NEC
ELECTRON DEVICE

MOS INTEGRATED CIRCUIT
μPD17052

ON-CHIP IMAGE DISPLAY CONTROLLER
4-BIT SINGLE-CHIP MICROCONTROLLER FOR VOLTAGE SYNTHESIZER

μPD17052 includes the image display controller (IDC) which is capable of various displays and the 4-bit single-chip microcontroller for the voltage synthesizer in which the 14-bit D/A converter for voltage synthesizer is incorporated.

The CPU has such functions as 4-bit parallel addition, logical operation, multiple-bit testing, carry-flag set/reset, powerful interrupt and timer functions.

Incorporated with a user programmable image display controller for on-screen display, the CPU can control various displays with a simple program.

The appearance, which is 64-pin plastic sealing DIP, has the various I/O ports and serial interface functions controlled by powerful input/output commands as well as the 4-bit A/D converter and 6-bit D/A converter (PWM) outputs.

FEATURE

- 4-bit single-chip microcontroller for digital tuning system
- 14-bit D/A converter for voltage synthesizer
- Program memory (ROM): 8192 words x 16 bits
- Data memory (RAM): 448 words x 4 bits
- Stack level: 6
- Instruction set comprising 35 easy-to-understand
- Decimal operation capability instruction
- Instruction execution time: 2 μs (when 8 MHz oscillator is connected)
- On-chip IDC (user programmable)
  - Number of display characters : Up to 99 characters per screen
  - Display location : 14 lines x 19 columns
  - Character type : 128 (A single screen can contain up to 64 simultaneously.)
  - Character format : 10 x 15 dots (burster trimming possible)
  - Color : 8
  - Character size : Four types can be set vertically or horizontally independent of each other. (15, 30, 45, 60 H) (2.5, 5.0, 7.5, 10 μs)
- On-chip 8-bit serial interface (1 system: 3-wire or 2-wire)
- On-chip 6-bit D/A converter: 4 outputs (PWM)
- On-chip 4-bit A/D converter: 8 inputs
- On-chip horizontal synchronous signal counter
- On-chip commercial power frequency counter
- On-chip power failure detection and power on reset circuits
- Interrupt input for remote control signals (Noise canceller attached)
- Various I/O ports
  - Input/output ports : 20
  - Input ports : 4
  - Output ports : 20
- 5 V ±10 %
- CMOS low power consumption

ORDERING INFORMATION

Order Code	Package	Quality Grade
μPD17052CW-xxx	64-pin plastic shrink DIP (750 mil)	Standard

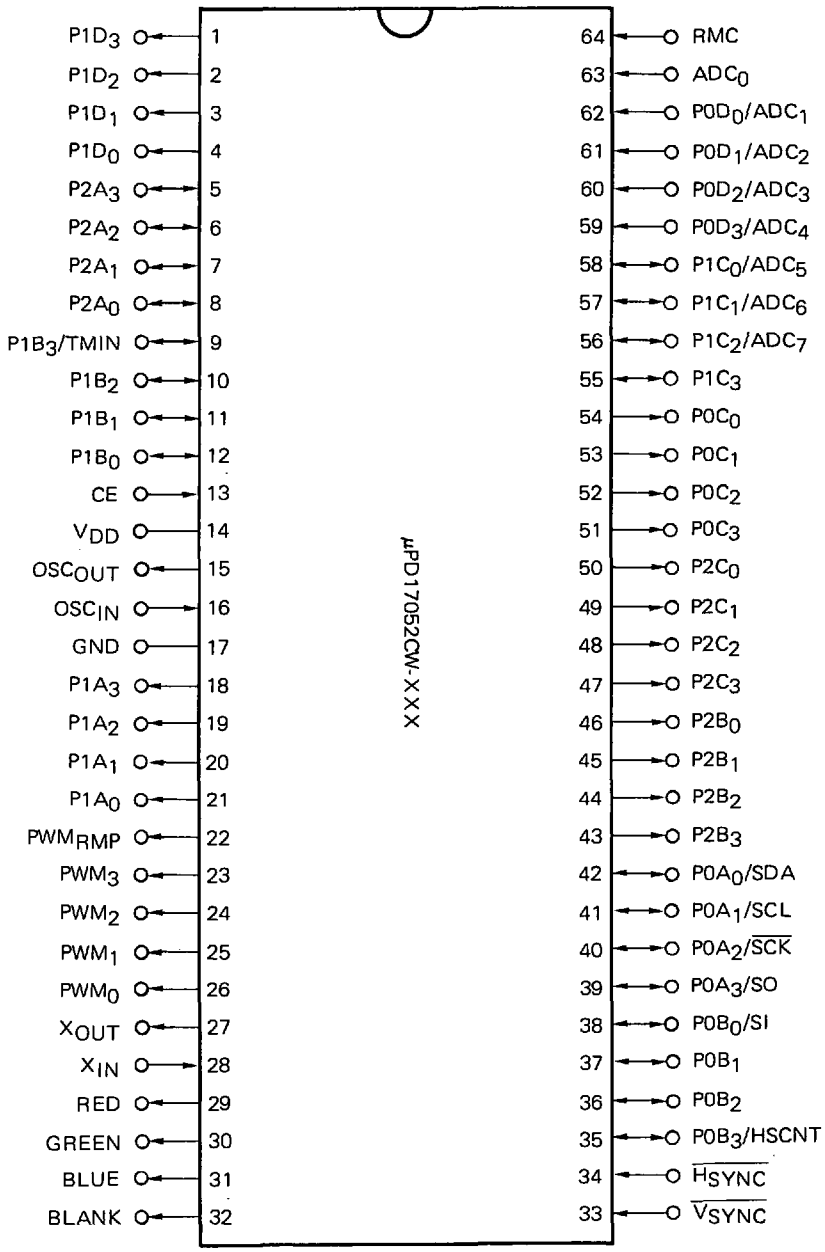
The information in this document is subject to change without notice.

Document No. IC-2423
(O.D.No. IC-7888)
Date Published October 1990 P
Printed in Japan

FUNCTION OUTLINE

Item	Function
Program memory	• 8192 words x 16 bits Table reference area: : 256 words x 16 bits CROM shared area : 2048 words x 16 bits
Data memory	• 448 words x 4 bits Data buffer : 4 words x 4 bits General-purpose register : 16 words x 4 bits VRAM shared area : 224 words x 4 bits
System register	• 12 words x 4 bits
Register file	• 25 words x 4 bits (control register)
Port register	• 11 words x 4 bits
Instruction execution time	• 2 μs (8 MHz ceramic oscillator used)
Stack level	• 6 levels (stack manipulation possible)
General-purpose port	• 20 I/O ports • 4 input ports • 20 output ports
IDC (Image Display Controller)	• Number of display characters : Up to 99 characters per screen • Display position : 14 lines x 19 digits • Character type : 128 types (User programmable) (However, a signal screen can contain up to 64 types simultaneously.) • Character format : 10 x 15 dots • Color : 8 colors • Character size : Vertically 4 sizes (15, 30, 45, 60 H) Horizontally 4 sizes (2.5, 5.0, 7.5, 10 μs) Can be set vertically or horizontally independently of each other.
Serial interface	• 1 system (2 channels) 8-bit 3-wire : 1 channel 8-bit 2-wire : 1 channel
D/A converter	• 14 bits x 1 converter (PWM output, withstand voltage 12.5 V max.) • 6 bits x 4 converters (PWM output, withstand voltage 12.5 V max.)
A/D converter	• 4 bits x 8 converters (non-periodical comparison method based on the software)
Interrupt	• 4 channel (maskable interrupt) External interrupt : 2 channels (RMC pin, $\overline{V_{SYNC}}$ pin) Internal interrupt : 2 channels (timer, serial interface)
Timer	• 2 systems Internal timer : 5, 20, 100 ms External timer : 1/5 and 1/6 of the frequency input in the $P1B_3$/TMIN pin
Reset	• Power ON reset (When turning on the power) • Reset by CE pin (CE pin: Low level → high level) • Power failure detection function
Power supply voltage	5 V ± 10 %
Package	64-pin plastic shrink DIP (750 mil)

PIN CONFIGURATION (Top View)



ADC_0 to ADC_7	:	A/D converter input
CE	:	Chip enable
RMC	:	Interrupt signal input
X_{IN}, X_{OUT}	:	Clock oscillation
OSC_{IN}, OSC_{OUT}	:	LC oscillation
TMIN	:	External timer input
PWM_0 to PWM_3	:	D/A converter output
PWM_{RMP}	:	Channel select D/A converter output
RED	:	Character signal output
GREEN	:	Character signal output
BLUE	:	Character signal output
BLANK	:	Blanking signal output
$\overline{H_{SYNC}}$	:	Horizontal synchronous signal input
$\overline{V_{SYNC}}$	:	Vertical synchronous signal input
HSCNT	:	Horizontal synchronous signal counter input
V_{DD}	:	Power input
GND	:	Ground
SI	:	Data input
SO	:	Data output
$\overline{SCK}$	:	Shift clock input/output
SCL	:	Shift clock input/output
SDA	:	Data input/output
$P0A_0$ to $P0A_3$	:	Port 0A
$P0B_0$ to $P0B_3$	:	Port 0B
$P0C_0$ to $P0C_3$	:	Port 0C
$P0D_0$ to $P0D_3$	:	Port 0D
$P1A_0$ to $P1A_3$	:	Port 1A
$P1B_0$ to $P1B_3$	:	Port 1B
$P1C_0$ to $P1C_3$	:	Port 1C
$P1D_0$ to $P1D_3$	:	Port 1D
$P2A_0$ to $P2A_3$	:	Port 2A
$P2B_0$ to $P2B_3$	:	Port 2B
$P2C_0$ to $P2C_3$	:	Port 2C

