MSX2 TECHNICAL HANDBOOK

Edited by: ASCII Systems Division
Published by: ASCII Coprporation - JAPAN
First edition: March 1987

Text file typed by: Nestor Soriano (Konami Man) - SPAIN
October 1997

Changes from the original:

- In description of SUBROM routine, comment "see page 352" has been changed to "see appendix 2..."

- In description of SLTATR and SLTWRK work areas, expressions for calculate the concrete work area for a given slot and page have been added.

- In the first line after beginning of section 7.2.3, "The following routines..." has been corrected to "The following addresses..."

- In Figure 5.52, indication of F380H address was placed in the middle of the user's area. It has been moved to the beginning of system work.

-=-

CHAPTER 5 - ACCESS TO PERIPHERALS THROUGH BIOS (Part 7)

7. SLOTS AND CARTRIDGES

The CPU (Z80) used in the MSX can access an address space of only 64K bytes (0000H to FFFFH). MSX is set up to access an effective space of 1M bytes. This is accomplished by using "slots", which allocate more than one memory byte or device to the same address.

This chapter introduces the use of the slot and information necessary to connect the cartridge software or the new device to MSX via the slot.

7.1 Slots

A slot is an interface to effectively use a large address space, and to interface any memory or devices connected to the MSX address bus installed via the slot. The BASIC ROM inside the machine or RAM at MSX-DOS mode are not exceptions. The place at which the cartridge software is installed is also one of the slots. The following descriptions introduce how the software and the devices are connected to the slot.

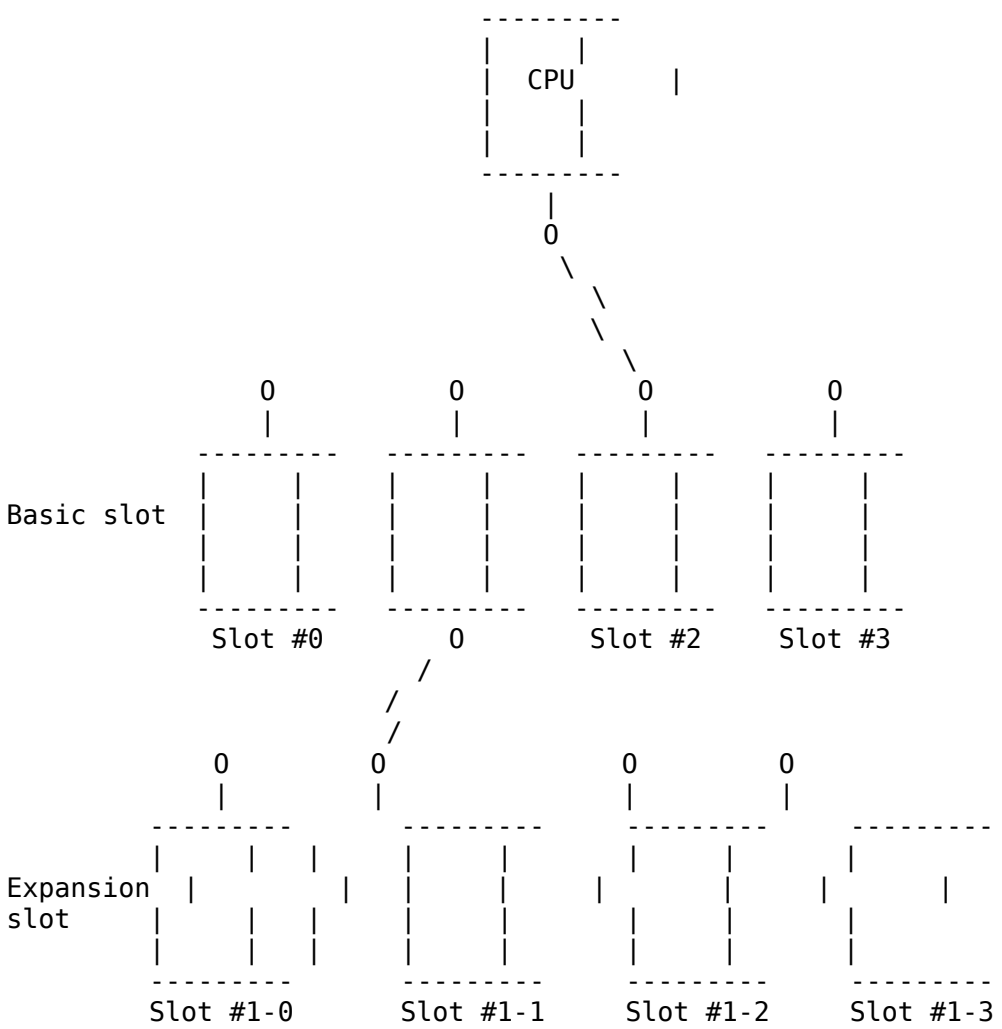
7.1.1 Basic slot and expansion slot

The slot is either a basic slot or a expansion slot. A "basic slot" is a slot directly connected to the CPU address bus, as shown in Figure 5.36. The standard MSX machine can have up to four basic slots. The basic slot can be expanded up to four slots by connecting a slot expansion box (in some cases, the expansion is already done inside the machine), and is called "expansion

slots". When each of four basic slots is expanded to four expansion slots, the maximum number of slots is 16. If you multiply 16 slots x 64K bytes you will get 1M bytes of accessible address space.

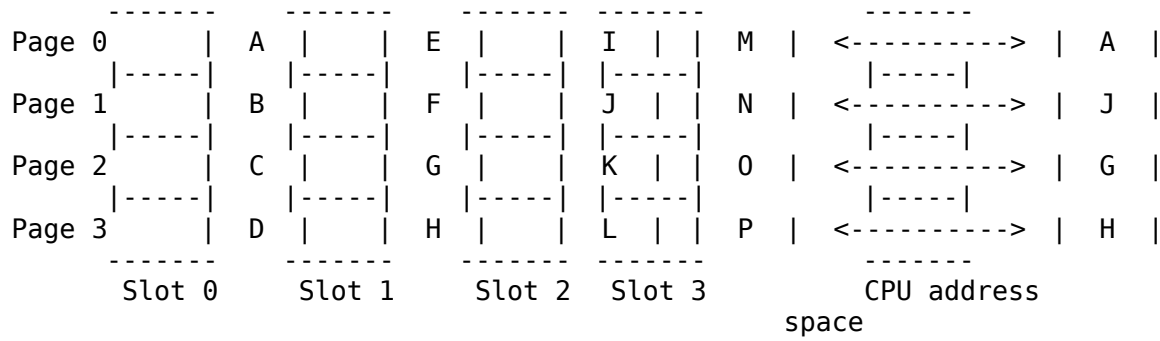
Note that the system itself cannot be started when expansion slot boxes are connected to the expansion slot. So the user should only connect expansion slot boxes to the basic slots. (Though the slot for the standard MSX cartridge is always a basic slot, in some cases the connector for the optional hardware of each machine might be connected to the expansion slot).

Figure 5.36 Basic slot and expansion slot



Each slot has 64K bytes from 0000H to FFFFH of address space and MSX manages it by dividing it into four "pages" of 16K bytes each. The CPU can select and access any slot for each page, and, as shown in Figure 5.37, it is possible to select and combine portions of several slots. Note that a pages with a given page number cannot be assigned to a page with a different page number (that is, page n of each slot is also page n to the CPU).

Figure 5.37 Example of the page selection



7.1.2 Selecting slots

Selecting slots is different for the basic slot than for the expansion slot. For basic slots, it is done through the I/O port at A8H (see Figure 5.38), and for expansion slots, it is done through the "expansion slot selection register (FFFFH)" of the installed expansion slot (see Figure 5.39). It is not recommended to change them directly, so the user should not switch the slots without careful planning. When the program switches the slot of the page in which it resides, the action is not always predictable. To call the program in another slot, use the inter-slot call described in the next section.

