Develop a program in cartridge ROM

This page was last modified 11:19, 5 March 2022 by Gdx. Based on work by Mars2000you and Grauw and others.

Category: **Programming**

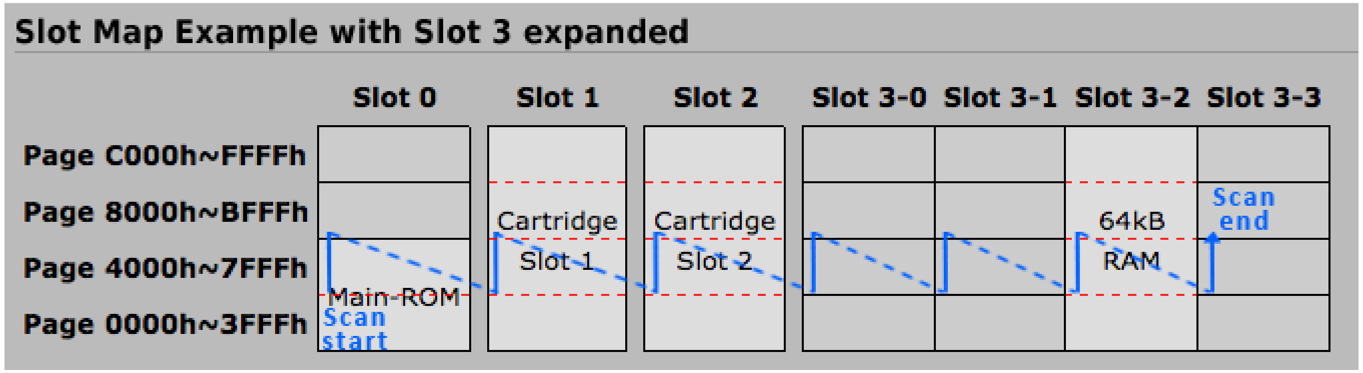
Contents

Develop a program in cartridge ROM

MSX cartridge ROM can take a multitude of forms depending on the needs. Only the programming aspect will be explained here.

A ROM needs a header to be auto-executed by the system when the MSX is initialized.

After **finding the RAM** and initializing the **system variables**, the MSX system looks for the ROM headers in all the slots on the memory pages 4000h-7FFFh and 8000h-BFFFh. The search is done in ascending order. When a primary Slot is expanded, the search is done in the corresponding secondary Slots before going to the next Primary Slot.



When the system finds a header, it selects the ROM slot only on the memory page corresponding to the address specified in INIT then, runs the program in ROM at the same address by an inter-slot call.

The ROM Header

A header consists of 16 bytes and should be placed at 4000h or 8000h as below.

Header	Name	Use
+0	ID	Put these first two bytes at 041H and 042H ("AB") to indicate that it is an additional ROM.
+2	INIT	Address of the routine to call to initialize a work area or I/O ports, or run a game, etc. The system calls the address from INIT of each ROM header during the MSX initialisation **in that order**.

+4	STATEMENT	Runtime address of a program whose purpose is to add instructions to the MSX-Basic using CALL. STATEMENT is called by CALL instructions. It is ignored when 0000h. It is not called at MSX start up.
+6	DEVICE	Execution address of a program used to control a device built into the cartridge. For example, a disk interface. It is not called at MSX start up.
+8	TEXT	Pointer of the tokenizen Basic program contained in ROM. TEXT must be always an address more than 8000h and be specified in the header of the page 8000h-BFFFh. **In other cases, it must always be 0000h** under penalty of causing crash or bug.
+10	Reserved	6 bytes reserved for future updates.

Note: Unused addresses and reserved bytes have to set to 0000h.

INIT

This is the first address taken into account. When this address is greater than 0000h, the system selects the ROM slot on the memory slot corresponding to the address and executes the program in ROM at the same address. At the time of program execution, the C register contains the slot number of the ROM in the form F000SSPP. The routine must end with a RET. All registers can be modified by routine except SP. In place of an initialization routine, the ROM may very well contain a game.

Trick if your ROM has a size of 32K (4000h-BFFFh):

1. When INIT is an address between 4010h-7FFFh, you can select the second part (8000h-BFFFh) by running the routine below at start.

```
        call    RSLREG  ; Read the primary slots register
        rrca
        rrca
        and     3
        ld      c,a
        ld      b,0
        ld      hl,EXPTBL       ; HL = Address of the secondary slot flags table
        add     hl,bc
        ld      a,(hl)
        and     80h     ; Keep the bit 7 (secondary slot flag)
        or      c
        ld      c,a
        inc     hl
        inc    hl
        inc    hl
        inc    hl     ; HL =  Address of the secondary slot register in the secondary slot register table
        ld     a,(hl)
        and    0Ch
        or     c
        ld     h,080h
        call   ENASLT ; Select the ROM on page 8000h-BFFFh
```

Or this method that is shorter (see note below).

```
        ld     a,c
        ld     h,080h ; The ENASLT routine does not take into account the register L
        call   ENASLT        ; Select the ROM on page 8000h-BFFFh
```

2. When INIT is an address between 8010h-BFFFh, you can select the first part (4000h-7FFFh) by running the routine below at start.

```
        call   RSLREG ; Read the primary slots register
        rrca
        rrca
        rrca
        rrca
        and    3      ;Keep bits corresponding to the page 4000h-7FFFh
        ld     c,a
        ld     b,0
        ld     hl,EXPTBL     ; HL = Address of the secondary slot flags table
        add    hl,bc
        ld     a,(hl)
        and    80h    ; Keep the bit 7 (secondary slot flag)
        or     c
        ld     c,a
        inc    hl
        inc    hl
        inc    hl
        inc    hl     ; HL =  Address of the secondary slot register in the secondary slot register table
        ld     a,(hl)
        and    0Ch
        or     c
        ld     h,040h ; The ENASLT routine does not take into account the register L
        call   ENASLT        ; Select the ROM on page 4000h-7FFFh
```

Or this method that is simpler (see note below).

```
        ld      a,c
        ld      h,040h  ; The ENASLT routine does not take into account the register L
        call    ENASLT  ; Select the ROM on page 4000h-7FFFh
```

Note: For both examples, the first method is considered as the most standard. I give the alternative method because I remember seeing it in a old documentation and it seems actually 100% reliable. I tested it on all configurations emulated by blueMSX. Given that I can no longer find this doc to confirm that this register as well as HL can be used, and as it is not documented in the MSX-Data pack, the main technical source today, please use this method only if you are running out of memory in your ROM in the meantime, waiting for a more established confirmation.

STATEMENT

Processing program of the instruction must reside on the page 4000h-7FFFh.

A instruction called by CALL must have the following format:

```
CALL <Instruction name> [(variable[, variable][,...])]
```

Name of the instruction can be up to 15 characters. When the BASIC interpreter finds the instruction CALL, it copies its name into the PROCNM work area (0FD89h) and then searches the slots in ascending order for a STATEMENT address greater than 0000h to transmit the control for that instruction. At this point, the double register HL contains the address of the parameter that follows the name of the statement in the listing. The instruction can be processed. At the output, HL must indicate the next instruction to be processed and the Carry flag must indicate if there has been an error.

Here is an example of a procedure with CALL NAME(0,0) followed by A=0:

1. The listing therefore contains "CALL NAME (0,0): A=0".
   HL points to the character "(".
   Carry flag = 1.
   PROCNM = "N","A","M","E",00h (00h can be also 3Ah)
2. Processing of the instruction by the routine at the address specified at STATEMENT.
   If the name does not match then leave HL as is and put Carry at 1 before handing over to the interpreter (by a RET).
   If name matches, execute the statement routine and its parameters then, point the next statement with HL and set Carry to 0 if there is no error in the parameters.

3. End of treatment.
   HL must point the variable A of A=0.
   Give back to the interpreter (by a RET).

Note: Avoid giving an already existing name to your instruction because according to the position of the ROM in the slots, it could not be taken into account or even cause an error because of the parameters.

DEVICE

This address must be between 4000h-7FFFh. The system can control up to 4 devices per cartridge. The device name must be 15 characters maximum. When the BASIC interpreter encounters a device name, it copies it to the PROCNM work area (0FD89h), then sets the register A to 255, and searches the slots in ascending order for a DEVICE address greater than 0000h to transmit control of the corresponding device.

