MSX Supervised by magazine editorial department

MSX turbo R Technical Hand Book

MSX turbo R Technical handbook

ASCII Publishing Bureau

MSX turbo R Technical Hand Book

This is a Work In Progress translation for this book from japanese to english.

It's made possbile by translating small chunks of texts using Google's Image translation services, to maintain original scanned resolution, and also, to improve translations, going paragraph by paragraph, editing page content to avoid bad translation in some texts because paragraphs spanning two pages.

Original japanese book was download from archive.org, so thanks to the people who scanned and made this book available.

I'm translating it to english to help this knowledge to be of use for a wider audience :-)

htdreams - 2025/05

ASCII Publishing Bureau

This book was typeset using ASCII's "Japanese TEX" except for the title page, colophon, and some illustrations. I would like to thank ishii@cts.dnp.co.jp, the author of msdos.sty, and the Publishing Technology Department. I would also like to thank Melhen Maker, who kindly agreed to do the illustrations on the title page.

In order to avoid making the book too long, I have omitted the description of "Japanese MSX-DOS2" and memory mappers. These will be explained in the "Japanese MSX-DOS2 Technical Handbook (tentative title)" to be released soon.

Introduction

Welcome to the world of MSX turbo R. This book provides detailed information on the internals of the MSX personal computer, which has become incredibly powerful thanks to its high-speed CPU and large-capacity memory, and is necessary to make the most of it.

1. This technique maximizes the performance of the R800, a high-speed CPU with a 16-bit internal configuration that delivers processing speeds more than 10 times faster than previous MSX models.

2. Information and techniques for making full use of the PCM and FM sound sources that come standard with the MSX turbo R.

3. The mechanism of the SLOT mechanism, which is essential for mastering the MSX, and how to use it.

4. The Kanji BASIC mechanism is necessary for developing software that handles Japanese.

5. How to master VDP to bring out your techniques in on-screen displays.

MSX turbo R was the first personal computer to achieve dramatically higher performance by upgrading the CPU to 16-bit without making major changes to the architecture of previous models.

Others changed their architecture completely when they went from 8-bit to 16-bit, which meant that all the software and know-how that had been developed by many people on 8-bit machines was thrown away.

We felt that it was necessary to make the CPU 16-bit to improve the performance of MSX, but that we should not throw away the software and hardware assets developed for MSX, nor the know-how of our users. In order to achieve this, we felt that a CPU that was upwardly compatible with the Z80 was necessary for the new MSX, and so we developed the R800. And to achieve complete compatibility with previous MSXs, we developed the MSX turbo R, which is equipped with the conventional Z80 as well as the newly developed R800.

In this way, MSX turbo R maintains ideal upward compatibility with the conventional MSX. Therefore, users can get many times the performance improvement by simply running the software assets they have accumulated so far on MSX turbo R.[1] In addition, the knowledge required for software development can be used as it is, but by using some of the know-how explained in this book, it will be possible to further extract the machine's performance and realize a system that demonstrates outstanding cost performance.

Ryozo Yamashita, General Manager of the 1st Product Division, Systems Business Division

[1] Commercially available software for MSX may not run as fast as the R800 would be too fast and would not be compatible, so it automatically switches to Z80 instead.

table of contents

Figure table of contents

table of contents

第1章 MSX turbo R

This chapter is a re-edited version of the articles "MSX turbo R Technical Analysis" and "PCM Maximum Usage" from the November 1990 and December 1990 issues of MSX Magazine.

1.1 MSX turbo R hardware

The MSX turbo R has been much talked about, with its newly developed 16-bit CPU "R800", 256KB of main RAM, and MSX-DOS2 with hierarchical directory support as standard equipment. Here is an overview of the system configuration of this notable machine.

1.1.1 These are the features of the MSX turbo R!

- Z80 In addition, by incorporating the high-speed CPU "R800" which is upwardly compatible, it achieves an average speed of 4 to 5 times, and up to 10 times, faster (compared to MSX2+).

- Along with MSX-DOS1, it also comes with Japanese MSX-DOS2 and Kanji drivers, and supports MS-DOS compatible hierarchical directories and environment variables.

- It comes standard with 256KB of main RAM that supports memory mappers. The slot configuration has also been standardized.

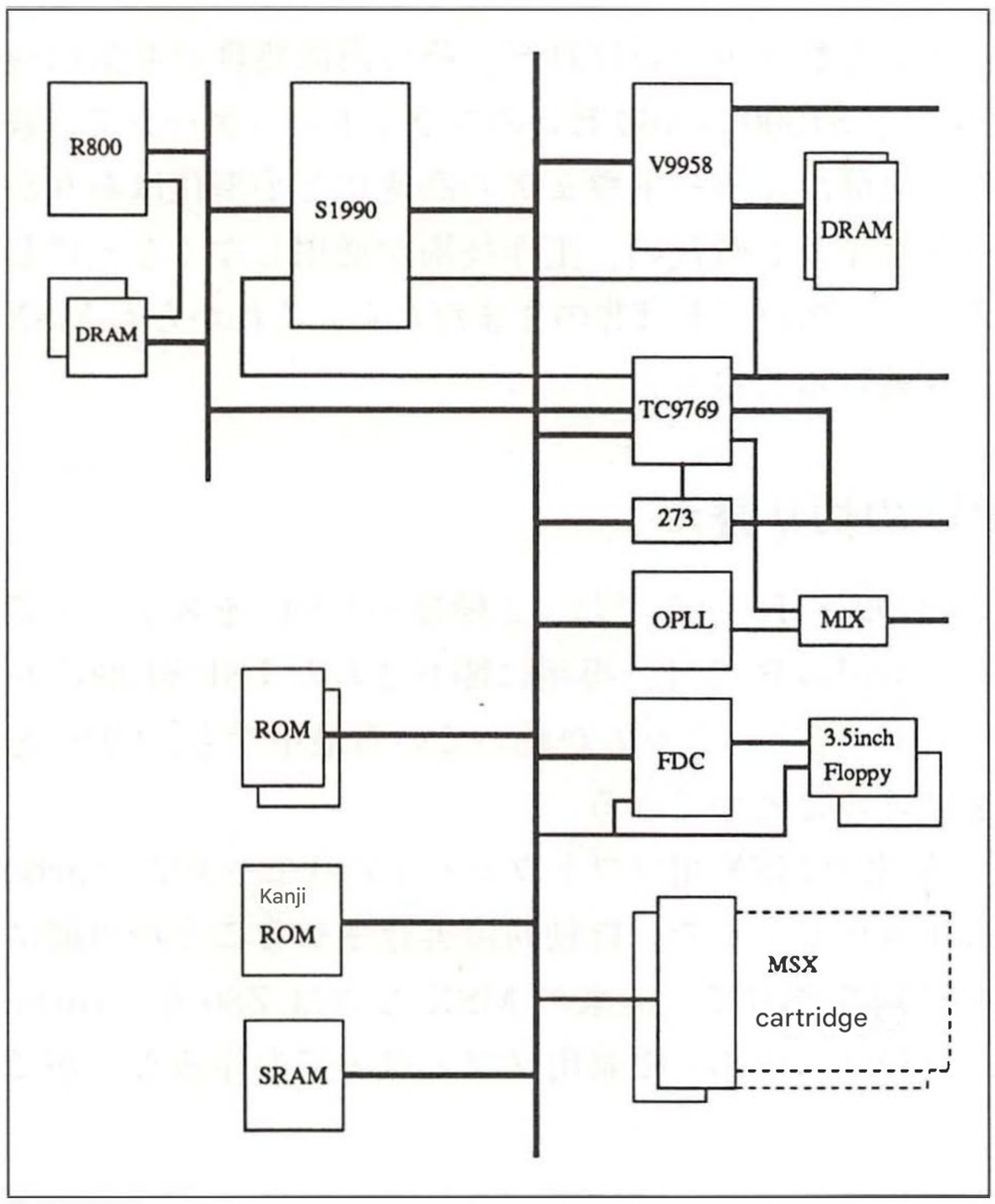
- PCM recording/playback function is standard equipment. MSX-MUSIC, which was previously optional, is now also standard equipment.

1.1.2 MSX turbo R system configuration

The hardware configuration of MSX turbo R (hereafter referred to as turbo R) is shown in Figure 1.1. It includes the same "Z80" compatible CPU as the previous MSX, and the newly developed "R800" CPU. Contrary to the rumors in the industry that "the next MSX will be equipped with Zilog's Z280 or Hitachi's HD64180 (both high-speed Z80-compatible CPUs)," ASCII actually made the CPU.

The performance of these hardware is comparable to older 16-bit machines, and the CPU speed is on par with the V30 (a 16-bit CPU developed by NEC). Also, putting the kanji conversion dictionary in ROM to save RAM and disk space is a traditional MSX design policy. It is also adopted in some recent notebook computers. To evaluate the turbo R hardware in one word, it would be "everyone has been aiming for this."

Figure 1.1: MSX turbo R system configuration



Here is a simplified representation of the turbo R hardware configuration, omitting all the detailed control signal lines. The output of the V9958 is the video signal, the lines connected to the TC9769 (Z80) are the keyboard and joystick, the output of the 273 is the printer port, and the output of MIX is the audio signal.

To explain the hardware configuration in a little more detail, first, the "TC9769" in the diagram is a Toshiba CMOS-LSI (a type of low-power digital LSI) judging from the model number. It is an LSI that includes a Z80-compatible CPU and a PSG sound source, commonly called the "MSX-Engine." In the following, whenever the term "Z80" appears in this book, it refers to this chip.

Below that, "273" is the bus buffer for controlling the printer, "OPLL" is the FM sound source, and "FDC" is the floppy disk controller. Also, "SRAM" is memory that stores the results of the kanji dictionary even when the power is turned off, but this SRAM and the compound phrase conversion dictionary are manufacturer optional features.

By the way, the R800 and main RAM are connected to the bus (the long vertical line in the diagram) through the S1990. For example, when the R800 operates the VDP, the S1990 relays the signal, and if necessary, sends a wait signal to the R800 to synchronize the signal timing with that of the Z80. Conversely, when the Z80 uses the main RAM, the S1990 and R800 relay the signal and handle memory mapping.

The reason why the turbo R has such a complex configuration is to maintain compatibility with existing hardware and software. I think they managed to do this. Bravo.

As you can see, the turbo R has few parts, but its internal processing has become very complicated. Furthermore, the S1990 is a 160-pin flat package, so it is impossible to solder it by hand. Although we are grateful for the faster and smaller hardware, the crafting techniques of the good old days of single-board microcomputers are no longer applicable. However, the MSX cartridge slot remains the same as it was in the past, so the MSX will continue to be a useful teaching material for beginners to hardware.

1.1.3 Elegant CPU switching

JVC's MSX2 machines, the HC-90 and HC-95, used two different CPUs, which could be switched with a switch. However, the turbo R uses a specially developed LSI, the "S1990", to manage the system, so even when the power is on and a program is running, the CPU can be switched and the program can continue to run.

Thanks to this hardware, it is possible to automatically run conventional MSX software in Z80 mode, and turbo R software in the faster R800 mode. It is also possible to create MSX2 / turbo R compatible software that checks the hardware type and selects Z80 for conventional MSX and R800 for turbo R.

1.1.4 Everything packed into MSX turbo R ROM configuration

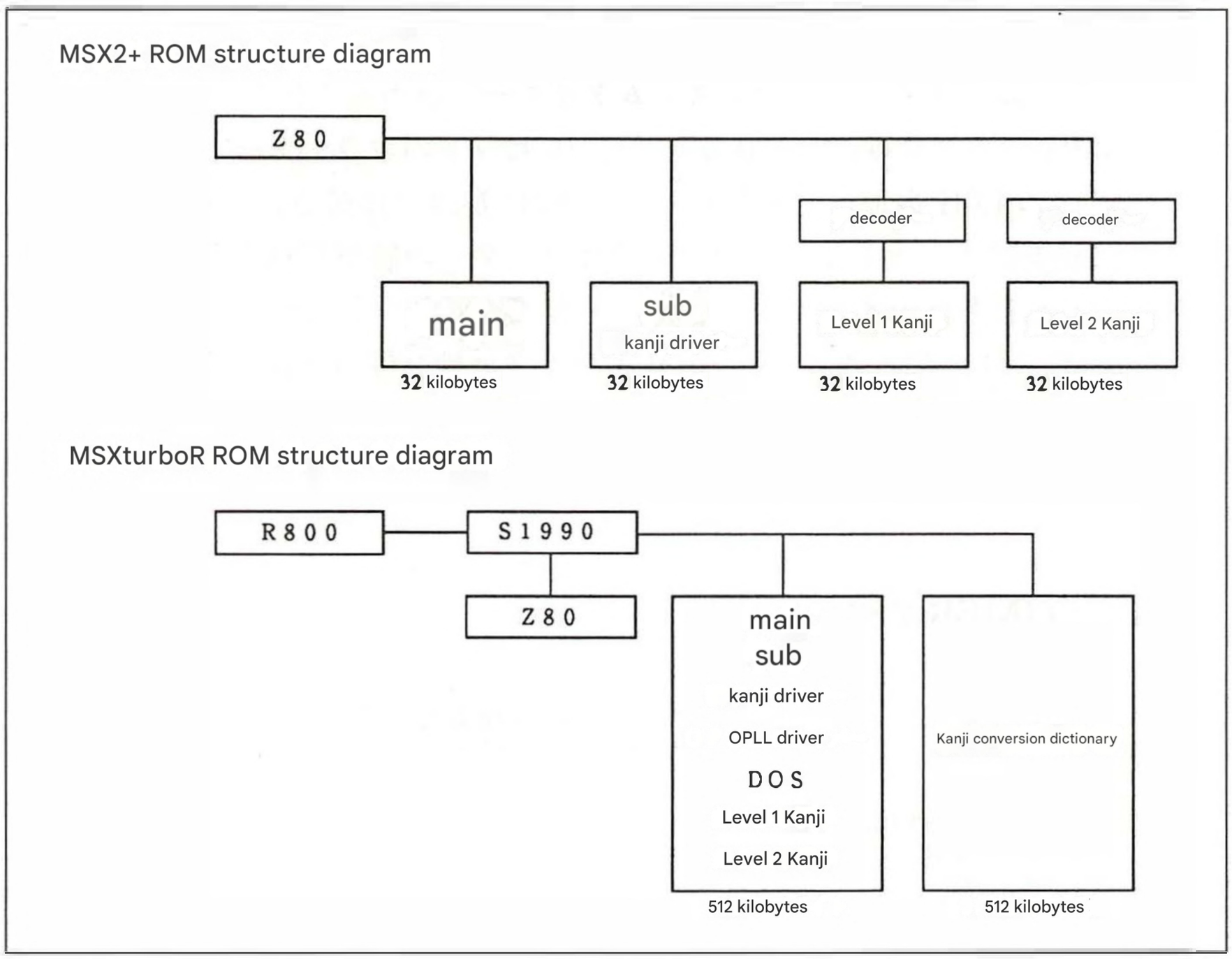
The turbo R should have a lot of ROMs built in, but when you open it up, you'll find that the number of ROMs is surprisingly small. The reason for this is the mega ROM control function of the S1990.

The MSX2+ has a built-in ROM like the one shown in the upper part of Figure 1.2. The main ROM and sub ROM are connected to different slots, and the kanji ROM is connected to an I/O port, so they need to be different ROMs regardless of the total capacity. However, using one 128KB ROM is cheaper than using four 32KB ROMs, and it also reduces the board area and power consumption.

So, in turbo R, the main, sub, OPLL driver, DOS, kanji level 1, and kanji level 2 were all packed into one 512KB ROM, as shown in the lower part of Figure 1.2. However, because of the mega ROM control function of the S1990, which is placed between the CPU and the ROM, from the software's point of view, for example, the kanji level 1 ROM appears to be connected to the I/O port addresses D8H and D9H.

In addition, a total of 64 KB of DOS ROM (16 KB for MSX-DOS and 48 KB for MSX-DOS2) was connected to the 16 KB space in slot 3-2 using a 4-bank switching system.

