

MSX Supervised by magazine editorial department

MSX turbo R Technical Hand Book

MSX turbo R Technical handbook

MSX turbo R Technical Hand Book

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

It's made possible 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



- MSX and MSX-DOS are trademarks of ASCII.
 - MS-DOS is a trademark of Microsoft Corporation.
 - OS-9 is a trademark of Microware Systems, Inc. and Motorola, Inc.
 - TEX is a trademark of the American Mathematical Society.
 - MicroTEX is a trademark of Addison-Wesley Publishing, USA.
 - Other CPU names, system names, product names, etc. used in this manual are generally trademarks of their respective developers.
- The TM and R marks are not stated in the text.

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

¹Commercially 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

1	MSX turbo R	15
1.1	MSX turbo R hardware	16
1.1.1	Here are the features of the MSX turbo R!	16
1.1.2	MSX turbo R system configuration	16
1.1.3	Elegant CPU switching	18
1.1.4	Everything packed into MSX turbo R ROM configuration	18
1.1.5	System timer to adjust speed	19
1.1.6	MSX turbo R I/O port	20
1.1.7	DRAM mode for maximum speed	22
1.1.8	Here are the features of the R800!	23
1.1.9	All about the R800	23
1.2	MSX turbo R <small>How to use</small>	27
1.2.1	Programming that takes advantage of the speed of the R800	27
1.2.2	Precautions and issues when using the R800	27
1.2.3	Added BIOS and its function description.	28
1.2.4	What BIOS was changed or removed?	31
1.2.5	Notes on application development	32
1.2.6	Example of a program that switches CPUs	33
1.3	How to use PCM to its limits	37
1.3.1	Basics.....How to use in BASIC	37
1.3.2	PCM-related BASIC instructions	38
1.3.3	Play the BEEP sound with PCM!	39
1.3.4	<small>Advanced edition</small>PCM in machine language!	41
2	SLOT	47
2.1	What is a slot?	48

2.1.1	How is the CPU and memory connected?	48
2.1.2	Exploring the inside of the 8-bit CPU Z80	48
2.1.3	There are various types of memory depending on the function	50
2.1.4	What are MSX slots like?	50
2.1.5	The secret to MSX's expandability lay in its slots	52
2.1.6	The MSX2+ slot has changed	53
2.1.7	Expand your slots	55
2.2	Try switching slots	57
2.2.1	To switch slots	57
2.2.2	How to specify the slot number	57
2.2.3	BIOS functions to operate slots	58
2.2.4	How to know the slot configuration	60
2.2.5	Explore the system work area	61
2.2.6	MSX2+ hardware specifications	64
2.2.7	Device enable to prevent collisions	65
2.3	MSX Turbo R slot configuration	67
2.3.1	Finally, the slot configuration has been unified	67
3	Kanji BASIC	71
3.1	Analyze Kanji BASIC	72
3.1.1	Hardware required for Kanji BASIC	72
3.1.2	What is MSX-JE compatible software?	73
3.1.3	Explaining the operating principle of the Kanji driver	73
3.1.4	JE compatible hardware & software	75
3.1.5	Various screen modes available in Kanji BASIC	76
3.1.6	Kanji text and kanji graphics	77
3.1.7	This is the correct way to use the Kanji Driver	78
4	V9958 VDP	81
4.1	V9958 Register List.	83
4.2	What's new in V9958	85
4.2.1	horizontal scroll	85
4.2.2	Weight	87
4.2.3	command	87
4.2.4	YJK style display	87
4.3	Discontinued functions of V9958	89