Here is an example of a procedure with OPEN "NAME:":

1. The Basic listing contains OPEN "NAME:"
   A = Instruction number (see table below)
   Carry flag = 1
   PROCNM = "N","A","M","E",00h (00h can be also 3Ah)
2. If the name does not match then set register A to 255 and Carry to 1 before returning the hand to the interpreter (by a RET).
   If name does match, run the control routine then, point the next statement with HL and set Carry to 0 if there is no error in the settings.
3. End of treatment.
   A = Device identifier (0-3)
   Carry flag = 0 if no error
   Give back to the interpreter (by a RET).

Instruction numbers:

Register A	Instruction
0	OPEN
2	CLOSE
4	Random access

6	Sequential output
8	Sequential entry
10	LOC function
12	LOF function
14	EOF function
16	Function FPOS
18	Backup character

TEXT

This TEXT pointer indicates the beginning of the Basic program to be executed automatically at MSX start. First byte of the program is always zero. The program can not have a maximum size of 16K and should be between 8000h-BFFFh. It must also be a tokenized format and not an ASCII text format. In addition, the addresses corresponding to the program line numbers must indicate the actual destination addresses in the program.

Method to put a Basic program in ROM:

1. A Basic program starts at 08000h on a 32k MSX or more by default. It must be shifted at least to desired address (08012h for this example) to insert the header of the ROM. To do this, enter the following line under Basic:

   ```
   POKE &HF676,&H13: POKE &HF677,&H80: POKE &H8012,0: NEW
   ```

2. Load the Basic program to ROM by entering the following instruction.

   LOAD"Name.BAS"
3. Then save the program by entering the following instruction.

   SAVE"Name2.BAS"
4. Put the 08012h address to TEXT in the header of the page 8000h-BFFFh, then replace the first byte (FFh) by 00h in the file "Name2.BAS" and copy its content to the ROM at 08012h.

Example in assembler:

```
        org 08000h

ROMheader:
        db      "AB"
        dw      0,0,0,08012h,0,0,0

        nop
        nop

BasicPRG:
        incbin "NAME2.BAS"     ; The first byte (FFh) must be previously replaced by 00h

        ds      4000h - ($ - ROMheader),0
```

Create a ROM without mapper

In the chapter **The Rom Header** you can see that the ROM header can be placed to 4000h or 8000h, or even both. In addition, your program can start from almost any address since the system is making an inter-slot call to the address specified by INIT. The only constraints are the header and interrupts. Indeed, the system interrupt routine is at address 0038h. If you put 03000h to INIT, your ROM will need to have a replacement interrupt routine since it will be selected on page 0000h-3FFFh to be executed. The problem also occurs on page C000h-FFFFh because of Hooks and system variables. You need have a high mastery of system and hardware to choose these pages. Better to choose an address between 4000h and BFFFh and if necessary, use the other two pages to put data (text and graphics for example).

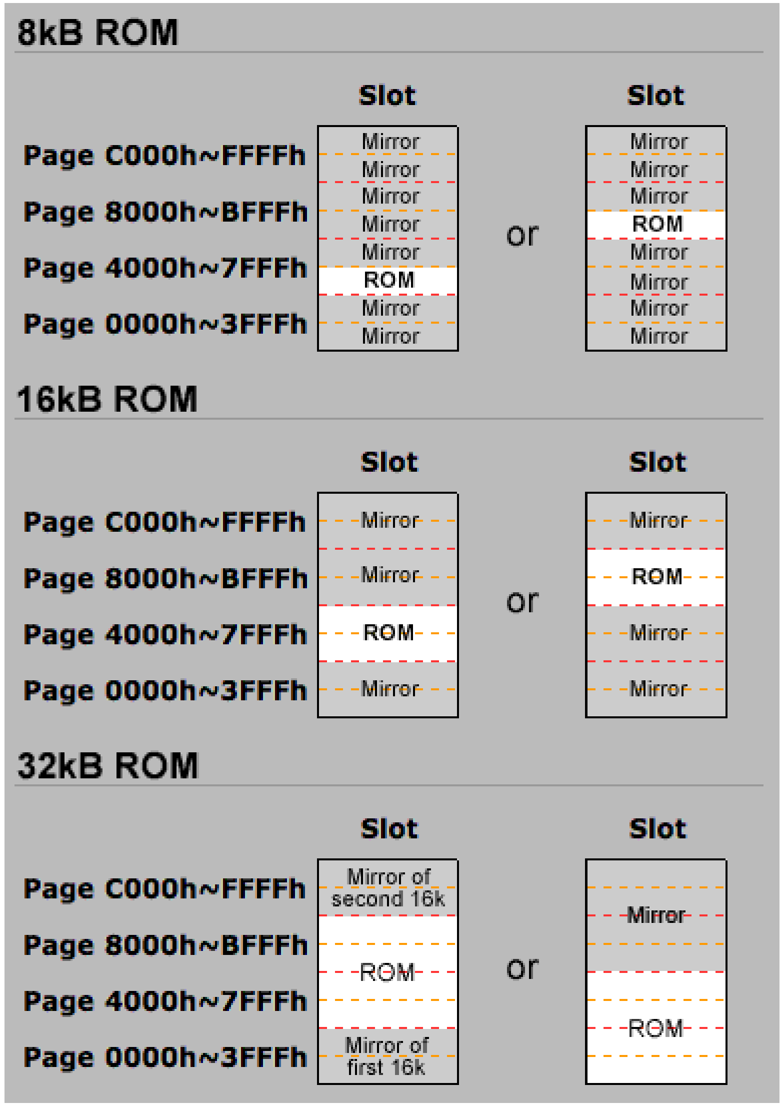
The size of a ROM without mapping can vary in theory from 1kB to 64kB. In practice, it is difficult to find ROMs of less than 16kB.

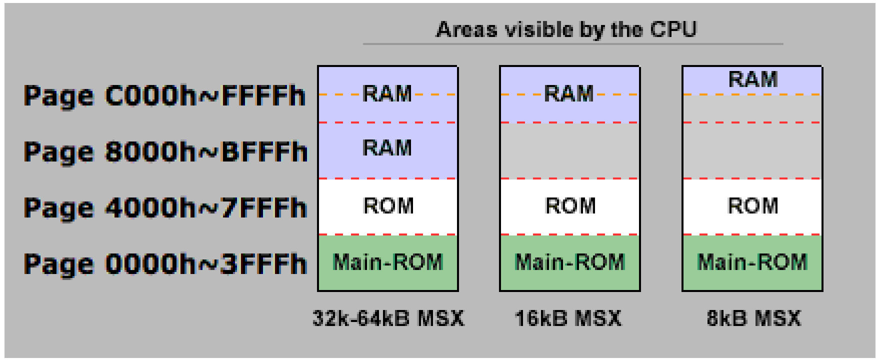
If the ROM is 48kB, the C000h-FFFFh page will contain undefined values (usually 0FFh).

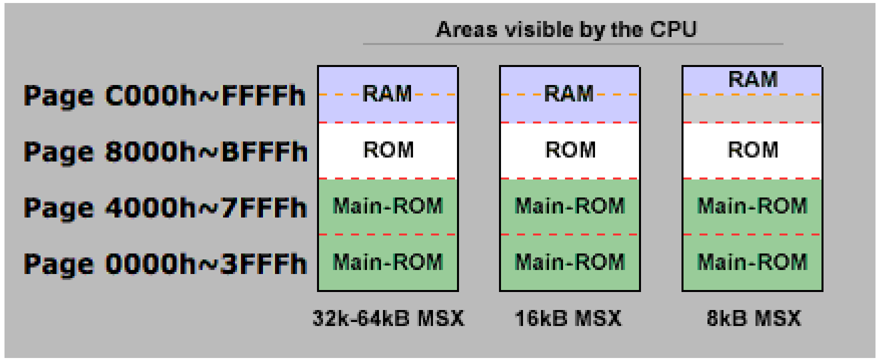
If the ROM is 32kB or less, depending on the hardware connections the unused parts will contain undefined values or mirrors as shown below.

You can use these mirrors to confuse people looking to disassemble your program. MegaROMs can have the same mirrors as a 32kb ROM since they are often connected the same way with an additional mapper. Also note that I once saw a MegaROM whose mapper is controlled by an EPM with the mirrors reversed.

When a Rom is executed on page 4000h-7FFFh, the CPU can see the half of the Main-ROM on the page 0000h-3FFFh and the available Main-RAM on the other pages.