Figure 5.38 Selecting the basic slot

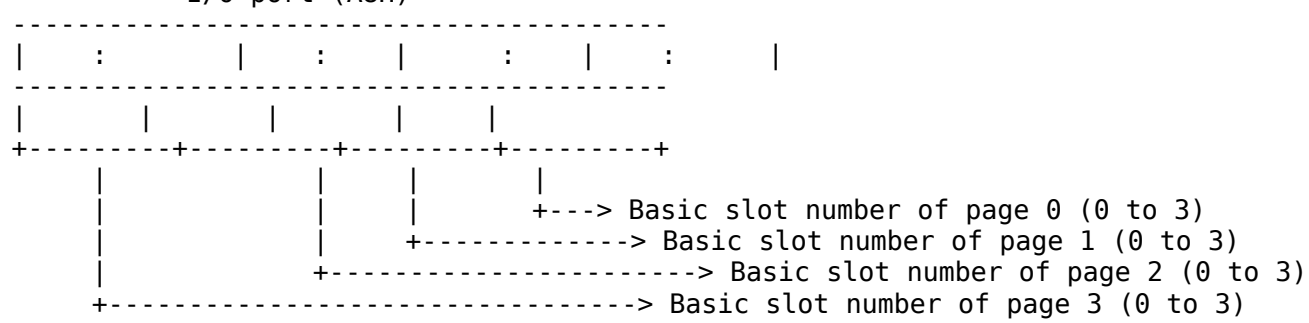
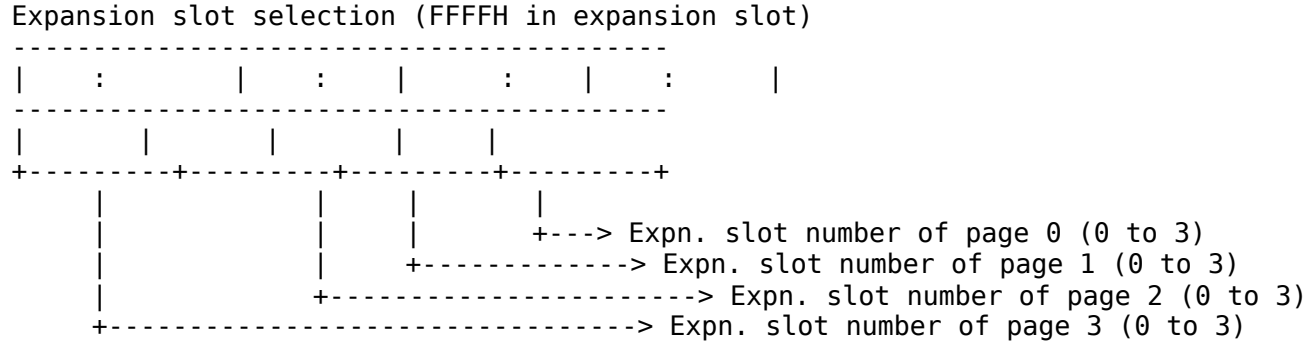


Figure 5.39 Selecting the expansion slot



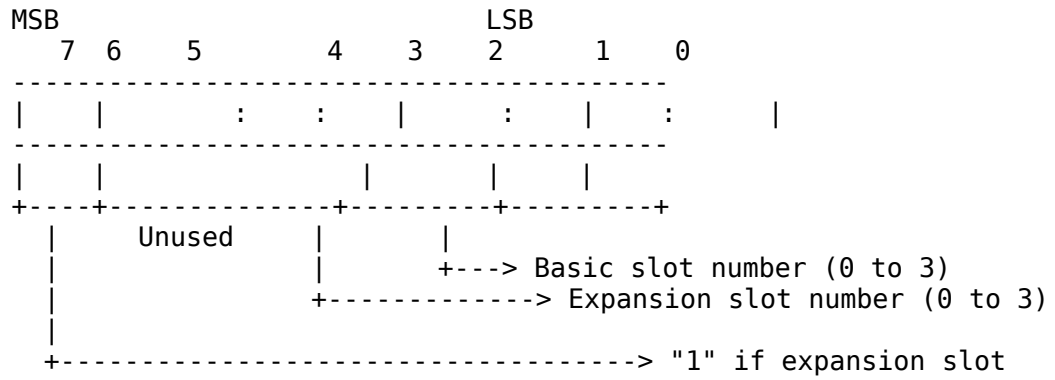
Note: to identify the kind of slot, in the case of the expansion slot, the value written is read as the reversed value. The value of this register is the same inside the basic slot.

The slot where MAIN-ROM or RAM is installed and the slot number of the slot for the cartridge depend on the machine. Refer to the appropriate manual to see how slots are used on your MSX. But the MSX standard guarantees the normal operation no matter what is in the slots, so it is not necessary to worry about the slot use, as long as you are following the standard.

In some cases, however, it is required to know the slot number of the specified software. For example, in the previous version, BASIC MAIN-ROM was placed in basic slot #0 or in expansion slot #0-0 when basic slot #0 was expanded. So when MSX1 is upgraded to have the MSX2 functions by installing MSX-VIDEO and BASIC ver 2.0 ROM, the MAIN ROM should be placed somewhere other than slot #0 or slot #0-0. The slot where MSX2 SUB-ROM resides depends on the machine, but the information about the slot where BASIC interpreter ROM resides can be obtained by referring to the work area described below (the slot information can be obtained in the format shown in Figure 5.40). When calling BIOS from DOS, examine the slot of MAIN-ROM in this way.

* EXPTBL (FCC1H, 1)   the slot of MAIN-ROM
* EXBRSA (FAF8H, 1)   the slot of SUB-ROM (0 for MSX1)

Figure 5.40 Format for the slot



When a given routine resides over page 1 and page 2 (4000H to BFFFH), the same slot for page 2 as the one for page 1 should be selected when the jump from page 1 to page 2 occurs within this routine. To do this, you need to examine the slot, in page 1, where the program resides and then to switch page 2 to that slot. To obtain information about the slot where the program currently is, execute the program shown in List 5.11.

List 5.11 Program to know the current slot

```
;*********************************************************
;  List 5.11   to know where you are
;*********************************************************
;     Suppose your program cartridge is 32K bytes
;     long (4000h..0BFFFH). You set the ID at 4000H
;     and 4001H and the execution start address within
;     page 1 (4000h..7FFFH), MSX passes control
;     to this address so the part which resides in
;     page 2 is not yet enabled at this point. You
```

```
;     have to know where you are (in what primary
;     slot, in what secondary slot) and enable the
;     part at page 2. Below is the sample program
;     to do this.
;
ENASLT      EQU   0024H       ;enable slot
RSLREG      EQU   0138H       ;read primary slot select register
EXPTBL      EQU   0FCC1H            ;slot is expanded or not

;----- program start -----

ENAP2:
      CALL  RSLREG            ;read primary slot #
      RRCA              ;move it to bit 0,1 of [Acc]
      RRCA
      AND   00000011B
      LD    C,A
      LD    B,0
      LD    HL,EXPTBL   ;see if this slot is expanded or not
      ADD   HL,BC
      LD    C,A         ;save primary slot #
      LD    A,(HL)            ;See if the slot is expanded or not
      AND   80H
      OR    C           ;set MSB if so
      LD    C,A         ;save it to [C]
      INC   HL          ;Point to SLTTBL entry
      INC   HL
      INC   HL
      INC   HL
      LD    A,(HL)            ;Get what is currently output
                        ;to expansion slot register
      AND   00001100B
      OR    C           ;Finally form slot address
      LD    H,80H
      JP    ENASLT            ;enable page 2

      END
```

==========

7.2 Inter-slot Calls (calls between slots)

As described above, programs reside in different slots, so a program not in the current slot might be needed in some cases. The most common cases are listed below:

(1) calling BIOS in MAIN-ROM from MSX-DOS level
(2) calling BIOS in SUB-ROM from BASIC level (only the case of MSX2)
(3) calling BIOS in MAIN-ROM or SUB-ROM from cartridge software

In doing such calls, to switch slots easily and safely, there is a group of BIOS routines called the inter-slot calls, which can be called from the routine in any slot. This section describes the use of the inter-slot calls.

7.2.1 Inter-slot call operation

When calling BIOS in MAIN-ROM from MSX-DOS, the state of slots changes as described below.