4.4	V9958 Hardware Specifications (Changes)	90
4.5	V9958 and MSX2+	91
4.5.1	There are 12 screen modes in total	91
4.5.2	Controls the VDP register	92
4.5.3	V9958 Register	95
4.5.4	Horizontal scrolling with VDP	95
4.5.5	Whatever you do, don't use any tricks	98
4.6	Dissecting the YJK Method	99
4.6.1	Television broadcasting and the YJK system	99
4.6.2	RGB and YJK data structures	99
4.6.3	color sample program	101
4.6.4	It's a deadly logical operation	103
4.6.5	So-called discoloration	105
4.6.6	What is the difference between SCREEN 10 and 11?	105
4.6.7	To use subtitles in SCREEN 11	106
4.6.8	A trick to display text on SCREEN 12	108
4.6.9	YJK method and VDP register	108
4.7	Studying Scanline Interrupts	110
4.7.1	How does the monitor screen display?	110
4.7.2	Interlaced TV broadcasting	111
4.7.3	Interlaced screen on MSX2	112
4.7.4	Exploring the principles of scan line interruption	113
4.7.5	Here is an example of scan line interruption	114
4.7.6	Let's finally get to the practical part!	116
4.7.7	VDP register used for scan line interrupts	116
4.7.8	How to assemble and how the BASIC part works	119
4.7.9	This is how the assembler part works	121
4.7.10	This is the machine routine for the scan line interrupt	128
5	MSX-MUSIC	129
5.1	What is an FM synthesis source?	130
5.1.1	The history of electronic musical instruments leading to FM synthesis.	130
5.1.2	Let's analyze the sound of the instrument	132
5.1.3	The pitch is not necessarily equal tempered	134
5.1.4	Let's analyze MSX-MUSIC	135
5.1.5	Try making rhythm sounds using FM sound source	137

5.2	Control FM sound source	139
5.2.1	Making sounds with a machine language program	139
5.2.2	Give an overview of the library	141
5.2.3	Let's compile with MSX-C	149
5.3	FM sound source data structure	150
5.3.1	Let's create FM sound data	150
5.3.2	To specify percussion data	152
5.3.3	Let's specify the instrument sound data	154
5.3.4	What the OPLL driver cannot do	156
5.3.5	Adding Sound Data	156
5.3.6	Explaining sample data	158
5.4	Various things related to FM synthesis	160
5.4.1	Correction to the content of the powerful usage method	160
5.4.2	List of MSX-MUSIC tone data	162
A	R800 instruction table	165
A.1	How to use the instruction sheet	166
A.2	8-bit movement instructions	168
A.3	16-bit move instruction	169
A.4	Exchange instruction	171
A.5	stack manipulation instructions	171
A.6	Block transfer instruction	172
A.7	block search instruction	172
A.8	multiplication instruction	172
A.9	addition instruction	173
A.10	subtraction instruction	175
A.11	comparison command	176
A.12	logical operation instructions	177
A.13	bit manipulation instructions	178
A.14	Rotate instruction	179
A.15	shift instruction	181
A.16	branch instruction	182
A.17	call instruction	183
A.18	input/output instructions	185
A.19	CPU control instructions	186

Figure table of contents

1.1	MSX turbo R system configuration	17
1.2	Changes in ROM configuration in MSX turbo R	19
1.3	R800 internal block diagram	25
1.4	Differences in memory access methods between Z80 and R800	26
2.1	Z80 CPU memory	49
2.2	MSX slot configuration (part 1)	51
2.3	MSX slot configuration (Part 2)	52
2.4	Example of MSX2+ slot configuration (when expanding only slot 3)	54
2.5	Example of MSX2+ slot configuration (when expanding slots 0 and 3)	55
2.6	How to specify slot number	58
2.7	device enable	65
2.8	MSX turbo R slot configuration	68
3.1	Kanji driver operating principle	75
3.2	Switching screen mode	78
4.1	Horizontal scrolling (when SP2=0)	85
4.2	Horizontal scrolling (when SP2=1)	86
4.3	List of control register functions added to V9958	96
4.4	Two types of side-scrolling mechanics	97
4.5	Data structure of RGB screen	99
4.6	Data structure of YJK style screen	101
4.7	Data structure of mixed screen	101
4.8	Scanning lines on a television screen	110
4.9	This is what happens in interlaced mode	112
4.10	This is the principle of scan line interruption	114