When a Rom is executed on page 8000h-CFFFh, the CPU can see the Main-ROM on the pages 0000h-3FFFh and 4000h-7FFFh, and the available Main-RAM on the top page.

Bottom of the RAM that the CPU can see is indicated by BOTTOM (0FC48h) system variable.

Typical examples to make a 32kB ROM

Below is an example for a ROM that start from page 4000h-7FFFh.

```
LF:     equ     0Ah
CR:     equ     0Dh

CHPUT:  equ     00A2h   ; Address of character output routine of BIOS
ENASLT: equ     0024h
INIT32: equ     006Fh
RSLREG: equ     0138h

PageSize:       equ  4000h      ;  16kB

LINL32: equ     0F3AFh
EXPTBL: equ     0FCC1h          ;  Extended slot flags table (4 bytes)

        org  4000h

; ### ROM header ###
```

```
        db "AB"                 ; ID for auto-executable ROM
        dw INIT                 ; Main program execution address.
        dw 0            ; STATEMENT
        dw 0            ; DEVICE
        dw 0            ; TEXT
        dw 0,0,0        ; Reserved

INIT:   ; Program code entry point label

        ld      a,32
        ld      (LINL32),a      ; 32 columns
        call    INIT32          ; SCREEN 1

; Typical routine to select the ROM on page 8000h-BFFFh from page 4000h-7BFFFh

        call    RSLREG
        rrca
        rrca
        and     3       ;Keep bits corresponding to the page 4000h-7FFFh
        ld      c,a
        ld      b,0
        ld      hl,EXPTBL
        add     hl,bc
        ld      a,(hl)
        and     80h
        or      c
        ld      c,a
        inc     hl
        inc     hl
        inc     hl
        inc     hl
        ld      a,(hl)
        and     0Ch
        or      c
        ld      h,080h
        call    ENASLT          ; Select the ROM on page 8000h-BFFFh

        ld      hl,Page4000hTXT        ; Text pointer into HL
        call    Print           ; Call the routine Print below

        jp      08000h  ; Jump to above page.

Print:
        ld      a,(hl)          ; Load the byte from memory at address indicated by HL to A.
        and     a               ; Same as CP 0 but faster.
        ret     z               ; Back behind the call print if A = 0
        call    CHPUT           ; Call the routine to display a character.
        inc     hl              ; Increment the HL value.
        jr      Print           ; Relative jump to the address in the label Print.

; Message data
Page4000hTXT:                   ; Text pointer label
        db "Text from page 4000h-7FFFh",LF,CR,0         ; Zero indicates the end of text

; Padding with 255 to make a fixed page of 16K size
; (Alternatively, include macros.asm and use ALIGN 4000H)

        ds PageSize - ($ - 4000h),255           ; Fill the unused aera in page with 0FFh

; Begin of page 8000h-BFFFh

        ld      hl,Page8000hTXT         ; Text pointer
        call    Print           ; Call the routine Print

Finished:
        jr      Finished        ; Jump to itself endlessly.

Page8000hTXT:                   ; Text pointer label
        db "Text from page 8000h-BFFFh",0       ; Zero indicates the end of text.

        ds PageSize - ($ - 8000h),255           ; Fill the unused aera in page with 0FFh
```

Note: "**ds PageSize - ($ - 4000h),255**" is here just for the example. You can remove it and replace "**ds PageSize - ($ - 8000h),255**" at end by "**ds PageSize - ($ - 4000h),255**" to make your own ROM.

Below is an example for a ROM that start from page 8000h-BFFFh.

```
LF:     equ     0Ah
CR:     equ     0Dh

CHPUT:  equ     00A2h   ; Address of character output routine of BIOS
ENASLT: equ     0024h
INIT32: equ     006Fh
RSLREG: equ     0138h

PageSize:       equ 4000h       ; 16kB

LINL32: equ     0F3AFh
EXPTBL: equ     0FCC1h          ; Extended slot flags table (4 bytes)

        org 4000h
        ld      hl,Page4000hTXT         ; Text pointer into HL
        call    Print           ; Call the routine Print

Finished:
        jr      Finished        ; Jump to itself endlessly.

; Message data
Page4000hTXT:                   ; Text pointer label
        db "Text from page 4000h-7FFFh",0       ; Zero indicates the end of text

; Padding with 255 to make a fixed page of 16K size
; (Alternatively, include macros.asm and use ALIGN 4000H)

        ds PageSize - ($ - 4000h),255   ; Fill the unused aera in page with 0FFh

; Begin of page 8000h-BFFFh

; ### ROM header ###

        db "AB"                 ; ID for auto-executable ROM
        dw INIT                 ; Main program execution address.
        dw 0            ; STATEMENT
        dw 0            ; DEVICE
        dw 0            ; TEXT
        dw 0,0,0        ; Reserved

INIT:   ; Program code entry point label

        ld      a,32
        ld      (LINL32),a      ; 32 columns
        call    INIT32          ; SCREEN 1

; Typical routine to select the ROM on page 4000h-7FFFh from page 8000h-BFFFh

        call    RSLREG
        rrca
        rrca
        rrca
        rrca
        and     3       ;Keep bits corresponding to the page 8000h-BFFFh
        ld      c,a
        ld      b,0
        ld      hl,EXPTBL
        add     hl,bc
        ld      a,(hl)
        and     80h
        or      c
        ld      c,a
        inc     hl
        inc     hl
        inc     hl
        inc     hl
        ld      a,(hl)
        and     0Ch
        or      c
        ld      h,040h
        call    ENASLT          ; Select the ROM on page 4000h-7FFFh

        ld      hl,Page8000hTXT        ; Text pointer
        call    Print           ; Call the routine Print below

        jp      04000h  ; Jump to below page.

Print:
        ld      a,(hl)          ; Load the byte from memory at address indicated by HL to A.
        and     a               ; Same as CP 0 but faster.
        ret     z               ; Back behind the call print if A = 0
        call    CHPUT           ; Call the routine to display a character.
        inc     hl              ; Increment the HL value.
        jr      Print           ; Relative jump to the address in the label Print.

Page8000hTXT:                   ; Text pointer label
        db "Text from page 8000h-BFFFh",LF,CR,0      ; Zero indicates the end of text.

        ds PageSize - ($ - 8000h),255       ; Fill the unused aera with 0FFh
```