Figure 1.2: Changes in ROM configuration in MSX turbo R



1.1.5 System timer to adjust speed

When the R800 tries to use the V9958 (the LSI that controls the screen display) at intervals of less than 8 microseconds, the VDP interface circuit built into the S1990 automatically puts the R800 into a wait state. This means that there is no risk of the V9958 malfunctioning due to the CPU processing being too fast.

However, other peripheral LSIs do not have an automatic wait function, so the software itself must adjust the timing.

```
EX    (SP),HL
EX    (SP),HL
```

or

```
PUSH  HL
POP   HL
```

The timing was adjusted by embedding instructions that take time but have no side effects, such as the following, into the program. However, as I will explain later, the execution time of the R800's instructions is uncertain, so it is impossible to achieve timing in this way. Therefore, the turbo R has a "system timer" to adjust the speed.

This is a 16-bit counter that increments every 3.911 microseconds, with the lower byte connected to the I/O port at address E6H and the upper byte connected to address E7H. However, it would be inconvenient if the counter value changed while trying to read the 16-bit value, so it is best to use either the lower byte or the upper byte.

Listing 1.1 shows an example of a program that waits for 3.911 microseconds times the value of the B register. If you rewrite the program to use the high byte of the counter instead of the low byte, you can create a program that waits for 1001.2 microseconds times the value of the B register.

list 1.1 (TIMER.Z80)

```
        .Z80
COUNTLOW        EQU     0E6H    ;       Counter lower 8 bits
COUNTHIGH       EQU     0E7H    ;       Upper 8 bits of counter
;
; B register value 3.911uS wait
; The error is -3.911uS..+OS
; Do not specify 0
; Interrupts must be disabled
; C, A, F are destroyed
;
WAIT:
        IN      A,(COUNTLOW)    ; Get the current value of the counter
        LD      C,A             ; and save it
WAIT_LOOP:
        IN      A,(COUNTLOW)    ; Get the current value of the counter
        SUB     C               ; Calculate the elapsed
        CP      B               ; time Has a specified time elapsed?
        JR      C,WAIT_LOOP     ; Loop if it has not elapsed
        RET
```

1.1.6 MSX turbo R I/O port

The turbo R press release did not include an I/O map, so the editorial team added the information obtained from interviews and hardware analysis to the MSX2+ I/O map to create the I/O map shown in Table 1.1. Items with an "R" note are the I/O ports newly added to the turbo R.

First, the "D/A converter" is an I/O port for controlling PCM recording and playback without going through the BIOS. We'll explain the details later. "Pause key control" is an I/O port for disabling or allowing the pause key to stop a program. It appears to have been provided to prevent situations where a program is interrupted during disk input/output, resulting in the disk being destroyed.

Table 1.1: MSX turbo R I/O map

address	Purpose	Note	
00H～3FH	homemade hardware		
40H～7BH	Manufacturer options		
7CH～7DH	OPLL	+	B
80H～87H	RS-232C	1	B、X
88H～8BH	external VDP	2	XX
90H～93H	printer	2	
98H～9BH	VDP	+	
A0H～A2H	PSG	1	
A4H～A5H	D/A converter	R	
A7H	pause key control	R	
A8H～ABH	8255	1	B
ACH～AFH	MSX-Engine	2	
B0H～B3H	SONY の SRAM	1	XX
B4H～B5H	clock	2	
B8H～BBH	light pen	2	XX
BCH～BFH	VHD control	2	XX
C0H～C1H	MSX-Audio	2	XX
C8H～CCH	MSX-Interface	2	XX
D0H～D7H	floppy disk	2	—、XX
D8H～D9H	No. 1 level Kanji ROM	2	
DAH～DBH	No. 2 level kanji ROM	2	
DCH	Kanji ROM expansion	R	—、X
E3H～E5H	?	R	?
E6H～E7H	system timer	R	
F4H	reset status	+	B
F5H	device enable	2	
F6H～F7H	AV control	2	X
FCH～FFH	memory mapper	2	B

According to a survey by the MSX Magazine editorial team

1 $\mathbf{MSX_1}$ Compatible

2 $\mathbf{MSX_2}$ Compatible

+ $\mathbf{MSX_{2+}}$ Compatible

R **turbo R** Newly established

B Always operate through the BIOS.

— It should not be operated by application programs.

X This is a factory-installed option, but was not implemented in the turbo R prototype we investigated.

XX It was included in the previous model but has been removed from the turbo R model.

? Something is connected, but it's not mentioned in the specs. It looks like there's a register for hardware testing.

"Kanji ROM Expansion" seems to be a reserved function for 24-dot Kanji ROM and JIS Level 3 Kanji ROM that may be created in the future. The "?" below it is not written in any of the documents, but it seems that some hardware works inside the S1990 when reading and writing to the I/O port. I think it is an I/O port for testing turbo R hardware at the factory. And "System Timer" is as I have already explained.

Although this is not mentioned in the table, in order to keep up with the speed of turbo R, you should use the BIOS to operate the PSG, joystick, mouse, printer, keyboard, and clock (battery-backed clock IC).

Next, there is a feature shared with the MSX2+ that I would like to explain in more detail: the "reset status." This is an I/O port that distinguishes between a hardware reset and a restart caused by jumping to address 0 of the main ROM. Specifically, when address 17AH of the main ROM is called, the value of this reset status is read into the A register, and when address 17DH is called, the value of the A register is written into the reset status. For example,

```
CALL  17AH
OR    80H
CALL  17DH
RST   0H
```

By following this procedure, setting bit 7 of the reset status to 1 and then jumping to address 0, you can reliably restart the MSX.

By the way, the reason why you can't use the reset status without going through the BIOS is because the logic of the hardware signal for the reset status is reversed depending on the machine. The BIOS compensates for that difference.

With the turbo R, which comes standard with DOS2, the "memory mapper" has become increasingly important. It requires a somewhat complicated procedure to use the extended BIOS and to operate.

Finally, as a side note, Table 1.1 includes functions that were once put to practical use or prototyped, but are not included in recent MSXs. The author thinks that the recent MSX has become a standard computer, and there are too few cute peripherals, but what do you think?

1.1.7 DRAM mode for maximum speed

Each memory has a minimum time interval between reads and writes, called the "access time." If the CPU speed is too fast, a "wait" must be inserted to match the CPU speed to the memory. This access time varies by type, and the faster the memory, the more expensive it is. In general, RAM has a shorter access time than ROM.

To take advantage of the speed of the R800, it is better to store programs in RAM rather than ROM. Therefore, a "DRAM mode" was provided that transfers the contents of the BIOS, BASIC, sub-ROM, and Kanji driver ROMs to DRAM (main RAM) for use.

This separates the last 64KB of main RAM from the memory mapper, transfers the ROM contents, then write-protects it and connects it to the CPU. From the CPU's perspective, it appears that the normal ROM has been replaced with a high-speed ROM. When executing programs written in BASIC, the ROM containing the BIOS and BASIC interpreter is often used, so the speed of the DRAM mode can be utilized.

However, when running machine language programs, especially DOS programs, the time that ROM is used is relatively short, so it may be more advantageous to use the extra memory for a RAM disk or similar, rather than using DRAM mode.

Also, programs on ROM cartridges will run faster if they are transferred to RAM, but with turbo R disk-based software will likely become more mainstream than ever before.

1.1.8 Here are the features of the R800!

- It is object-compatible with the Z80, so Z80 software will also work, except for parts that depend on CPU timing.

- The CPU clock speed is 7.16 MHz. However, since the number of clocks per instruction is significantly reduced compared to the Z80, it is equivalent to 29 MHz in Z80 terms (when there are no wait states).

- Supports multiplication instructions with precision from 16 bits x 16 bits to 32 bits, enabling a significant improvement in calculation processing speed.

- Access to the upper/lower 8 bits of the IX/IY registers, which was undefined on the Z80, is now officially guaranteed.

1.1.9 All about R800

The R800 used as the CPU for turbo R is a high-speed CPU that is software compatible with the conventional Z80. In other words, unless the CPU is too fast, software developed for the Z80 can be run at high speed on the R800 as is.

The features added to the Z80 include a 16-bit multiplication instruction and an instruction for byte access of the IX/IY register, which was considered a "trick" in the Z80. For details, please refer to the R800 instruction table in the appendix of this book.

The clock frequency of the conventional MSX is 3.58 MHz, and that of the turbo R is 7.16 MHz. From this alone, it seems that the speed has only doubled, but in fact, that is not the case. The number of clocks required to execute one instruction is reduced with the R800, and since there is no M1 cycle wait to access RAM, program execution speed is even faster. With the conventional Z80, a clock frequency of about 29 MHz is required to achieve the same processing speed as the R800, so this is a significant speed increase.

Table 1.2: Comparing the operating speeds of Z80 and R800

instruction		MSX2+ (unit μs)	turbo R (unit μs)	magnification
LD	r,s	1.40	0.14	x10.0
LD	r,(HL)	2.23	0.42	x 5.3
LD	r,(IX+n)	5.87	0.70	x 8.4
PUSH	qq	3.35	0.56	x 6.0
LDIR	(BC $\neq$ 0)	6.43	0.98	x 6.6
ADD	A,r	1.40	0.14	x10.0
INC	r	1.40	0.14	x10.0
ADD	HL,ss	3.35	0.14	x24.0
INC	ss	1.96	0.14	x14.0
JP		3.07	0.42	x 7.3
JR		3.63	0.42	x 8.7
DJNZ	(B $\neq$ 0)	3.91	0.42	x 9.3
CALL		5.03	0.84	x 6.0
RET		3.07	0.56	x 5.5
MULTU	A,r	—	1.96	—
MULTUW	HL,rr	—	5.03	—

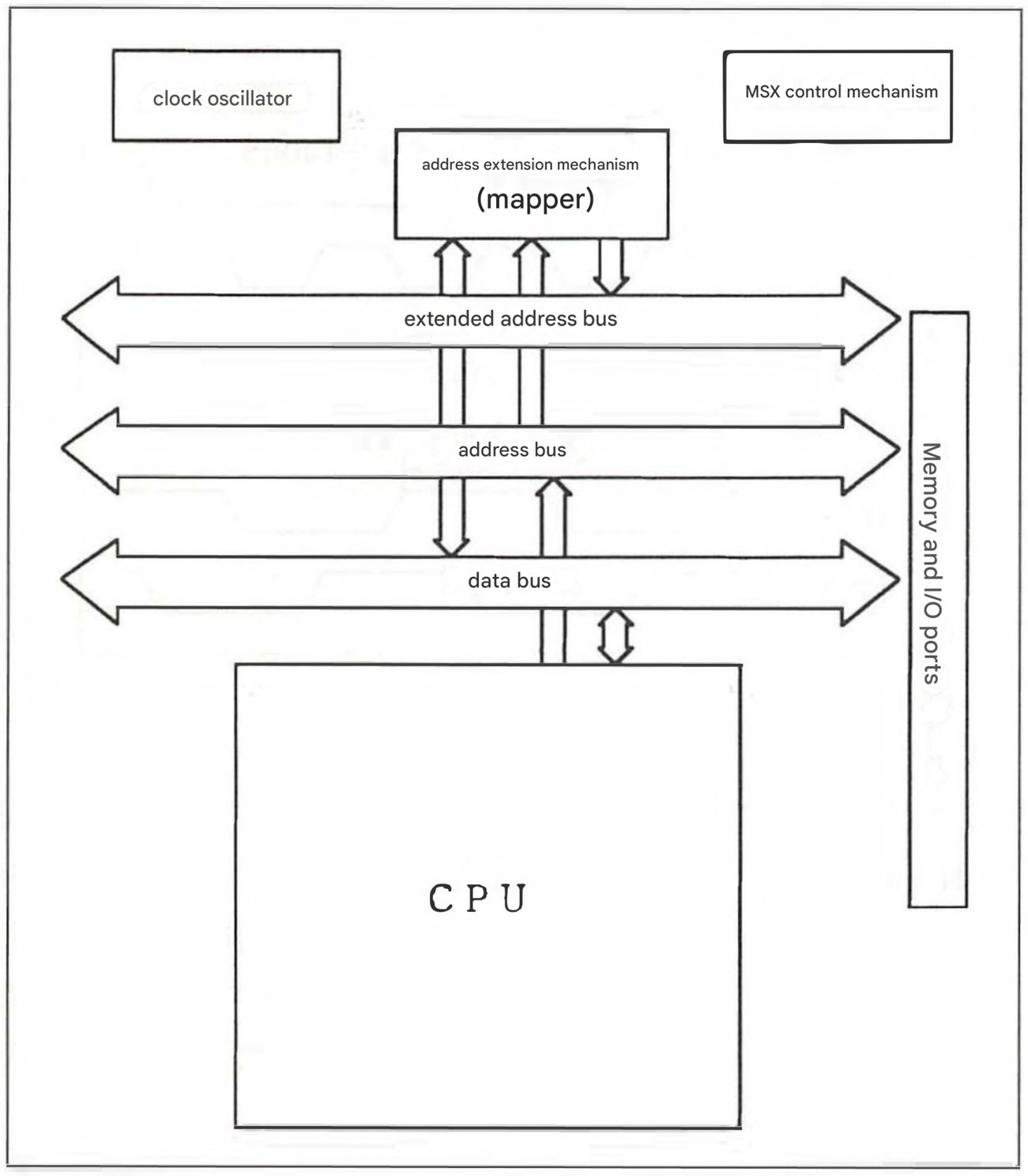
Table 1.2 shows the results of comparing the speed of the Z80 and the R800 for each type of instruction. It is worth noting that data transfer between registers (LD instruction) and addition are 10 times faster. However, the values in this table are measurements of the speed when the R800 operates with no waits. In reality, the speed may decrease due to waits, so be careful. The conditions under which waits occur and how to avoid them will be explained in detail later.

The internal structure of the R800 is shown in Figure 1.3. In the R800, the external data bus is 8 bits, but the data bus inside the CPU is 16 bits, so a 16-bit add instruction is processed in one cycle.

Looking at this hardware configuration, the R800 appears to be closer to 16-bit CPUs with 8-bit external data buses, such as Intel's "8088" or Motorola's "MC68008," than to the 8-bit CPU of the Z80.

At the top of Figure 1.3, there is something called an "address extension mechanism (mapper)", but this seems to have been prepared so that the R800 could be used for things other than MSX. When using it with turbo R, the slot control mechanism and memory mapper built into the S1990, not the R800, will control the system.

figure 1.3: R800 internal block diagram

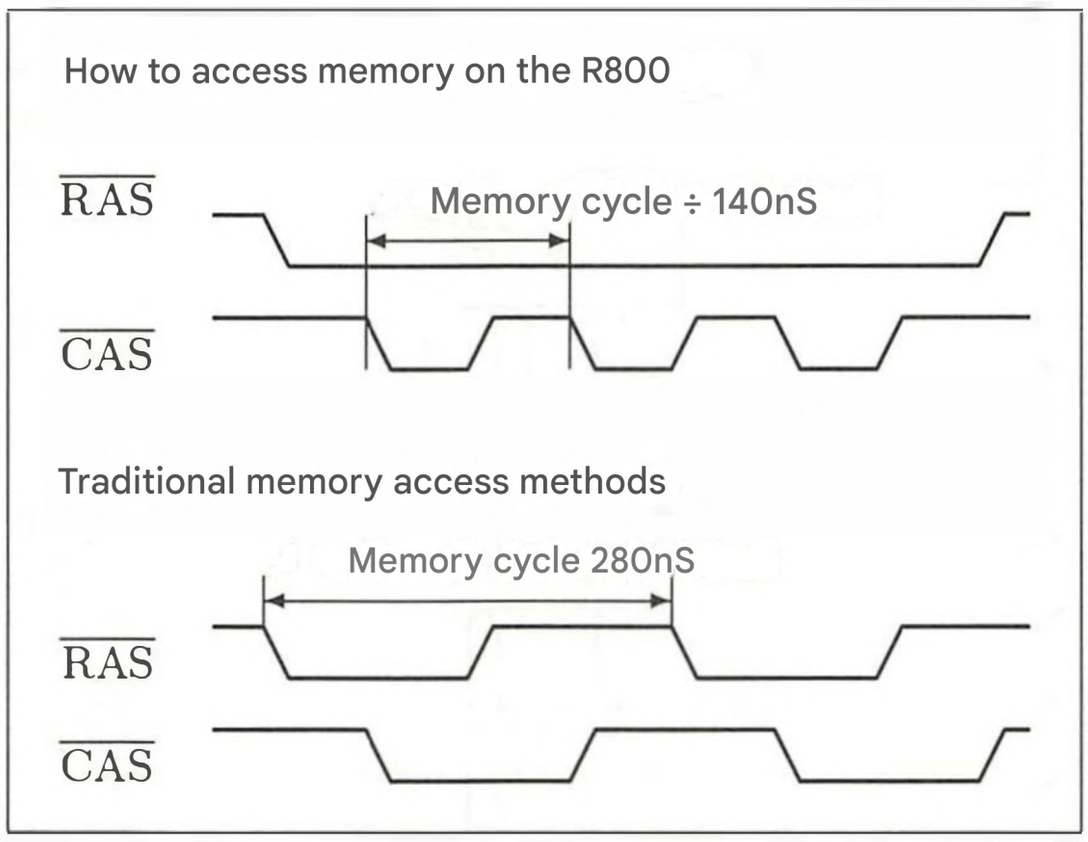


Next, let's explain "DRAM page access" in detail. First, the bottom part of Figure 1.4 shows the previous method of memory access using the Z80. The upper byte of the address (row address) is sent to the DRAM, the RAS (row address strobe) signal is set to LOW, the lower byte of the address (column address) is sent to the DRAM, and then the CAS (column address strobe) signal is set to LOW. This specifies the memory address.

