MSX Article

The MSX Memory (I)

Summary

This article aims at describing how the MSX memory works, showing how the ROMs and RAMs share only 64 KB memory space. This is the first of three articles, which addresses the primary slots. The second article will analyze the secondary slots, Memory Mapper and Megaram. The last article proposes some memory experiments on MSX.

1- Introduction

We all know that MSX can only address 64 KB of memory. But if the MSX ROM BIOS has 32 KB and the RAM 64 KB, isn't it addressing more than 64 KB? In addition, if we insert a game or application cartridge with 16 or 32 KB, doesn't the memory amount still raising?

The answer is yes. So, in this case, is the Z-80 processor able to access more than 64 Kb? Yes, it is possible, but not directly. In fact, Z-80 is able only to address 64 KB due to the memory addressing schema of 16 bits. According to that, it is only possible to represent numbers between 0 and 65535, or, 64 KB.

In order to access more than 64 KB, we must have up to four memory banks attached on the system, limited to 64 KB each. So, if we want to access a given memory bank, we have to switch the active memory to the desired one. For example, the Z-80 stops accessing the ROM and starts to read/write on the RAM. Nevertheless, the memory changing is not applied to the whole memory, but only to a part of it.

MSX engineers didived the whole memory into 4 parts, called pages, having all the same size. Now, each page can access a different memory bank, resulting on a more flexibile memory switching system. Figure 1 shows the MSX page system.

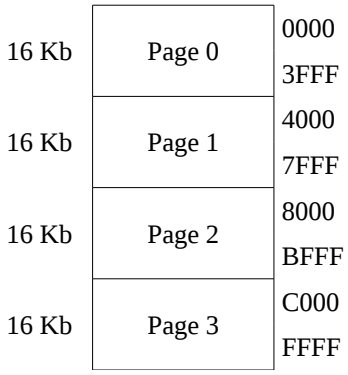

Figure 1. MSX memory division into pages.

The pages are numbered from 0 to 3, each one having 16 KB. The initial and final addresses of each page is described on figure 1.

As told before, it is possible to to share up to four memory banks limited to 64 each (although it is possible to share more memories using the subslots system, described on the second article). Each memory bank is called slot.

The Brazilian Gradiente Expert MSX slot and page configuration is shown on the figure 2. The shadowed areas indicates the pages used by each memory. This configuration varies from other MSX systems, but always having BIOS on slot 0.

	Slot 0 ROM - BIOS	Slot 1 Cartridge A	Slot 2 RAM	Slot 3 Cartridge B	
16 Kb	Page 0	Page 0	Page 0	Page 0	0000 - 3FFF
16 Kb	Page 1	Page 1	Page 1	Page 1	4000 - 7FFF
16 Kb	Page 2	Page 2	Page 2	Page 2	8000 - BFFF
16 Kb	Page 3	Page 3	Page 3	Page 3	C000 - FFFF

Figure 2. MSX memory division into slots and pages.

A memory bank neither have to use all 64 KB nor start on the page 0. An application or game cartridge, for example, starts on the page 1 and have 16 or 32 KB.

2- Memory switching

Once we are able to select the active memory on each page, we may have, for example, the pages 0 and 1 acessing the ROM on slot 0 and the page 2 and 3 acessing the RAM on slot 3. The PPI is the responsible for memory switching, telling the Z-80 what it will see on each page. The port &HA8 port configures each page, as seen on figure 3.

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Page	3		2		1		0	

Figure 3. Port &HA8 data configuration.

Each page is represented by a 2-bit value, which contains the number of the active slot. The next example will show how to configure the port A8.

When the Brazilian Expert MSX starts, the memory is configured as shown on the figure 4. The pages 0 and 1 are accessing the ROM on slot 0, and the pages 3 and 4 are accessing the RAM on slot 2. The shadowed areas detaches the current memory accessed by the Z-80 processor.

	Slot 0 ROM - BIOS	Slot 1 Cartridge A	Slot 2 RAM	Slot 3 Cartridge B	
16 Kb	Page 0	Page 0	Page 0	Page 0	0000 - 3FFF
16 Kb	Page 1	Page 1	Page 1	Page 1	4000 - 7FFF
16 Kb	Page 2	Page 2	Page 2	Page 2	8000 - BFFF
16 Kb	Page 3	Page 3	Page 3	Page 3	C000 - FFFF

Figure 4. Brazilian Expert MSX initial configuration.