4.11	Scan line interrupt procedure	115
4.12	VDP register that generates a scan line interrupt	116
4.13	VDP register to detect scan line interrupt	117
4.14	VDP registers that control screen switching	117
4.15	Hardware vertical scrolling mechanism	118
5.1	Exploring the Structure of Four Types of Electronic Musical Instruments	131
5.2	Let's analyze the basic sound	132
5.3	Instrument and Synth Envelopes	134
5.4	Percussion instrument sound data	153
5.5	tone data	157
5.6	List of OPLL registers	161

table of contents

1.1	MSX turbo R I/O map	21
1.2	Comparing the operating speed of the Z80 and R800	24
1.3	List of BIOS and BASIC changes in MSX turbo R	32
1.4	I/O port for PCM	45
2.1	System work area for slots	61
2.2	MSX2+ I/O ports	64
3.1	MSX-JE built-in hardware list	73
3.2	Kanji BASIC screen mode	77
3.3	Hooks used by Kanji drivers	79
4.1	VDP mode and BASIC screen mode	82
4.2	Mode register	83
4.3	command register	84
4.4	status register	84
4.5	Change of terminal of V9958	90
4.6	DC characteristics of V9958	90
4.7	MSX2+ screen mode	91
4.8	VDP I/O Ports	92
4.9	Control register storage location	93