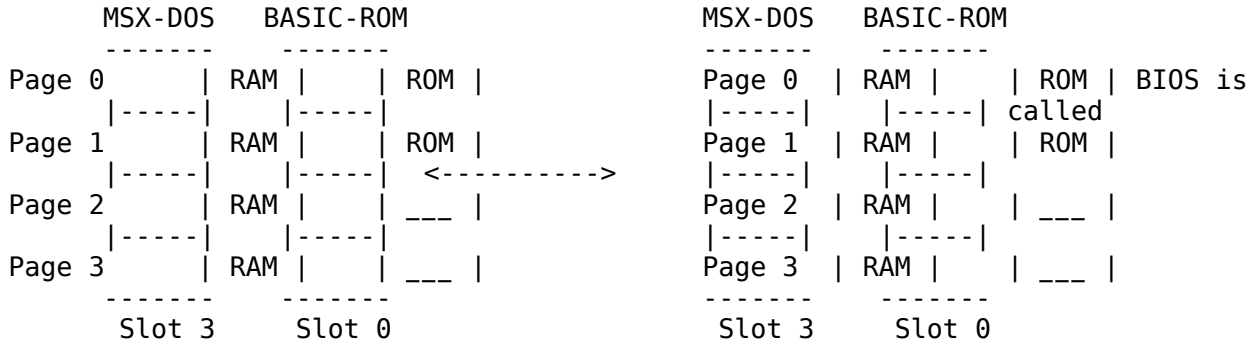
(1) Since, in the initial MSX-DOS mode, RAM is selected all over 64K address space, BASIC-ROM cannot be accessible in this state (see Figure 5.41-a).

(2) To call BIOS in ROM, switch page 0 to MAIN-ROM or BASIC to access. Then, call BIOS (see Figure 5.41-b).

(3) Restore the original status after BIOS operations and return to the initial address.

Figure 5.41 Inter-slot call

a) When using MSX-DOS



This is easily done when the program resides in other than page 0, but there can be some problem when the calling program resides in page 0 which is the same page as BIOS of the called program. Care is needed to prevent the program from disappearing itself and generating unpredictable results when it is switched to page 0. The inter-slot call settles this problem by jumping to page 3 temporarily and then switching slots.

7.2.2 Inter-slot call use

There are several ways to execute inter-slot calls, as described below. They are included in MAIN-ROM as BIOS calls. Some of them are offered in MSX-DOS, so inter-slot calls can be executed under MSX-DOS.

(1) Inter-slot call routines in BIOS

* RDSLT (000CH/MAIN) ..... read value at specified address of specified slot

```
Input:          A register <-- slot specification
            HL register <-- address to be read at
Output:     A register <-- value which has been read out
Use:        AF, BC, DE
Function:   reads the value at the specified address of the specified
            slot and stores it in the A register. The slot specification
            is done using the A register in the form shown in Figure
            5.40. At this point, when the objective slot is the basic
```

slot, set "0" to the 6 high order bits and define the slot #0 to #3 using the 2 low order bits. When specifying the expansion slot, specify the basic slot by bit 0 and bit 1 and the expansion slot by bit 2 and bit 3 and set bit 7 to "1".

* WRSLT (0014H/MAIN) ..... write value at specified address of specified slot

Input: A register <-- slot specification (same format as in Figure 5.40)
HL register <-- address to be written in
E register <-- data to be written
Output: ---
Use: AF, BC, D
Function: writes E register value in the address specified by HL register of the slot specified by the A register (the specification format is the same as in Figure 5.40).

* CALSLT (001CH/MAIN) ..... call specified address of specified slot

Input: 8 high order bits of IY register <-- slot (same format as in Figure 5.40)
IX register <-- address to be called
Output: depends on the result of the called program
Use: depends on the result of the called program
Function: calls the routine at the address specified by IX register of the slot specified by the 8 high order bits of IY register (the specification format is the same as in Figure 5.40).

* ENASLT (0024H/MAIN) ..... swicth slots

Input: A register <-- slot (same format as in Figure 5.40)
HL register <-- specifies the page to switch the slot by 2 high order bits
Output: ---
Use: all
Function: switches the page specified by the 2 high order bits of the HL register to the slot specified by the A register.

* CALLF (0030H/MAIN) ..... call specified address of specified slot

Input: specifies the slot and the address in the inline parameter format
Output: depends on the result of the called program
Use: depends on the result of the called program
Function: calls the specified address of the specified slot, but, different from CALSLT described above, the slot and the address is specified in the inline parameter format, as described below. That is, parameters are passet by one byte (same format as RDSLT) to specify that the slot is placed just after the instruction which calls CALLF and the next two bytes to specify the address are placed. "CALL 0030H"

may be replaced by the RST (restart) instruction, "RST 30H". In this case, the inter-slot call is done in 4 bytes.

Figure 5.42 Example of the inter-slot call execution

```
-------------------------------------------------------------
|                                                           |
|     RST   30H         ;interslot call                     |
|     DB    00000000B   ;select slot#0                |
|     DW    006CH       ;call address = 006CH         |
|                                                           |
-------------------------------------------------------------
```

```
* RSLREG (0138H/MAIN) ................. read the basic slot selection
                              register

Input:            ---
Output:     A register <-- value which has been read
Use:        ---
Function:   reads the contents of the basic slot selection register and
            stores it in the A register.


* WSLREG (013BH/MAIN) ................. write in the basic slot selection
                              register

Input:            A register <-- value to be written
Output:     ---
Use:        ---
Function:   writes the A register value in the basic slot selection
            register and selects the slot.


* SUBROM (015CH/MAIN) ................. call specified address in SUB-ROM

Input:            IX register <-- address to be called, PUSH IX
                        (see Appendix 2, SUB-ROM list)
Output:     depends on the result of the called program
Use:        background register and IX, IY registers are reserved
Function:   This is the routine to call BASIC SUB-ROM especially. The
            slot where SUB-ROM resides is automatically examined.
            Normally, EXTROM, described below, is used.


* EXTROM (015FH/MAIN) ................. call specified address in SUB-ROM

Input:            IX register <-- address to be called
Output:     depends on the result of the called program
Use:        background register and IY register are reserved
Function:   This is the routine to call BASIC SUB-ROM. The difference
            between this and SUB-ROM above is the point whether the
            IX register value is pushed.
```

(2) Inter-slot call in MSX-DOS

In MSX-DOS, five kinds of inter-slot calls are offered and their entry addresses are defined at jump vectors of MSX-DOS. These are the same as ones in BIOS, so refer to BIOS above for their functions or use. Note that these routines should not be used when calling routines in SUB-ROM from MSX-DOS.

* RDSLT (000CH) ..... read value at specified address of specified slot
* WRSLT (0014H) ..... write value at specified address of specified slot
* CALSLT (001CH) ..... call specified address of specified slot
* ENASLT (0024H) ..... make specified slot available
* CALLF (0030H) ..... call specified address of specified slot

List 5.12 Calling BIOS from MSX-DOS

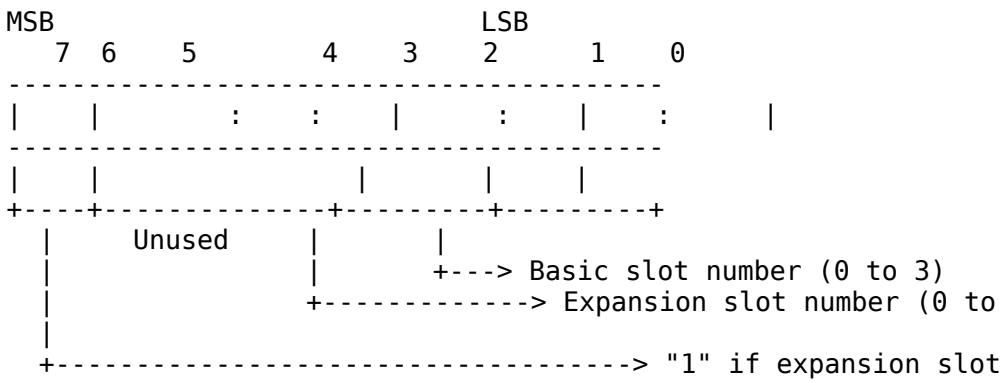
```
;*********************************************************
;  List 5.12      How to use BIOS from MSX-DOS.
;*********************************************************
;
CALSLT      EQU   001CH       ;Inter slot call
EXBRSA      EQU   0FAF0H            ;Slot address of BIOS (main) ROM
EXPTBL      EQU   0FCC1H            ;Slot address of extended ROM
;
;     LD    IY,(EXPTBL-1)     ;Load slot address of the BIOS ROM
;                         ;in high byte of IY
;     LD    IX,address of the BIOS jump table
;     CALL  CALSLT
;
;----- Sample program to set text mode -----

INITXT      EQU   006CH
LINL40      EQU   0F3AEH
;
TOTEXT: LD  B,40
      LD    A,(EXBRSA)  ;slot address of SUB-ROM
      OR    A           ;0 if MSX1
      JR    Z,TO40
      LD    B,80

TO40: LD    (LINL40),B  ;set width into work area
      LD    IX,INITXT
      LD    IY,(EXPTBL-1)     ;get expanded slot status to IYH
      CALL  CALSLT            ;perform an inter-slot call
      EI                ;because CALSLT do DI
      RET

      END
```

7.2.3 Work area to obtain the slot status

The following addresses involve the slot work area.