CONTENTS

1. PIN FUNCTION

1.1 DESCRIPTION OF PIN FUNCTIONS

PIN NO.	SYMBOL	PIN NAME	DESCRIPTION	OUTPUT TYPE
1 to 4	$P1D_3$ to $P1D_0$	Port 1D	These are 4-bit output ports. Port 1D's latch is located in address 73H of BANK1 of the data memory (RAM). The output state at the time of power-on reset is undefined.	CMOS push-pull
5 to 8	$P2A_3$ to $P2A_0$	Port 2A	These are 4-bit I/O ports. It is possible to specify input/output for these ports bit-wise. Input/output is set with the P2ABIO words (34H) on the register file. The latch of this port is located in address 70H of BANK2 of the data memory.	CMOS push-pull
9 10 11 12	$P1B_3$/TMIN $P1B_2$ $P1B_1$ $P1B_0$	Port 1B	These are 4-bit I/O ports. It is possible to specify input/output for these ports bit-wise. Input/output is set with the P1BBIO words (35H) on the register file. The latch of this port is located in address 71H of BANK1 of the data memory. $P1B_3$/TMIN can be used as an external timer input as well. It is possible to interrupt the frequency that is input to this pin with 1/5 or 1/6 of the frequency. Normally, the frequency of the commercial power is input to this pin to be used as a reference clock.	CMOS push-pull (I/O)
13	CE	Chip enable	This is the device selection signal input pin. The device is set to the high level for a normal operation and to the low level when not using it. If the STOP instruction is executed when this pin is at the low level, the clock oscillation is stopped thus making the backup at a low power consumption possible. The STOP instruction is valid only when the CE pin is at the low level. When it is at the high level, it functions in the same manner as the NOP instruction. This pin plays the role of a reset pin as well. Therefore, if the CE pin is changed from the low level to high, the device is reset, thus restarting the program from address 0. If the device is reset, the bank becomes 0, thus placing the I/O port in the input mode. However, the low level from 188 μs and below is not accepted.	Input
14	V_{DD}	Power input	This is the device power pin. A voltage of 5 V ± 10 % is supplied to make the device operate all the functions. A voltage between 4 to 5.5 V is supplied when IDC is not used. It is possible to lower the voltage to about 2.2 V when holding the RAM data (when the clock oscillation is stopped). As μPD17052 is incorporated with the power-on reset circuit, if this is changed from 0 to 4.0 V, the system is reset thus making the program operate from address 0. Restrict the start-up time for changing from 0 to 4.0 V within 500 ms to make the power-on reset circuit operate normally.	–

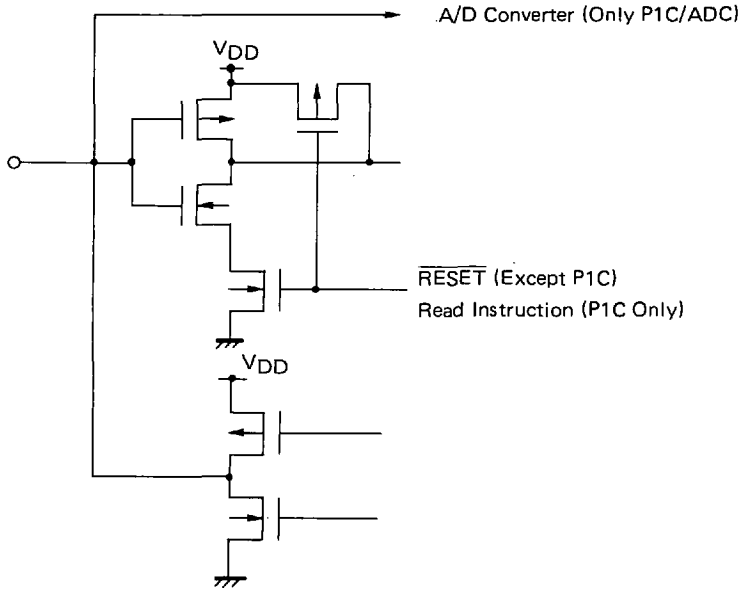
PIN NO.	SYMBOL	PIN NAME	DESCRIPTION	OUTPUT TYPE
15 16	OSC_{OUT} OSC_{IN}	LC oscillation	These are the LC oscillation circuit pins for IDC. They are oscillated at 4 MHz.	
17	GND	Ground	This is the ground pin of the device.	—
18 to 21	$P1A_3$ to $P1A_0$	Port 1A	These are 4-bit output ports. The latch of this port is located in address 70H of BANK1 of the data memory. The format is N-ch open-drain (middle-voltage, high current).	N-ch open-drain
22	PWM_{RMP}	Channel select D/A converter output	This is the 14-bit D/A converter output or 1-bit output port for voltage synthesizer. The D/A converter outputs the pulse which has combined the 9-bit PWM and the 5-bit RMP (Rate Multiplier). Therefore, D/A conversion can be carried out by externally connecting a simple CR filter. The low level is output for power-on reset or clock stop.	N-ch open-drain
23 to 26	PWM_3 to PWM_0	D/A converter	These are the VDP (Variable Duty Port) or 1-bit output ports. The VDP function is to output the 15.625 kHz pulse continuously and is capable of varying the dury of this pulse in 64-step programs.	N-ch open-drain
27 28	X_{OUT} X_{IN}	Clock oscillation	These are connection pins of ceramic oscillators or crystal resonators. Ensure to use 8 MHz.	CMOS push-pull (X_{OUT}), input (X_{IN})
29 30 31	RED GREEN BLUE	Character signal output	These are the output pins of the character data corresponding to R, G and B. Output is made at active high.	CMOS push-pull
32	BLANK	Blanking signal output	This is the output pin of the blanking signal to cut image signals. Output is made at active high.	CMOS push-pull
33	$\overline{V_{SYNC}}$	Vertical synchronous signal input	This is the input pin of the vertical synchronous signal for IDC. Ensure to make the input at active low. Interrupt can be applied with this signal.	Input
34	$\overline{H_{SYNC}}$	Horizontal synchronous signal input	This is the input pin of the horizontal synchronous signal for IDC. Ensure to make the input at active low.	Input
35 36 37 38	$P0B_3$/HSCNT $P0B_2$ $P0B_1$ $P0B_0$/SI	Port 0B	These are 4-bit I/O ports. It is possible to specify input/output bit-wise for these ports. This set-ting is made with the P0BBIO words (36H) on the register file. The latch of this port is located in address 71H of BANK0 of the data memory. Pin $P0B_0$/SI can be used as the data input pin of the serial interface (serial I/O mode) as well. Pin $P0B_3$/HSCNT can be used as the input pin of the horizontal synchronous signal counter as well. At this time, the own bias ($V_{DD}/2$) is applied to the HSCNT pin. Port 0B is ready for input in power-on reset, clock stop and CE reset.	CMOS push-pull (I/O) However, the own bias is applied to HSCNT to make the input.

PIN NO.	SYMBOL	PIN NAME	DESCRIPTION	OUTPUT TYPE
39 40 41 42	$P0A_3$/SO $P0A_2/\overline{SCK}$ $P0A_1$/SCL $P0A_0$/SDA	Port 0A	These are 4-bit I/O ports. It is possible to specify input/output bit-wise for these ports. This set-ting is made with the P0ABIO words (37H) on the register file. The latch of this port is located in address 70H of BANK0 of the data memory. Pin $P0A_3$/SO can be used as the data output pin of the serial interface (serial I/O mode) and $P0A_2/\overline{SCK}$ can be used as the shift clock I/O pin. Pin $P0A_0$/SDA can be used as the data I/O pin of the serial interface (two-wire mode and serial I/O mode) and pin $P0A_1$/SCL can be used as the shift clock I/O pin.	$P0A_3$/SO $P0A_2/\overline{SCK}$ CMOS push-pull (I/O) $P0A_1$/SCL $P0A_0$/SDA N-ch open drain (I/O)
43 to 46	$P2B_3$ to $P2B_0$	Port 2B	These are 4-bit output ports. The latch of this port is located in address 71H of BANK2 of the data memory. The format is N-ch open-drain (middle-voltage).	N-ch open-drain
47 to 50	$P2C_3$ to $P2C_0$	Port 2C	Thise are 4-bit output ports. The latch of this port is located in address 72H of BANK2 of the data memory. The format is N-ch open-drain (middle-voltage).	N-ch open-drain
51 to 54	$P0C_3$ to $P0C_0$	Port 0C	These are 4-bit output ports. The latch of this port is located in address 72H of BANK0 of the data memory. The output state is undefined in power-on reset.	CMOS push-pull
55 56 57 58	$P1C_3$ $P1C_2/ADC_7$ $P1C_1/ADC_6$ $P1C_0/ADC_5$	Port 1C	These are 4-bit I/O ports or A/D converter input pins. Input/output setting is carried every 4 bits, specifying with the P1CGIO bits (27H's #0 bit) on the register file. Ensure to specify when used as the A/D converter. The latch of this port is located in address 72H of BANK1 of the data memory. Port 1C is ready for input in power-on reset, clock stop and CE reset.	CMOS push-pull (I/O)
59 60 61 62	$P0D_3/ADC_4$ $P0D_2/ADC_3$ $P0D_1/ADC_2$ $P0D_0/ADC_1$	Port 0D	These are 4-bit input ports or A/D converter input pins. The pull-down resistor is installed (100 kΩ TYP.) when these are used as ports. The latch of Port 0D is located in address 73H of BANK0 of the data memory.	Input (Equipped with pull-down resistor)
63	ADC_0	A/D converter input	This is the A/D converter input pin. It is incorporated with the 4-bit A/D converter of the program-based random comparison method. The reference voltage of the A/D converter is V_{DD}.	Input
64	RMC	Interrupt signal input	This is the interrupt input pin equipped with a noise canceller. If the signal is noisy such as the remote control signal, the program is made easier by using this pin. It is possible to specify in the program whether to apply an interrupt at the rise or the fall of the input signal to this pin. The interrupt is applied at the rise if the IEG flag is reset, and at the fall if the IEG flag is set. If the CE is reset, the IEG flag is reset thus applying the interrupt at the rising edge.	Input