4.10	Other useful system work areas	94
4.11	System work area added and modified in MSX2+	94
4.12	Details of OFAFCH Address (MODE)	94
4.13	Logical operation	104
5.1	Comparing the performance of electronic musical instruments	131
5.2	Relationship between scale and frequency	134

5.3	List of temperaments that can be set in MSX-Music	135
5.4	Data structure for 6 instruments + 1 percussion sound	150
5.5	Example of data structure for 6 instruments + 1 percussion sound	151
5.6	9 Instrument sound data structure	152
5.7	Instrument sound data	155
5.8	Example of musical instrument sound data	155
5.9	Tone data list	163

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!

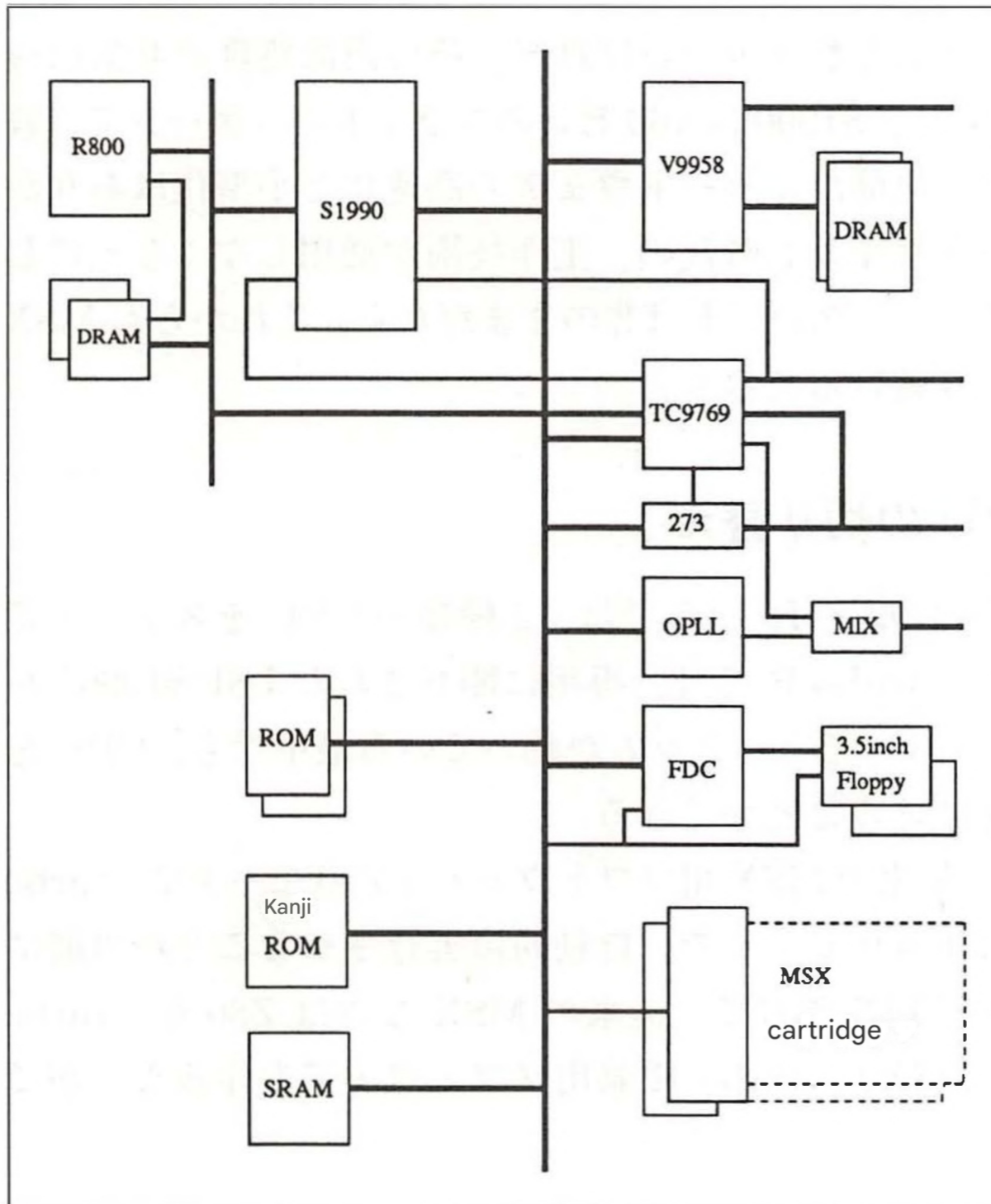
- **Z80** In addition, by incorporating the high-speed CPU "R800" which is upwardly compatible, it achieves an average speed of 4 to 5 times, and up to 10 times, faster (compared to MSX2+).
- Along with MSX-DOS1, it also comes with Japanese MSX-DOS2 and Kanji drivers, and supports MS-DOS compatible hierarchical 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