Meanwhile, the upper part of Figure 1.4 shows DRAM page access on the R800. By keeping the upper byte of the address and the RAS signal fixed and varying only the lower byte of the address and the CAS signal, the DRAM is used twice as fast as with the conventional method. In this way, with the R800, when the DRAM is used continuously without changing the upper byte of the address, page access is performed automatically.

Now, the types of DRAM that are easy to connect and use with the R800 include 256kbit (32kB), 1Mbit (128kB), 4Mbit (512kB), etc. Even though the turbo R has a minimum main RAM capacity of 256kB, you can get by with just two 1Mbit DRAMs.

Figure 1.4: Differences in memory access methods between Z80 and R800



The first MSX, developed in 1983, used eight 16-kilobit DRAMs, but the main RAM capacity was only 16 kilobytes. Considering that, today's technology is amazing, as it can achieve a RAM capacity of 256 kilobytes with just two DRAMs. The functionality of the MSX has increased, but the size and power consumption of the hardware has decreased. The application of Japan's latest semiconductor technology can be seen in the much talked about notebook computers and turbo R.

1.2 MSX turbo R How to use

1.2.1 Programming that takes advantage of the speed of the R800

The R800 is certainly fast, but to get the most out of its speed, that is, to avoid waits and utilize the capabilities of the R800, you need to be creative with your programming. Remember that there are three waits for access to an external slot, two waits for access to the internal ROM, and one wait when the internal DRAM cannot be page-accessed.

Ideally, programs should be placed in the same 256-byte range (page-accessible range) of the upper bytes of the internal RAM addresses, and data should be placed in the registers. In this case, no memory access is required for data, and memory access for the CPU to read the program is also performed in page mode, so there is no wait time for the CPU. It is difficult to write all programs like this, but it is a good idea to try to get as close as possible to this condition for at least the subroutines, which require the highest speed.

Now, whether a page can be accessed or not depends on the addresses of the program, data, and stack. do. for example,

```
PUSH    HL
```

The execution time of an instruction is 4 clocks if the high byte of the address where that instruction is located matches the high byte of the stack pointer. If they don't match, it's 5 clocks. There's probably little need to think this through when creating a program, but it's important to remember that the execution time of an instruction varies depending on the situation.

1.2.2 Precautions and issues when using the R800

The Z80 refreshed the DRAM after each instruction, but the R800 takes 280 nanoseconds to refresh the DRAM every 31 microseconds. Note that because of the time it takes to refresh, and the DRAM page accessibility conditions mentioned above, it is not possible to accurately predict the execution time of an R800 program.

So to adjust the speed of the program, we will use something called a "system timer". I will explain how to use this system timer and how to adjust the speed between the CPU and the VDP later, so please wait.

Also, as with any new CPU, one of the problems with the R800 is the lack of development equipment. In particular, it is inconvenient that the "ICE (in-circuit emulator)" which is so useful when developing software cannot be used for debugging.

Therefore, to create software for the turbo R, it is best to first thoroughly debug using the conventional MSX and Z80 ICE, and then rewrite the programs that should work for the turbo R. Create a program that can also be used for the Z80, check that it works, and then rewrite only the parts that use multiplication for the R800. At this point, it is also a good idea to break it down into subroutines and check their operation. Then, if you put the whole thing together and it doesn't work, you'll have to look at the source listing and think about it.

1.2.3 Added BIOS and its function description

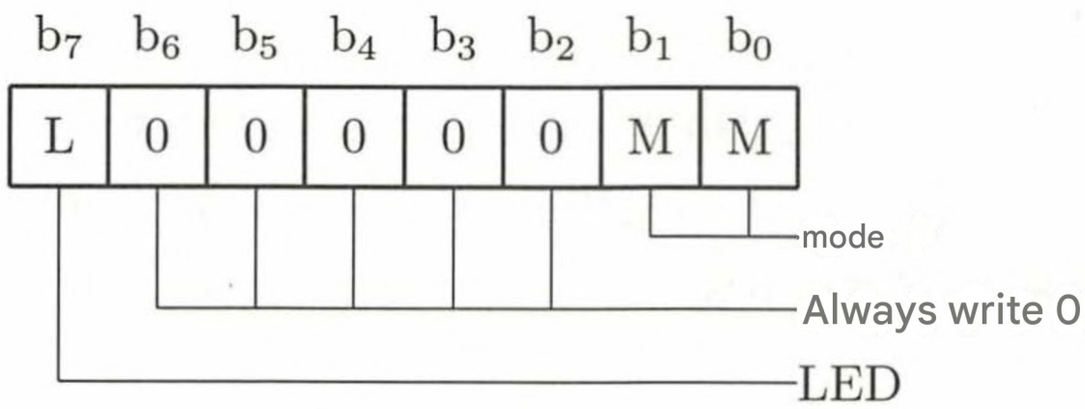
To control the turbo R's new hardware features, a BIOS was added for CPU switching and PCM recording and playback.

Here, we will explain the BIOS name (label), entry address, function, and each register in that order. The symbols used to describe BIOS functions are as follows. First, E is a register that should be set before calling the BIOS. R is a register to which the BIOS returns a value, and M is a register to which the BIOS writes a meaningless value, i.e., the original contents are destroyed. IYH represents the upper byte of the IY register, and the contents of the lower byte are ignored.

CHGCPU 0180H street address

function Switch CPU.

E Bits 1 and 0 of the A register set the mode as follows. "R800 DRAM" is a mode in which the contents of the BIOS ROM are transferred to DRAM.



mode	
00	Z80
01	R800 ROM
10	R800 DRAM

Also, if bit 7 of the A register is 1, the LED indicating which CPU is running will change. Conversely, if bit 7 of the A register is 0, the CPU will be switched, but the LED will not change.

R none

M AF

Note The contents of the registers before switching the CPU are carried over to the new CPU, except for AF and R. Also, interrupts are enabled after switching. Note that you should take note of the following points when switching CPUs, which will be explained in detail later.

GETCPU 0183H address

function Check the running CPU.

E none

R Depending on the CPU being used, the following values will be returned in the A register:

0	Z80
1	R800 ROM
2	R800 DRAM

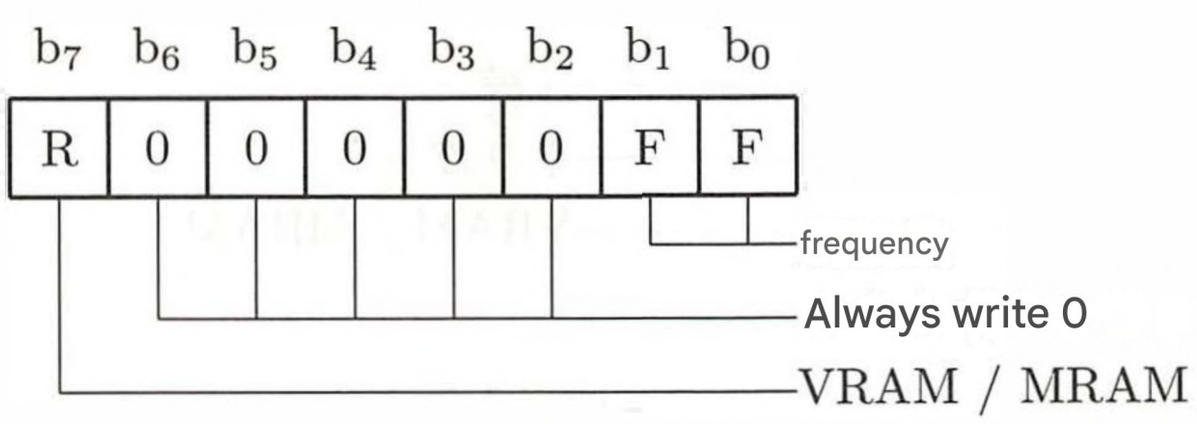
M F

Note You need to make sure your hardware is turbo R as explained later before calling up this BIOS.

PCMPLY 0186H address

function Plays PCM sound.

E A



EHL (data address)

DBC (data length)

If bit 7 of the A register is 1, the PCM audio data is stored in the video RAM, if it is 0, the PCM audio data is stored in the main RAM. Note that the values of the D and E registers are only meaningful if there is data in the video RAM.

Bits 1 and 0 of the A register set the sampling frequency, with 15.75 kHz only available when the turbo R is running in R800 DRAM mode.

00	15.75	kilohertz
01	7.875	kilohertz
10	5.25	kilohertz
11	3.9375	kilohertz

R carry flag

0 Normal termination

1 Abnormal termination

A (Cause of abnormality)

1 Frequency specification error

2 Interruption with STOP key

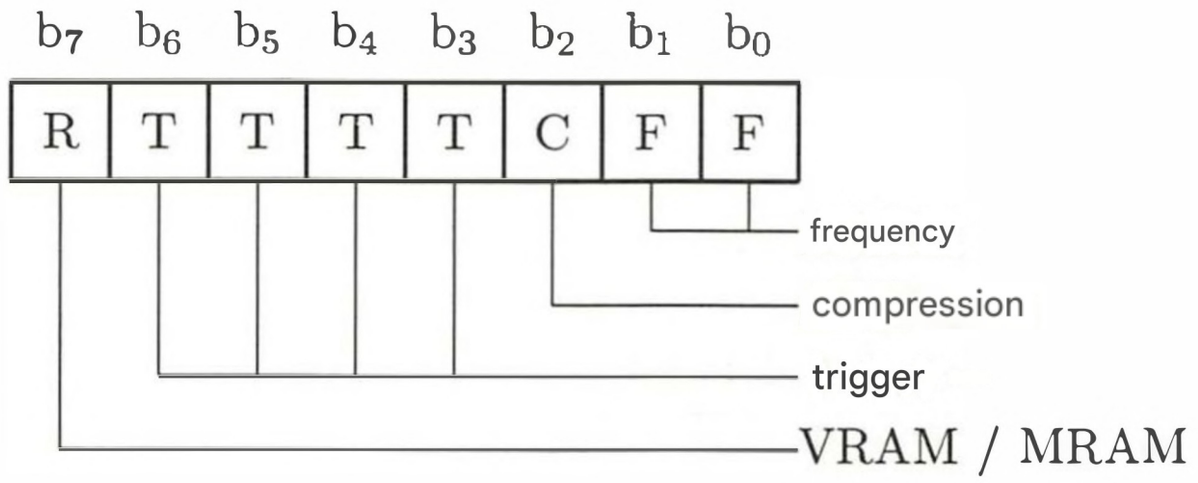
EHL (interrupted address)

M all

PCMREC 0189H address

function Records PCM sound.

E A



EHL (data address)

DBC (data length)

The method for setting bits 7, 1, and 0 of the A register is the same as that explained for PCMPLY. Bits 6 to 3 of the A register are called the "trigger level" and specify the volume of the sound that triggers recording to begin. If this value is 0, recording will begin immediately.

Also, if bit 2 of the A register is 1, the recorded data is compressed. If it is 0, it is not compressed.

R — carry flag

- 0 Normal termination
- 1 Abnormal termination

A (Cause of abnormality)

- 1 Frequency specification error
- 2 Interruption with STOP key

EHL (interrupted address)

M — all

1.2.4 What BIOS was changed or removed?

Table 1.3 lists the BIOS that were changed or removed in turbo R. We will briefly explain each one below.

First, the turbo R no longer has a cassette tape interface, so if you call "TAPION", "TAPIN", "TAPIOF", "TAPOON", "TAPOUT", or "TAPOOF" that were in the previous BIOS, the carry flag will be set and the program will return with an error. Also, "STMOTR" no longer exists, so if you call it, it will return without doing anything.

In addition, the paddle and light pen BIOS were removed to allow new features to be added without changing the capacity of the main ROM. When the BIOS "GTPDL" is called, the A register is always set to 0 and the function returns. Similarly, when "GTPAD" or "NEWPAD" is called with a value of 8 to 11 in the A register to specify the light pen, the A register is always set to 0 and the function returns.

The BIOS that was changed is the "ROM version ID" to know the version of MSX being used. This can be found in the contents of address 002DH of the main ROM, and has been changed to 03H in the case of turbo R. If you are developing a program for turbo R, first make sure that the value of this address is 03H or higher. If it is not, run it as an MSX2 program, or display an error message and abort.

Note that programs that only work when the content of address 002DH is 03H will no longer work when MSX is updated in the future, so they must be written to work on 03H or higher. In general, when it comes to hardware and OS versions, you should program your software to work if it obtains a value equal to or higher than the version number you require.

Table 1.3: List of BIOS and BASIC changes in MSX turbo R

Added BIOS entries	
CHGCPU	0180H
GETCPU	0183H
PCMPLY	0186H
PCMREC	0189H

Modified BIOS entries	
ROM version ID	002DH

Deleted BIOS Entries	
GTPDL	00DEH
TAPION	00E1H
TAPIN	00E4H
TAPIOF	00E7H
TAPOON	00EAH
TAPOUT	00EDH
TAPOOF	00F0H
STMOTR	00F3H
GTPAD	00DBH
NEWPAD	SUB 01ADH

Added statement
CALL PCMREC CALL PCMPLAY CALL PAUSE

modified statement
COPY

deleted statement
CLOAD CSAVE MOTOR

This has actually happened in the past, but due to incorrect checking of the MSX version, some MSX2 programs did not work on MSX2+, and some applications did not work when combined with MSX-JE with learning function. To avoid this, please remember to make sure that the program works on 03H or later.

As with the BIOS, some BASIC features have been added, changed, or removed in the turbo R. For details, see Table 1.3 or the BASIC manual that came with your machine.

1.2.5 Notes on application development

In MSX turbo R, the R800 does not always operate with no waits. There are three waits when accessing an external slot, two waits when accessing internal ROM, and one wait when the internal DRAM causes a page break. Therefore, to speed up a program, you must work while thinking about how to reduce these waits as much as possible. Here are three points to keep in mind for this purpose.

The first is to transfer the program itself to RAM before executing it. Software provided on floppy disks is not a problem because it will inevitably run in RAM, but you need to be careful with programs provided on ROM cartridges in the slots. By transferring only the necessary parts to RAM before executing them, it is possible to significantly increase the speed.

It is also important to code in a way that does not cause a page break. The R800 has a dedicated bus that supports DRAM page access, so make the most of this function. Specifically, it is effective to program memory access so that only the lower 8 bits of the address change, that is, in the range of 256 bytes from ??00H to ??FFH.