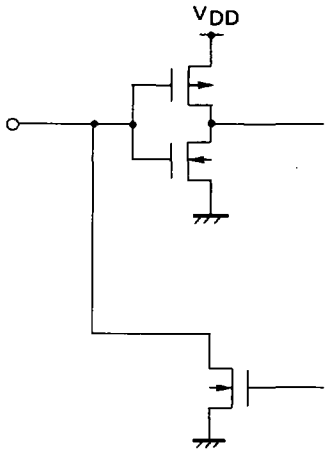
1.2 PIN'S EQUIVALENT CIRCUIT

P0A ($P0A_3$/SO, $P0A_2$/$\overline{SCK}$)
P0B ($P0B_2$, $P0B_1$, $P0B_0$/SI)
P1B ($P1B_2$, $P1B_1$, $P1B_0$)
P1C ($P1C_3$, $P1C_2$/ADC_7, $P1C_1$/ADC_6, $P1C_0$/ADC_5)
P2A ($P2A_3$, $P2A_2$, $P2A_1$, $P2A_0$)

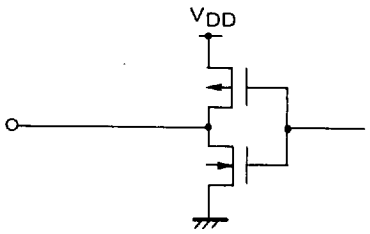
} (Input/output)



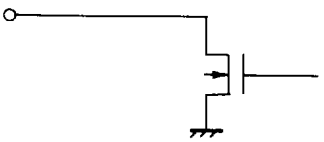
P0A ($P0A_1$/SCL, $P0A_0$/SDA): (Input/output)



P0C ($P0C_3$, $P0C_2$, $P0C_1$, $P0C_0$)
P1D ($P1D_3$, $P1D_2$, $P1D_1$, $P1D_0$)
RED, GREEN, BLUE, BLANK
} (Output)

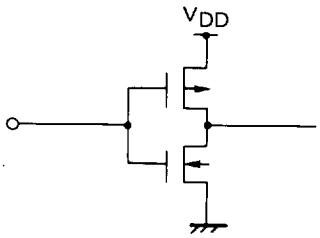


PWM (PWM_3, PWM_2, PWM_1, PWM_0, PWM_{RMP})
P1A ($P1A_3$, $P1A_2$, $P1A_1$, $P1A_0$)
P2B ($P2B_3$, $P2B_2$, $P2B_1$, $P2B_0$)
P2C ($P2C_3$, $P2C_2$, $P2C_1$, $P2C_0$)
} (Output)

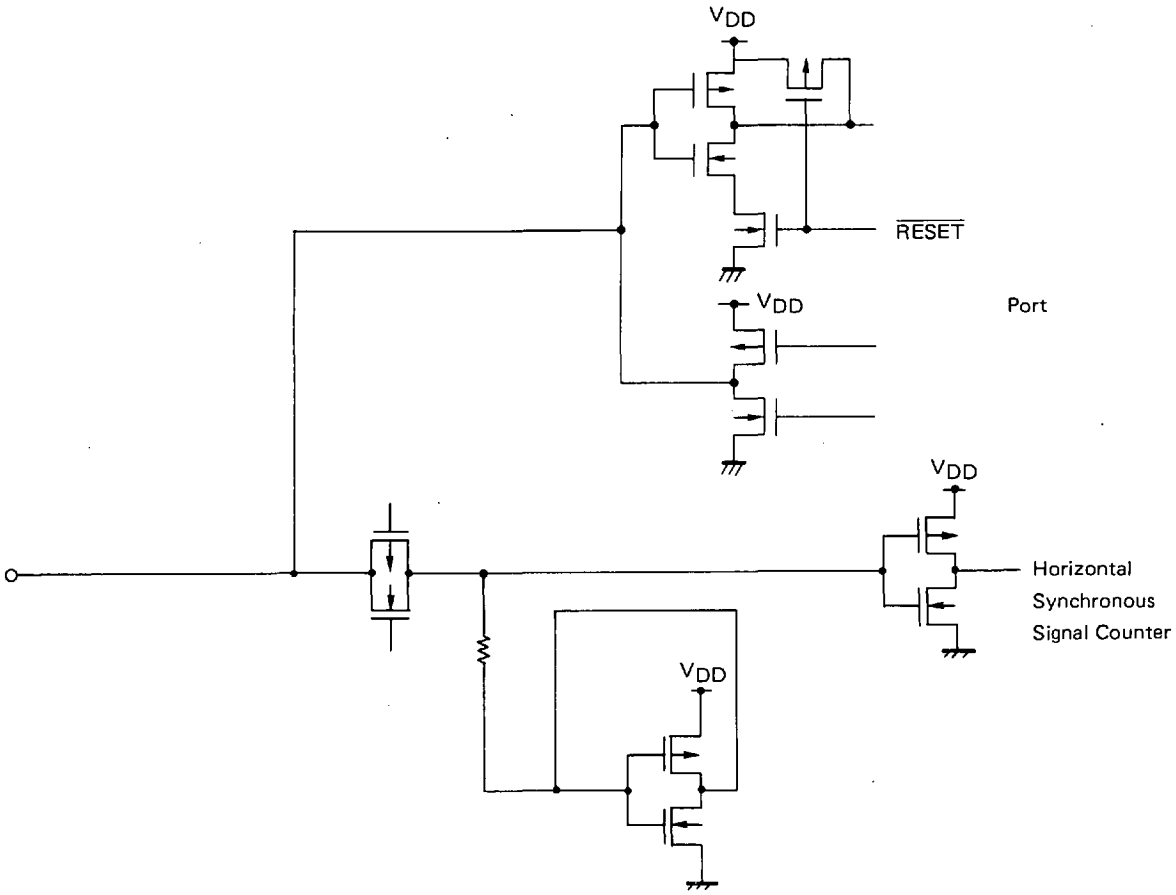


P0D ($P0D_3/ADC_4$, $P0D_2/ADC_3$, $P0D_1/ADC_2$, $P0D_0/ADC_1$): (Input)

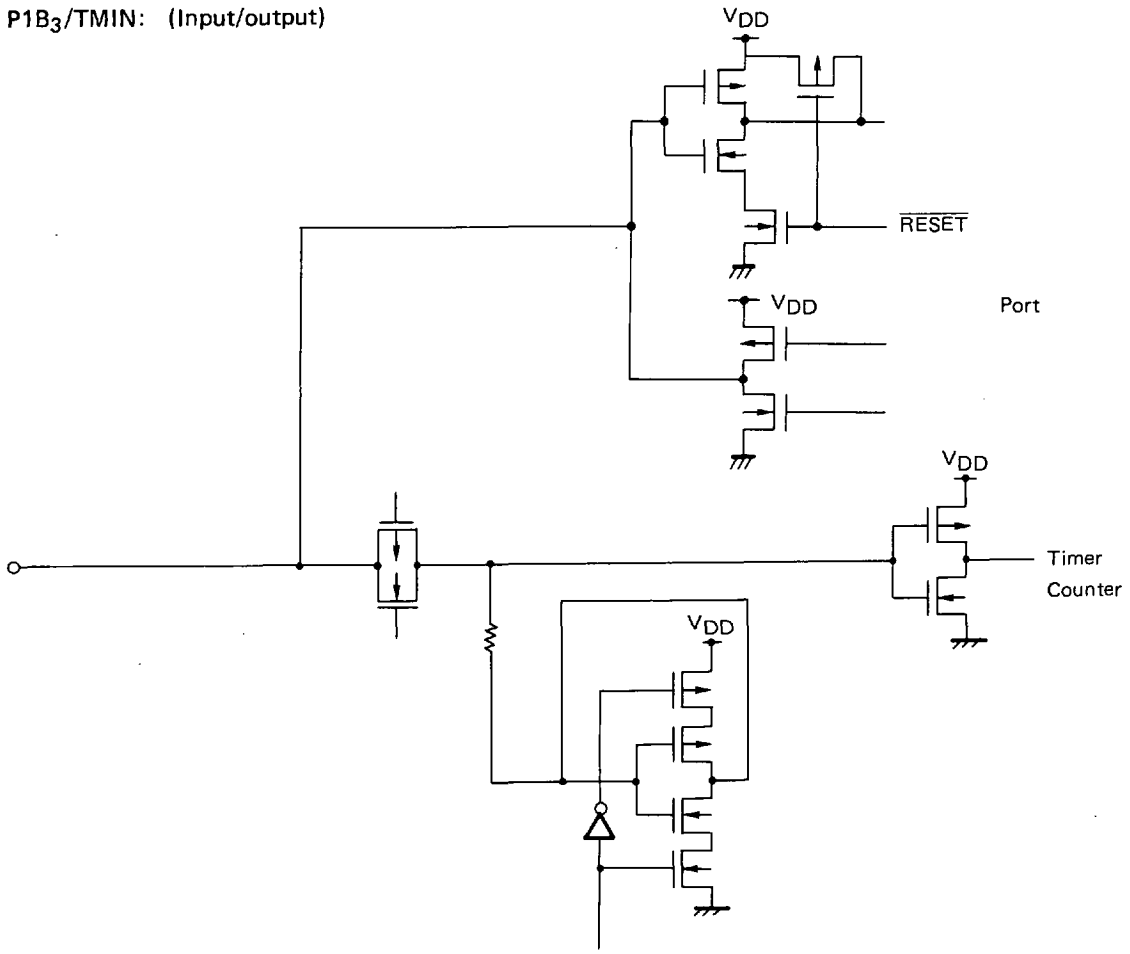
ADC_0: (Input)