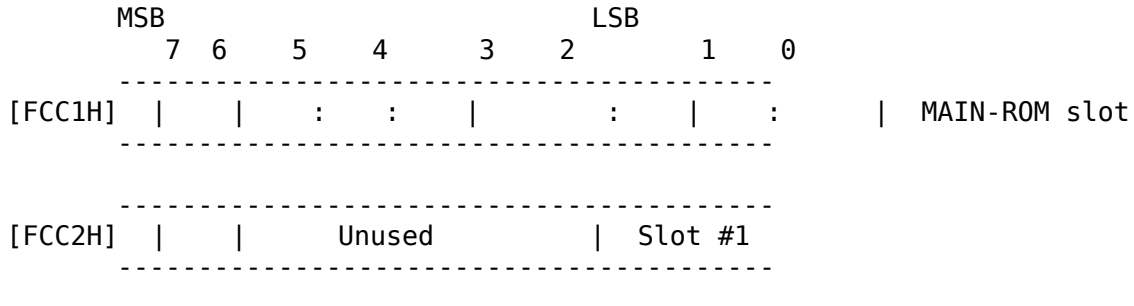
* EXBRSA (FAF8H, 1) ..... SUB-ROM slot

Figure 5.43 SUB-ROM slot



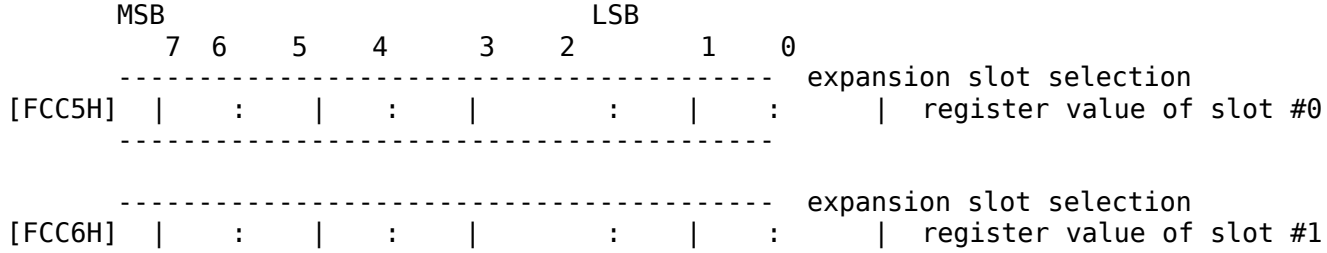
* EXPTBL (FCC1H, 4) ..... whether the basic slot is expanded or not

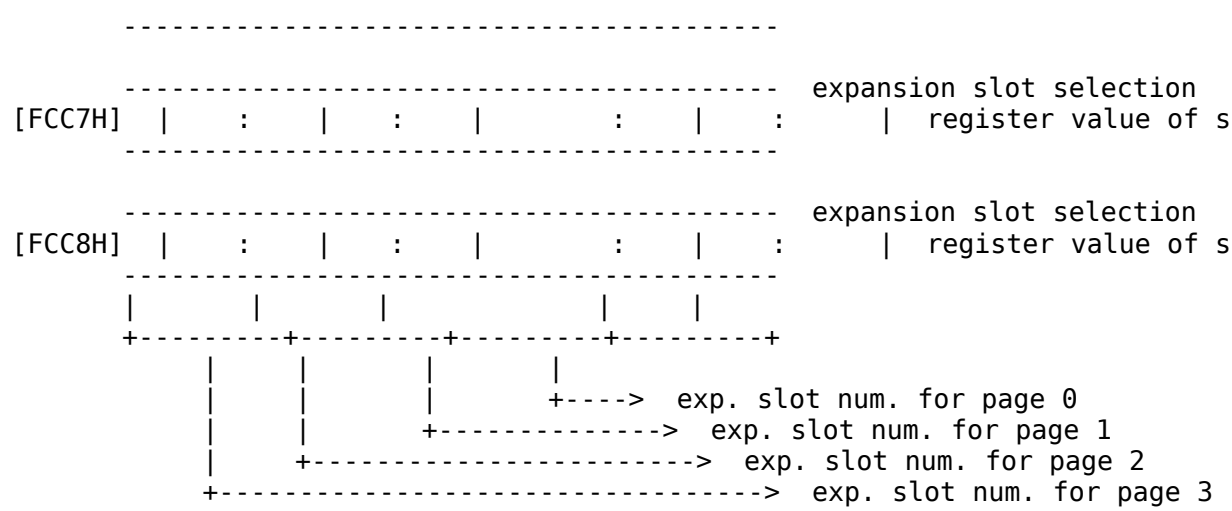
Figure 5.44 Selecting the basic slot



* SLTTBL (FCC5H, 4) ..... preservation area for the expansion slot selection register value

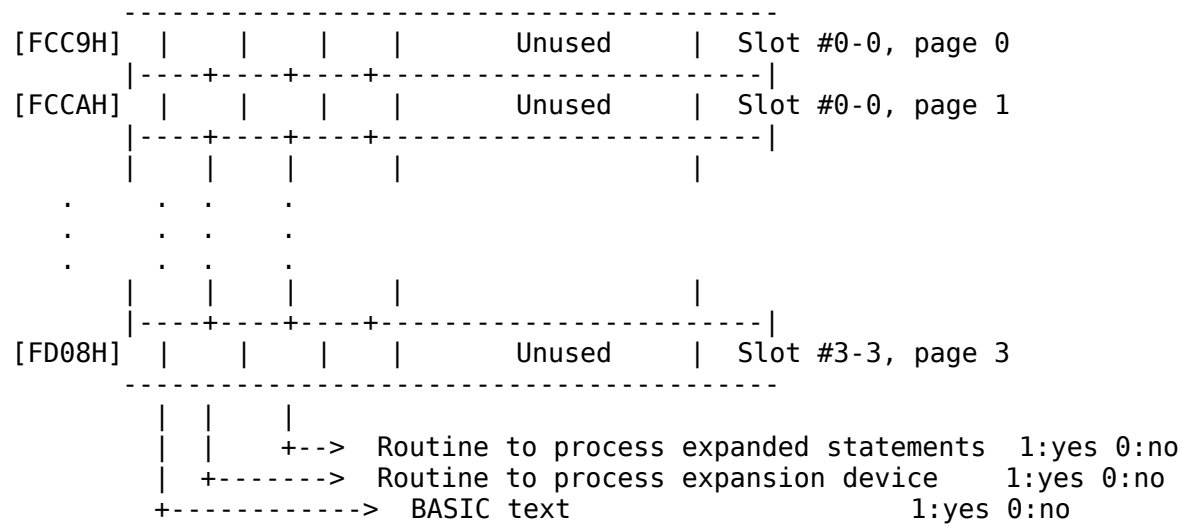
Figure 5.45 Selecting the expansion slot





* SLTATR (FCC9H, 64) ..... test for existance of application in each slot/page

Figure 5.46 Test for existence of application

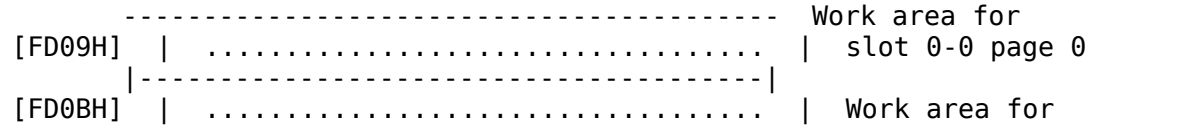


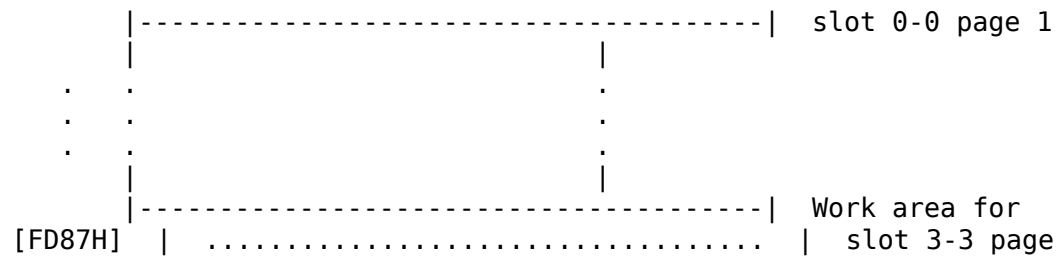
The concrete work area for a given slot and page can be obtained with the following expression:

SLTATR address = FCC9H + 16*basic slot + 4*expansion slot + page

* SLTWRK (FD09H, 128) ..... work area for application

Figure 5.47 Work area for application





The concrete work area for a given slot and page can be obtained with the following expression:

SLTWRK address = FD09H + 32*basic slot + 8*expansion slot + 2*page

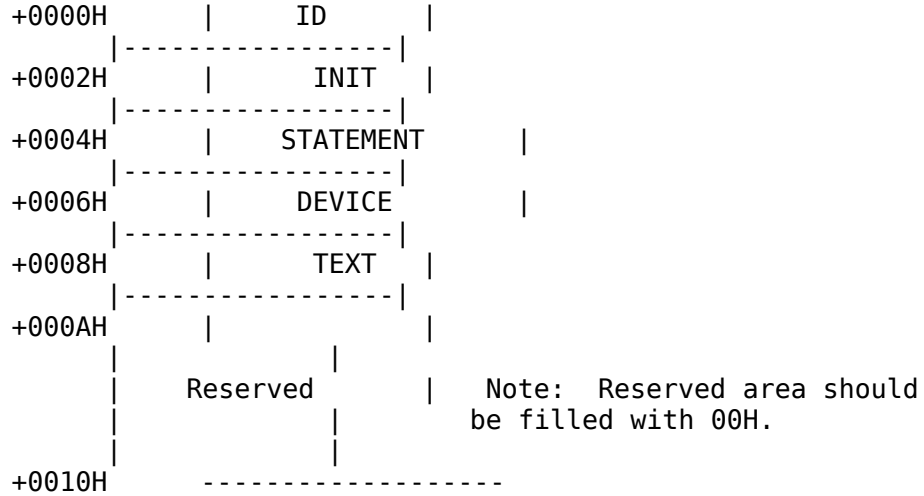
7.3 Developing Cartridge Software

MSX machines usually have at least one external slot and the hardware to be put there is called a "cartridge". There are cartridges such as the ROM cartridges for application programs or games, input-output device cartridges for a disk or RS-232 interface, RAM expansion cartridges for expanding RAM, and slot expansion cartridges for expanding slots. These cartridges make the MSX easy to upgrade. BASIC and assembly language programs can also be stored in ROM cartridge easily. This section describes how to develop cartridge software.

7.3.1 Catridge header

MSX cartridges have a 16-bye common header and, when the system is reset, the cartridge is initialised by the information written in this header. For ROM cartridges of BASIC or assembly language programs, they can be automatically started by using the information written in the header. Figure 5.48 shows the cartridge header configuration.

Figure 5.48 Program cartridge header



* ID

In the case of ROM cartridges, these two bytes have codes "AB" (41H, 42H). For SUB-ROM cartridges, the ID is "CD".

* INIT

When the cartridge is made to initialise the work area or I/O, these two bytes are the addresses for the initialization routine; otherwise 0000H is assumed. After instructions such as getting the work area in the initialization routine are placed, end with "RET". All registers except the SP register may be destroyed. For assembly language programs, such as games, which loop within the cartridge, it is possible to execute the object program from here.

* STATEMENT

When the cartridge is made to expand the CALL statement, these two bytes are the address for the statement expansion routine; otherwise 0000H is assumed. If so, the statement expansion routine should reside at 4000H to 7FFFH.

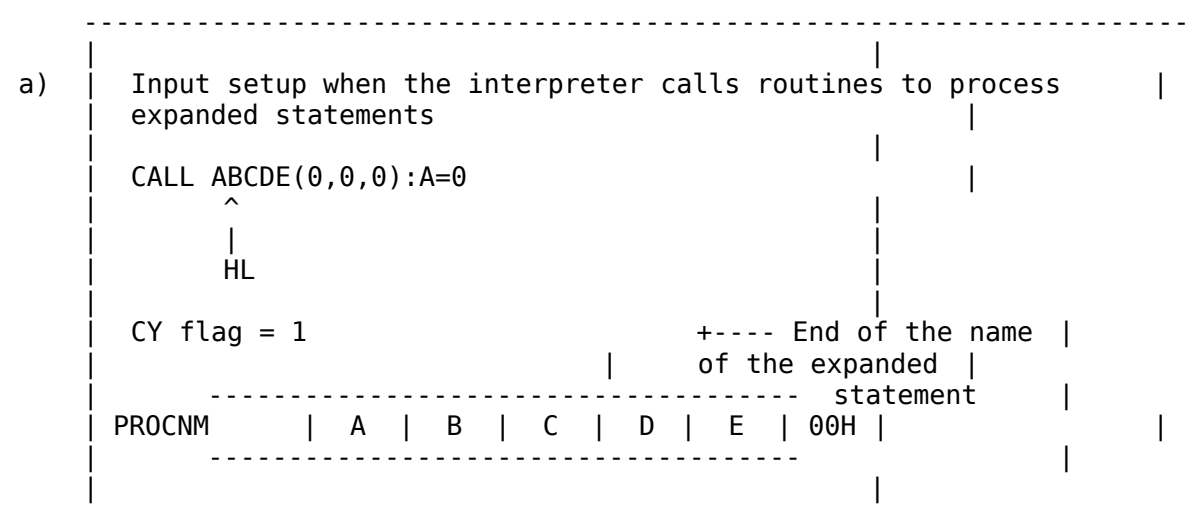
The CALL statement is described in the following format:

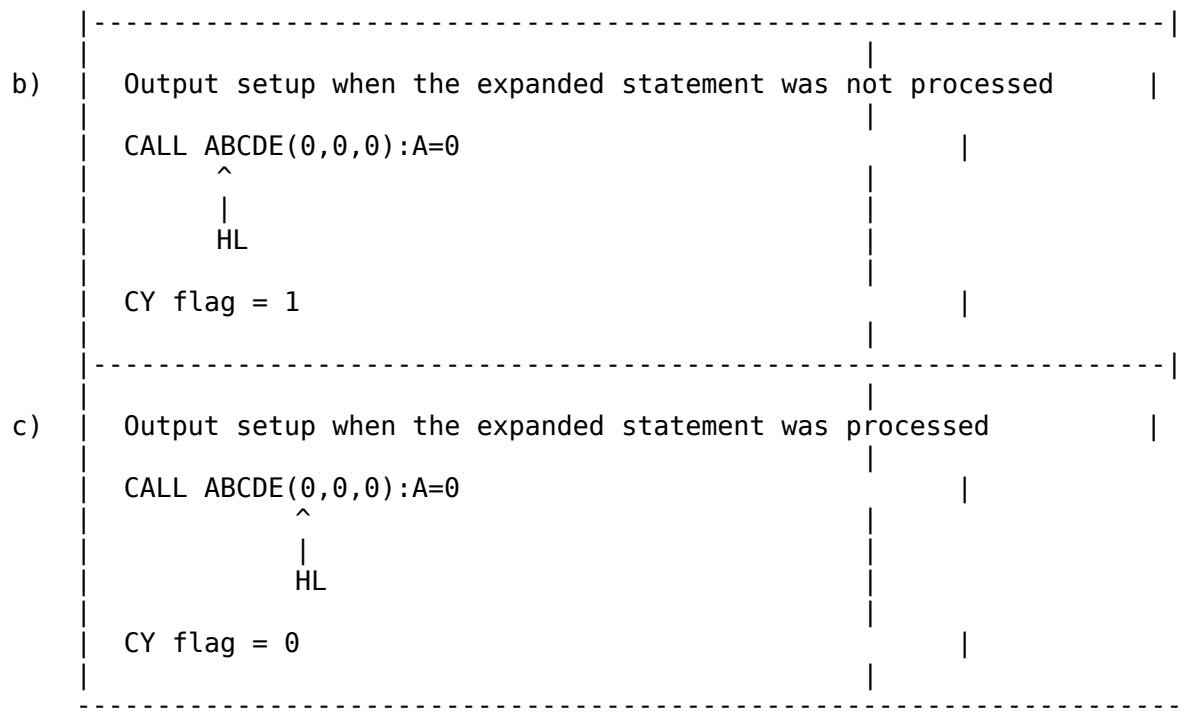
CALL <expression statement name> [(<argument>[, <argument>...])]