According to the presented configuration, we have:

Bit	7	6	5	4	3	2	1	0
Binary value	1	0	1	0	0	0	0	0
Slot	2		2		0		0	
Page	3		2		1		0	

So, the value sent to the port &HA8 is 160 (&B10100000).

If we read the value on port &HA8, we get the current slots configuration. If we write, we will change the slots configuration.

3- Some care when changing slots

By the time we change the active slot on a memory page, all the region related to this page starts to access another memory bank. Although the previous memory content is not really lost, it turns unaccessible after the memory switching. According to that, we cannot change the page where the current program is executing or we will loose the program's control.

For illustrating that, suppose two programs in Basic located on the same page, but in different slots. Suppose also that the stack pointer (SP) points to a Basic line instead of pointing to a memory address.

Program on slot 2	Program on slot 1
8000 PRINT"I will change the slot" 8010 OUT &HA8, &B10010000 8020 END	8000 INPUT"What is your name";N$ 8010 INPUT"How old are you";A 8020 PRINT"Name: ";N$ 8030 PRINT"Age: ";A 8040 END

Both programs are located on the same memory region. After start running the program on slot 2, the following executing flow would be resulted, marked as blue.

Program on slot 2	Program on slot 1
8000 PRINT"I will change the slot" 8010 OUT &HA8, &B10010000 8020 END	8000 INPUT"What is your name";N$ 8010 INPUT"How old are you";A 8020 PRINT"Name: ";N$ 8030 PRINT"Age: ";A 8040 END

The program located on the slot 1 gained the control, while the program located on the slot 2 lost the control.

If we fall on the situation above, the solution is to jump the program's execution to a page different from that one we want to change the slot, and then change the slot. In this case, the control is not lost. That strategy was successfully applied on Slot View [1].

Program on slot 2	Program on slot 1
8000 PRINT"I will change the slot" 8010 GOTO C000 8020 END	8000 INPUT"What is your name";N$ 8010 INPUT"How old are you";A 8020 PRINT"Name: ";N$ 8030 PRINT"Age: ";A 8040 END
C000 OUT &HA8, &B10010000 C010 END	

4- Iter-slot calls

The BIOS is the native MSX operating system, which contains a lot of ready-made instructions designed to basic hardware management and MSX peripherals communications such as writing on screen, generate sounds, control joystick etc. The BIOS prevents the developer from coding complexes programs, once most of necessary code is ready to be used. The BIOS is located at page 0, slot 0.

The Basic interpreter is located on the pages 0 and 1, slot 0, and it used together with the BIOS when the Basic mode is active. The Basic interpreter is not necessary for an Assembly program, except when some specific calls to the Basic interpreter is performed by this program. So, the page 1 is free for programs written in Assembly. The page 0 can also be used by an Assembly program, but it may experience some problems if some BIOS calls are made.

The MSX BIOS has routines designed to perform secure and efficient calls to programs located on another slot. These are the inter-slots calls.

The most common inter-slot calls are [2]:

- The BIOS is called from MSX-DOS.
- The BIOS is called from SUB-ROM in Basic (MSX 2)
- The BIOS is called from a cartridge.

MSX-DOS sets all four pages to use the RAM. In that case, BIOS access is not possible. In order to access the BIOS, we must change the page 0 to the slot containing the BIOS and then perform the subroutine call. When finished, we must return to the original slots configutration.

This operation is quite easy when the current execution line is loacted on a page different from 0. Nevertheless, we may experience some problems when our program is located on the page 0, the same page used by the BIOS. As seen before, the correct BIOS access in this situation demands a lot of care. The inter-slot calls solve this problem for us, while it jumps the execution to the page 3 to the temporary slot changing. Some of inter-slot calls are included on the MSX-DOS [2].

Some inter-slot call examples are [4]: RDSLT (&H000C), WRSLT (&H0014), CALSLT (&H001C) and ENASLT (&H0024).