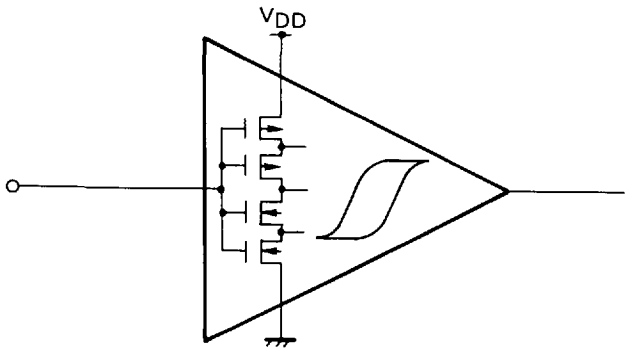
P0B$_3$/HSCNT: (Input/output)



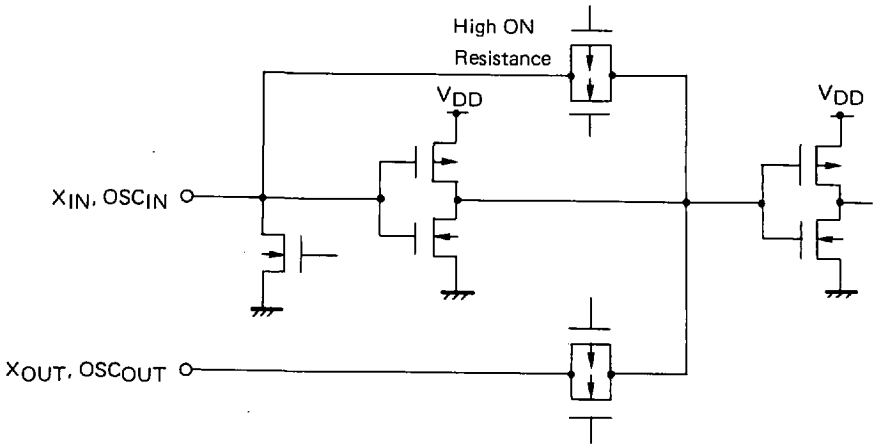
P1B$_3$/TMIN: (Input/output)



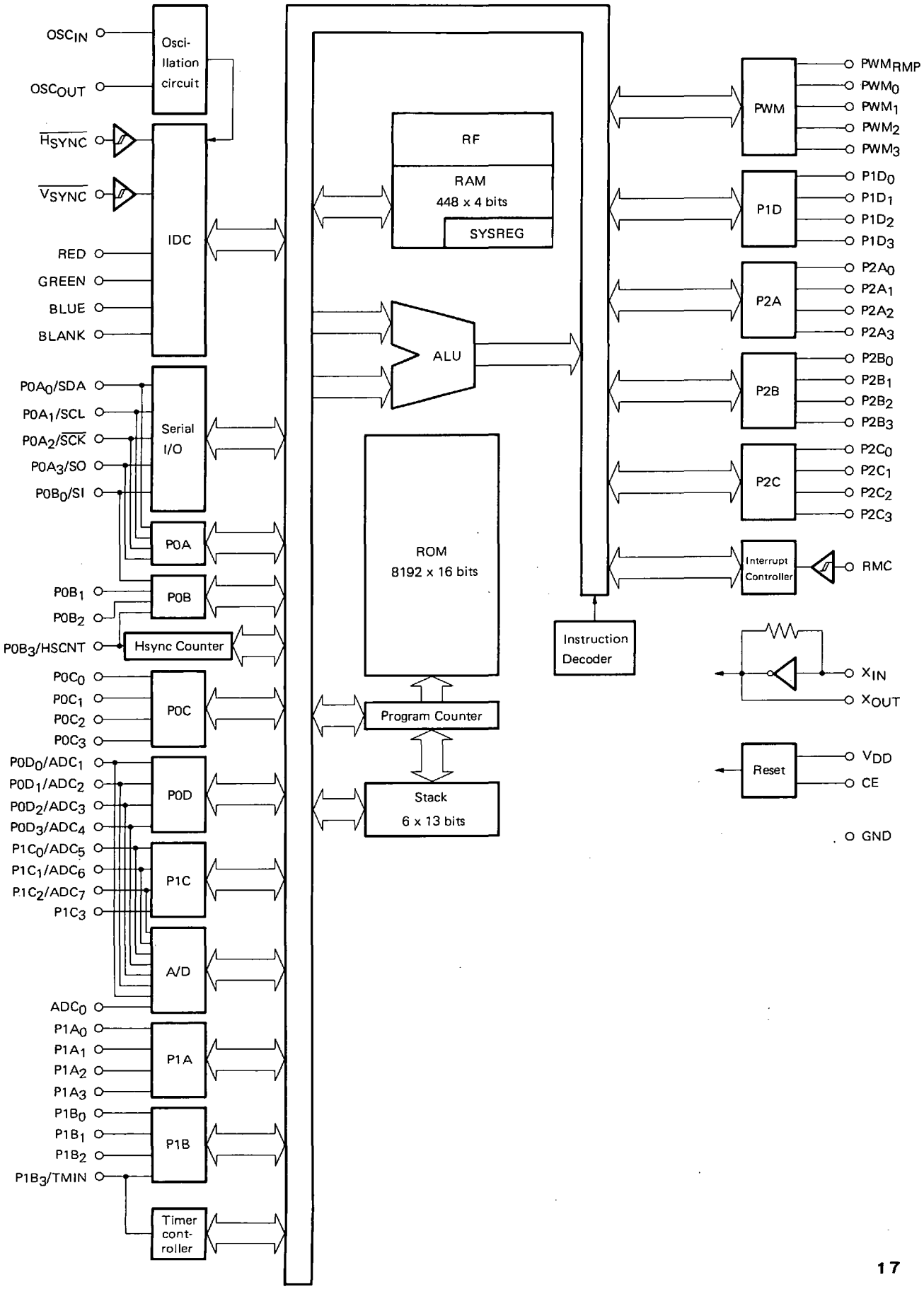
$\overline{H_{SYNC}}$, $\overline{V_{SYNC}}$, RMC, CE: (Schmitt trigger input)



X_{IN}, OSC_{IN} : (Input)
X_{OUT}, OSC_{OUT} : (Output)



2. BLOCK DIAGRAM



3. PROGRAM MEMORY (ROM)

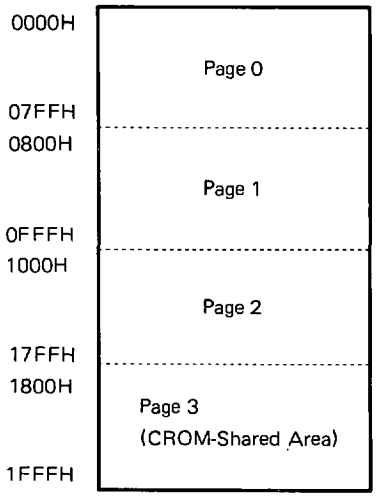
3.1 PROGRAM MEMORY CONFIGURATIONS

ROM is configured with 8192 words x 16 bits and is used to store programs. The usable ROM address range is 8192 words of addresses 0000H to 1FFFH. Every 2048 words are divided into 4 pages.

1800H to 1FFFH are also used as the CROM (Character ROM) area to store the display patterns for IDC. When not used as CROM, they can be used as the program area.

0000H to 00FFH are the area for table reference and used with BR, MOVT, PUSH and POP instructions.

Fig. 3-1 ROM Configurations



3.2 PROGRAM MEMORY FUNCTION

The program memory has the following two major functions.

(1) To store programs
(2) To store constant data

A program is a collection of "instructions" for operating the CPU. The CPU processes the work sequentially in accordance with the "instructions" written in the program.

In other words, the CPU reads "instructions" sequentially from a program stored in the program memory and executes the work in accordance with each "instruction".

As all instructions are in "one word" of 16 bit-length, it is possible to store one instruction in one address of the program memory.

Constant data are pre-determined data such as the patterns for display. By using the special-purpose MOVT instruction, the content of the program memory can be read into the data buffer (DBF) on the memory (RAM). As such, reading the constant data on the program memory is called "table reference".

As the program memory is the memory for reading only, it cannot be reloaded by the instruction. Therefore, the program memory and the ROM are used with the same meaning.