The expression statement name can have up to 15 characters. As an abbreviation for CALL, "_" (underscore) is available.

When the BASIC interpreter finds a CALL statement, it puts the expansion statement name in PROCNM (FD89H, 16) in the work area and passes the control to cartridges, whose contents of STATEMENT of the header is other than 0, in the order from the one with the smaller slot number. The HL register points to the text address next to the expansion statement name at this point (see Figure 5.49a).

Figure 5.49 Input-output of the operation routine of the expansion statement





To develop the statement expansion routine, recognise the name of the expansion statement written in PROCNM first, then return with setting "1" to the carry flag without modifying the HL register if the statement is not to be handled (see Figure 5.49b); otherwise, handle it properly and set the HL register (text pointer) to the next handled statement (where 00H or 3AH is placed usually), then return after setting "0" to the carry flag (see Figure 5.49c).

The BASIC interpreter determines the status of the carry flag whether a CALL statement has been executed, and, if not, calls the next cartridge. When all cartridges have not executed the statement (when the carry flag has been "1" all the time), it displays "SYNTAX ERROR". To test arguments of the statement, it is convenient to use "internal routines for the statement expansion" in section 4.4 of chapter 2.

* DEVICE

These two bytes are the addresses of the device expansion routine, when the cartridge does the device expansion (the input-output device expansion); otherwise 0000H is used. When doing the device expansion, the device expansion routine should be in 4000H-7FFFH. One cartridge can have up to 4 devices. The name of the expansion device should be less than 16 characters.

When the BASIC interpreter finds an undefined device, it stores that in PROCNM (FD89H, 16) and put FFH in the A register and passes control to the cartridge whose contents is not 0 in the order from the one with the smaller slot number (see Figure 5.50a).

When creating device expansion routines, identify the file descriptor of PROCNM first, and, when it is not for the device to be processed, return with setting 1 to the carry flag (see Figure 5.50b). Otherwise, process it and set the device ID (0-3) in the A register, then return with setting 0 to the

carry flag (see Figure 5.50c).

The BASIC interpreter determines by the status of the carry flag whether or not it is processed, and, if not, call the next cartridge. When all cartridges were not processed (that is, when the carry flag was "1" all the time), "Bad file name ERROR" is displayed.

When the actual input-output operations are done, the BASIC interpreter sets the device ID (0-3) in DEVICE (FD99H) and sets the request to the device in the A register (see Table 5.6), then calls the device expansion routine. The device expansion routine should refer to it to handle the request.

Figure 5.50 Input-output to the device expansion routine

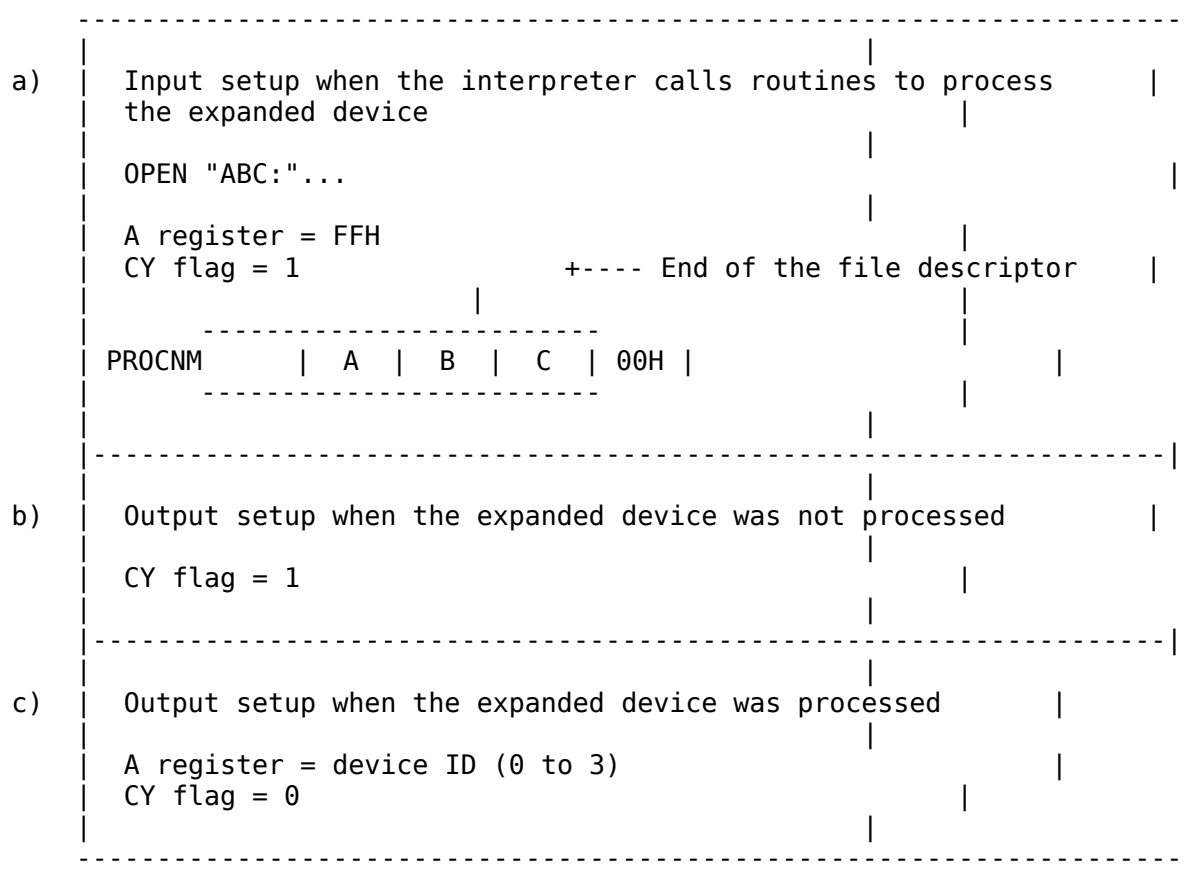


Table 5.6 Requests to the device

Register A	Request
0	OPEN
2	CLOSE
4	Random access
6	Sequential output

8	Sequential input
10	LOC function
12	LOF function
14	EOF function
16	FPOS function
18	Backup character

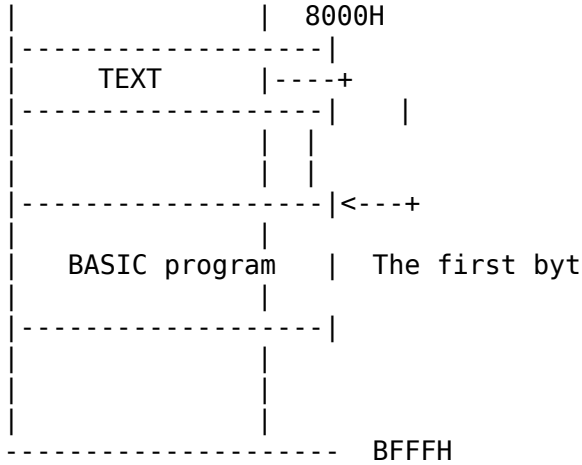
* TEXT

These two bytes are text pointers of the BASIC program, when the BASIC program in the cartridge would be auto-started (executed at reset); otherwise they are 0000H. The size of the program must be under 16K bytes, 8000H to BFFFH.

The BASIC interpreter examines the contents of TEXT of the header after the initialization (INIT) and after the system is started. When they are not 0000H, it begins the execution from the address as BASIC text pointer (see Figure 5.51). BASIC programs should be stored in the form of the intermediate code and the beginning of it (the address pointed by TEXT) must be 00H, which indicates the beginning of the program.

The execution speed of the program will be improved when the objective line number of statements such as GOTO is the absolute address of the objective text pointer.