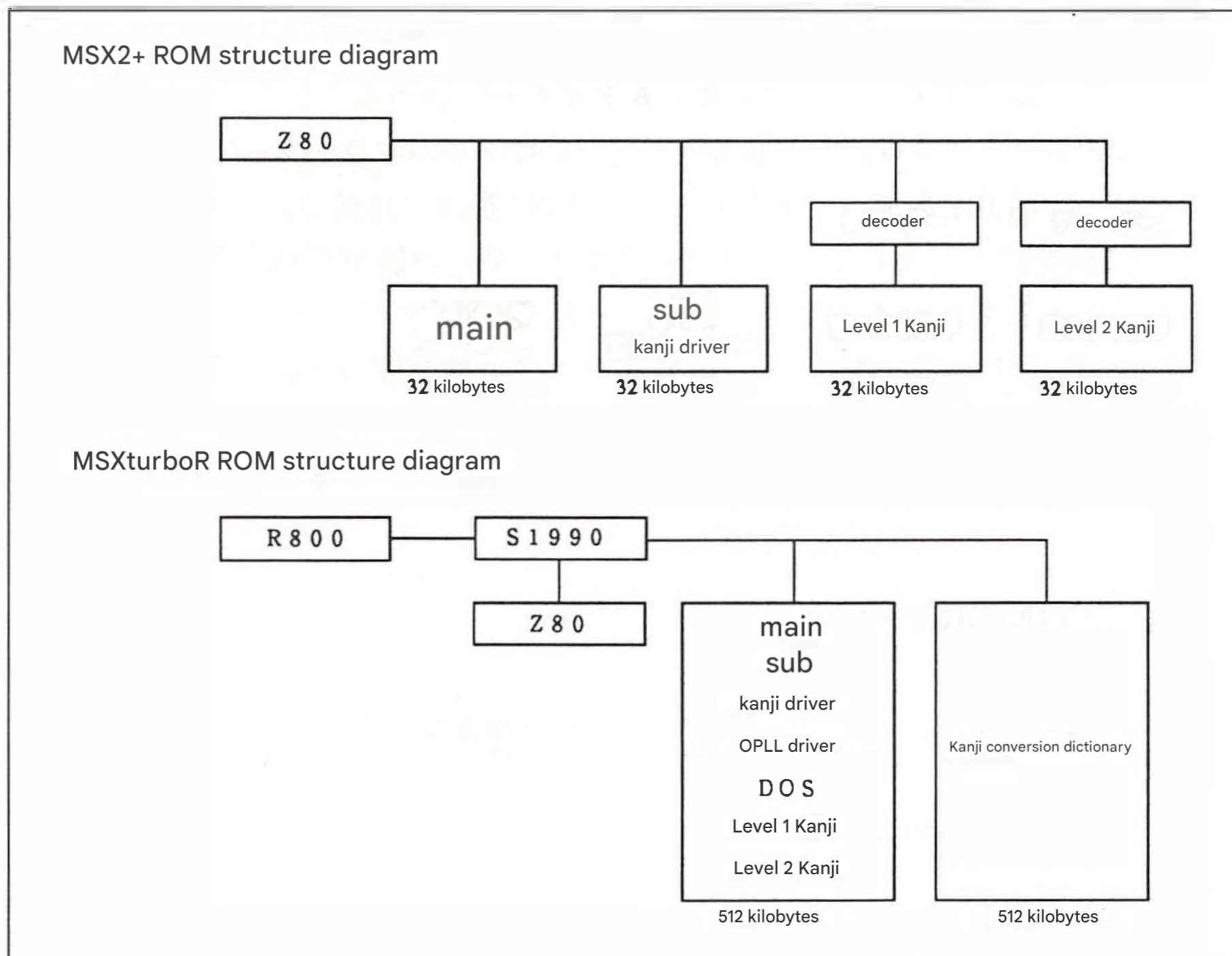
The turbo R should have a lot of ROMs built in, but when you open it up, you'll find that the number of ROMs is surprisingly small. The reason for this is the mega ROM control function of the S1990.