3.3 PROGRAM FLOW

Programs stored in the program memory are normally executed address by address starting from address 0000H.

However, when, for example, executing a different program for a certain condition, it is necessary to branch the program flow. In this case, the branch instruction is used.

If the same program needs to be executed repeatedly, using the same program each time the execution is performed will reduce the efficiency of the program memory. In this case, keep the program in a certain location. Then, the same program can be executed as many as desired by calling it with the dedicated CALL instruction. This program is called the "subroutine". In contrast, a normally executed program is called the "main routine".

When a program needs to be executed because a certain condition is satisfied regardless of the program flow, the interrupt function is used. With the interrupt function, is possible to branch to a decided address (called header address) regardless of the program flow if conditions are met.

Such a program flow as was described above is controlled by the program counter (PC) which specifies the program memory address.

3.4 BRANCH INSTRUCTION

Branch instructions (BR) include two types, i.e., the direct branch instruction (BR addr) which directly branches into the program memory address specified with the instruction operand (addr) and the indirect branch instruction (BR @AR) which branches to the program memory address specified with the address register (AR).

The direct branch instruction specifies the program memory address of the branch destination with 11 bits of the instruction operand. The addresses that can be specified with the 11 bits are one page portion of 0000H to 07FFH.

Therefore, the direct branch instruction is available for each page to be branched and each has a different operation code. However, when programming with the μPD17000 series assembler (AS17K), the BR instruction can be used without the page concept.

The indirect branch instruction specifies the address of the branch destination with the 8 bits of the address register. Therefore, the branch destination addresses of the indirect branch instruction are limited to addresses 0000H to 00FFH.

(1) Precautions in debugging

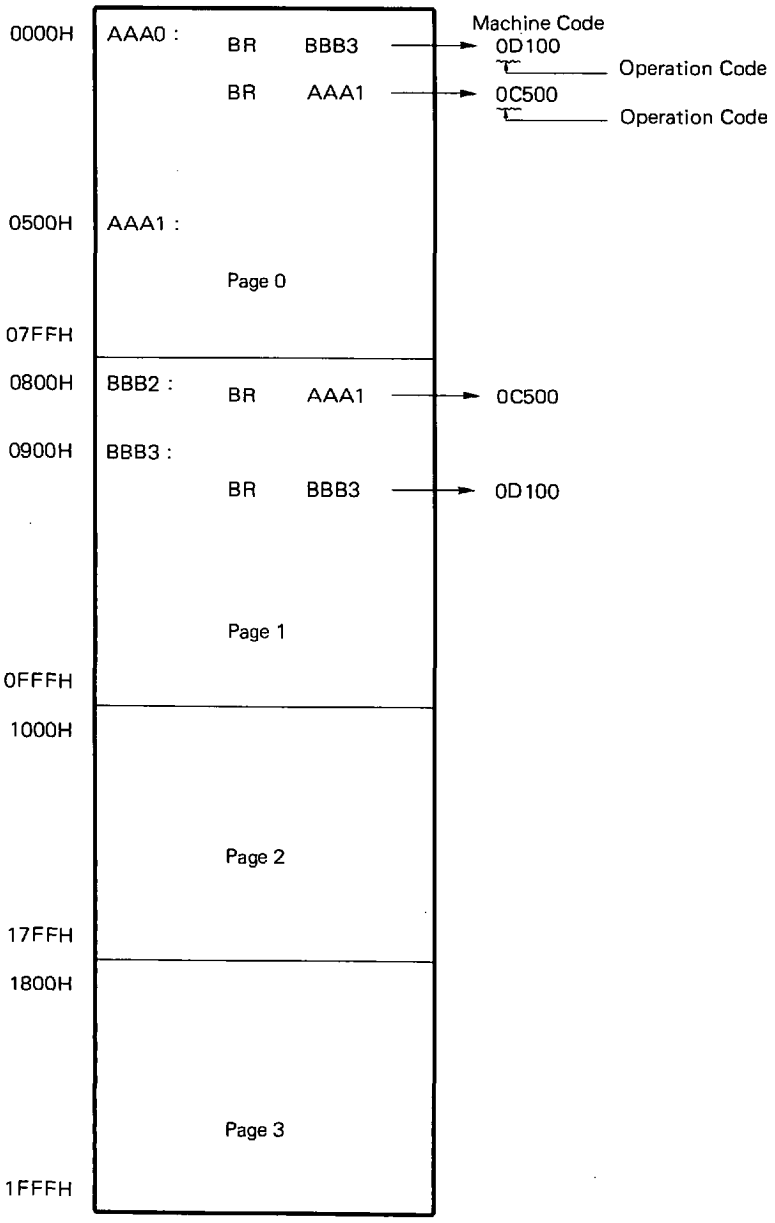
As shown in Fig. 3-2, the direct branch instruction can be used with the same description between data memory addresses 0000H to 1FFFH without the page concept when describing with the assembler.

However, the operation code of the direct branch instruction differs depending on the page to be branched.

For example, the operation code of the direct branch instruction into page 0 is "0CH", while the operation code of the direct branch instruction into page 1 is "0DH". If the 17K series assembler is used for assembling, these are automatically converted by the assembler referring to the jump destination. Here, what requires attention is that, if patch correction is performed in debugging, the program itself must decide on the branch destination page. At this time, the operation code is selected according to the branch destination page of the direct branch page and the lower 11 bits of the address is used as the operand.

For example, if the assembler description is "BR BBB2" (BBB2 is address 0800H), "0D000" is input to perform the patch correction. "BR AAA0" (AAA0 is address 0000H) is "0C000".

Fig. 3-2 Machine Code of Direct Branch Instruction



3.5 SUBROUTINE

The subroutine is used by dedicated subroutine call (CALL) and subroutine return (RET, RETSK) instructions.

Subroutine call instructions include the direct subroutine call instruction (CALL addr) which directly calls the program memory address specified with the instruction's operand (addr) and the indirect subroutine call instruction (CALL @AR) which calls the program memory address specified by the content of the address register.

As the return instruction from the subroutine, the RET instruction and the RETSK instruction are used. By executing the RET or RETSK instruction, the operation is returned to the program memory address following the address on which the subroutine call instruction (CALL) was executed. At this time, the RETSK instruction executes the first returned instruction as the no operation instruction (NOP).

3.5.1 Examples of Using the Subroutine

Fig. 3-3 shows the example of using the subroutine.

When using the "CALL addr" instruction, it is necessary to place its call address, i.e., the subroutine's start address within page 0 (addresses 0000H to 07FFH). The subroutine whose start address is located in any of the other pages (addresses 0800H to 1FFFH) cannot be called.

However, the return instruction (RET, RETSK) can be placed on any page. The CALL instruction itself can be on any page, too.

Example 1: Subroutine start address is within page 0:

If the subroutine start address is located within page 0 as shown in Fig. 3-3, both the return address and the return instruction can be located either within page 0 or page 1.

As long as the subroutine start address is in page 0, the CALL instruction can be used without any page concept. However, if is not possible to place the subroutine start address within page 0 for reasons related to program preparation, the method shown in example 2 is used.

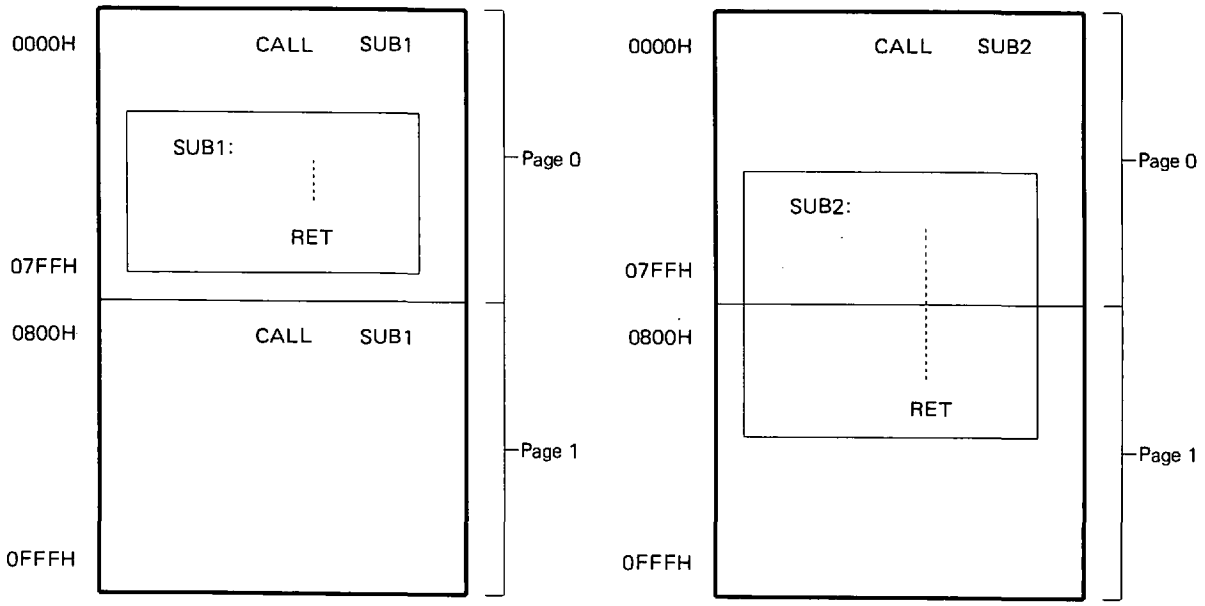
2: Subroutine start address is within page 1:

As shown in the example in Fig. 3-3, this method actually calls the subroutine (SUB1) via the BR instruction which is set within page 0.