Example to make a 48kB ROM

Below is an example for a 48kB ROM that start from page 4000h-7FFFh. In this example, interrupts are disabled during page 0000h-3FFFh is selected, and since BIOS routines are absent the text is displayed by making direct access to the VDP. You will also find a routine to put back the BIOS. Better to add an interrupt routine if page 0 needs to be selected longer.

```
LF:     equ     0Ah
CR:     equ     0Dh

CHPUT:  equ     00A2h   ; Address of character output routine of BIOS
ENASLT: equ     0024h
INIT32: equ     006Fh
RSLREG: equ     0138h
```

```
SETWRT: equ     0053h   ; set address to write in VRAM

PageSize:       equ     4000h   ; 16kB

LINL32: equ     0F3AFh
T32NAM: equ     0F3BDh
CSRX:   equ     0F3DDh
CSRY:   equ     0F3DCh
EXPTBL: equ     0FCC1h          ; Extended slot flags table (4 bytes)

        org 0000h

        ld      hl,Page0000hTXT         ; Text pointer into HL
        call    PrintP0         ; Call the routine Print for page 0
        ret

PrintP0:
        ld      a,(hl)          ; Load the byte from memory at address indicated by HL to A.
        cp      LF
        jr      z,Code_LF
        cp      CR
        jr      z,Code_CR
        and     a               ; Same as CP 0 but faster.
        ret     z               ; Back behind the call print if A = 0
        out     (098h),a        ; Call the routine to display a character.
        inc     hl              ; Increment the HL value.
        push    hl
        ld      hl,CSRX
        inc     (hl)
        pop     hl
        jr      PrintP0         ; Relative jump to the address in the label Print.

Code_CR:
        push    af
        ld      a,1
        ld      (CSRX),a
        pop     af
        inc     hl              ; Increment the HL value.
        jr      PrintP0
Code_LF:
        push    hl
        ld      hl,CSRY
        inc     (hl)
        pop     hl
        inc     hl              ; Increment the HL value.
        jr      PrintP0
```

```
; Message data
Page0000hTXT:                ; Text pointer label
        db "Text from page 0000h-3FFFh",LF,CR,0     ; Zero indicates the end of text

        ds PageSize - $,255  ; Fill the unused aera with 0FFh

;-------------------------
; Begin of page 4000h-3FFFh
;-------------------------

; ### ROM header ###

        db "AB"              ; ID for auto-executable ROM
        dw INIT              ; Main program execution address.
        dw 0          ; STATEMENT
        dw 0          ; DEVICE
        dw 0          ; TEXT (Unused on this page)
        dw 0,0,0      ; Reserved

INIT:   ; Program code entry point label

        ld     a,32
        ld     (LINL32),a    ; 32 columns
        call   INIT32        ; SCREEN 1
        ld     hl,(T32NAM)
        call   SETWRT        ; Set the VRAM address to write the texte

; Routine to select the ROM on page 0000h-3FFFh (from page 4000h-7FFFh)

        ld     a,(0FFFFh)
        cpl                  ; reverse all bits
        ld     d,a           ; Store the current secondary slots register

        in     a,(0A8h)
        ld     e,a           ; Store the current primary slots register

        and    03Ch          ; 00xxxx00
        ld     b,a
        ld     a,e
        and    0Ch           ; 0000xx00
        rrca
        rrca                 ; 000000xx
        ld     c,a
        rrca
        rrca                 ; xx000000
        or     c             ; xx0000xx
        or     b
        di
        out     (0A8h),a        ; Select the primary slot of ROM on page 0000h-3FFFh and C000h-FFFFh

        ld      a,(0FFFFh)
        ld      b,a
        cpl
        ld      (0FFFFh),a
        ld      a,(0FFFFh)
        cp      b
        jr      nz,NO_SS        ; Jump if primary slot

        cpl
        and     0FCh            ; xxxxxx00
        ld      b,a
        ld      a,(0FFFFh)
        cpl
        and     0Ch             ; 0000xx00
        rrca
        rrca                    ; 000000xx
        or      b
        ld      (0FFFFh),a      ; ROM Selection (Secondary Slot)
NO_SS:

; Routine to re-select the Main-RAM on page C000h-7FFFh

        ld      a,e
        and     0C0h            ; xx000000
        ld      b,a
        in      a,(0A8h)
        and     03Fh            ; 00xxxxxx
        or      b
        out     (0A8h),a        ; Select the prim slot of Main-RAM on page C000h-FFFFh

        ld      a,(0FFFFh)
        cpl
        and     03Fh            ; 00xxxxxx
        ld      b,a
        ld      a,d
        and     0C0h            ; xx000000
        or      b
        ld      (0FFFFh),a      ; Select the secondary of Main-RAM slot register

        call    0000h

; Routine to re-select the Main-ROM on page 0000h-3FFFh

        ld      a,e
        out     (0A8h),a        ; Restore the register as at start

        ld      a,d
        ld      (0FFFFh),a      ; Restore the register as at start
NO_SS2:
        ei

; Typical routine to select the ROM on page 8000h-BFFFh from page 4000h-7FFFh

        call    RSLREG
        rrca
        rrca
        and     3       ;Keep bits corresponding to the page 4000h-7FFFh
        ld      c,a
        ld      b,0
        ld      hl,EXPTBL
        add     hl,bc
        ld      a,(hl)
        and     80h
        or      c
        ld      c,a
        inc     hl
        inc     hl
        inc     hl
        inc     hl
        ld      a,(hl)
        and     0Ch
        or      c
        ld      h,080h
        call    ENASLT          ; Select the ROM on page 8000h-BFFFh

        ld      hl,Page4000hTXT         ; Text pointer into HL
        call    Print           ; Call the routine Print below

        jp      08000h  ; Jump to above page.

Print:
        ld      a,(hl)          ; Load the byte from memory at address indicated by HL to A.
        and     a               ; Same as CP 0 but faster.
        ret     z               ; Back behind the call print if A = 0
        call    CHPUT           ; Call the routine to display a character.
        inc     hl              ; Increment the HL value.
        jr      Print           ; Relative jump to the address in the label Print.


; Message data
```

```
Page4000hTXT:                  ; Text pointer label

        db "Text from page 4000h-7FFFh",LF,CR,0      ; Zero indicates the end of text

; Padding with 255 to make a fixed page of 16K size
; (Alternatively, include macros.asm and use ALIGN 4000H)

        ds PageSize - ($ - 4000h),255        ; Fill the unused area in page with 0FFh

;--------------------------
; Begin of page 8000h-BFFFh
;--------------------------

        ld      hl,Page8000hTXT        ; Text pointer
        call    Print          ; Call the routine Print

Finished:
        jr      Finished       ; Jump to itself endlessly.

Page8000hTXT:                  ; Text pointer label
        db "Text from page 8000h-BFFFh",0    ; Zero indicates the end of text.

        ds PageSize - ($ - 8000h),255        ; Fill the unused aera with 0FFh
```

Create a ROM with disks support

There are two methods to create a ROM with disks support. The first uses the hook H.STKE and the second is to launch the ROM from a little BASIC program in ROM.

Method that uses the hook H.STKE

H.STKE (0FEDAh) is called after searching in each slot the executable ROMs when initializing the MSX, just before the system starts the Basic environment. This Hook can therefore allow you to automatically run your ROM with the installed disks.

This example below saves 16 bytes (C500h-C500Fh) in the file "DATA.DAT" on the current disk. In addition of the errors indicated at the BDOS call output (Software errors), the error to know if floppy disk is inserted in current drive or not is also handled but if you need to know more about hardware errors handling see **this site (http://map.grauw.nl/articles/dos-error-handling.php)** .

```
LF:     equ     0Ah
CR:     equ     0Dh

```

```
INIT32: equ     006Fh
CHPUT:  equ     00A2h   ; Address of character output routine from Main-Rom BIOS

RomSize:        equ 4000h       ; 16kB
FCBinRAM:       equ 0C000h

ERRADR: equ     0F323h
FCBBASE:        equ     0F353h
LINL32: equ     0F3AFh
BDOS:   equ     0F37Dh
EXPTBL: equ     0FCC1h          ; Extended slot flags table (4 bytes)
H_STKE: equ     0FEDAh
H_PHYD: equ     0FFA7h
NEWKEY: equ     0FBE5h

        org 8000h       ; Your disk errors handling routine can not be on the page 4000h-7FFFh

; ### ROM header ### (Put 0000h as address when unused)

        db "AB"                 ; ID for auto-executable ROM
        dw INIT                 ; Main program execution address.
        dw 0            ; STATEMENT
        dw 0            ; DEVICE
        dw 0            ; TEXT (Unused on this page)
        dw 0,0,0        ; Reserved

; Setup the hook H.STKE to run the ROM with disk support

INIT:   ; Program entry point label

        ld      a,c             ; Get the ROM slot number

        ld      hl,NewH_STKE
        ld      de,H_STKE
        ld      bc,4
        ldir            ; Copy the routine to execute the ROM to the hook

        ld      (H_STKE+1),a    ; Put the ROM slot number to the hook

        ret     ; Back to slots scanning

; Routine to execute the ROM

NewH_STKE:
        rst     030h    ; Inter-slot call
        db      1       ; This byte will be replaced by the slot number of ROM
        dw      ROM_Exe ; Address to execute the ROM

; Start of your program in ROM

ROM_Exe:
        ld      a,0C9h
        ld      (H_STKE),a     ; Remove the hook

        ld      hl,(ERR_Routine)
        ld      (ERRADR),hl    ; Catches the Error routine

        ld      a,32
        ld      (LINL32),a     ; 32 columns
        call    INIT32         ; SCREEN 1

        ld      a,(H_PHYD)
        cp      0C9h           ;
        jr      nz,DSK_Found   ; Jump if disk installed

        ld      hl,NoDisk_TXT  ; Text pointer into HL
        call    Print          ; Call the routine Print below

        jr      NeverEndLoop

DSK_Found:
        ld      hl,Save_TXT    ; Text pointer into HL
        call    Print          ; Call the routine Print below

        ld      hl,InsDisk
        ld      (0F1E6h),hl    ; Set address to jump to insert disk routine

        ld      hl,FCBinRAM
        ld      (FCBBASE),hl   ; Set FCB pointer to 0C000h
        ex      hl,de
        ld      hl,FCB
        ld      bc,128
        ldir            ; Initialises the FCB data

        ld      c,1Ah
        ld      de,0C500h      ; pointer to data to save
        call    BDOS
Write:
        ld      c,016h  ; Create file
        ld      de,FCBinRAM
        call    BDOS_WE
        or      a
        jp      nz,ERROR
        ld      hl,1
        ld      (FCBinRAM+14),hl       ; Record size = 1 byte

        ld      c,026h  ; Write file
        ld      de,FCBinRAM
        ld      hl,10h
        call    BDOS_WE ; Save 16 bytes (0C500h-0C50Fh)
        or      a
        jp      nz,ERROR

        ld      c,010h  ; Close file
        ld      de,FCBinRAM
        call    BDOS_WE
        or      a
        jp      nz,ERROR

        ld      hl,SaveOK_TXT  ; Text pointer into HL
        jr      SaveMES
ERROR:
        ld      hl,SaveERR_TXT
SaveMES:
        call    Print          ; Call the routine Print below

NeverEndLoop:
        jr      NeverEndLoop

; Your

InsDisk:
        ld      sp,(0D000h)    ; Restore SP register

        ld      a,c            ; Get error flags
        and     2
        jp      z,ERROR ; Jump if disk is present in drive

        ld      hl,InsDisk_TXT
        call    Print          ; Call the routine Print below

RET_KEY:
        ld      a,(NEWKEY+7)
        bit     7,a
        jr      nz,RET_KEY

        jp      Write

; Print the text pointed by HL
```

```
Print:
        ld     a,(hl)         ; Load the byte from memory at address indicated by HL to A.
        and    a              ; Same as CP 0 but faster.
        ret    z              ; Back behind the call print if A = 0
        call   CHPUT          ; Call the routine to display a character.
        inc    hl             ; Increment the HL value.
        jr     Print          ; Relative jump to the address in the label Print.

BDOS_WE:
        ld     (0D000h),sp    ; Store SP register
        jp     BDOS

; Data

NoDisk_TXT:                   ; Text pointer label
        db "No disk installed!",LF,CR
        db "Turn off the MSX.",0     ; Zero indicates the end of text
Save_TXT:
        db "Saving",022h,"DATA.DAT",022h,"...",LF,CR,0
SaveOK_TXT:
        db "File saved",LF,CR,0
SaveERR_TXT:
        db "File error!!!",LF,CR,0
InsDisk_TXT:
        db "Insert the floppy disk",LF,CR
        db "then press RETURN",LF,CR,0

ERR_Routine:
        dw InsDisk

FCB:
        db 0,"DATA    DAT"
        ds 116,0      ; Fill the rest of FCB with 00h

        ds RomSize - ($ & (RomSize-1)),255   ; Fill the unused aera in page with 0FFh
```