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** MSX₁ Compatible
- 2** MSX₂ Compatible
- +** MSX₂₊ 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 μ s)	turbo R (unit μ s)	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	—

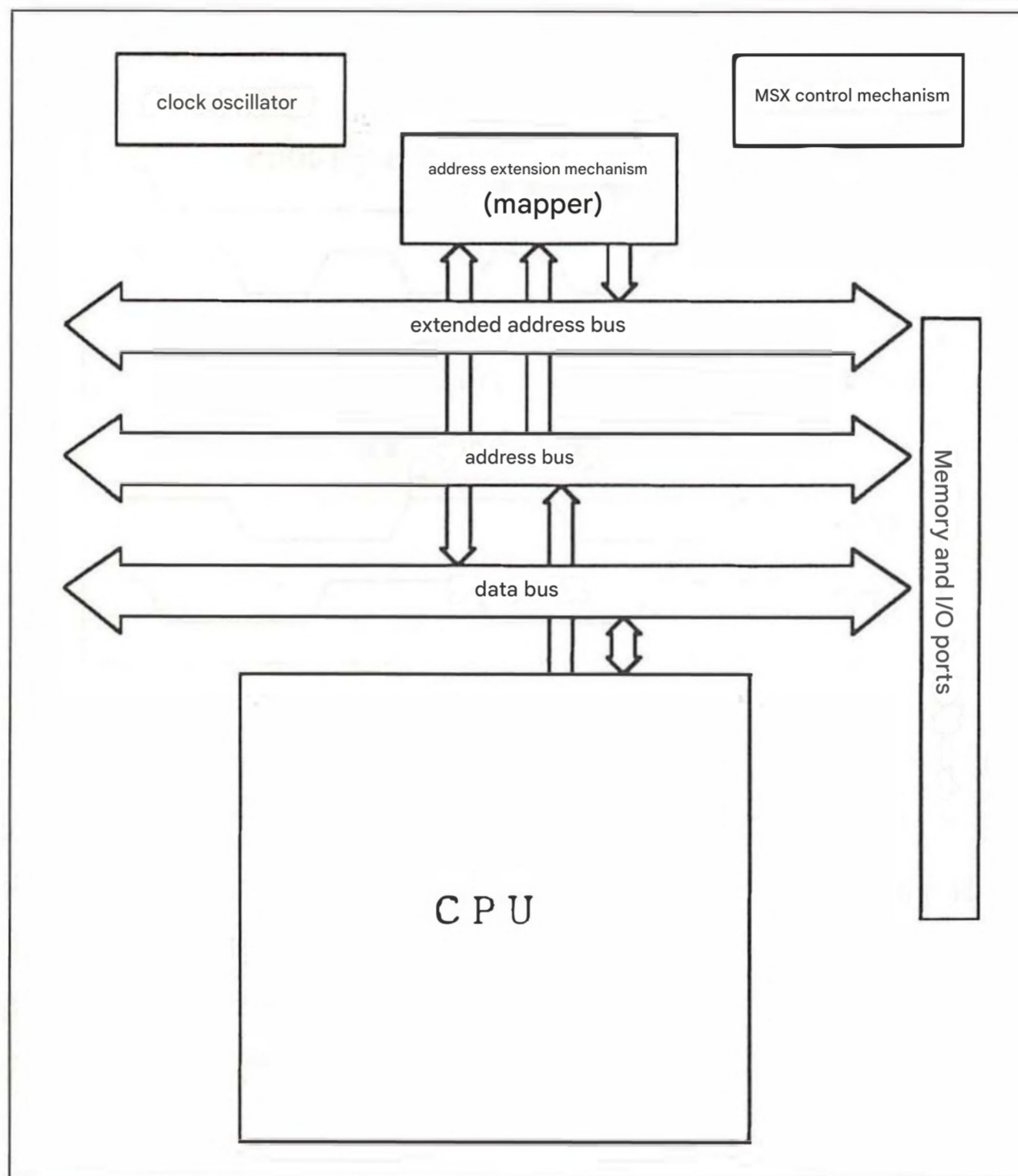
Table 1.2 shows the results of comparing the speed of the Z80 and the R800 for each type of instruction. It is worth noting that data transfer between registers (LD instruction) and addition are 10 times faster. However, the values in this table are measurements of the speed when the R800 operates with 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

