmenuje skoro úplně jinak* (to já jen aby vás to příliš nemátlo). Nyní jsem se slušně omluvil a veškeré invektivy, které si ke mě pan Petr Koudelka dovolil v manuálu k programu TOOLS 40 si vyprošuji!

Od minula nám přibylo několik nových reklam a tak se jich budu držet.

1) Už jste zkusili BLEND-A-MED? Po použití si můžete klidně několik hodin máchat zuby v roztoku, který připomíná kyselé prostředí ve vašich ústech aniž by došlo k jejich změknutí, BLEND-A-MED totiž váže vápník a tím posiluje vaši sklerózu (nebo snad sklovinu, už nevím, viděl jsem to naposledy před dvěma hodinami).

2) Chlapečci se počůrávají dopředu nahoru a holčičky dolů doprostřed, zkuste si to někdy, je to úžasné! (kam na to ti lidé jenom chodí?!)

3) Vypadá to na dočasnou ztrátu inspirace.

4) Vypadá to na trvalejší ztrátu inspirace.

5) Vypadá to na chronickou ztrátu inspirace,.... . . . . .

# *Obsah*

Úvodem	1
Hýbeme obrazem	3
Volba ovládání	22
Šipka	29
Jemná grafika	34
Proporční tisk	53
Plníme obrazovku	64
Spritová grafika	77
Přerušení	88
Multitasking	92
Vnitřní, pracovní obrazovka	99
Pomluva	105
Obsah	106





---

<b>Název knihy:</b>	<b>Assembler a ZX Spectrum II</b>
<b>Autor:</b>	<b>Tomáš Vilím</b>
<b>Vydavatel:</b>	<b>PROXIMA - software nové dimenze post box 24, pošta 2 Ústí nad Labem 400 21</b>
<b>Vyšlo:</b>	<b>v říjnu 1992</b>
<b>Vydání:</b>	<b>první</b>

---