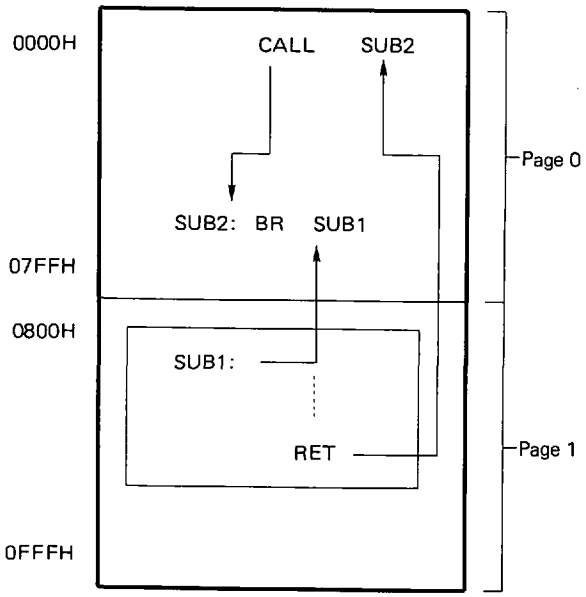
The subroutine call instruction (CALL @AR) which is based on indirect address specification calls the subroutine of the address specified by the contents of the address register. The address register consists of 8 bits. Therefore, the program memory addresses that can be called by the CALL @AR instruction are 0000H to 00FFH.

Fig. 3-3 Examples of Using the Subroutine

(a) Subroutine start address is within page 0:



(b) Subroutine start address is within page 1:



3.6 TABLE REFERENCE

The table reference is used when referring to the constant table in the program memory.

If the "MOVT DBF, @AR" instruction is executed, the content of the program memory address specified by the address register is stored in the data buffer.

The contents of the program memory are 16-bit configurations. Therefore, the constant data stored in the data buffer by the MOVT instruction becomes 16 bits (4 words). The address register consists of 8 bits.

Therefore, the program memory addresses that can be referenced by the MOVT instruction become 0000H to 00FFH.

Be careful because the stack is used temporarily when executing the MOVT instruction.

3.7 PRECAUTIONS IN ASSEMBLER DESCRIPTION

When using the assembler, error occurs if the program memory address (address based on numerals) is specified directly in the operand of the branch instruction (BR) or subroutine call instruction (CALL).

Example 1: Case of error occurrence

```
        BR          0005H ;┐
        CALL        00F0H ;┘  Error occurs in the assembler.
```

2: Case of non-error

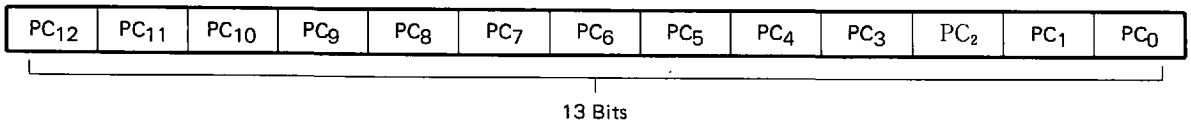
```
        BR      LOOP1       ;┐
LOOP1 :                     ;│  Levels are used in the program. And the BR or CALL
SUB1 :                      ;│  instruction is executed to these labels.
        CALL    SUB1        ;┘
        LOOP2   LAB 0005H   ;┐  0005H is allocated to LOOP2 as the label type.
        BR      LOOP2       ;┘
        BR      LD  0005H ;  Numeric value of the operand is converted to the label type.
```

For details, see **"AS17K User's Manual"**.

4. PROGRAM COUNTER (PC)

The program counter addresses the program memory, i.e., the program and consists of 13-bit binary counters.

Fig. 4-1 Program Counter



Normally, one increment is made by executing one instruction. However, when the jump instruction or the subroutine call instruction is executed, the address specified in the operand part is loaded. When the skip instruction has been executed, the address of the instruction following the skip instruction is specified regardless of the content of the skip condition. At this time, if the condition is to be skipped, the instruction following the skip instruction is regarded as the NOP instruction. In other words, execution of the NOP instruction results in specification of the address of the following instruction.

When accepting the interrupt request, addresses 1 to 4 (differing on the interrupt factor) is unconditionally loaded on the PC. The program counter is reset to 0 in power-on reset or CE reset.

Table 4-1 Vector Address in Interrupt

Priority Order	Interrupt Factor	Vector Address
1	RMC pin	4H
2	Internal timer	3H
3	$\overline{V_{SYNC}}$ pin	2H
4	Serial interface	1H

5. STACK

The stack is a register to save the return address of the program and contents of the system register to be described later when accepting the subroutine call or interrupt.

5.1 CONFIGURATIONS

The stack consists of a stack pointer (SP) which is a 4-bit binary counter, six 13-bit address stack registers (ASR) and two 3-bit interrupt stack registers.

5.2 STACK POINTER (SP)

The stack pointer is located in address 01H on the register file and specifies the address stack register. It is made −1 by the PUSH manipulation (CALL, MOVT, PUSH instruction and interrupt acceptance) and +1 by the POP manipulation (RET, RETSK, RETI, MOVT and POP instructions).

The upper 1 bit of the stack pointer is always "0". The obtainable stack pointer values include eight values of 0H to 7H. However, there is no stack matching 6H and 7H.

Fig. 5-1 Stack Pointer Configurations

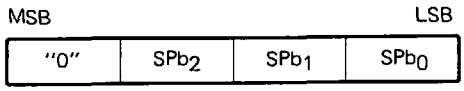


Table 5-1 Stack Pointer Operation

Instruction	Stack Pointer Value
CALL addr CALL @AR MOVT DBF, @AR PUSH AR Interrupt acceptance	SP − 1
RET RETSK MOVT DBF, @AR POP AR RETI	SP + 1

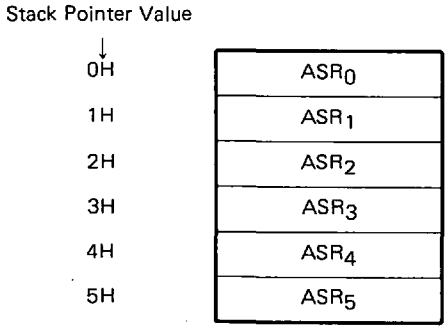
5.3 ADDRESS STACK REGISTER (ASR)

The address stack register consists of 6 x 13 bits. When executing the subroutine call instruction or accepting the interrupt request, the value of the content of the program counter plused by +1, in other words, the return address is stored here. The content of the stack register is loaded on the program counter by executing the return instruction to be returned to the flow of the original program.

The stack register is used for both the subroutine call and the interrupt. Therefore, if two levels are used for the interrupt, the remaining four levels of the stack register are available for the subroutine call.

If the MOVT instruction is executed, the stack register is used temporarily.

Fig. 5-2 Address Stack Register Configurations



5.4 INTERRUPT STACK REGISTER

The interrupt stack register is configured with 2 x 3 bits as shown in Fig. 5-3.

If the interrupt is accepted, the 2 bits of the bank register (BANK) and 1 bit of the index enable flat (XE) in the system register to be described later are saved.

Then, if the interrupt return instruction (RETI) is executed, the content to the interrupt stack register is returned to the system register's bank register and the index enable flag.

The interrupt stack register does not have the address specified by the stack pointer as in the address stack register. It saves data everytime an interrupt is accepted as shown in Fig. 5-4. If an interrupt exceeding two levels is received, the initial data is omitted forth. Therefore, it is necessary to save it in the program.

The contents of the interrupt stack register is undefined in power-on reset. The previous state is maintained when resetting the CE or executing the CLOCK STOP instruction.

Fig. 5-3 Interrupt Stack Register Configurations

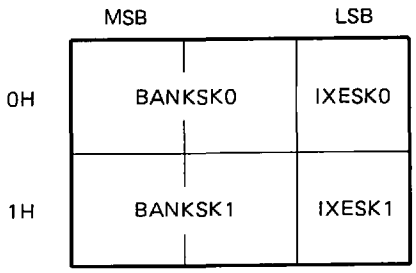
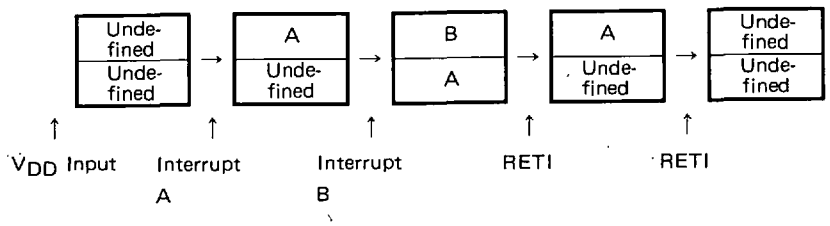


Fig. 5-4 Interrupt Stack Register Operation



6. DATA MEMORY (RAM)

The data memory stores such data as operation and control, etc. Data are constantly written or read by the instruction.

6.1 DATA MEMORY CONFIGURATIONS

Fig. 6-1 shows the data memory configurations.

As shown in Fig. 6-1, the data memory is divided in four units called "bank". These four banks are called BANK0, BANK1, BANK2 and BANK3 respectively.

Each bank is allocated with the address at every 4 bits of data. The upper 3 bits are called "row address" and the lower 4 bits "column address". For example, the data memory whose row address is 1H and column address is AH is called the data memory of address 1AH. One address consists of the memory of 4 bits, which is called "1 nibble".