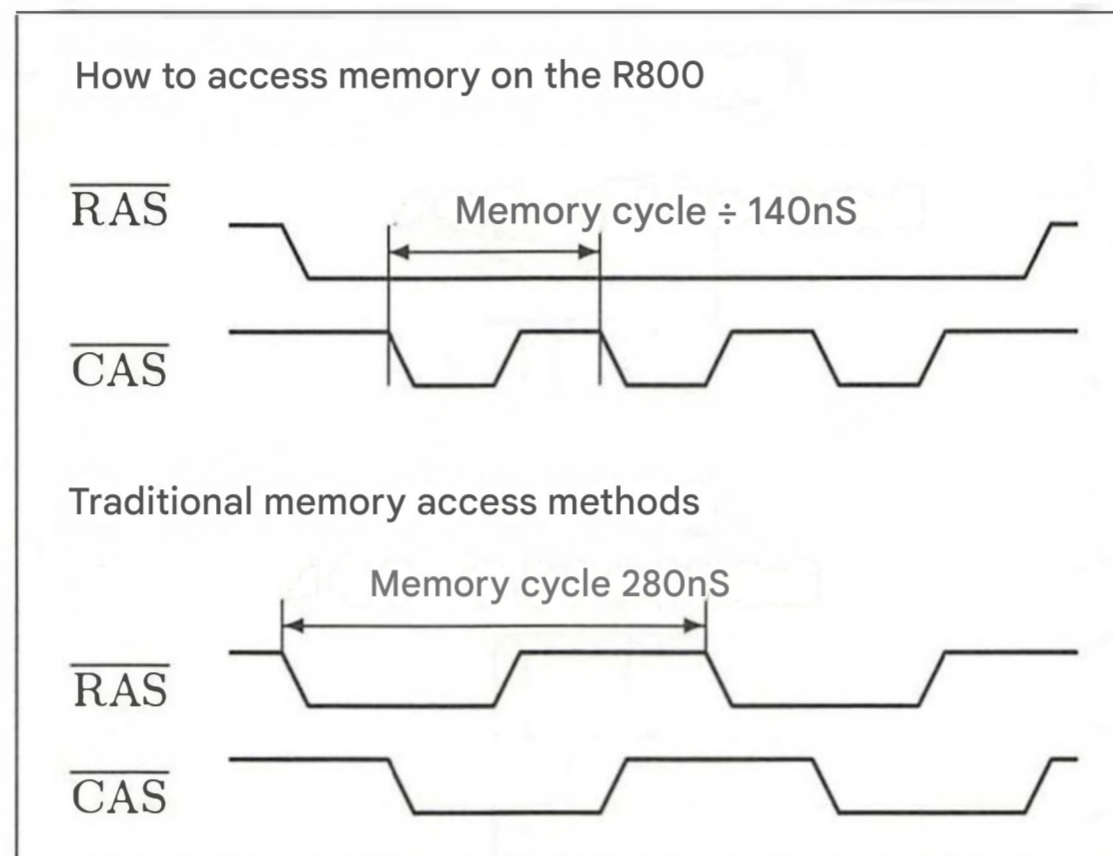


Next, let's explain "DRAM page access" in detail. First, the bottom part of Figure 1.4 shows the previous method of memory access using the Z80. The upper byte of the address (row address) is sent to the DRAM, the RAS (row address strobe) signal is set to LOW, the lower byte of the address (column address)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

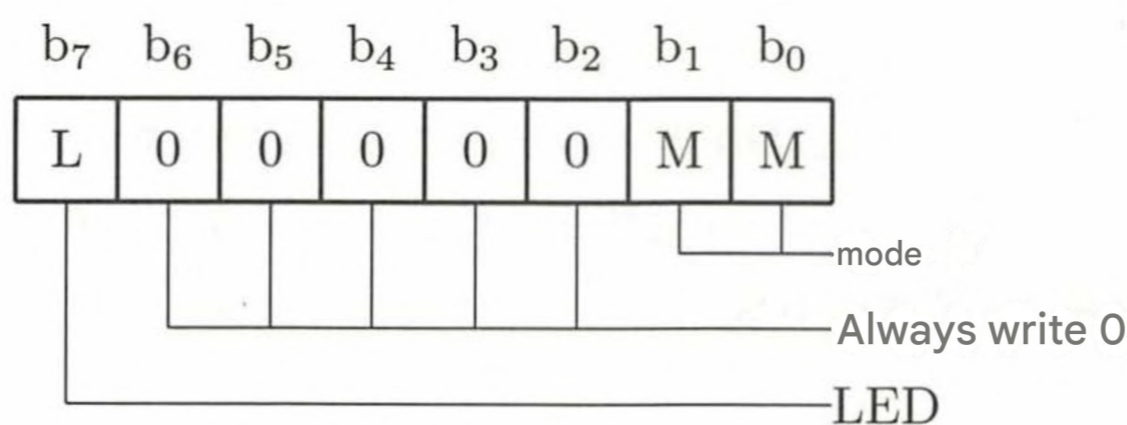
To control the turbo R's new hardware features, a BIOS was added for CPU switching and PCM recording and playback.

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 street 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.