By the way, a page break occurs when memory access is performed outside this range, in other words, when the upper 8 bits of the address change.

As I mentioned briefly before, unlike MSX2+, turbo R does not know the exact execution time of instructions at the program coding stage. The reason for this is that DRAM page breaks occur unpredictably, and unlike Z80, DRAM refresh is performed asynchronously with instruction execution.

Also, when creating a program that will run on both turbo R and MSX2+, it is not recommended to use a software loop for timing. Therefore, turbo R now has a new system timer that counts up every 3.911 microseconds. From now on, let's use this system timer for timing.

1.2.6 Example of program to switch CPU

Listing 1.2 is the source listing for “CHGCPU.COM” which switches CPU. When DOS2 is running in turbo R,

```
CHGCPU 0
```

In Z80 mode,

```
CHGCPU 1
```

The ROM mode of the R800 is

```
CHGCPU 2
```

The R800's DRAM mode is selected with these. The program gets the first character of the first parameter of the command from address 5DH of the DOS work area (specifically the default FCB area), sets the value of the A register accordingly, and calls the BIOS "CHGCPU" at address 180H of the main ROM.

To make the program more practical, a process to check the DOS version number was added. Specifically, it first checks that the content of address 2DH in the main ROM is 03H or higher, meaning it is turbo R, and then uses the DOS system call 6FH to check that the DOS kernel version number is 2 or higher.

list 1.2 (CHGCPU.Z80)

```
        .Z80
RDSLT   EQU     0000CH          ; inter slot read
CALSLT  EQU     001CH           ; inter slot call
EXPTBL  EQU     0FCC1H          ; slot # of main ROM
;
        ld      a,(EXPTBL)      ;
        ld      hl,2dh          ; address to read
        call    RDSLT           ; read version
        cp      3
        jr      nc,TURBOR
        ld      de,MSG_NOTR
        ld      c,9             ; _STROUT
        call    5
        rst     0               ; return to DOS
TURBOR:
        ld      c,6fh           ; _DOSVER
        call    5               ;
        ld      a,b             ; version of DOS kernel
        cp      2
        jr      c,NOTDOS2       ;
        ld      a,d             ; version of MSXDOS.SYS
        cp      2
        jr      c,NOTDOS2

        ld      a,(005ch+1)     ; command parameter
        sub     '0'             ; 0:Z80, 1:R800ROM, 2:R800RAM
        ret     c               ; abort if parameter < '0'
        cp      3
        ret     nc              ; abort if '3' <= parameter
        or      80h               set change-LED flag
        ld      ix,180h         ; address of CHGCPU
        ld      iy,(EXPTBL-1)   ; slot of main ROM
        call    CALSLT          ; inter-slot call
        rst     0               ; return to DOS
;
NOTDOS2:
        ld      de,MSG_NOTDOS2
        ld      c,9             ; _STROUT
```

```
        call    5
        rst     0               ; return to DOS
;
MSG_NOTR:
        DB      'not MSX turbo R', 0dh, 0ah, '$'
MSG NOTDOS2:
        DB      'not MSX-DOS 2', 0dh, 0ah, '$'
        END
```

Similarly, the following Listing 1.3 is the source listing for "GAMEBOOT.COM", which tricks MSX2 programs into running in R800 mode. This program is used to start programs on other disks when DOS2 is started and R800 is selected. In other words, it is used to forcibly run programs (such as games) that do not include the DOS2 system in R800 mode.

To briefly explain the program, it first displays a message on the screen and waits for the disk to be swapped. Then it reads the boot sector of the swapped disk and executes it. The environment is the same as when the boot sector is called the second time in the normal way: page 1 is DOS ROM, the other pages are RAM, and the carry flag is set.

In addition, the DOS work area (address F323H) for storing the pointer to the error handling program is set in the HL register, and the address (address F368H) of the program that switches page 1 from RAM to DOS ROM is set in the DE register.

list **1.3 (GAMEBOOT.Z80)**

```
        .z80

_conin          equ     01h
_strout         equ     09h
_setdta         equ     1ah
_rdabs          equ     2fh

dos             equ     0005h
enaslt          equ     0024h

notfirst        equ     0f340h
master          equ     0f348h

        ld      sp,(6)
        ld      de,prompt       ; print prompt message
        ld      c,_strout
        call    dos
        ld      c,_conin        ; wait for key in
        call    dos
        ld      de,0c000h       ; read boot sector at 0c000h
        ld      c,_setdta
        call    dos
        ld      de,0            ; logical sector 0
        ld      l,0             ; drive A:
        ld      h,1             ; read 1 sector
        ld      c,_rdabs
        call    dos
        ld      h,40h
        ld      a,(master)
        call    enaslt
        ld      hl,0f323h
        ld      de,0f368h
        xor     a
        ld      (notfirst),a
        scf
        jp      0c01eh
prompt:
        db      'Insert game disk in drive A:,',0dh,0ah
        db      'and press any key $'

        end
```

1.3 How to use PCM to its limits

A new feature added to turbo R is PCM. It is human nature to want to get the most out of this feature. Here, we will introduce how to use PCM, from BASIC to machine language and special uses that utilize horizontal scan line interrupts.

1.3.1 Basics: How to use in BASIC

Let's start by looking at some basic things using BASIC.

In the first place, PCM converts audio input from a microphone or other device into digital data and stores it in memory. It stores the information in a file and allows you to play it back at will.

In turbo R, PCM data is stored in either main RAM or video RAM. There are four sampling rates to choose from: 15.75 kHz, 7.875 kHz, 5.25 kHz, and 3.9375 kHz. The higher the value, the higher the quality of the sampling.

To use PCM from BASIC, there are two commands you need to remember. Instructions on how to use them are listed below, so please refer to them. Basically, you can record and play PCM just by executing these commands. However, you need to be very careful when setting the start and end addresses where the data will be stored.

First of all, you must reserve memory space for PCM data with the BASIC "CLEAR" command, otherwise the program will definitely go out of control. For example, if you want to use addresses C000H to D000H for PCM data,

```
CLEAR 200,&HC000
```

For now, let's put a simple sample program in List 1.4.

I've prepared this for you, so you can enter it and have a play around with it.

Of course, if you use video RAM for PCM data, you can put the data at any address, so you don't need to worry about the start and end addresses. Also, with video RAM, you can see the PCM data. First,

```
SCREEN 8
```

If you set the screen mode like this and then record PCM, the data will be displayed on the screen in a row, which might be interesting.

If you pay attention to the above, you can record and play back the basic PCM. You can also change the playback sampling rate to play back at four different speeds. However, the problem occurs when playing back PCM. The turbo R is completely occupied with that, so unfortunately you can't do anything while playing back PCM.

list 1.4 (PCM1.BAS)

```
10 CLEAR100,&H9000
20 PRINT " It's time for a new start.";
30 A$=INPUT$(1):PRINT
40 _PCMREC  (@&H9000,&HCFFF,0)
50 PRINT " Restart.";
60 A$=INPUT$(1):PRINT
70 _PCMPLAY (@&H9000,&HCFFF,0)
80 GOTO 20
```

1.3.2 PCM related BASIC instructions

CALL PCMREC

Format

- Record to main RAM or video RAM.

  CALL PCMREC(@start address, end address, sampling rate [, [trigger level], compression switch [, S])

- Recording to an array variable.

  CALL PCMREC(Array variable name, [Length], Sampling rate [, [Trigger level], Compression switch])

Setting the sampling rate	
Specified value	sampling rate
0	15.7500KHz
1	7.8750KHz
2	5.2500KHz
3	3.9375KHz

The trigger level sets the input level when recording begins. The value can be from 0 to 127. Recording will begin when the input level reaches or exceeds this value, and will begin immediately if this is 0 or omitted. The compression switch is set to 1 to compress silent parts, and 0 or omitted to not compress.

CALL PCMPLAY

Format

- Playback from main RAM or video RAM

  CALL PCMPLAY(@start address, end address, sampling rate [, S])

- Playback from array variables

  CALL PCPLAY(Array variable name, [length], sampling rate)

For both PCMREC and PCMPLAY, if the mode is not high-speed, it will temporarily switch to high-speed mode before execution, and will return to the original state when it is finished. Also, if 15.75KHz is specified in the ROM mode of the R800, an error will occur.

If the STOP key is pressed during recording or playback, program execution will be interrupted. In the PCM data format, normal data is from 1 to 255, with 0 being special. The following byte outputs level 0 (127) the number of times specified.

1.3.3 Play BEEP sounds via PCM!

You know that when you press the "CTRL" and "STOP" keys simultaneously while executing a BASIC program to interrupt the program, a "beep" sound is emitted. The same "beep" sound is emitted when you display a list with the "LIST" command and stop the program with "CTRL + (STOP)". You can change the sound by using the "SET BEEP" command in BASIC, but none of the four sounds available are particularly impactful.

So, what happens if you play this BEEP sound with PCM? If you set it to some funny dialogue, the MSX will talk at every opportunity, which might be pretty loud and fun.

So, if you run the program in Listing 1.5, you will be able to play the BEEP sound on the PCM. Of course, this is only for turbo R. It's not a very long program, so please try your best to type it in.

This program is stored in page 1 of the main RAM (addresses 4000H to 60FFH), so after executing the program, you can use the "CLEAR" command to set the upper limit of the user area above address B000H. However, you cannot use memory disks, so be careful not to accidentally use "CALL MEMINI". Also, when using the BASIC "BEEP" command,

```
PRINT CHR$(7)
```

If you do not use it instead, the BEEP sound will not be PCM.

Explain how to use the program.

1 PCM BEEP set

After this, the BEEP sound will be PCM. Once you execute this command, the setting will remain valid until the power is turned off. You can also change the CLEAR statement setting.

2 PCM BEEP reset

BEEP Returns the sound to its original state. Be sure to execute this command when you change to DOS or DOS2 with “CALL SYSTEM”.

3 PCM data playback

Plays the currently set PCM data. Use it for confirmation.

4 PCM data recording

Record PCM data at 15.75 kilohertz. When recording, memory from addresses B000H to CFFFH is used.

5 PCM data LOAD

Reads PCM data saved in BSAVE format with the extension ".PCM".

6 PCM data SAVE

The PCM data recorded using "PCM Data Recording" is written to a disc.

0 END

End the program. You can also use CTRL + STOP.

A simple message will be displayed on the screen for your reference.

list **1.5 (PCM2.BAS)**

```
10 SCREEN0:WIDTH40:DEFINT A-Z
20 CLEAR100,&HB000
30 DEFUSR=&HD800:DEFUSR1=&HD806:DEFUSR2=&HD803
40 FOR I=&HD800 TO &HD87F
50 READ A$:POKE I,VAL("&H"+A$):NEXT
100 PRINT
110 PRINT"1) PCM BEEP set "
120 PRINT"2) PCM BEEP reset"
130 PRINT"3) PCM DATA playback"
140 PRINT"4) PCM DATA recording "
150 PRINT"5) PCM DATA LOAD"
160 PRINT"6) PCM DATA SAVE"
170 PRINT"0) END"
180 PRINT" ....HIT 0-6 KEY=";
190 A$=INPUT$(1):I=ASC(A$)-ASC("0")+1
200 IF I>0 AND I<8 THEN ELSE190
210 ON I GOTO 230,240,310,220,340,390,430
220 PRINTCHR$(7);:GOTO 190
230 GOSUB 470:END
240 GOSUB 470:I=USR1(0):I=USR(0)
250 PRINT"PCM BEEP can be used."
260 PRINT"When using DOS or DOS2, please reset PCM BEEP"
270 PRINT"PCM BEEP / Nearing data page1 (4100H to 60FFH)."
280 PRINT"The CLEAR command can be set to above B000H without any issue,
but commands related to memory disks, such as CALL MEMINI, cannot be
executed."
```

```
290 PRINT"Now, to execute the BEEP command, please use PRINT CHR$(7)."
300 END
310 GOSUB 470:POKE &HFDA4,&HC9
320 PRINT"PCM BEEP has been reset."
330 GOTO 100
340 GOSUB 470
350 PRINT"Starting PCM recording. (HIT ANY KEY!)";
360 A$=INPUT$(1):PRINT:_PCMREC(@&HB000,&HCFFF,0):I=USR(0)
370 PRINT"Recording completed."
380 GOTO 100
390 GOSUB 470
400 PRINT"PCM data LOAD"
410 INPUT" FILE NAME (8 characters)=";A$
420 BLOAD A$+".PCM":I=USR(0):GOTO 100
430 GOSUB 470
440 PRINT"PCM data SAVE"
450 INPUT" FILE NAME (8 characters)=";A$
460 I=USR2(0):BSAVE A$+".PCM",&HB000,&HCFFF:GOTO 100
470 PRINT CHR$(I+47):PRINT:PRINT:RETURN
480 DATA C3,4D,D8,C3,5F,D8,CD,6D
490 DATA D8,3A,42,F3,32,2A,D8,21
500 DATA 2E,D8,11,00,40,01,00,01
510 DATA ED,B0,CD,76,D8,21,29,D8
520 DATA 11,A4,FD,01,05,00,ED,B0
530 DATA C9,F7,00,00,40,C9,FE,07
540 DATA C0,01,00,20,21,00,41,3E
550 DATA 03,D3,A5,F3,DB,A4,D6,01
560 DATA 38,FA,7E,D3,A4,23,0B,79
570 DATA B0,20,F1,FB,C9,CD,6D,D8
580 DATA 21,00,B0,11,00,41,01,00
```

1.3.4 Advanced level: PCM in machine language!

The easiest way to use PCM in machine language is to use the BIOS. The settings for sampling rate, trigger level, etc. are almost the same as those in BASIC, so there should be no problem.

Since this is an advanced topic, we will introduce two programs that can record and play PCM without using the BIOS.

When using the BIOS, you can only choose from four sampling rates: 15.75 kHz, 7.875 kHz, 5.25 kHz, and 3.9375 kHz. This is because the BIOS uses a counter whose value changes every 63.5 microseconds, so you cannot set more than four sampling rates. The program introduced here replaces this counter with a system timer whose value changes every 3.911 microseconds.

First, let's explain how to use the recording program. The HL register is set to the top address of the memory that stores the PCM data, and the BC register is set to the size of the data to be recorded. The E register is set to the number of counts to wait in the system timer. 16 is roughly equivalent to 15.75 kHz.

The playback program is the same. Set the start address of the PCM data to be played in the HL register, the size of the data in the BC register, and the wait count in the E register.

In the middle of the list of PCM recording programs,

```
OEDH,70H
```

There is a strange thing called

```
IN (HL),(C)
```

This is an instruction unique to the R800 that reads a value from the port of the C register and reflects it only in the flags.