Also, the data memory is divided into functional blocks as shown in **6.1.1** to **6.1.5** below.

6.1.1 System Register (SYSREG) Configurations

The system register consists of 10 nibbles allocated to addresses 74H to 7FH of the data memory. System registers are allocated regardless of the bank. In other words, any bank has the same system registers in its addresses 74H to 7FH.

The configurations are shown in Fig. 6-2.

6.1.2 Data Buffer (DBF) Configurations

The data buffer consists of 4 nibbles allocated to addresses 0CH to 0FH of BANK0 of the data memory.

The configurations are shown in Fig. 6-3.

6.1.3 General Register (GR) Configurations

The general register consists of 16 nibbles specified by any row addresses of the data memory.

The arbitrary row address is specified by the general register pointer in the system register.

The configurations are shown in Fig. 6-4.

6.1.4 Port Data Register (Port Register) Configurations

The port data register consists of 16 nibbles allocated to addresses 70H to 73H of each bank of the data memory. The configurations are shown in Fig. 6-5.

As shown in Fig. 6-5, the same port register is allocated to BANK0' address 73H, BANK2's address 73H, BANK1 and BANK3. Therefore, in reality, the register consists of 11 nibbles.

6.1.5 General-Purpose Data Memory Configurations

The general purpose data memory refers to the remaining area of the data memory excluding the system register and the port register.

It consists of total 448 nibbles of the 112 nibbles of each of BANK0 to BANK3.

Fig. 6-1 Data Memory Configuration

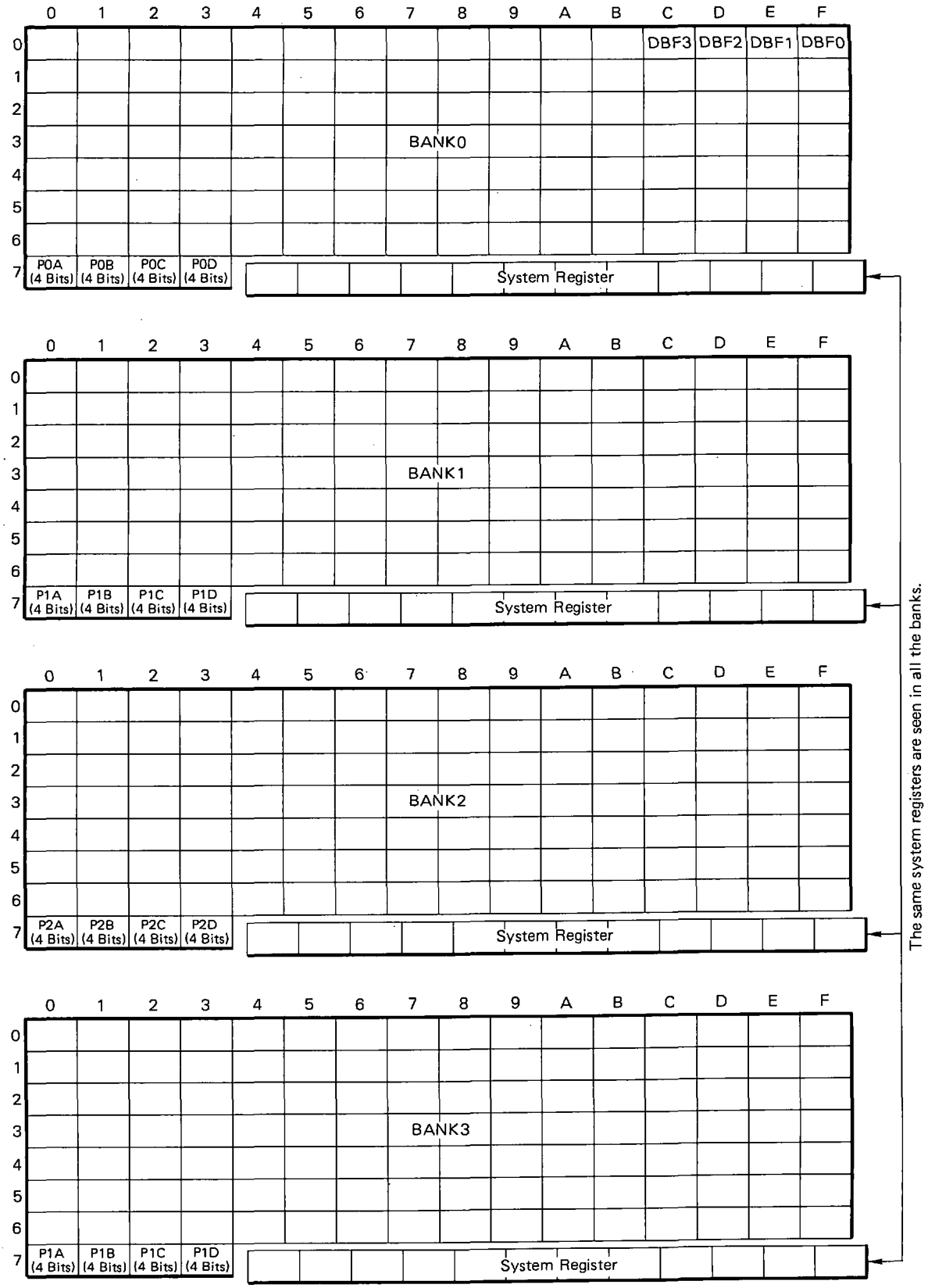


Fig. 6-2 System Register Configurations

System Register (SYSREG)												
Address	74H	75H	76H	77H	78H	79H	7AH	7BH	7CH	7DH	7EH	7FH
Name (Symbol)	Address Register (AR)				Window Register (WR)	Bank Register (BANK)	Index Register (IX) / Data Memory Row Address Pointer (MP)			General Register Pointer (RP)		Program Status Word (PSWORD)

Fig. 6-3 Data Buffer Configurations

Data Buffer (DBF)				
Address	0CH	0DH	0EH	0FH
Symbol	DBF3	DBF2	DBF1	DBF0

Fig. 6-4 General Register (GR) Configurations

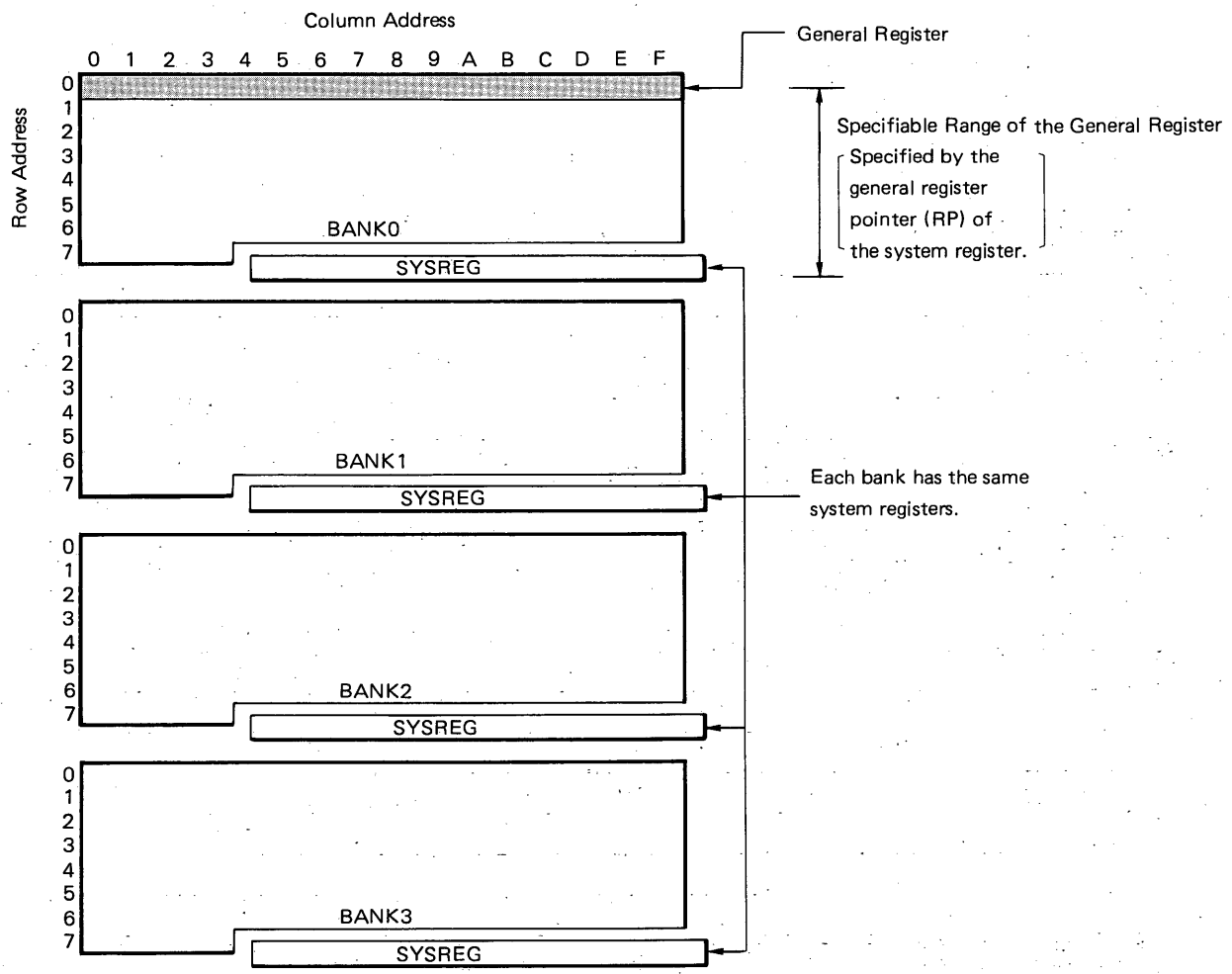


Fig. 6-5 Port Register Configurations

Port Register					
Address		70H	71H	72H	73H
Symbol	BANK0	P0A	P0B	P0C	P0D
	BANK1	P1A	P1B	P1C	P1D
	BANK2	P2A	P2B	P2C	P0D
	BANK3	P1A	P1B	P1C	P1D

6.2 DATA MEMORY FUNCTION

The data memory can perform 4-bit operation, comparison, decision and transmission between the data on the data memory and the immediate data (arbitrary data) with a single instruction by executing the data memory operation instruction shown in Table 6-1.