Method that uses a BASIC program

Following example use the program "10 DEFUSR=&H8024:?USR(0)" to execute the machine program of the ROM. Aside from this difference, it does the same thing as the previous program.

```
LF:     equ     0Ah
CR:     equ     0Dh

INIT32: equ     006Fh
CHPUT:  equ 00A2h       ; Address of character output routine from Main-Rom BIOS

RomSize:        equ 4000h       ; 16kB
FCBinRAM:       equ 0C000h

ERRADR: equ     0F323h
FCBBASE:        equ     0F353h
LINL32: equ     0F3AFh
BDOS:   equ     0F37Dh
EXPTBL: equ     0FCC1h          ; Extended slot flags table (4 bytes)
H_PHYD: equ     0FFA7h
NEWKEY: equ     0FBE5h

        org 8000h       ; Your disk errors handling routine can not be on the page 4000h-7FFFh

; ### ROM header ### (Put 0000h as address when unused)

        db "AB"                 ; ID for auto-executable ROM
        dw 0            ; INIT
        dw 0            ; STATEMENT
        dw 0            ; DEVICE
        dw 08010h       ; TEXT
        dw 0,0,0        ; Reserved

INIT:   ; Program entry point label

; BASIC Program Data for "10 DEFUSR=&H8024:?USR(0)"

        db 0,22h,80h,0Ah,0,97h,0DDh,0EFh,0Ch,24h,80h,3Ah,91h,0DDh,28h,11h,29h,0,0,0

; Start of your program in ROM (08024h)

ROM_Exe:

        ld      hl,(ERR_Routine)
        ld      (ERRADR),hl     ; Catches the Error routine

        ld      a,32
        ld      (LINL32),a      ; 32 columns
        call    INIT32          ; SCREEN 1

        ld      a,(H_PHYD)
        cp      0C9h            ;
        jr      nz,DSK_Found    ; Jump if disk installed

        ld      hl,NoDisk_TXT   ; Text pointer into HL
        call    Print           ; Call the routine Print below

        jr      NeverEndLoop

DSK_Found:
        ld      hl,Save_TXT     ; Text pointer into HL
        call    Print           ; Call the routine Print below

        ld      hl,InsDisk
        ld      (0F1E6h),hl     ; Set address to jump to insert disk routine

        ld      hl,FCBinRAM
        ld      (FCBBASE),hl    ; Set FCB pointer to 0C000h
        ex      hl,de
        ld      hl,FCB
        ld      bc,128
        ldir            ; Initialises the FCB data

        ld      c,1Ah
        ld      de,0C500h       ; pointer to data to save
        call    BDOS
Write:
        ld      c,016h  ; Create file
        ld      de,FCBinRAM
        call    BDOS_WE
        or      a
        jp      nz,ERROR

        ld      hl,1
        ld      (FCBinRAM+14),hl        ; Record size = 1 byte

        ld      c,026h  ; Write file
        ld      de,FCBinRAM
        ld      hl,10h
        call    BDOS_WE ; Save 16 bytes (0C500h-0C50Fh)
        or      a
        jp      nz,ERROR

        ld      c,010h  ; Close file
        ld      de,FCBinRAM
        call    BDOS_WE
        or      a
        jp      nz,ERROR
        ld      hl,SaveOK_TXT   ; Text pointer into HL
        jr      SaveMES
ERROR:
        ld      hl,SaveERR_TXT
SaveMES:
        call    Print           ; Call the routine Print below

NeverEndLoop:
        jr      NeverEndLoop

; Your

InsDisk:
        ld      sp,(0D000h)     ; Restore SP register

        ld      a,c             ; Get error flags
        and     2
        jp      z,ERROR ; Jump if disk is present in drive

        ld      hl,InsDisk_TXT
        call    Print           ; Call the routine Print below

RET_KEY:
        ld      a,(NEWKEY+7)
        bit     7,a
        jr      nz,RET_KEY

        jp      Write

; Print the text pointed by HL

Print:
        ld      a,(hl)          ; Load the byte from memory at address indicated by HL to A.
        and     a               ; Same as CP 0 but faster.
        ret     z               ; Back behind the call print if A = 0
        call    CHPUT           ; Call the routine to display a character.
        inc     hl              ; Increment the HL value.
        jr      Print           ; Relative jump to the address in the label Print.

BDOS_WE:
        ld      (0D000h),sp     ; Store SP register
        jp      BDOS

; Data

NoDisk_TXT:                     ; Text pointer label
        db "No disk installed!",LF,CR
        db "Turn off the MSX.",0       ; Zero indicates the end of text
Save_TXT:
        db "Saving",022h,"DATA.DAT",022h,"...",LF,CR,0
SaveOK_TXT:
        db "File saved",LF,CR,0
SaveERR_TXT:
        db "File error!!!",LF,CR,0
InsDisk_TXT:
        db "Insert the floppy disk",LF,CR
        db "then press RETURN",LF,CR,0

ERR_Routine:
        dw InsDisk

FCB:
        db 0,"DATA     DAT"
        ds 116,0       ; Fill the rest of FCB with 00h

        ds RomSize - ($ & (RomSize-1)),255    ; Fill the unused aera in page with 0FFh

```

Create a ROM with mapper

MegaRom's mappers are not standardized. The main existing mappers are described on the page **here (https://www.msx.org/wiki/MegaROM_Mappers)** . In addition, how to assemble your program to the MegaRom format also depends on the assembler used. Please refer to the manual for how to manage the segments. Below are some examples. If your assembler can not create a ROM for mapper, you will have to assemble each segment separately and merge them together with concat, or use the instruction INCBIN in an extra program in assembler. You can create a jump table to make the link between the segments for example.