Regardless of the principles of the program, try using it. By changing the value of the E register, you can enjoy changing the sound in various ways.

list 1.6 (PCMREC.MAC)

```
PMDAC   EQU 0A4H
PMCNT   EQU 0A4H
PMCNTL  EQU 0A5H
PMSTAT  EQU 0A5H
SYSTML  EQU 0E6H        ; system timer port

REC:
        LD    A,00001100B
        OUT   (PMCNTL),A        ; A/D MODE
        DI
        XOR   A
        OUT   (SYSTML),A        ; reset timer

REC1:
        IN    A,(SYSTML)
        CP    E
        JR    C,REC1            ; wait
        XOR   A
        OUT   (SYSTML),A        ; reset timer

        PUSH  BC
        LD    A,00011100B
        OUT   (PMCNTL),A        ; DATA HOLD
        LD    A,80H
        LD    C,PMSTAT

        OUT   (PMDAC),A         ; BIT CONVERT
```

```
        DEFB  0EDH,70H          ; IN (HL),(C)
        JP    M,RECAD0
        AND   01111111B
RECAD0:
        OR    01000000B

        OUT   (PMDAC),A
        DEFB  0EDH,70H
        JP    M,RECAD1
        AND   10111111B
RECAD1:
        OR    00100000B

        OUT   (PMDAC),A
        DEFB 0EDH,70H
        JP   M,RECAD2
        AND  11011111B
RECAD2:
        OR   00010000B

        OUT  (PMDAC),A
        DEFB 0EDH,70H
        JP   M,RECAD3
        AND  11101111B
RECAD3:
        OR   00001000B

        OUT  (PMDAC),A
        DEFB 0EDH,70H
        JP   M,RECAD4
        AND  11110111B
RECAD4:
        OR   00000100B

        OUT  (PMDAC),A
        DEFB 0EDH,70H
        JP   M,RECAD5
        AND  11111011B
RECAD5:
        OR   00000010B

        OUT  (PMDAC),A
        DEFB 0EDH,70H
        JP   M,RECAD6
        AND  11111101B
RECAD6:
        OR   00000001B

        OUT  (PMDAC),A
        DEFB 0EDH,70H
        JP   M,RECAD7
        AND  11111110B
RECAD7:
        OR   00000000B
```

```
        LD    (HL),A
        LD    A,00001100B
        OUT   (PMCNTL),A

        POP   BC
        INC   HL
        DEC   BC
        LD    A,C
        OR    B                    ; end of data ?
        JR    NZ,REC1              ; next data

        LD    A,00000011B
        OUT   (PMCNTL),A   ;D/A MODE
        EI
        RET

        END
```

list 1.7 (PCMPLAY.MAC)

```
PMDAC   EQU 0A4H
PMCNT   EQU 0A4H
PMCNTL  EQU 0A5H
PMSTAT  EQU 0A5H
SYSTML  EQU 0E6H          ; system timier port

PLAY:
        LD    A,00000011B
        OUT   (PMCNTL),A          ; D/A MODE
        DI
        XOR   A
        OUT   (SYSTML),A          ; reset timer

PLAY1:
        IN    A,(SYSTML)
        CP    E
        JR    C,PLAY1             ; wait
        XOR   A
        OUT   (SYSTML),A          ; reset timer

        LD    A,(HL)
        OUT   (PMDAC),A           ; play 1 byte
        INC   HL
        DEC   BC
        LD    A,C
        OR    B                   ; end of data ?
        JR    NZ,PLAY1            ; next data
        EI
        RET

        END
```

Table 1.4: I/O Ports for PCM

street address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0A5H Write	0	0	0	SMPL	SEL	FILT	MUTE	ADDA
0A5H Read	COMP	0	0	SMPL	SEL	FILT	MUTE	BUFF
0A4H Write	DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0
0A4H Read	0	0	0	0	0	0	CT1	CT0

- ADDA (BUFF): Buffer mode
  Specifies the output of the D/A converter.
  Set this to 0 (double buffer) for D/A, and 1 (single buffer) for A/D. Note that the output is set to double buffer when reset.

- MUTE: Muting control
  Turns system-wide sound output on or off.

  0: Audio output off (at reset)
  1: Audio output on

- FILT: Selection of sample-and-hold circuit input signal
  Select whether the signal input to the sample-and-hold circuit during A/D conversion is the filter output signal or the reference signal. 0 is the reference signal, 1 is the filter output signal. Reset is 0.

- SEL: Filter input signal selection
  Select whether the signal input to the low pass filter is the output signal of the D/A converter or the output signal of the microphone amplifier. 0 is the D/A converter output signal, 1 is the microphone amplifier output signal sound.

- SMPL: Sample and hold signal Select
  whether to sample or hold the input signal.

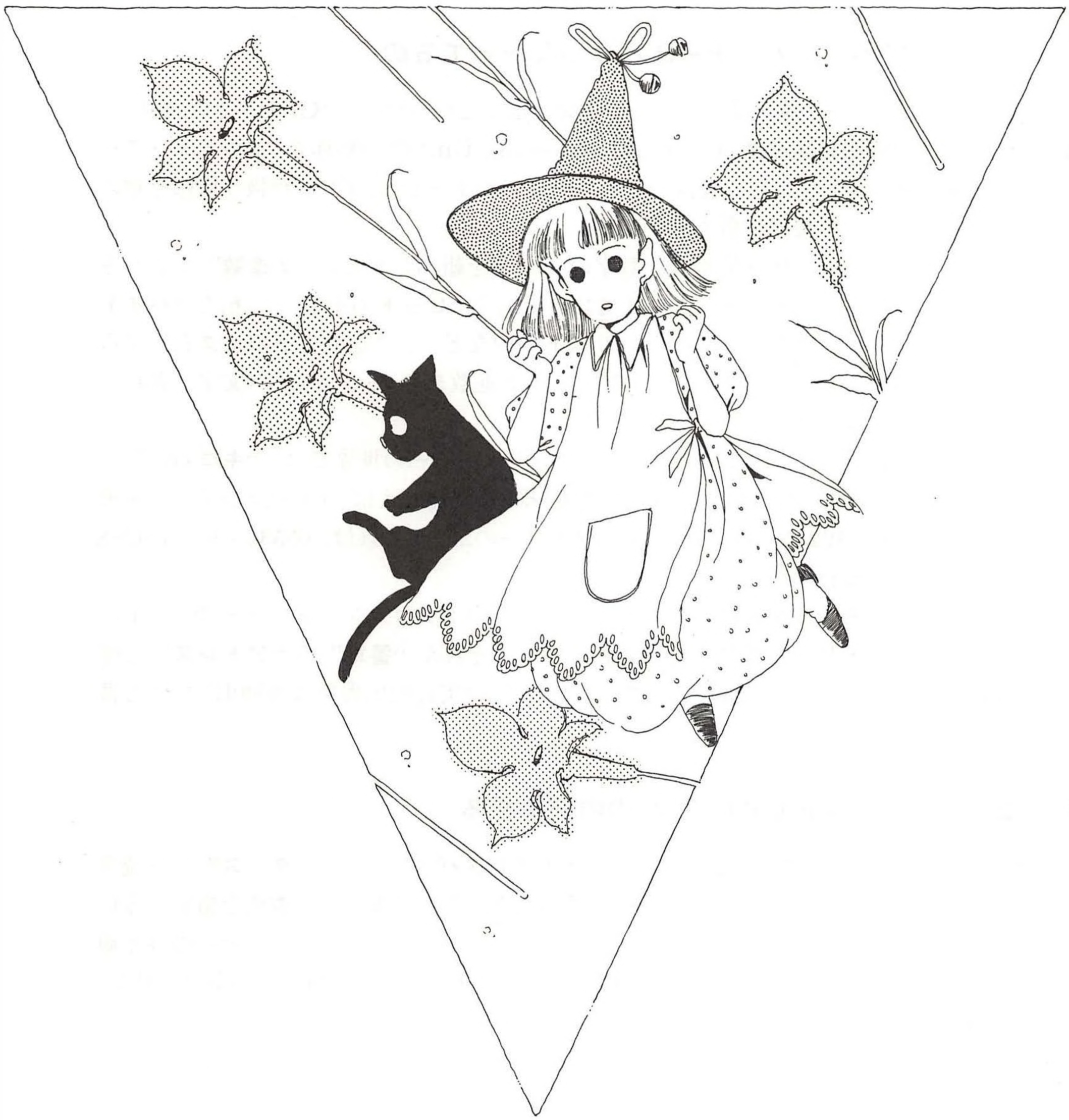
  0: Sample (at reset)
  1:Hold

- COMP: Comparator output signal. Compares the sample and hold output signal with the D/A converter output signal.
  0: D/A output > Sample hold output
  1: D/A output < sample hold output

- DA7~DA0: D/A output data
  When playing PCM data, you can play PCM sound by outputting the prepared data here. The data format is absolute binary, with 127 corresponding to level 0.

- CT1, CTO: Counterdata
  It counts up every 63.5 microseconds. During D/A conversion, it is synchronized with the count up and the data written to address 0A4H is repeatedly output. Also, writing data to 0A4H clears the counter.

第2章 SLOT



This chapter is a re-edited version of the articles "MSX2+ Technical Exploration" from the February 1989 and March 1989 issues of MSX Magazine, and "Technical Analysis" from the November 1990 issue.

2.1 What is a slot?

The hole for inserting the cartridge into the MSX is called the "cartridge slot." But the slot also serves the function of managing the MSX's memory. In this chapter, we will explain the most important and complex of the slots.

2.1.1 How is the CPU and memory connected?

The most important components of a computer are the CPU and memory. CPU is an abbreviation for Central Processing Unit, and is the device that manages the entire computer and performs calculations. Memory, on the other hand, is a pad-like device that stores the information that the CPU handles.

As you may know, the information handled by computers is represented as "binary numbers" combining the numbers 0 and 1. One digit of this binary number is called a "bit" and eight digits are called a "byte." Also, since expressing binary numbers as is in program lists etc. would result in too many digits, "hexadecimal numbers" are often used, which represent 4-bit binary numbers using the letters 0-9 and A-F.

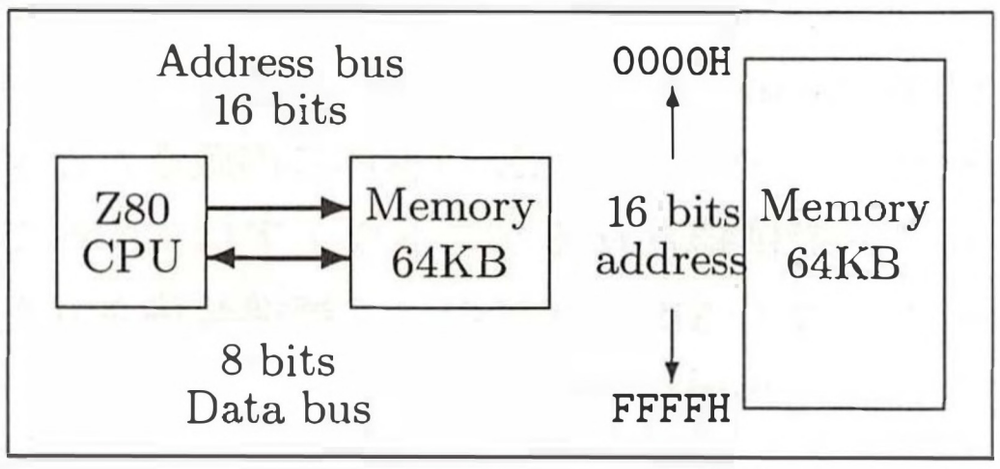
Make no mistake, in the computer world, the unit "kilo" (K) means 1024 times, not 1000 times. For example, 64 kilobytes of memory means 64 x 1024 = 65536 bytes of memory, which is also 65536 x 8 = 524288 bits.

To manage these memories, many microcomputers assign numbers to each byte of memory. These numbers are called "locations" or "addresses." In machine language programs, you often see things like "The execution start address is 8000H."

2.1.2 Exploring the inside of the 8-bit CPU Z80

As shown in Figure 2.1, the CPU and memory are connected by an "address bus" and a "data bus." The address bus is an electrical wire that sends signals from the CPU to memory specifying the memory address that the CPU wants to read or write. The data bus is an electrical wire that communicates the contents of memory. Note that the former is a one-way bus from the CPU to memory, while the latter is bidirectional.

figure 2.1: Memory for Z80 CPUs



The Z80 CPU used in the MSX can basically connect to 64KB of memory. A 16-bit address bus signal can specify one byte of memory between 0000H and FFFFH. The data bus for transferring data is 8-bit.

The "Z80", the CPU of MSX before turbo R, has an 8-bit (physically 8 wires) data bus and a 16-bit address bus. This means that it can read and write 64 kilobytes of memory one byte at a time. Such a CPU is called an "8-bit CPU", and a computer with an 8-bit CPU is called an "8-bit computer". That's why MSX is an 8-bit computer.

Now, to explain the address bus in detail, it is specified by a 16-bit address bus. The memory address that can be set is a binary number.

0000000000000000B

From (B is the symbol for binary numbers)

1111111111111111B

In decimal, this ranges from 0 to 65535, and in hexadecimal, it ranges from addresses 0000H to FFFFH (H is the symbol for hexadecimal). Each address contains 8 bits (=1 byte), which in decimal represents a value from 0 to 255. Furthermore, these memories expressed in bytes are converted to kilobytes, which equals 64. Therefore, an 8-bit computer can be connected to 64 kilobytes of memory.

I mentioned before that MSX is an 8-bit computer, but recently 16-bit computers (computers with 16-bit CPUs) have also become popular. In this case, the data bus is 16 bits, so twice as much information can be read and written at one time compared to 8-bit computers. There are also many address buses, which has the advantage of being able to connect more memory. However, due to the complex wiring, it is relatively expensive. Also, many large computers have 32-bit or 64-bit data buses and address buses.

Although the MSX turbo R was equipped with a 16-bit CPU, the R800, the data bus remained 8-bit so that conventional cartridges could be connected.

2.1.3 There are various types of memory depending on the function

There are many types of memory. First, they are classified according to the type of component: "ROM" and "RAM". ROM (Read Only Memory) is memory whose contents cannot be rewritten, but remain even when the power is turned off. All software built into the MSX (BASIC, etc.) and software supplied on cartridges is written into this ROM. Kanji ROM, which became standard equipment with the MSX2+, is a dedicated ROM with the shapes of kanji characters written into it.

RAM (Random Access Memory) is a type of memory whose contents can be freely rewritten, but whose contents are lost when the power is turned off. It is used to temporarily store the results of calculations in a program, or to load and execute a program from a floppy disk. For example, if you type in a short program from M Magazine to play a game, it will be stored in this RAM.

"SRAM" is a type of RAM that consumes little power and is used in battery-powered laptops and portable word processors, as well as battery-backed game cartridges for MSX and Famicom.

Memory can also be classified according to how it is used. As shown in Figure 2.1, memory that is directly connected to the CPU is called "primary memory" or "main memory." In other words, all ROM other than the MSX's kanji ROM and the 64 kilobytes of main RAM are the MSX's main memory.

MSX also has another memory called "Video RAM (VRAM)". Video RAM is RAM used to store the graphics and text to be displayed on the TV screen. Depending on the computer model, the video RAM may be directly connected to the CPU, but in MSX the CPU and video RAM are connected via a dedicated component called the VDP (short for Video Display Processor).

The basics explained so far are the most basic knowledge not only about MSX but also about computers. I think the introductory book on BASIC that comes with MSX will explain it in detail, so please refer to it.