5- How does a ROM type cartridge work

MSX computers have, at least, one external slot and the hadware attached there is called cartridge. The cartridge types are [2]:

- ROM – used by game and applications.
- I/O – as disk-drives or RS-232 interface.
- RAM – RAM memory expansion.
- Expansion – devices used to expand the number of cartridges.

Generally the games developed in Assembly starts at the memory address &H4000 (page 1). When the cartridge contains a program in Basic, it starts at the address &H8000 (page 2).

The ROM file is composed by a header and the software content:

Header	Software content

The header has 16 bytes and describes a ROM format cartridge. So, when the system starts, the Basic interpreter analyzes the cartridge's header and then jumps to the program located inside the ROM, based on these informations. The header is discribed on the figure 5.

2 bytes	ID	+ 0000
2 bytes	INIT	+ 0002
2 bytes	STATEMENT	+ 0004
2 bytes	DEVICE	+ 0006
2 bytes	TEXT	+ 0008
6 bytes	Reserved	+ 000A

Figure 5. cartridge ROM *Header*.

Header description [2]:

- ID – in case of ROM type, the identification is “AB” (&H41, &H42). For the SUB-ROM type cartridges, the identification is “CD”.
- INIT – contains the initial address of the routine to be executed, when the cartridge is auto-started. Otherwise, this value is 0.
- STATEMENT – when the cartridge has a CALL function to start a routine from the Basic, these two bytes are the address for such routine. Otherwise, this value is 0.
- DEVICE – initial address for the device expansion routine. Otherwise, the value is 0.
- TEXT – pointer to the program in Basic when he cartridge is auto-started. Otherwise, the value is 0.

6- Loading ROMs from Basic

In the previous section, we saw that ROM type cartridges programs in Assembly are located on page 1. The question now is how to load these binary programs from the Basic, once this program must run on the page 1? We must take in account that Basic sets the pages 0 and 1 as ROM.

The solution is to copy the ROM block to the page 1 and then start the program, but first of all we have to change the page 1 to RAM. Once this is not achieved easily in Basic, we must write a program in Assembly to change the page 2 to Assembly, copy the ROM to that place and then start the program.

We may add such program at the bottom of the ROM block and set the starting execution address to those indicated by the ROM's header. Figures 6 and 7 ilustrates this proccess. On figure 6, the ROM block (yellow) plus the program (green) are loaded on the page 2, and the program responsible for all those tasks (green) starts to execute.

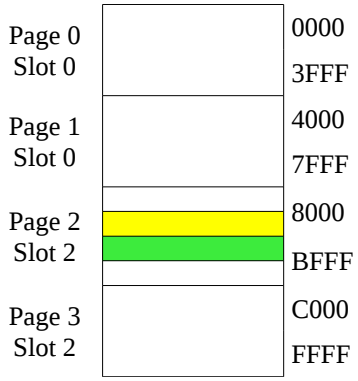

Figure 6. Loading ROM on page 1.

On figure 7, the ROM block is copied to the page 1 (RAM) and then initialized.

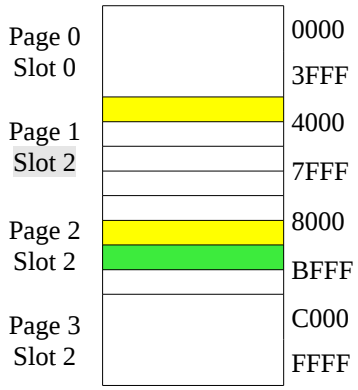

Figure 7. Process finished.

7- Credits

This article was originally written in Portuguese and translated into English by Marcelo Silveira, Systems and Computer Engineer, graduated at Universidade do Estado do Rio de Janeiro, Brazil.

Original written on: May, 2004.
Revised on: July, 2017.
E-mail: flamar98@hotmail.com
Homepage: http://marmsx.msxall.com

References:

[1] – Slot View, MarMSX Development, em http://marmsx.mxsall.com
[2] – MSX 2 Technical Handbook, ASCII Corporation, 1987.
[3] – O Livro Vermelho do MSX, editora Mc Graw Hill.
[4] – MSX Top Secret 2, Edison Pires Moraes, 2004.