Examples to make a 128kB ROM for ASCII 16k mapper

Example for Glass assembler

```
; Example to create an MegaRom of 128kB that use an ASCII 16K Mapper
; for glass assembler

Seg0:   ds      4000H
Seg1:   ds      4000H
Seg2:   ds      4000H
Seg3:   ds      4000H
```

```
Seg4:   ds      4000H
Seg5:   ds      4000H
Seg6:   ds      4000H
Seg7:   ds      4000H

LF:     equ     0Ah
CR:     equ     0Dh

CHPUT:  equ     00A2h   ; Address of character output routine of main Rom BIOS
ENASLT: equ     0024h
INIT32: equ     006Fh
RSLREG: equ     0138h

Seg_P8000_SW:   equ     7000h   ; Segment switch on page 8000h-BFFFh (ASCII 16k Mapper)

LINL32: equ     0F3AFh
EXPTBL: equ     0FCC1h          ; Extended slot flags table (4 bytes)


        SECTION Seg0

        org     4000h

        db      41h,42h
        dw      INIT,0,0,0,0,0,0

INIT:
        ld      a,32
        ld      (LINL32),a      ; 32 columns
        call    INIT32          ; SCREEN 1

; Typical routine to select the ROM on page 8000h-BFFFh from page 4000h-7FFFh

        call    GetSlotPage1
        ld      h,080h
        call    ENASLT          ; Select the ROM on page 8000h-BFFFh

        ld      a,1
LOOP:
        ld      (Seg_P8000_SW),a        ; Select the segment on page 8000h-BFFFh

        push    af
        ld      hl,Seg1_TXT     ; Text pointer into HL
        call    Print           ; Call the routine Print below
        pop     af

        inc     a       ; Increment segment number
        cp      8
        jr      nz,LOOP ; Jump to LOOP if A<8

Finished:
        jr      Finished        ; Jump to itself endlessly.

; Gets the slot selected in page 1 (4000h-7FFFh)
; a <- slot ID
GetSlotPage1:
        call    RSLREG
        rrca
        rrca
        and     3       ;Keep bits corresponding to the page 4000h-7FFFh
        ld      c,a
        ld      b,0
        ld      hl,EXPTBL
        add     hl,bc
        ld      a,(hl)
        and     80h
        or      c
        ld      c,a
        inc     hl
        inc     hl
        inc     hl
        inc     hl
        ld      a,(hl)
        and     0Ch
        or      c
        ret

Print:
        ld      a,(hl)          ; Load the byte from memory at address indicated by HL to A.
        and     a               ; Same as CP 0 but faster.
        ret     z               ; Back behind the call print if A = 0
        call    CHPUT           ; Call the routine to display a character.
        inc     hl              ; Increment the HL value.
        jr      Print           ; Jump to the address in the label Print.

        ENDS

        SECTION Seg1
        org     8000h

Seg1_TXT:                       ; Text pointer label
        db "Text from segment 1",LF,CR,0        ; Zero indicates the end of text.
        ENDS
        SECTION Seg2
        org     8000h

        db "Text from segment 2",LF,CR,0
        ENDS

        SECTION Seg3
        org     8000h

        db "Text from segment 3",LF,CR,0
        ENDS

        SECTION    Seg4
        org     8000h

        db "Text from segment 4",LF,CR,0
        ENDS

        SECTION    Seg5
        org     8000h

        db "Text from segment 5",LF,CR,0
        ENDS

        SECTION    Seg6
        org     8000h

        db "Text from segment 6",LF,CR,0
        ENDS

        SECTION    Seg7
        org     8000h

        db "Text from segment 7",LF,CR,0
        ENDS
```

Example for Sjasm assembler

Do not forget the space or tabulation in the front of directives defpage and page.

```
; Example to create an MegaRom of 128kB that use an ASCII 16K Mapper
; for Sjasm assembler

        output ASC16tst.ROM
```

```
LF:     equ     0Ah
CR:     equ     0Dh

ENASLT: equ     0024h
INIT32: equ     006Fh
CHPUT:  equ     00A2h   ; Address of character output routine of main Rom BIOS
RSLREG: equ     0138h

PageSize:       equ     04000h  ; 16kB
Seg_P8000_SW:   equ     07000h  ; Segment switch for page 8000h-BFFFh (ASCII 16k Mapper)

LINL32: equ     0F3AFh
EXPTBL: equ     0FCC1h          ; Extended slot flags table (4 bytes)


        defpage 0,4000H,PageSize
        page 0

        db      41h,42h
        dw      INIT,0,0,0,0,0,0

INIT:
        ld      a,32
        ld      (LINL32),a      ; 32 columns
        call    INIT32          ; SCREEN 1

; Typical routine to select the ROM on page 8000h-BFFFh from page 4000h-7BFFFh

        call    RSLREG
        rrca
        rrca
        and     3       ;Keep bits corresponding to the page 4000h-7FFFh
        ld      c,a
        ld      b,0
        ld      hl,EXPTBL
        add     hl,bc
        ld      a,(hl)
        and     80h
        or      c
        ld      c,a
        inc     hl
        inc     hl
        inc     hl
        inc     hl
        ld      a,(hl)
        and     0Ch
        or      c
        ld      h,080h
        call    ENASLT          ; Select the ROM on page 8000h-BFFFh

        ld      a,1
LOOP:
        ld      (Seg_P8000_SW),a        ; Select the segment on page 8000h-BFFFh

        push    af
        ld      hl,Seg1_TXT     ; Text pointer into HL
        call    Print           ; Call the routine Print below
        pop     af

        inc     a       ; Increment segment number
        cp      8
        jr      nz, LOOP        ; Jump to LOOP if A<8

Finished:
        jr      Finished        ; Jump to itself endlessly.

Print:
        ld      a,(hl)          ; Load the byte from memory at address indicated by HL to A.
        and     a               ; Same as CP 0 but faster.
        ret     z               ; Back behind the call print if A = 0
        call    CHPUT           ; Call the routine to display a character.
        inc     hl              ; Increment the HL value.
        jr      Print           ; Jump to the address in the label Print.

        defpage 1,8000H,PageSize
        page 1

Seg1_TXT:                       ; Text pointer label
        db "Text from segment 1",LF,CR,0       ; Zero indicates the end of text.

        defpage 2,8000H,PageSize
        page 2

        db "Text from segment 2",LF,CR,0

        defpage 3,8000H,PageSize
        page 3

        db "Text from segment 3",LF,CR,0

        defpage 4,8000H,PageSize
        page 4
        db "Text from segment 4",LF,CR,0

        defpage 5,8000H,PageSize
        page 5

        db "Text from segment 5",LF,CR,0

        defpage 6,8000H,PageSize
        page 6

        db "Text from segment 6",LF,CR,0

        defpage 7,8000H,PageSize
        page 7

        db "Text from segment 7",LF,CR,0
```