2.1.4 What are MSX slots like?

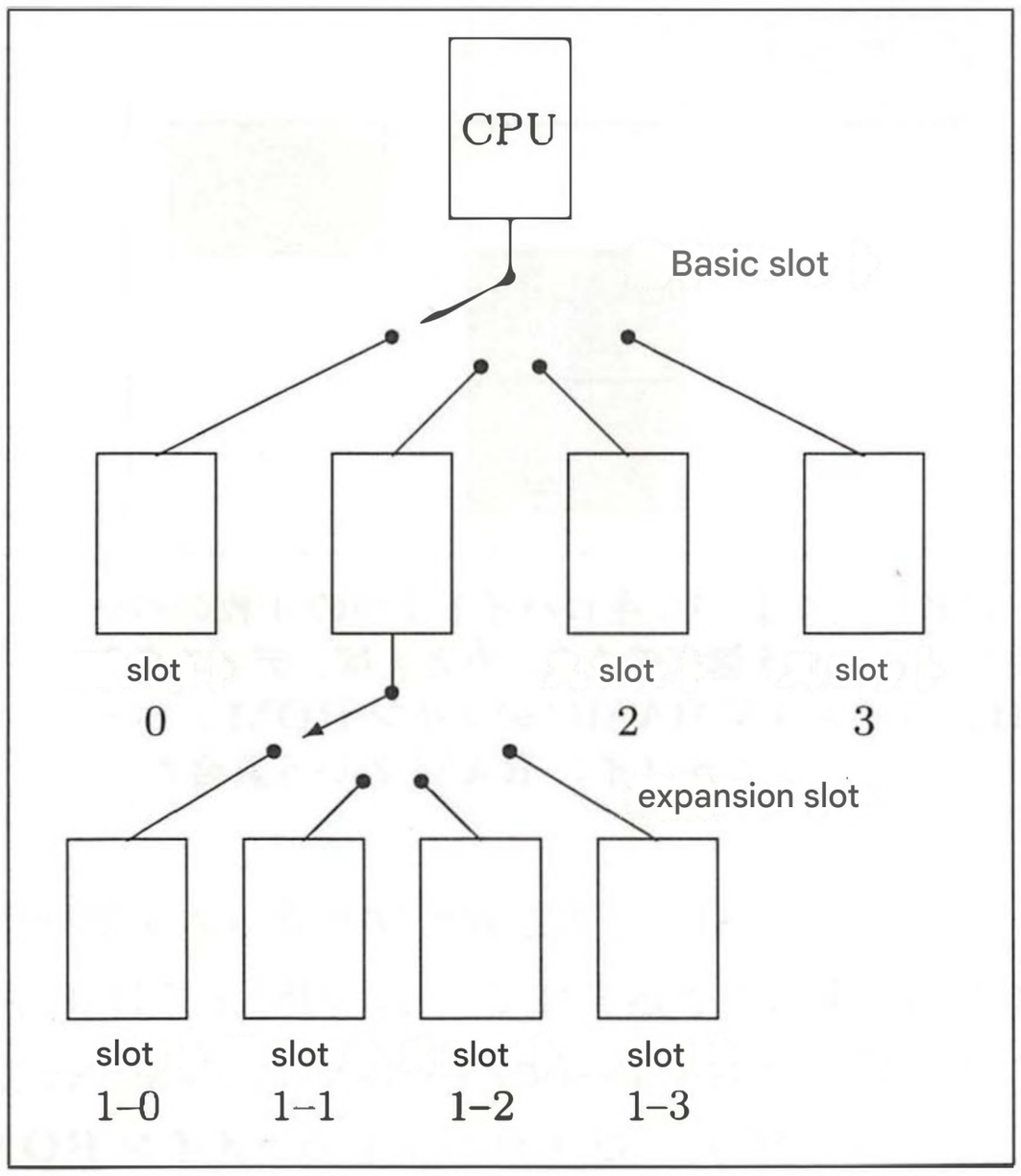
As mentioned at the beginning, the main memory that can be connected to an 8-bit CPU is 64KB. However, this was only the case for early 8-bit computers. Nowadays, there are various methods that allow you to connect more than 64KB of memory.

In the case of MSX, this is called "slot switching." As shown in Figure 2.2, four sets of 64 KB memory are prepared and can be switched between, making it possible to handle a maximum of 256 KB of memory.

These memories are called "basic slots"

They are also assigned to cartridge slots provided on the machine.

figure 2.2: MSX slot configuration (part 1)



In MSX, to handle memory exceeding 64 kilobytes, a method called “slot switching” is used. By switching between four 64-kilobyte memory blocks, up to 256 kilobytes of memory can be handled. These four memory blocks are called “primary slots,” and each of them can be expanded into four “secondary slots,” referred to as “expanded slots.”

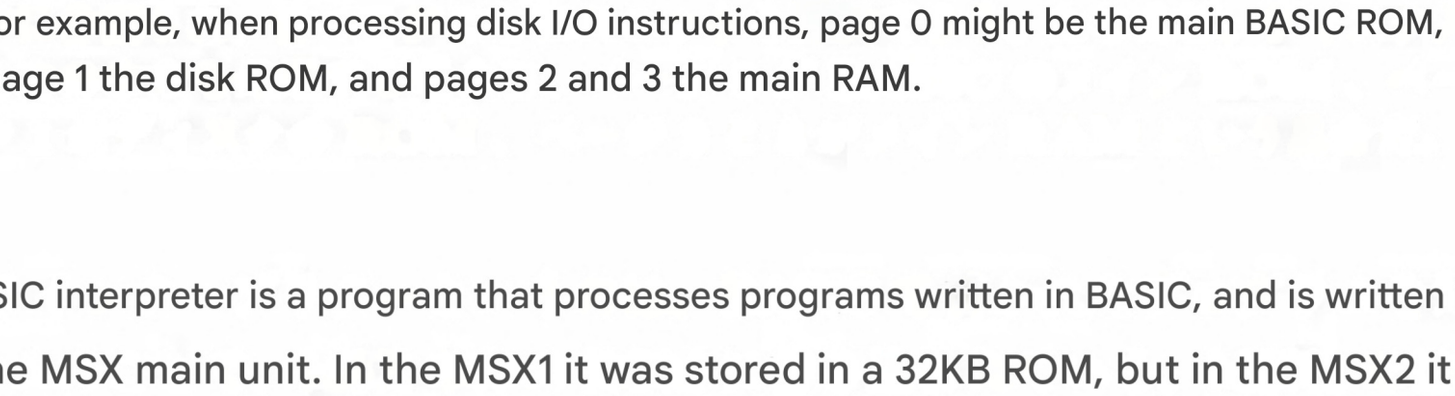
Additionally, you can use four sets of "expansion slots" instead of one basic slot, which gives you a total of 16 sets of 64KB memory, meaning you can connect a maximum of 1MB (1024KB) of memory. However, you cannot expand the expansion slots further.

Now, it's fine to be able to handle memory over 64KB by switching slots, but it's inconvenient to have the entire memory switched at the same time. So in MSX, memory is divided into units called "pages" and handled accordingly.

The 16KB from memory addresses 0000H to 3FFFH is page 0, and the 16KB from memory addresses 4000H to 8000H is page 1. Similarly, addresses 8000H to BFFFH is page 2, and addresses C000H to FFFFH is page 3. Each page has 16KB in one block, and a different slot can be selected for each page.

For example, when a BASIC disk I/O instruction is being processed, page 0 becomes the BASIC interpreter's main ROM, page 1 becomes the disk interface ROM, and pages 2 and 3 become main RAM (see Figure 2.3).

figure 2.3: MSX slot configuration (part 2)



The 64KB memory address is divided into four 16KB pages, with a selectable slot for each page. For example, when processing disk I/O instructions, page 0 might be the main BASIC ROM, page 1 the disk ROM, and pages 2 and 3 the main RAM.

The BASIC interpreter is a program that processes programs written in BASIC, and is written into the ROM of the MSX main unit. In the MSX1 it was stored in a 32KB ROM, but in the MSX2 it is 48KB. Therefore, in the MSX2 the ROM is divided into two, with the parts common to the MSX1 stored in the 32KB "main ROM" and the functions extended by the MSX2 stored in the 16KB "sub ROM".

Also, the MSX with built-in disk drive and the interface cartridge for external drives have a 16KB ROM built in. This is where the program (DISK-BASIC) for processing disk I/O in BASIC is written. Therefore, during disk I/O, the state shown in Figure 2.3 appears.

2.1.5 The secret to MSX's expandability was its slots

The MSX slots are used not only to add memory, but also to expand the functions of the MSX. The slots are also used to connect game cartridges and modem cartridges.

As I wrote above, when you connect a disk interface cartridge to the MSX, the 16KB ROM in the cartridge is connected to a slot. Then, when disk I/O is performed, the memory is automatically switched to the disk interface ROM slot. Therefore, whether the disk interface itself is built into the main unit or connected as a cartridge, there is no problem with the operation of the program.

Also, when a disk interface is connected, the "CALL FORMAT" command becomes available, and when a communications cartridge is connected, the "CALL TELCOM" extended BASIC command becomes available. These extended commands are processed by the ROM in the cartridge. With most computers other than MSX, when using peripheral devices, it is necessary to load the programs that control them from the disk. However, with MSX, simply connecting an interface cartridge automatically extends the BASIC commands.

In addition, "Japanese MSX-DOS2", an improved version of MSX-DOS, "HALNOTE", a popular integrated software, and "MSXView", a GUI (Graphical User Interface) for turbo R, were also among the software supplied on cartridges. In this way, the ability to easily expand functions by plugging ROM cartridges into the slot is an advantage of MSX that other personal computers do not have.

Although slots are a useful feature, developing a program that uses them effectively is quite difficult. Even for a programmer who is fluent in writing machine language programs for the Z80 CPU, it may take more than a year to truly understand the concept of slots.

2.1.6 How the MSX2+ slot has changed

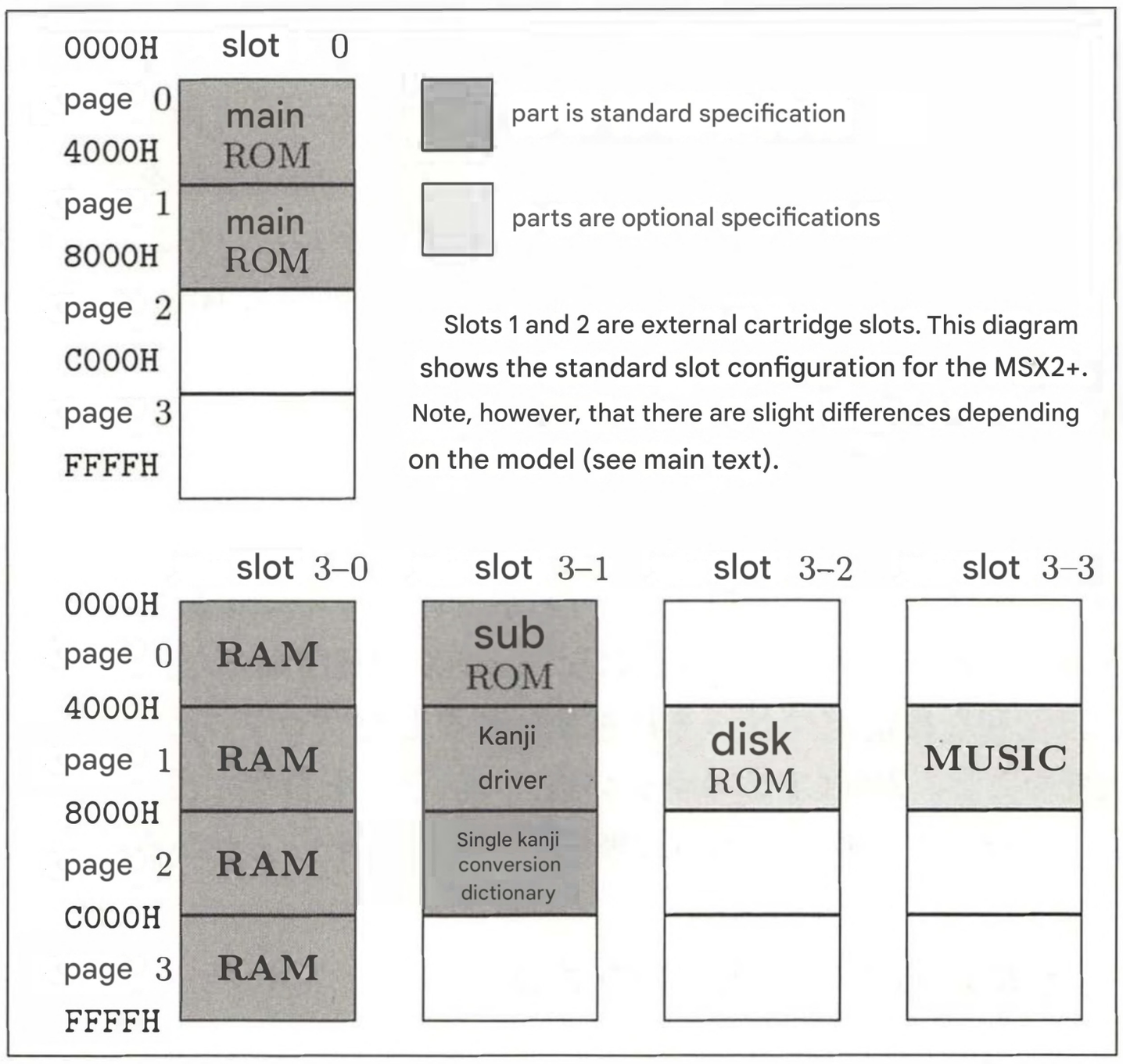
As I have written so far, the slots gave the MSX machine a lot of expandability and character. However, these slots were also a weakness of the MSX. That is, "Because the slot configuration differed depending on the model in the past, software compatibility problems were likely to occur."

For example, certain software may not work on models where slots 1 and 3 are assigned as cartridge slots. Also, if RAM is placed in the expansion slot (slot 3), the sub-ROM functions may not be available from DOS. These problems should be avoided by carefully creating a program and checking that it works on all MSX machines. However, making a program compatible with machines with all slot configurations may result in longer programs and slower execution speeds. Slots are not a straightforward matter.

With the arrival of the MSX2+, a certain degree of standard for slot configuration was finally established. Figures 2.4 and 2.5 show examples of MSX2+ slot configurations. Depending on the amount of software built into the console, there are cases where only slot 3 is expanded, and cases where both slot 0 and slot 3 are expanded.

Figure 2.4 shows an example where only slot 3 is expanded. The main BASIC ROM is placed in basic slot 0, and slots 1 and 2 are used as external cartridge slots.

Figure 2.4: Example of MSX2+ slot configuration (when expanding slot 3 only)



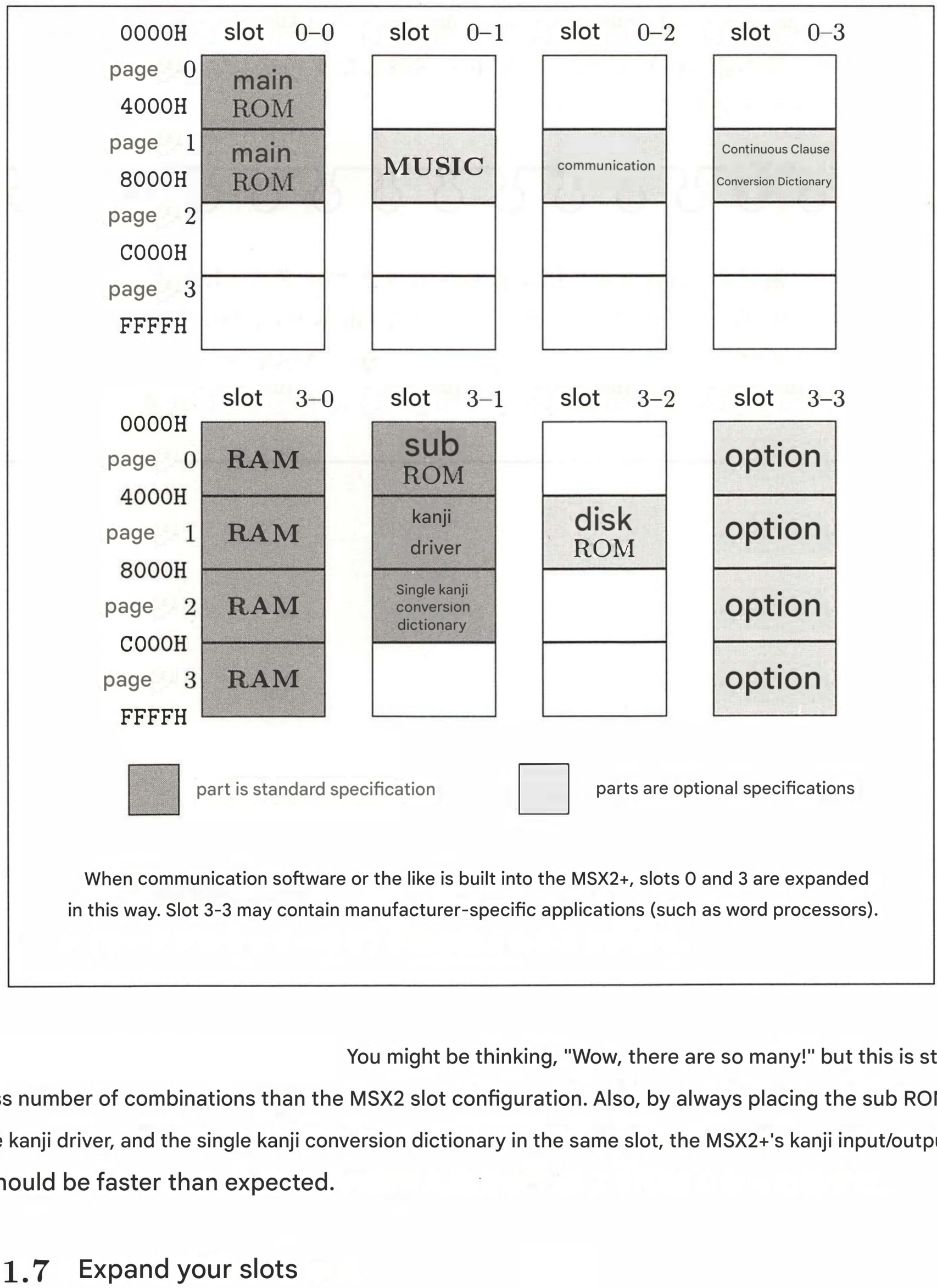
Then, place 64KB of RAM (main RAM) in one of the expansion slots of slot 3, so that pages 0 to 3 always have RAM in the same slot. Similarly, place a total of 48KB of ROM, including the sub-ROM, kanji driver, and single-kanji conversion dictionary, in one of the expansion slots of slot 3. In the diagram, the RAM is placed in slot 3-0 (the 0th expansion slot of basic slot 3), and the ROM in slot 3-1, but this will differ depending on the model.