Figure 5.51 Executing BASIC program cartridge



* How to place BASIC programs in ROM

1. Change the starting address of BASIC text to 8021H.

```
POKE &HF676,&H21 : POKE &HF677,&H80 : POKE &H8020,0 : NEW
```

Note: these statements must be in one line

2. Load the objective BASIC program.

```
LOAD "PROGRAM"
```

3. Create ID.

```
AD = &H8000
FOR I = 0 TO 31    ----+
 POKE AD + I, 0        |  clears ID area
NEXT I               ----+
POKE &H8000,ASC("A")
POKE &H8001,ASC("B")
POKE &H8008,&H20
POKE &H8009,&H80
```

4. Put 8000H to BFFFH in ROM.

7.3.2 notes on the creation of the cartridge software

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.

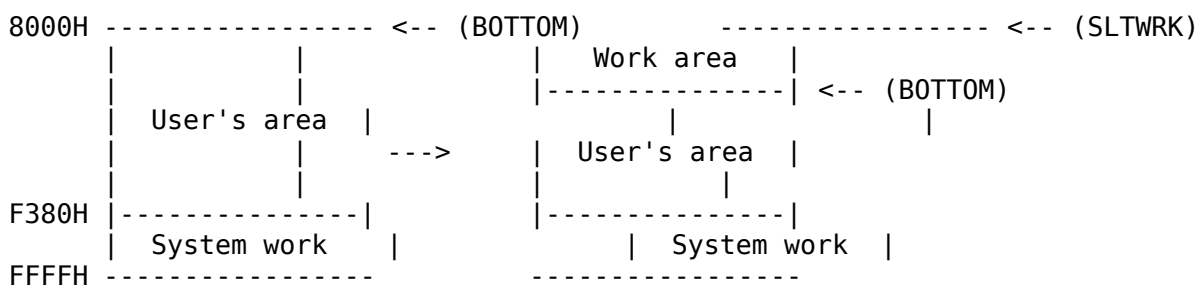
But in programs which are executed by using BASIC interpreter functions, the same area cannot be shared as the work area. To do this, there are three methods:

(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 to ...) as the work area.

(3) When more than three bytes are needed for the work area, allocates it from RAM used by BASIC. To do this, put the contents of BOTTOM (FC48H) to the area corresponding to SLTWRK (FD09H to ...), and increase the value of BOTTOM by the needed work area, then allocate it for the work area (see figure 5.52).

Figure 5.52 Allocating the work area



See the following list for the reference of (2) and (3).

List 5.13 Example of allocating the work area
==========

```
;*******************************************************
;  List 5.13  subroutines to support slot
;            for ROM in 1 page
;*******************************************************

RSLGREG EQU 0138H
EXPTBL      EQU   0FCC1H
BOTTOM      EQU   0FC48H
HIMEM EQU   0FC4AH
SLTWRK      EQU   0FD09H

;-------------------------------------------------------
;
;     GTSL1 Get slot number of designated page
;     Entry None
;     Return      A     Slot address as follows
;     Modify      Flags
;
;         FxxxSSPP
;         | ||||
;         | ||++-- primary slot # (0-3)
;         | ++---- secondary slot # (0-3)
;         |        00 if not expanded
;         +--------- 1 if secondary slot # specified
;
;     This value can later be used as an input parameter
;     for the RDSLT, WRSLT, CALSLT, ENASLT and 'RST 10H'
;
      PUBLIC      GTSL10
GETSL10:
      PUSH  HL          ;Save registers
      PUSH  DE

      CALL  RSLREG            ;read primary slot #
      RRCA
      RRCA
      AND   11B         ;[A]=000000PP
      LD    E,A
      LD    D,0         ;[DE]=000000PP
      LD    HL,EXPTBL
      ADD   HL,DE       :[HL]=EXPTBL+000000PP
      LD    E,A         ;[E]=000000PP
      LD    A,(HL)            ;A=(EXPTBL+000000PP)
      AND   80H         ;Use only MSB
      JR    Z,GTSL1NOEXP
      OR    E           ;[A]=F00000PP
      LD    E,A         ;save primary slot number
      INC   HL          ;point to SLTTBL entry
      INC   HL
      INC   HL
```

```
      INC   HL
      LD    A,(HL)            ;get current expansion slot register
      RRCA
      RRCA
      AND   11B         ;[A] = 000000SS
      RLCA
      RLCA              ;[A] = 0000SS00
      OR    E           ;[A] = F000SSPP
;
GTSL1END:
      POP   DE
      POP   HL
      RET
GTSL1NOEXP:
      LD    A,E         ;[A] = 000000PP
      JR    GTSL1END


;-----------------------------------------------------
;
;     ASLW1 Get address of slot work
;     Entry None
;     Return      HL    address of slot work
;     Modify      None
;
      PUBLIC      ASLW10
ASLW10:
      PUSH  DE
      PUSH  AF
      CALL  GTSL10            ;[A] = F000SSPP, SS = 00 if not expanded
      AND   00001111B   ;[A] = 0000SSPP
      LD    L,A         :[A] = 0000SSPP
      RLCA
      RLCA
      RLCA
      RLCA              ;[A] = SSPP0000
      AND   00110000B   ;[A] = 00PP0000
      OR    L           ;[A] = 00PPSSPP
      AND   00111100B   ;[A] = 00PPSS00
      OR    01B         ;[A] = 00PPSSBB
;
;     Now, we have the sequence number for this cartridge
;     as follows.
;
;     00PPSSBB
;       ||||||
;       ||||++-- higher 2 bits of memory address (1)
;       ||++---- seconday slot # (0..3)
;       ++------ primary slot # (0..3)
;
      RLCA              ;*=2
      LD    E,A
      LD    D,0         ;[DE] = 0PPSSBB0
      LD    HL,SLTWRK
      ADD   HL,DE
      POP   AF
      POP   DE
      RET
```

```
;-------------------------------------------------------
;
;     RSLW1 Read slot work
;     Entry None
;     Return      HL    Content of slot work
;     Modify      None
;
      PUBLIC      RSLW10
RSLW10:
      PUSH  DE
      CALL  ASLW10            ;[HL] = address of slot work
      LD    E,(HL)
      INC   HL
      LD    D,(HL)            ;[DE] = (slot work)
      EX    DE,HL       ;[HL] = (slot work)
      POP   DE
      RET


;-------------------------------------------------------
;
;     WSLW1 Write slot work
;     Entry HL    Data to write
;     Return      None
;     Modify      None
;
      PUBLIC      WSLW10
WSLW10:
      PUSH  DE
      EX    DE,HL       ;[DE] = data to write
      CALL  ASLW10            ;[HL] = address of slot work
      LD    (HL),E
      INC   HL
      LD    (HL),D
      EX    DE,HL       ;[HL] = data tow write
      POP   DE
      RET


;-------------------------------------------------------
;
; How to allocate work area for cartridges
; If the work area is greater than 2 bytes, make the SLTWRK point
; to the system variable BOTTOM (0FC48H), then update it by the
; amount of memory required. BOTTOM is set up by the initization
; code to point to the bottom of equipped RAM.
;
;        Ex, if the program is at 4000H..7FFFH.
;
;     WORKB allocate work area from BOTTOM
;           (my slot work) <- (old BOTTOM)
;     Entry HL    required memory size
;     Return      HL    start address of my work area = old BOTTOM
;                 0 if cannot allocate
;     Modify      None
;
```

```
      PUBLIC      WORKB0
WORKB0:
      PUSH  DE
      PUSH  BC
      PUSH  AF

      EX    DE,HL       ;[DE] = Size
      LD    HL,(BOTTOM) ;Get current RAM bottom
      CALL  WSLW10            ;Save BOTTOM to slot work
      PUSH  HL          ;Save old BOTTOM
      ADD   HL,DE       ;[HL] = (BOTTOM) + SIZE
      LD    A,H         ;Beyond 0DFFFH?
      CP    0E0H
      JR    NC,NOROOM   ;Yes, cannot allocate this much
      LD    (BOTTOM),HL ;Updtae (BOTTOM)
      POP   HL          ;[HL] = old BOTTOM
WORKBEND:
      POP   AF
      POP   BC
      POP   DE
      RET
;
;     BOTTOM became greater than 0DFFFH, there is
;     no RAM to be allocated.
;
NOROOM:
      LD    HL,0
      CALL  WSLW10            ;Clear slot work
      JR    WORKBEND    ;Return 0 in [HL]

      END
```