Example for tniASM assembler

```
; Example to create an MegaRom of 128kB that use an ASCII 16K Mapper
; for tniASM assembler

        fname "ASC16tst.ROM"

LF:     equ     0Ah
CR:     equ     0Dh

ENASLT: equ     0024h
INIT32: equ     006Fh
CHPUT:  equ     00A2h   ; Address of character output routine of main Rom BIOS
RSLREG: equ     0138h

PageSize:       equ     04000h  ; 16kB
Seg_P8000_SW:   equ     07000h  ; Segment switch for page 8000h-BFFFh (ASCII 16k Mapper)

LINL32: equ     0F3AFh
EXPTBL: equ     0FCC1h          ; Extended slot flags table (4 bytes)


        org     4000h,7FFFh     ; Page 0

        dw        "AB",INIT,0,0,0,0,0,0

INIT:
        ld      a,32
        ld      (LINL32),a      ; 32 columns
        call    INIT32          ; SCREEN 1

; Typical routine to select the ROM on page 8000h-BFFFh from page 4000h-7BFFFh

        call    RSLREG
        rrca
        rrca
        and     3       ;Keep bits corresponding to the page 4000h-7FFFh
        ld      c,a
        ld      b,0
        ld      hl,EXPTBL
        add     hl,bc
        ld      a,(hl)
        and     80h
        or      c
        ld      c,a
        inc     hl
        inc     hl
        inc     hl
        inc     hl
        ld      a,(hl)
        and     0Ch
        or      c
        ld      h,080h
        call    ENASLT          ; Select the ROM on page 8000h-BFFFh

        ld      a,1
LOOP:
        ld      (Seg_P8000_SW),a

        push    af
        ld      hl,Seg1_TXT     ; Text pointer into HL
        call    Print           ; Call the routine Print below
        pop     af

        inc     a
        cp      8
        jr      nz, LOOP        ; Jump to LOOP if A<8

Finished:
        jr      Finished        ; Jump to itself endlessly.

Print:
        ld      a,(hl)          ; Load the byte from memory at address indicated by HL to A.
        and     a               ; Same as CP 0 but faster.
        ret     z               ; Back behind the call print if A = 0
        call    CHPUT           ; Call the routine to display a character.
        inc     hl              ; Increment the HL value.
        jr      Print           ; Jump to the address in the label Print.

        ds PageSize - ($ - 4000h),255        ; Fill the unused aera with 0FFh

        org     8000h,0BFFFh    ; page 1

Seg1_TXT:                       ; Text pointer label
        db "Text from segment 1",LF,CR,0     ; Zero indicates the end of text.
        ds PageSize - ($ - 8000h),255        ; Fill the unused aera with 0FFh

        org     8000h,0BFFFh    ; page 2

        db "Text from segment 2",LF,CR,0
        ds PageSize - ($ - 8000h),255

        org     8000h,0BFFFh    ; page 3

        db "Text from segment 3",LF,CR,0
        ds PageSize - ($ - 8000h),255

        org     8000h,0BFFFh    ; page 4

        db "Text from segment 4",LF,CR,0
        ds PageSize - ($ - 8000h),255

        org     8000h,0BFFFh    ; page 5

        db "Text from segment 5",LF,CR,0
        ds PageSize - ($ - 8000h),255

        org     8000h,0BFFFh    ; page 6

        db "Text from segment 6",LF,CR,0
        ds PageSize - ($ - 8000h),255

        org     8000h,0BFFFh    ; page 7

        db "Text from segment 7",LF,CR,0
        ds PageSize - ($ - 8000h),255
```

Example for Zasm assembler

```
; Example to create an MegaRom of 128kB that use an ASCII 16K Mapper
; for zasm assembler

LF:     equ     0Ah
CR:     equ     0Dh

ENASLT: equ     0024h
INIT32: equ     006Fh
CHPUT:  equ     00A2h   ; Address of character output routine of main Rom BIOS
RSLREG: equ     0138h

PageSize:       equ     04000h  ; 16kB
Seg_P8000_SW:   equ     07000h  ; Segment switch on page 8000h-BFFFh (ASCII 16k Mapper)

LINL32: equ     0F3AFh          ; 32 columns
EXPTBL: equ     0FCC1h          ; Extended slot flags table (4 bytes)


#target rom
#code   Seg0,04000h,PageSize

; ### ROM header ###

        db      41h,42h
        dw      INIT,0,0,0,0,0,0

INIT:
        ld      a,32
        ld      (LINL32),a      ; 32 columns
        call    INIT32          ; SCREEN 1

; Typical routine to select the ROM on page 8000h-BFFFh from page 4000h-7FFFh

        call    RSLREG
        rrca
        rrca
        and     3       ;Keep bits corresponding to the page 4000h-7FFFh
        ld      c,a
        ld      b,0
        ld      hl,EXPTBL
        add     hl,bc
        ld      a,(hl)
        and     80h
        or      c
        ld      c,a
        inc     hl
        inc     hl
        inc     hl
        inc     hl
        ld      a,(hl)
        and     0Ch
        or      c
        ld      h,080h
        call    ENASLT          ; Select the ROM on page 8000h-BFFFh

        ld      a,1
LOOP:
        ld      (Seg_P8000_SW),a        ; Select the segment on page 8000h-BFFFh

        push    af
        ld      hl,Seg1_TXT     ; Text pointer into HL
        call    Print           ; Call the routine Print below
        pop     af

        inc     a       ; Increment segment number
        cp      8
        jr      nz, LOOP        ; Jump to LOOP if A<8

Finished:
        jr      Finished        ; Jump to itself endlessly.

Print:
        ld      a,(hl)          ; Load the byte from memory at address indicated by HL to A.
        and     a               ; Same as CP 0 but faster.
        ret     z               ; Back behind the call print if A = 0
        call    CHPUT           ; Call the routine to display a character.
        inc     hl              ; Increment the HL value.
        jr      Print           ; Jump to the address in the label Print.

#code   Seg1,08000h,PageSize

Seg1_TXT:                       ; Text pointer label
        db "Text from segment 1",LF,CR,0        ; Zero indicates the end of text.

#code    Seg2,08000h,PageSize

        db "Text from segment 2",LF,CR,0

#code    Seg3,08000h,PageSize

        db "Text from segment 3",LF,CR,0

#code    Seg4,08000h,PageSize

```

```
        db "Text from segment 4",LF,CR,0

#code    Seg5,08000h,PageSize

        db "Text from segment 5",LF,CR,0

#code    Seg6,08000h,PageSize

        db "Text from segment 6",LF,CR,0

#code    Seg7,08000h,PageSize

        db "Text from segment 7",LF,CR,0

END
```

Search for RAM

For a ROM that supports disks you can use **this system variables (https://www.msx.org/wiki/How_to_detect_the_RAM)** .

For other ROMs you must search the RAM your self on each page as below example.

Note: Variables RAMAD0-RAMAD3 are used in examples but you can use any other free memory instead since these variables are used by the system only when a disk is installed.

```
; Routine of search for RAM on each page from MSX cartridge
;
; Output: RAMAD0-RAMAD3 = Slot number of Main-RAM for corresponding page

RDSLT:  equ     0000Ch  ; Read a byte in a Slot
RSLREG: equ     00138h  ; Read primary Slot REGister
WRSLT:  equ     00014h  ; Write a byte in a Slot
WSLREG: equ     0013Bh  ; Write primary Slot REGister

RomSize:        equ     04000h

EXPTBL: equ     0FCC1h  ; Expanded Slot Table
SLTTBL  equ     0FCC5h  ; Slot Table
KBUF:   equ     0F41Fh  ; Temporary data
RAMAD0: equ     0F341h  ; Main-RAM Slot (00000h~03FFFh)
RAMAD1: equ     0F342h  ; Main-RAM Slot (04000h~07FFFh)
RAMAD2: equ     0F343h  ; Main-RAM Slot (08000h~0BFFFh)
RAMAD3: equ     0F344h  ; Main-RAM Slot (0C000h~0FFFFh)
```

```
RAMSLT: equ     KBUF+3


        org     04000h  ; Can be also 8000h

; ### ROM header ###

        db      041h,042h
        dw      INIT,0,0,0,0,0,0

INIT:

def_RAMAD3:
        call    RSLREG
        and     0C0h
        rlca
        rlca                   ; A = Primary slot
        ld      c,a
        ld      b,0
        ld      hl,EXPTBL
        add     hl,bc
        ld      a,(hl)
        and     80h
        jr      z,No_SS3       ; Jump if slot is not secondary (page 3)

        ld      hl,SLTTBL
        add     hl,bc
        ld      a,(hl)         ; A = Value of current decondary slots register
        and     0C0h           ; Keep the bits for page 3
        rrca
        rrca
        rrca
        rrca                   ; Bits 2-3 of A = Current secondary slot (page 2)
        or      080h           ; Set the bit 7
No_SS3:
        or      c
        ld      (RAMAD3),a     ; Bit7=1 if extended Slot

def_RAMAD2:

        ld      hl,08000h
        call    ram_srch
        ld      (RAMAD2),a

def_RAMAD1:

        ld      hl,04000h
        call    ram_srch
        ld      (RAMAD1),a

def_RAMAD0:

        ld      hl,00000h
        call    ram_srch
        ld      (RAMAD0),a

NeverEndLoop:
        jr      NeverEndLoop

; Search RAM on a page
; Input: HL=0000h, 4000h or 8000h
; output: A=slot number and Carry = 0, Carry = 1 if Ram not found

ram_srch:
        ld      b,4     ;Slot primaire
ram_srch_loop:
        ld      a,b
        dec     a
        xor     3
        ld      (RAMSLT),a
        ld      e,a

        push    hl
        ld      hl,EXPTBL
        ld      d,0
        add     hl,de
        ld      a,(hl)
        ld      (KBUF),a        ; Save secondary slot flag

        pop     hl
        ld      a,h
        exx
        ld      h,a
        ld      l,0             ; Restore HL address

        ld      a,(KBUF)        ; Restore secondary slot flag
        rlca
        ld      b,1
        ld      a,(RAMSLT)
        jr      nc,PrimSLT

        ld      b,4     ;Slot secondaire
ram_srch_loop2:
        ld      a,b
        dec     a
        xor     3
        rlca
        rlca
        ld      c,a
        ld      a,(RAMSLT)
        or      c
        or      080h    ; Set bit 7
PrimSLT:
        ld      (KBUF+1),a
        push    bc
        call    RDSLT
        ld      (KBUF+2),a
        pop     bc
        cp      041h
        jr      nz,no_header   ; Jump if first byte = "A" (Rom?)

        inc     hl
        ld      a,(KBUF+1)
        push    bc
        call    RDSLT
        pop     bc
        dec     hl
        cp      042h
        jr      z,no_ram       ; Jump if second byte <> "B"
no_header:
        ld      a,(KBUF+1)
        push    bc
        call    RDSLT          ; Read first byte
        pop     bc
        ld      e,041h
        ld      a,(KBUF+1)
        push    bc
        call    WRSLT          ; Write "A" at first byte
        pop     bc
        ld      a,(KBUF+1)
        push    bc
        call    RDSLT          ; Read first byte
        pop     bc
        cp      041h
        jr      z,ram_found    ; Jump if first byte = "A"

no_ram:
        djnz    ram_srch_loop2 ; Go to next Slot if No RAM
        exx
        djnz    ram_srch_loop  ; Go to next Slot if No RAM
        scf                    ; Set Carry
        ret

ram_found:
        ld      a,(KBUF+2)
        ld      e,a
        ld      a,(KBUF+1)
        push    af
        or      080h
        call    WRSLT           ; Restore first byte value of RAM
        pop     af              ; A=Slot of Ram found (without Bit7)
        or      a               ; Reset Carry
        ret

        ds RomSize - ($ & (RomSize-1)),255   ; Fill the unused aera in page with 0FFh
        end
```