In contrast, Figure 2.5 shows the case where both slot 0 and slot 3 are expanded. The main ROM is placed in expansion slot 0 of the base slot 0. The configuration of slot 3 is almost the same as in Figure 2.4. Please note that if a disk interface is built in, the ROM will always be placed in the expansion slot 3, not the expansion slot 0.

The light gray parts in Figure 2.4 and Figure 2.5. In other words, the disk interface, MSX-MUSIC (FM sound source), communication, and multi-phrase conversion dictionary ROMs are optional for the MSX2+. Therefore, they are sometimes not built into the main unit, but are connected as external cartridges.

Now, with the MSX2+ having the same amount of built-in software as shown in Figure 2.5, there are theoretically 36 possible slot configurations.

Figure 2.5: Example of MSX2+ slot configuration (when expanding slots 0 and 3)



When communication software or the like is built into the MSX2+, slots 0 and 3 are expanded in this way. Slot 3-3 may contain manufacturer-specific applications (such as word processors).

You might be thinking, "Wow, there are so many!" but this is still less number of combinations than the MSX2 slot configuration. Also, by always placing the sub ROM, the kanji driver, and the single kanji conversion dictionary in the same slot, the MSX2+'s kanji input/output should be faster than expected.

2.1.7 Expand your slots

MSX machines have one or two external cartridge slots (where game cartridges are usually inserted) which are all basic slots. By connecting a "slot expander", it is possible to expand the number of slots to four. For example, if you connect a slot expander to both of the main body's two slots, you will have a total of eight slots.

However, you should be aware that some cartridges do not work in the expansion slot. Japanese MSX-DOS2 (type with built-in RAM) is one of them. Check the software you want to use before expanding.

The product name of the slot expander is "MSX Expansion Slot Box EX-4". It is currently on sale from Nippon Electronics (Tel: 03-3486-4181) for 29,800 yen (excluding tax). In addition to this, several MSX manufacturers have released slot expanders, but most are now unavailable.

2.2 Try switching slots

When talking about MSX, the concept of slots is unavoidable. Here, we will focus on how to control slots with software and the changes in specifications for MSX2+.

2.2.1 To switch slots

I've explained the meaning of slots in MSX, but I haven't explained how to switch between slots. So from here on, I'll show you how to do it. Of course, it's not something as unsophisticated as "a human switching it with a switch," but rather it can be switched by a program.

First, the MSX CPU, the Z80, has something called an "I/O port." This is like a telephone line that allows the CPU to communicate with the outside world (i.e. peripheral devices such as VDP and FM sound sources). The Z80 has a total of 256 I/O ports, and they are differentiated by assigning addresses from 0 to 255 (00H to FFH in hexadecimal).

To handle these ports in BASIC, you read a byte from an I/O port with the "INP" function and write a byte with the "OUT" command, and in machine language the "IN" and "OUT" commands perform the same function.

Now, the basic slot is switched depending on the value written to the A8H address of this I/O port. Conversely, by reading the value of this address, you can find out the current status of the slot. Bits 7 and 6 correspond to page 3, 5 and 4 to page 2, 3 and 2 to page 1, 1 and 0 to page 0, and so on, so writing the value 11110000B (B stands for binary) will switch page 3 and 2 to slot 3, and pages 1 and 0 to slot 0. Also, to switch expansion slots you use memory address FFFFH, but this is complicated so we won't go into it here.

However, directly switching slots using a program is not only tedious, but it is also prone to compatibility issues, such as the program not working on some models.

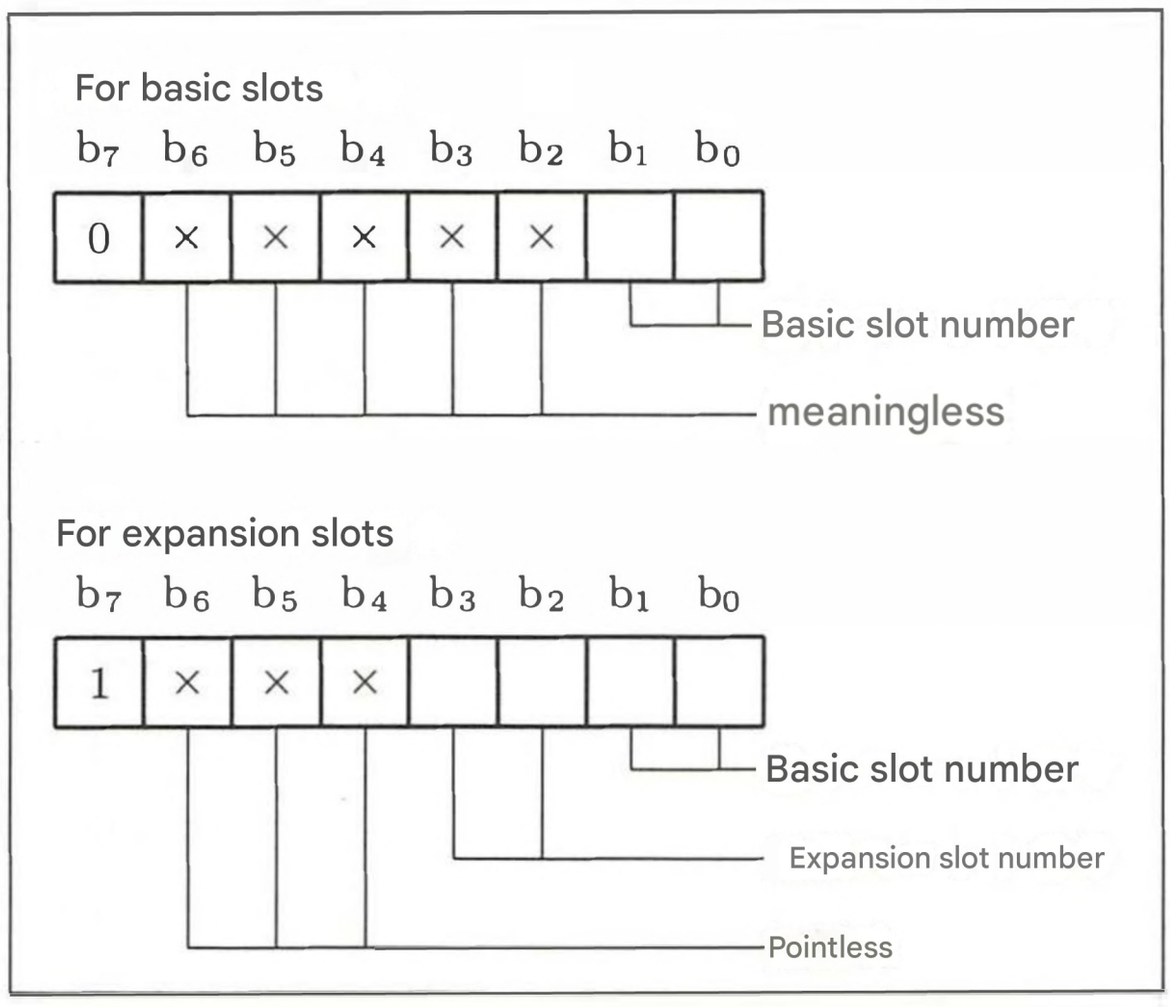
So, in reality, we switch slots using the "BIOS". BIOS stands for "Basic Input Output System". I'll explain it in detail later, but it's a set of machine language subroutines to control hardware. It has many functions other than switching slots, so let's check it out.

2.2.2 How to specify slot number

When using the BIOS to switch slots, you can specify the base slot number and the expansion slot number separately. However, this requires two registers (temporary data storage locations in the CPU), which is uneconomical. Therefore, a method is used to specify the base slot and expansion slot together by making good use of each bit of 8 bits (1 byte), as shown in Figure 2.6.

For example, to specify base slot 0, use the value 00000000B (00H in hexadecimal), and to specify base slot 3 and expansion slot 1, use the value 10000111B (87H).

figure 2.6: How to specify slot number



The slot number is represented by an 8-bit (= 1 byte) value as shown in the diagram. For example, to specify base slot number 0, use the value 00000000B, and to specify expansion slot 1 of base slot 3, use the value 10000111B. Note that the contents of each bit marked with an x in the diagram are ignored.

2.2.3 BIOS functions for handling slots

First, let's learn the symbols used to represent BIOS functions. E is the register that should be set before calling the BIOS. R is the register to which the BIOS returns the value. M is the register to which the BIOS writes a meaningless value, i.e. the original contents are destroyed. IYH is the upper byte of the IY register, and the contents of the lower byte are ignored. The address written at the beginning is the entry address for calling the BIOS.

RDSLT 000CH address

function Reads the contents of the address specified by the HL register in the slot specified by the A register.

E A slot number

HL address

R A read value

M AF, BC, DE

Note Interrupts are disabled.

WRSLT 0014H address

function Writes the contents of the E register to the address specified by the HL register in the slot specified by the A register.

E A slot number

HL address

EContents to be written

R none

M AF、BC、D

Note Interrupts are disabled.

CALSLT 001CH address

function Calls a subroutine in another slot.

E IX Address to call

IYH slot number

R Depends on who you call

M IX, IY, back register

Note The current state of the slot is saved on the stack and the desired subroutine is called. The contents of the AF, BC, DE, and HL registers are passed to the subroutine as is, and when the subroutine executes the RET instruction, it returns to the original program. In this case, the values of the AF, BC, DE, and HL registers are passed from the subroutine. The number of bytes of stack used depends on the slot configuration.

ENASLT 0024H address

function Switch slots.

E A slot number

H page (upper 2 bits)

R none

M AF、BC、DE、HL

Note For example, to switch between two pages, set a value between 80H and BFH in the H register. Interrupts are disabled.

CALLF 0030H address

function Calls a subroutine in another slot.

E As shown in the following program, write the slot number and address into the program after the "RST 30H" command.

```
RST   30H
DB    slot number
DW    address
```

R Depends on who you call

M IX, IY, back register

Note Except for the method of specifying the slot and address, it is the same as CALSLT. It is used for special purposes (hooks).

EXTROM 015CH address

function Call sub ROM.

E IX Address to call

R Depends on who you call

M IX, IY, back register

Note It works the same as CALSLT, except that the sub ROM slot is automatically selected.

The BIOS introduced above has some limitations. None of them can be used for page 3. They can only be used for page 0 when calling the main ROM from DOS. They can be used without problems for pages 2 and 3. Even though they "cannot be used", they may be usable depending on the slot configuration, so be careful not to create a program that only works on your MSX. In particular, if you use CALSLT to call the sub ROM from DOS, it may or may not work depending on the slot configuration and the type of disk interface.

2.2.4 How to know the slot configuration

As explained before, the slot configuration of the MSX varies depending on the model. There are also optional specifications such as disk interfaces, so it is no exaggeration to say that there are as many slot configurations as there are machines. Here we will introduce how to check the slot configuration of your machine and whether or not it has optional equipment.

Memory addresses F380H to FFFEH are called the "system work area", and important information for the BIOS and other devices is stored here. When a disk interface is connected, a "disk work area" is prepared at an address slightly smaller than the system work area. Information related to slots is stored in the system work area and disk work area, as shown in Table 2.1.

Although it is important to know which slot the main RAM is in, the “RAMAD0” in Table 2.1 is in the disk work area, so there is also the problem that you cannot know this information without the disk.

List 2.1 is a program that displays the slot configuration found from these system work areas in an easy-to-understand way. The configuration varies depending on the model, so try it out on your own MSX.

Other than those listed in Table 2.1, there are other system work areas that are useful for programs, but for details, please refer to the "MSX2 Technical Handbook". Also, unless specifically instructed, application programs should not rewrite these system work areas. Some programs use the system work area as a last resort when memory is insufficient, but be careful because this can cause compatibility issues.

Table 2.1: System work areas for slots

name	address	meaning
RAMAD0	F341H	RAM slot number for page 0 (1)
RAMAD1	F342H	RAM slot number for page 1 (1)
RAMAD2	F343H	Page 2 RAM slot number (1)
RAMAD3	F344H	Page 3 RAM Slot Number (1)
MASTER	F348H	Slot number of drive A's interface (1)
EXBRSA	FAF8H	Sub ROM slot number (0 for MSX1)
EXPTBL	FCC1H	Main ROM slot number
	FCC2H	Whether slot 1 is expanded (2)
	FCC3H	Whether slot 2 is expanded (2)
	FCC4H	Whether slot 3 is expanded (2)

(1) Valid only if disk is present.
(2) If it is extended then 80H, if not then .

2.2.5 Explore the system work area

Among the MSX2 games, there are some that display the title screen of SCREEN 12 when run on MSX2+. There are also programs that display a helpful error message if you try to communicate without a modem cartridge. In order to create such software, we will introduce a method for programs to check the type and configuration of the hardware.

First, to check whether a disk is present, the contents of address FFA7H are read. If it is C9H then there is no disk, if it is any other value there is a disk.

To check the "type of MSX", read address 2DH of the main ROM. 0 means MSX1, 1 means MSX2, 2 means MSX2+, and 3 means turbo R.

Generally, the contents of addresses 2BH and 2CH are 0, but in "MSX made for export to overseas", they contain numbers that indicate the type of keyboard or currency symbol. Since you only need to be concerned with these when making software for export, we will omit the list of numbers.

This is a bit off topic, but MSX computers have been exported in large numbers to Europe, the Soviet Union, and Kuwait in the Middle East. In neighboring South Korea, many of them are installed in schools and used in classes. It's a truly international machine.

Now, to find out the "video RAM capacity", read the FAFCH address. If bit 2 and bit 1 are 00, it's 16KB, if 01, it's 64KB, if 10, it's 128KB. Ignore the other bits, as they seem to be used for other purposes.

As shown in the list on the next page, we use "AND 6" to take the values of bits 2 and 1 and divide them by 2.

As an added bonus, this list also checks whether or not there is an "extended BIOS." This is a function for controlling optional hardware such as a communications modem, FM sound source, or kanji dictionary. If the content of bit 0 at address FB20H is 1 and the content of address FFCAH is not C9H, then there is some kind of extended BIOS function. Finding out what it is would require a complex machine language program, so we'll pass on that this time. Also, although this is not written in the specifications, the content of address FFCBH seems to be the slot number of a program that has an extended BIOS function.

By the way, we don't know what is written in bits that don't have a defined meaning, such as bits 6 to 4 of the value that represents the slot number. So we use "AND" to ignore those bits.