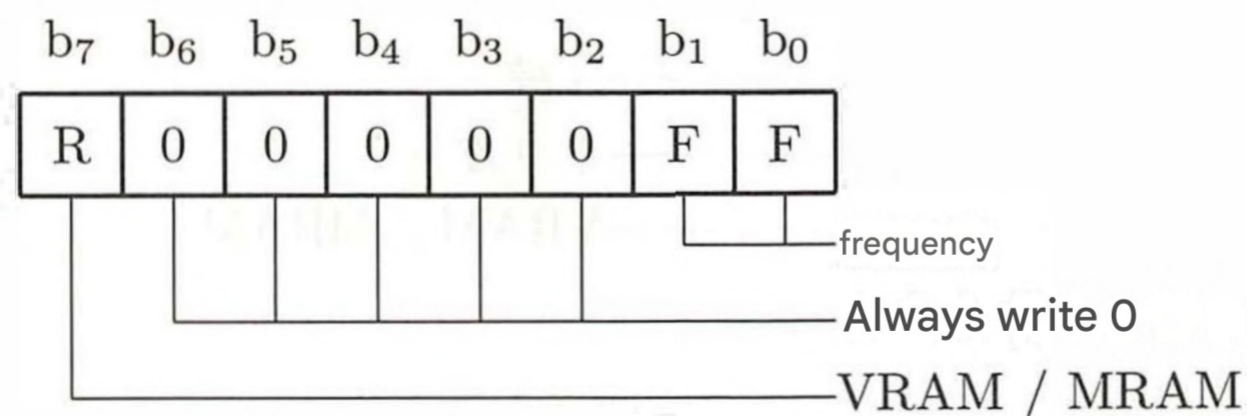
PCMPPLY 0186H street address

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

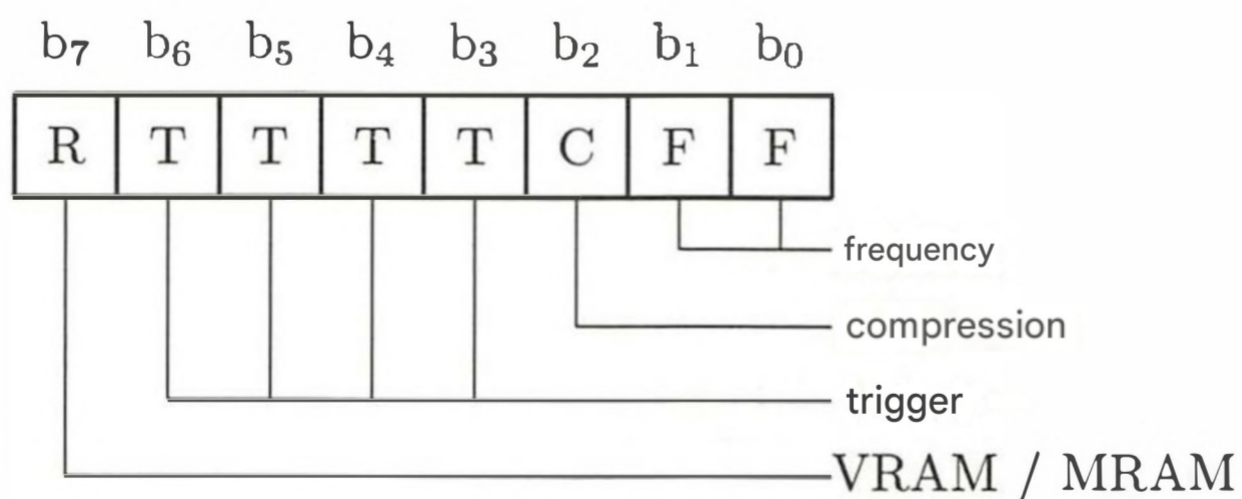
PCMREC 0189H street address

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 PCMPY. 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
PCMPY	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 PCMPY
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
RDSLTL EQU 0000CH ; inter slot read
CALSLTL EQU 0001CH ; inter slot call
EXPTBL EQU 0FCC1H ; slot # of main ROM
;
    ld    a,(EXPTBL) ;
    ld    hl,2dh      ; address to read
    call  RDSLTL     ; 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  CALSLTL   ; 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 PCMPLAY(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  OEDH,70H          ; IN (HL), (C)
                JP    M,RECADO
                AND   01111111B
RECADO:
                OR    01000000B

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

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

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

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

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

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

                OUT   (PMDAC),A
                DEFB  OEDH,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
SYSTM L EQU 0E6H           ; system timier port

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

PLAY1:
IN   A,(SYSTM L)
CP   E
JR   C,PLAY1             ; wait
XOR  A
OUT  (SYSTM L),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.