Use preferably the following example for the MSX Turbo R because it uses its internal memory by default and the access to RAM is faster in R800 mode.

```
; Routine of search for RAM on each page from MSX cartridge
;
; Output: RAMAD0-RAMAD3 = Slot number of Main-RAM for corresponding page

RDSLT:  equ     0000Ch  ; Read a byte in a Slot
WRSLT:  equ     00014h  ; Write a byte in a Slot
RSLREG: equ     00138h  ; Read primary Slot REGister
WSLREG: equ     0013Bh  ; Write primary Slot REGister
CHGCPU: equ     00180h

RAMAD0: equ     0F341h  ; Main-RAM Slot (00000h~03FFFh)
RAMAD1: equ     0F342h  ; Main-RAM Slot (04000h~07FFFh)
RAMAD2: equ     0F343h  ; Main-RAM Slot (08000h~0BFFFh)
RAMAD3: equ     0F344h  ; Main-RAM Slot (0C000h~0FFFFh)
EXPTBL: equ     0FCC1h  ; Expanded Slot Table
SLTTBL  equ     0FCC5h  ; Slot Table

        org     04000h  ; Can be also 8000h

; ### ROM header ###

        db      041h,042h
        dw INIT                 ; Main program execution address.
        dw 0            ; STATEMENT
        dw 0            ; DEVICE
        dw 0            ; TEXT (Unused on this page)
        dw 0,0,0        ; Reserved

INIT:
        ld      a,082h
        call    CHGCPU  ; Select R800 mode with DRAM

def_RAMADx:
        call    RSLREG
        and     0C0h
        rlca
        rlca                    ; A = Primary slot
        ld      c,a
        ld      b,0
        ld      hl,EXPTBL
        add     hl,bc
        ld      a,(hl)
        and     80h
        jr      z,No_SS3        ; Jump if slot is not secondary (page 3)

        ld      hl,SLTTBL
        add     hl,bc
        ld      a,(hl)          ; A = Value of current decondary slots register
        and     0C0h            ; Keep the bits for page 3
        rrca
        rrca
        rrca
        rrca                    ; Bits 2-3 of A = Current secondary slot (page 2)
        or      080h            ; Set the bit 7
No_SS3:
        or      c
        ld      (RAMAD3),a
        ld      (RAMAD2),a
        ld      (RAMAD1),a
        ld      (RAMAD0),a

NeverEndLoop:
        jr      NeverEndLoop

        ds RomSize - ($ & (RomSize-1)),255   ; Fill the unused aera in page with 0FFh
        end
```

Allocate RAM (workarea)

In programs not requiring software from other cartridges (stand-alone software such as games), the portion with the smaller address than the work area used by BIOS (F380H) can be used freely.

But in programs which are executed by using BASIC interpreter functions, the same area cannot be shared as the work area. To do this, there are three methods:

(1) Place RAM on the cartridge itself (the safest and most reliable method).

(2) When one or two bytes are needed for the work area, use two bytes corresponding to itself in SLTWRK (FD09h) as the work area.

(3) When more than two bytes are needed for the work area, allocates it from RAM used by BASIC.

Example code for method 2:
A page1 extension ROM (4000H-7FFFH) with other extensions in the slot address space:**[1] (https://sourceforge.net/p/msxsyssrc/git/ci/master/tree/examples/allocate_system_memory/alloc1.mac)**
A page2 extension ROM (8000H-BFFFH) with other extensions in the slot address space:**[2] (https://sourceforge.net/p/msxsyssrc/git/ci/master/tree/examples/allocate_system_memory/alloc2.mac)**
A page1 extension ROM (4000H-7FFFH) exclusive slot address space **(8 bytes of workarea)**:**[3] (https://sourceforge.net/p/msxsyssrc/git/ci/master/tree/examples/allocate_system_memory/allocs.mac)**

Example code for method 3:
A page1 extension ROM (4000H-7FFFH) not supporting the MSX disksystem:**[4] (https://sourceforge.net/p/msxsyssrc/git/ci/master/tree/examples/allocate_system_memory/allocn.mac)**
A page1 extension ROM (4000H-7FFFH) supporting the MSX disksystem:**[5] (https://sourceforge.net/p/msxsyssrc/git/ci/master/tree/examples/allocate_system_memory/allocd.mac)**

Useful system Variables

FCC9h	SLTATR	64	Slot attributes given during MSX boot process.
			`Bit 7 = 1 if Basic program, else 0`
			`Bit 6 = 1 if device extension, else 0`
			`Bit 5 = 1 if statement extension, else 0`
			`Bits 4~0 = Unused`

FD09h	SLTWRK	128	SLTWRK is a 128-byte variable array used to reserve a RAM work area in Main-RAM for ROM applications. This array consists of 8 bytes per slot (2 per memory page). Each of these 2 octets are provided to place an slot ID with flags on a byte (MSB) or an address on two bytes as follows.
			`SLTWRK+0 = Work area for slot 0-0, page 0000h~3FFFh`
			`SLTWRK+2 = Work area for slot 0-0, page 4000h~7FFFh`
			`SLTWRK+4 = Work area for slot 0-0, page 8000h~BFFFh`
			`SLTWRK+6 = Work area for slot 0-0, page C000h~FFFFh`
			`SLTWRK+8 = Work area for slot 0-1, page 0000h~3FFFh`
			`   .`
			`   .`
			`   .`
			`SLTWRK+124 = Work area for slot 3-3, page 8000h~BFFFh`
			`SLTWRK+126 = Work area for slot 3-3, page C000h~FFFFh`
			The pointer is used to reserve a work area from 8000h or higher to F37Fh.
			The slot ID is used to reserve a work area on the pages 0000h~3FFFh & 4000h~7FFFh).
			Slot ID format used in table SLTWRK:
			LSB = F RMD APP RES SS1 SS0 PS1 PS0
			MSB = 00h
			- PS = Primary slot number
			- SS = Secondary slot number
			- RES = Reserved
			- APP = Set if the RAM used by an application, 0 otherwise
			- RMD = Set if the RAM is used by instruction CALL MEMINI, 0 otherwise
			- F = Set if secondary slot, 0 if primary slot.

FD89h	PROCNM	16	Work aera of the instructions CALL and OPEN. Contents the instruction name or device name.

Retrieved from "**https://www.msx.org/wiki/Develop_a_program_in_cartridge_ROM**"

Navigation

- **Main page**
- **Community portal**
- **Current events**

- **Recent changes**
- **Random page**
- **Help**

Toolbox

- **What links here**
- **Related changes**
- **Special pages**
- **Printable version**
- **Permanent link**