As I've mentioned many times before, slot configurations can vary between machines, even from the same manufacturer. So after testing it on your own machine, try it on your friends' machines too. It's interesting to test it on many machines and compile a table of the results.

list **2.1 (WHO_AM_I.BAS)**

```
100 ' Analizing slot structure of MSX
110 ' by nao-i on 9. Jan. 1989
120 CLEAR : DEFINT A-Z : CLS
130 VE = PEEK(&H2D) : ' Version No. of BASIC
140 IF VE=0 THEN PRINT "I am MSX1"
150 IF VE=1 THEN PRINT "I am MSX2"
160 IF VE=2 THEN PRINT "I am MSX2+"
165 IF VE=3 THEN PRINT "I am MSX turbo R"
170 IF VE>3 THEN PRINT "Who am I ?"
180 VR = (PEEK(&HFAFC) AND 6) ¥ 2 : ' size of VRAM
190 IF VR=0 THEN PRINT "VRAM 16KB"
200 IF VR=1 THEN PRINT "VRAM 64KB"
210 IF VR>1 THEN PRINT "VRAM 128KB"
220 MR = PEEK(&HFC49) : ' size of main RAM
230 IF MR >= &HE0 THEN PRINT "RAM 8KB"
240 IF MR < &HE0 AND MR >= &HC0 THEN PRINT "RAM 16KB"
250 IF MR < &HC0 THEN PRINT "RAM >= 32KB"
260 FOR SS = 0 TO 3 : 'EXPTBL
270   PRINT USING "Slot # is "; SS;
280   FF = PEEK(&HFCC1 + SS) AND 128
290   IF FF THEN PRINT "expanded slot" ELSE PRINT "primary slot"
300 NEXT SS
310 PRINT
320 SS = PEEK(&HFCC1) : PRINT "Main ROM is in "; : GOSUB 530
330 SS = PEEK(&HFAF8) : PRINT "Sub  ROM is in "; : GOSUB 530
340 IF PEEK(&HFFA7) <> &HC9 THEN 360
350 PRINT "I have disk(s)." : GOTO 450
360 PRINT "I have no disk."
370 SS = PEEK(&HF348) : PRINT "FDC ROM is in "; : GOSUB 530
380 SS = PEEK(&HF341) : PRINT "P0  RAM is in "; : GOSUB 530
390 SS = PEEK(&HF342) : PRINT "P1  RAM is in "; : GOSUB 530
400 SS = PEEK(&HF343) : PRINT "P2  RAM is in "; : GOSUB 530
410 SS = PEEK(&HF344) : PRINT "P3  RAM is in "; : GOSUB 530
420 PRINT "Bottom address of disk work area is ";
430 PRINT RIGHT$("00"+HEX$(PEEK(&HFC4B)),2);
440 PRINT RIGHT$("00"+HEX$(PEEK(&HFC4A)),2)
450 ' detectiong extended BIOS
460 IF (PEEK(&HFB20) AND 1) = 0 THEN GOTO 520
470 IF PEEK(&HFFCB) = &HC9 THEN GOTO 520
480 PRINT : PRINT "I have extended BIOS."
490 SS = PEEK(&HFFCB)
500 PRINT "ROM of the extended BIOS may be in "
510 GOSUB 530
520 END
530 ' displaying slot number
540 PRINT USING "primary slot #"; SS AND 3;
550 IF (SS AND 128) = 0 THEN 570
560 PRINT USING " extended slot #"; (SS AND 12) ¥ 4;
570 PRINT : RETURN
```

2.2.6 MSX2+ Hardware Specifications

The MSX2+ included minor hardware improvements. These are summarized in Table 2.2. However, these are I/O ports for which new specifications have been defined or added.

Table 2.2: MSX2+ I/O ports

I/O address	Purpose
7CH	Built-in FM sound source
7DH	Built-in FM sound source
DAH	Level 2 Kanji ROM
DBH	Level 2 Kanji ROM
F4H	Controlling initialization
F5H	device enable

This is a new specification definition or addition. However, in actual programs, it is better to use the BIOS rather than using the I/O ports directly.

Addresses 7CH and 7DH are I/O ports for controlling the FM tone generator built into the unit. In contrast, the FM tone generator supplied by cartridge (if released in the future) will use the same I/O port as Panasonic's "FM-PAC".

To check if the unit has a built-in FM sound source, read addresses 4018H to 401FH for each slot. If the contents match the string "APRLOPLL", that slot contains a ROM program that controls the FM sound source, and the unit has a built-in FM sound source.

On the other hand, in the case of FM sound cartridges, the contents from address 4018H seem to be four letters indicating the product type and the letter "OPLL", such as "PAC2OPLL".

In "MSX-Write" and some modem cartridges, when you select "BASIC" in the first menu, the MSX title screen appears as if it was reset. This is because, for the sake of software preparation, the software jumps to address 0 of the main ROM and performs the same process as a reset.

Previously, there was no way to reliably distinguish between a jump to address 0 and a real reset, which could lead to software malfunctions. However, starting with the MSX2+, hardware was added to the I/O port at address F4H to check the reset state. However, in reality, the BIOS is added to the MSX2+ main ROM as follows:

```
CALL 17AH
OR   80H
CALL 17DH
JP   0
```

The ROM cartridge program called during initialization is

```
CALL 17AH
```

Then, if bit 7 of the A register is 0, it's a real reset. If it's 1, it's a jump to 0, and you know that you've been called.

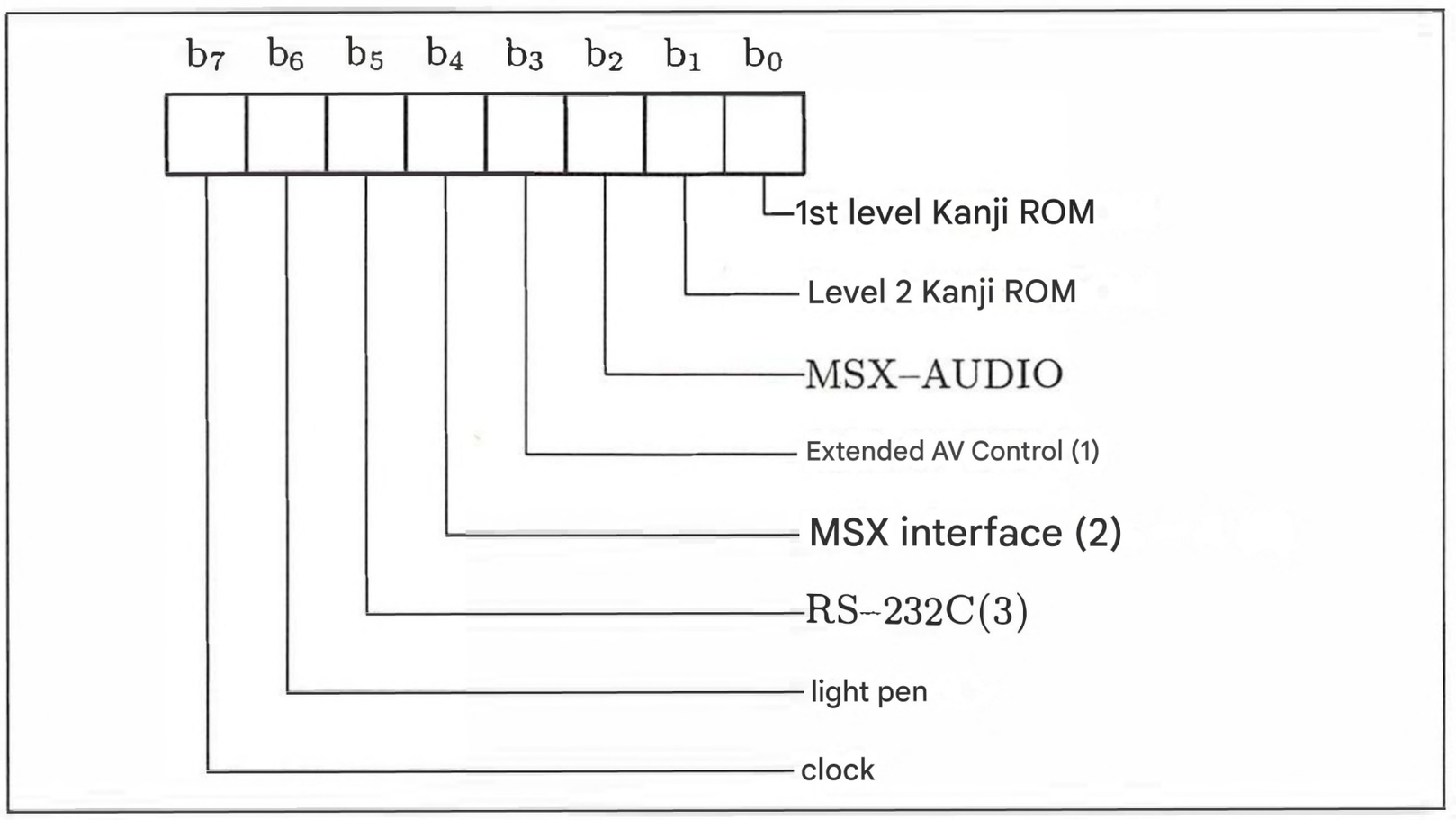
2.2.7 Device enable to prevent collisions

If you connect a kanji ROM cartridge to an MSX that has a built-in kanji ROM, not only will the kanji not be displayed correctly, but there is also the risk of hardware conflicts and failure. A useful feature to prevent this is "device enable," which is controlled by the I/O port address F5H.

The hardware shown in Figure 2.7 is disconnected from the bus at reset. Then, by writing a 1-byte (8-bit) value to the I/O port address F5H, the internal hardware corresponding to the bit that is set to 1 is connected to the bus. These processes are performed automatically after a reset or jump 0 (software transfers program execution to address 0 of the main ROM).

In MSX2, there was no provision for whether or not to disconnect already connected hardware when writing 0 to the I/O port address F5H. This meant that confusion could occur when commands such as "MSX-Write" tried to jump to address 0 of the ROM to reinitialize the BIOS.

figure 2.7: Device Enable



The I/O port address F5H is used to select whether to enable or disable the built-in hardware (the hardware corresponding to the bit with a 1 written to it is enabled).

(1) Superimpose function, etc., controlled by the F7H address of the I/O port.

(2) It is in the specifications but has not been put into practical use.

(3) This has nothing to do with modems.

Starting with MSX2+, however, the internal hardware is now unified so that writing 0 disconnects it from the bus. This means that by creating basic software for MSX2+ using I/O port addresses F4H and F5H, compatibility and reliability will be improved even more than before.

2.3 MSX Trubo R slot configuration

Here, I will explain the newly announced slot configuration of MSX turbo R. What is particularly noteworthy is that the slot configuration has finally been unified. This is quite significant.

2.3.1 Finally, the slot configuration has been unified.

Figure 2.8 shows the slot configuration of turbo R. The slot configuration was unified to accommodate faster CPU speeds and to make it easier to develop and debug application programs.

In this diagram, it looks like there is 64KB RAM in slot 3-0, but in reality, there is 256KB of main RAM connected through the memory mapper. The part exceeding 64KB is usually used for Japanese MSX-DOS2 work area, RAM disk, "DRAM mode" explained in another chapter, etc.

However, application programs can use the extended BIOS to switch mappers and use this RAM.

The DOS system ROM is stored in page 1 of slot 3-2. However, it also contains a 16KB DOS1 (MSX-DOS) ROM and a 48KB DOS2 ROM, which are automatically switched as needed.

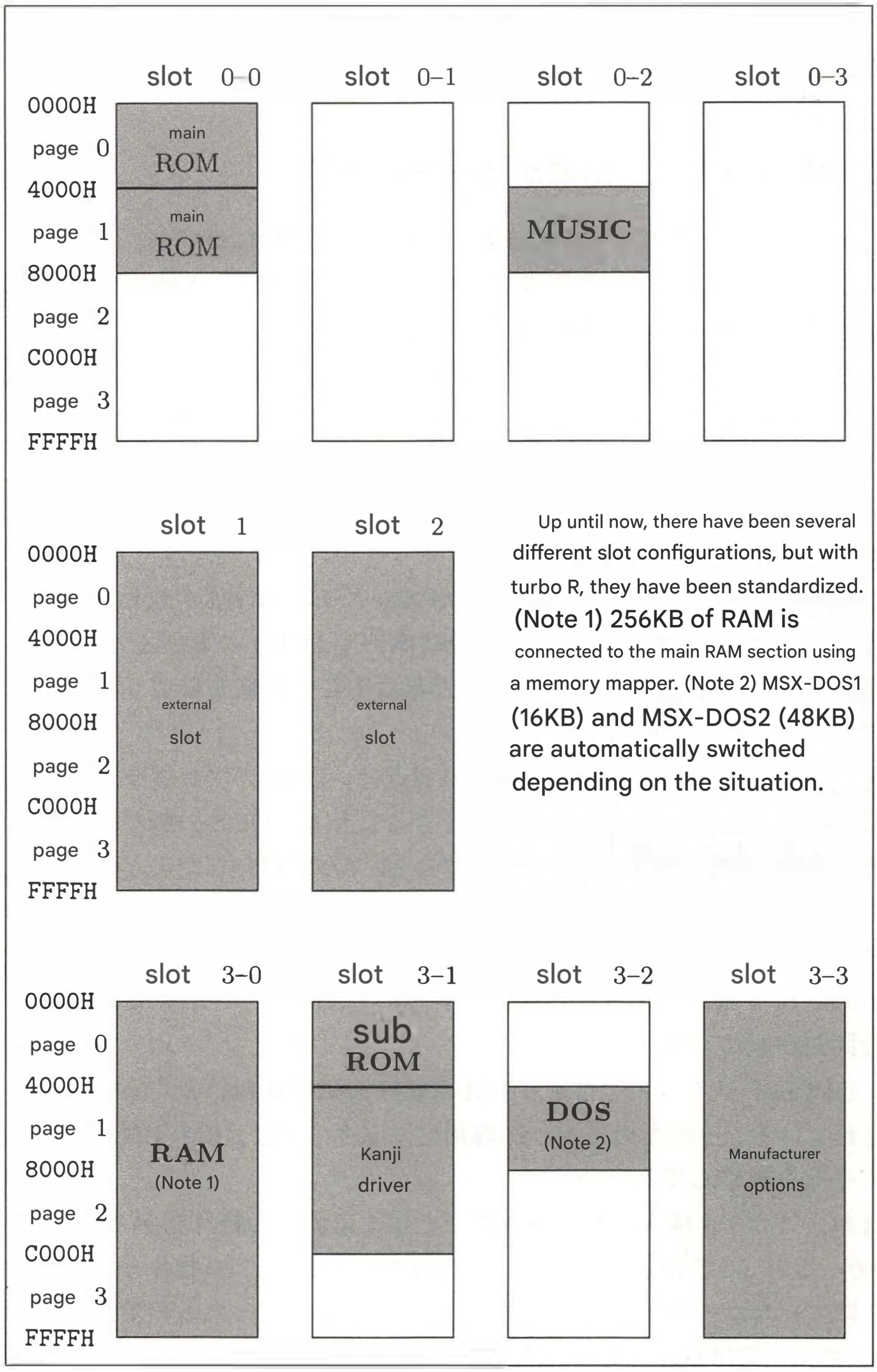
The biggest advantage of this standardized slot configuration is that DOS programs can make interslot calls to the subROM in the normal way, and that DOS interrupt processing programs can be placed in any address. As I mentioned in an old M Magazine, if there was RAM and subROM in the expanded slot 0, there was a possibility that the MSX-DOS interslot call function and interrupt processing programs would go out of control.

However, the turbo R has RAM and sub-ROM in the expanded slot 3, so this problem does not occur.

Also, the OPLL driver, or FM-BIOS ROM, is always placed in slots 0-2. Therefore, the turbo R dedicated software can omit the step of searching for the slot where FM-BIOS is located.

Another special example of the benefits of a unified slot configuration is the "Konami 10x cartridge." This required the 10x cartridge to be in slot 1 and the game cartridge in slot 2, and did not work on some MSX1 and MSX2 machines. However, the MSX2+ and turbo R fixed the external slots to slots 1 and 2, so this kind of special program could be easily and reliably implemented.

Figure 2.8: MSX turbo R slot configuration



And the biggest benefit for software houses is that they will be less bothered by bugs that arise with specific slot configurations, since the slot configuration is now standardized.
From the perspective of software developers, the standardization of slot configuration is far more pleasing than the faster CPU or the increased RAM capacity. Long live turbo R!