==========

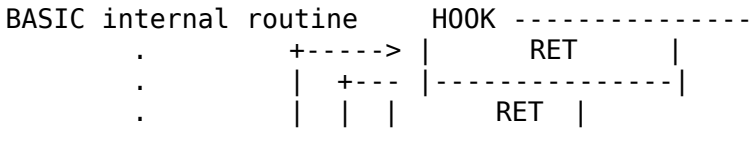
* Hook

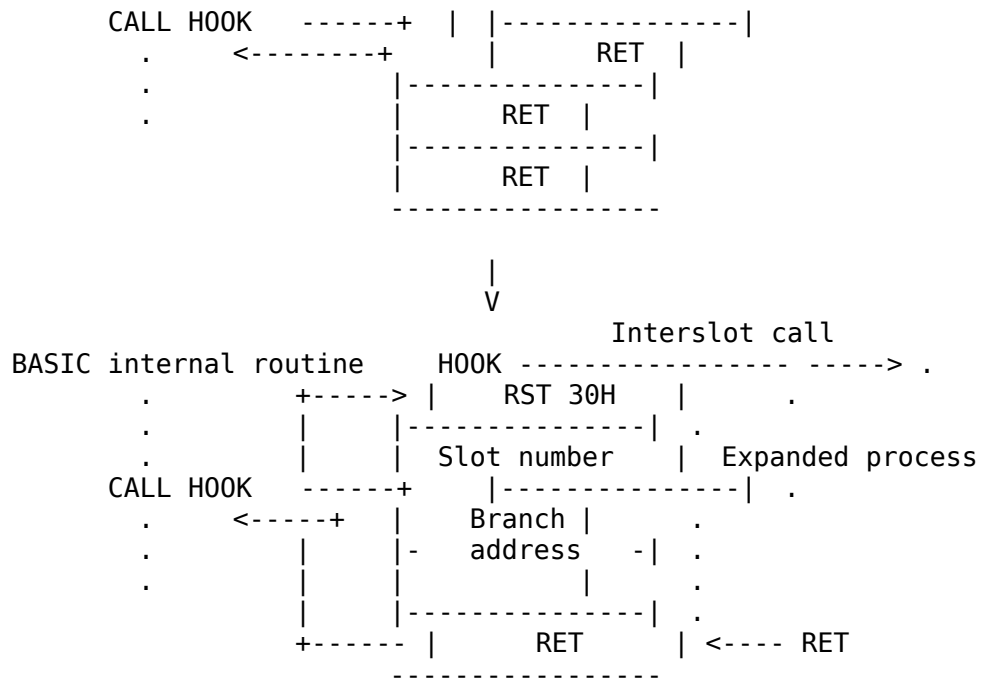
The area called "hook" is used for expanding BASIC functions in FD9AH to FFC9H of the work area used by MSX-BASIC. One hook has five bytes, which are normally "RET".

When MSX-BASIC does a certain operation (such as the one in the description about the hook of the work area), it calls this hook from there once. When the hook is "RET", the control returns immediately; but the function of BASIC can be expanded, when these five bytes were re-written to do the inter-slot call to the program inside the cartridge by the initialization routine (INIT) (see Figure 5.53).

List 5.14 shows an example of the program that the cartridge uses to hook H.KEYI for the timer interrupt ptocess.

Figure 5.53 Setting the hook





List 5.14 Using the hook

```
;*********************************************************
;  List 5.14  Sample program to use HOOK
;*********************************************************
;
; Start-up initialize entry
; This program will be called when system initializing.
;
H.KEYI      EQU   0FD9AH            ; interrupt hook
EXPTBL      EQU   0FCC1H            ; slots expanded or not
PSLTRG      EQU   0A8H        ; I/O port address of primary slot register
EXT   MYINT

INIT:
;     <<< Please insert other initialization routine here, if you need. >>>

; Set interrupt entry

      DI                ; start of critical region

; Get old interrupt entry inter-slot call hook

      LD    DE,OLDINT   ; get address of old int. hook saved area
      LD    HL,H.KEYI   ; get address of interrupt entry hook
      LD    BC,5        ; lenght of hook is 5 bytes
      LDIR              ; transfer

; Which slot address is this cartridge placed?

      CALL  GETMSLT     ; get my slot address

; Set new inter-slot call of interrupt entry
```

```
      LD    (H.KEYI+1),A      ; set slot address
      LD    A,0F7H            ; 'RST 30H' inter-slot call operation code
      LD    (H.KEYI),A  ; set new hook op-code
      LD    HL,INTENT   ; get our interrupt entry point
      LD    (H.KEYI+2),HL     ; set new interrupt entry point
      LD    A,0C9H            ; 'RET' operation code
      LD    (H.KEYI+4),A      ; set operation code of 'RET'
      EI                ; end of critical region
      RET


;-----------------------------------------------------
; Which slot address is the cartridge placed?
;  Entry: No
;  Action: Compute my slot address
;  Return: A = slot address
;  Modify: Flag

GTMSLT:
      PUSH  BC          ; save environment
      PUSH  HL
      IN    A,(PSLTRG)  ; read primary slot register
      RRCA              ; move it to bit 0,1 of A
      RRCA
      AND   00000011B   ; get bit 1,0
      LD    C,A         ; set primary slot No.
      LD    B,0
      LD    HL,EXPTBL   ; see if the slot is expanded or not
      ADD   HL,BC
      OR    (HL)        ; set MSB if so
      LD    C,A
      INC   HL          ; point to SLTTBL entry
      INC   HL
      INC   HL
      INC   HL
      LD    A,(HL)            ; get what is currently output to
                        ; expansion slot register

      AND   00001100B   ; get bits 3,2
      OR    C           ; finally form slot address

      POP   HL          ; restore environment
      POP   BC
      RET               ; return to main

;----- Interrupt entry -----

INTENT:
      CALL  MYINT       ; call interrupt handler
      JP    OLDINT            ; go to old interrupt handler

;----- HOOK save area -----

OLDINT: DS  5

      END
```

==========

* Stack pointer initialisation

When MSX has an internal disk, sometimes the disk interface ROM does the initialisation before the cartridge does, depending on the slot location, and pushes down the stack pointer in the direction of the low order address to allocate the work area. In this case, software not using the disk should set the stack pointer again after the cartridge received control; otherwise, the stack area might be exhausted and a system crash might occur. Remember to initialise the stack pointer at the beginning of the program.

* Testing the preformance of the expansion slot

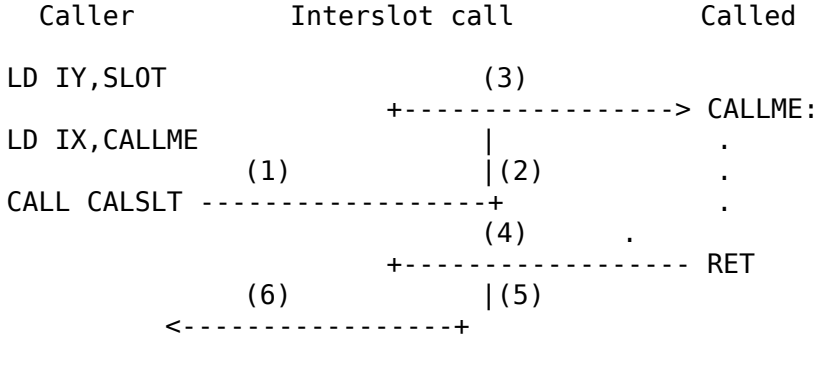
When general software in the market is put in the expansion slot or when RAM resides in the expansion slot, sometimes the application program do not work. Since most MSX2 machines use the expansion slot inside, problems may sometime result. Software to be sold in the market should be thoroughly tested in both cases that it is put in the expansion slot and that RAM resides in expansion slot.

Since the expansion slot register is placed in FFFFH, do not use it as if it were RAM. For example, setting the stack in FFFFH using "LD SP, 0" in the program causes machines using the expansion slot to go out of control.

* Notes on CALSLT use

Executing the inter-slot call in CALSLT and CALLF destroys the contents of IX, IY, and the background processing register. When returning from this routine, in MSX1 the interrupt is inhibited, but in MSX2 the state before the call is restored.

When using CALSLT or CALLF to execute the inter-slot call, the interrupt is always inhibited when calling the object program (see 2 in the figure below) and when returning to the calling program (see 6 below).



In MSX2, the state of the interrupt is reserved before and after the inter-slot call. That is, 3 in the figure is in the same state as 1, and 6 is in the same state as 4. Note when the called program executes "EI" or "DI".