By using the general register, the data memory can also perform 4-bit arithmetic operation, comparison and transmission between the data memory and the general register with a single instruction.

Examples are shown below. For details, please refer to **7 "General Register (GR)"** and **8 "ALU"**.

Example 1: Data memory operation

```
; ①
MOV  35H,  #0001B ;  Transmits (writes) the immediate data 0001B to the data memory
                     content of address 35H of the bank, which has been selected at the
                     time.
; ②
ADD  76H,  #0001B ;  Adds the immediate data 0001B to the data memory content of address
                     76H of the bank, which has been selected at the time.
```

The bank selected at the time in both ① and ② is specified by the bank register in the system register. For details of the bank register, please refer to **9 "System Register (SYSREG)"**.

② is the ADD instruction to the data memory of address 76H. Address 76H is the system register as well. As the system register exists regardless of BANK, this instruction results in adding 0001B to 76H of the system register regardless of the bank.

Example 2: Data memory and general register operation

When the general register exists at row address 1H of BANK0:

```
; ①
ADD  7H,  36H     ;  The data memory contents of address 36H of the bank which has been
                     selected at the time is added to the contents of address 17H of BANK0,
                     i.e., the general register whose column address 7H.
; ②
LD   7H,  36H     ;  Transmits the content of address 36H of the data memory to the
                     general register whose column address is 7H.
                     At this time, the general register becomes address 17H of BANK0.
```

It is possible to operate all of the system register, the data buffer, the general register and the port register as a data memory by means of the data memory operation instruction.

Functions are explained in **6.2.1** to **6.2.4.**

6.2.1 System Register (SYSREG) Function

The system register exists to control the CPU.

For example, the bank register shown in Fig. 6-2 specifies the bank of the data memory. The general register pointer specifies the row address of the general register.

For details, see **9 "System Register (SYSREG)"**.

6.2.2 General Register (GR) Function

The general register can perform arithmetic operation and data transmission between the data memory.

The general register's bank and row address are specified by the general register pointer on the system register.

For example, if the general register pointer is set to 0, the 16 nibbles of BANK0's row address 0, in other words, addresses 00H to 0FH of BANK0, are specified as the general register.

When using the general register, it is necessary to keep in mind that transmission and operation instructions cannot be issued between the general register and the immediate data. In other words, if transmission or arithmetic operation is desired between the general register and the immediate data, it is necessary to handle the general register as the data memory.

For example, if "ADD 00H, #1" is executed when the general register is at BANK0's row address 0H (general register pointer is 0) and the presently selected bank is BANK0 (bank register is 0), then the contents of address 00H of BANK0 specified in the general register is increased by +1. However, if this instruction is executed when the presently selected bank is BANK1 (bank register is 1), then the contents of address 00H of BANK1 is increased by +1.

For details, see **7 "General Register (GR)"**.

6.2.3 Data Buffer (DBF)

The data buffer exists to store the data to be transmitted to peripheral circuits, such as A/D converter comparison voltage setting data, and the data sent from peripheral circuits, such as the input data of the serial interface.

For details, see **11 "Data Buffer (DBF)"**.

6.2.4 General-Purpose Port Data Register (Port Register)

The port register sets the output data of various general-purpose I/O ports and reads the input data. The output of each pin is set by setting data for the port register corresponding to the pin which is set to the output port.

The input status of each pin can be detected by reading the port register corresponding to the pin which is set to the input port. The relationship between the port register and each port (each pin) is shown in Fig. 6-6.

For details, please refer to **16 "General-Purpose Port"**.

Table 6-1 Data Memory Operation Instruction List

Function		Instruction
Arithmetic operation	Addition	ADD ADDC
	Subtraction	SUB SUBC
	Logic	AND OR XOR
Comparison		SKE SKGE SKLT SKNE
Transmission		MOV LD ST
Decision		SKT SKF

Fig. 6-6 Relationship between Port Register and Each Port (Pin) (1/2)

General-Purpose Port Data Register					Object Port	Pin		
Bank	Address	Symbol	Bit Symbol			No.	Symbol	Input/Output
BANK0	70H	P0A	b_3	P0A3	Port0A	39	$P0A_3$	Input/Output (Bit I/O)
			b_2	P0A2		40	$P0A_2$	
			b_1	P0A1		41	$P0A_1$	
			b_0	P0A0		42	$P0A_0$	
	71H	P0B	b_3	P0B3	Port0B	35	$P0B_3$	Input/Output (Bit I/O)
			b_2	P0B2		36	$P0B_2$	
			b_1	P0B1		37	$P0B_1$	
			b_0	P0B0		38	$P0B_0$	
	72H	P0C	b_3	P0C3	Port0C	51	$P0C_3$	Output
			b_2	P0C2		52	$P0C_2$	
			b_1	P0C1		53	$P0C_1$	
			b_0	P0C0		54	$P0C_0$	
	73H	P0D	b_3	P0D3	Port0D	59	$P0D_3$	Input
			b_2	P0D2		60	$P0D_2$	
			b_1	P0D1		61	$P0D_1$	
			b_0	P0D0		62	$P0D_0$	
BANK1 BANK3	70H	P1A	b_3	P1A3	Port1A	18	$P1A_3$	Output
			b_2	P1A2		19	$P1A_2$	
			b_1	P1A1		20	$P1A_1$	
			b_0	P1A0		21	$P1A_0$	
	71H	P1B	b_3	P1B3	Port1B	9	$P1B_3$	Input/Output (Bit I/O)
			b_2	P1B2		10	$P1B_2$	
			b_1	P1B1		11	$P1B_1$	
			b_0	P1B0		12	$P1B_0$	
	72H	P1C	b_3	P1C3	Port1C	55	$P1C_3$	Input/Output (group I/O)
			b_2	P1C2		56	$P1C_2$	
			b_1	P1C1		57	$P1C_1$	
			b_0	P1C0		58	$P1C_0$	
	73H	P1D	b_3	P1D3	Port1D	1	$P1D_3$	Output
			b_2	P1D2		2	$P1D_2$	
			b_1	P1D1		3	$P1D_1$	
			b_0	P1D0		4	$P1D_0$	

Fig. 6-6 Relationship between Port Register and Each Port (Pin) (2/2)

General-Purpose Port Data Register					Object Port	Pin		
Bank	Address	Symbol	Bit Symbol			No.	Symbol	Input/Output
BANK2	70H	P2A	b_3	P2A3	Port2A	5	$P2A_3$	Input/Output (Bit I/O)
			b_2	P2A2		6	$P2A_2$	
			b_1	P2A1		7	$P2A_1$	
			b_0	P2A0		8	$P2A_0$	
	71H	P2B	b_3	P2B3	Port2B	43	$P2B_3$	Output
			b_2	P2B2		44	$P2B_2$	
			b_1	P2B1		45	$P2B_1$	
			b_0	P2B0		46	$P2B_0$	
	72H	P2C	b_3	P2C3	Port2C	47	$P2C_3$	Output
			b_2	P2C2		48	$P2C_2$	
			b_1	P2C1		49	$P2C_1$	
			b_0	P2C0		50	$P2C_0$	
	73H	P0D	b_3	P0D3	Port0D	59	$P0D_3$	Input
			b_2	P0D2		60	$P0D_2$	
			b_1	P0D1		61	$P0D_1$	
			b_0	P0D0		62	$P0D_0$	

6.3 PRECAUTIONS IN USING DATA MEMORY

6.3.1 Data Memory Address Specification

When using the 17K series assembler, error occurs if the data memory address is described directly in numerals in the operand of the data memory operation instruction as is shown in Example 1.

This is to reduce the bug factors in program correction, considering the program maintenance. However, actual addresses are described in program examples on this data sheet for easier understanding. For actual programming, please refer to the assembler manual.

Example 1: Error

```
; ①
MOV   2FH,    #0001B    ;  Directly specifies address 2FH.
; ②
MOV   0.2FH,  #0001B    ;  Directly specifies address 2FH of BANK0.
```

Non-error

```
; ③
M02F  MEM 0.2FH         ;  Symbol-defines BANK0's address 2FH in M02F as the memory
                           type.
MOV   M02F,   #0001B    ;
; ④
MOV   .MD.2FH, #0001B   ;  Converts address 2FH into the memory type by means of .MD.
                           However, this kind of method should be avoided to reduce bug
                           factors.
```

Therefore, it is necessary to symbol-define beforehand the data memory address with the MEM instruction (symbol definition pseudo-instruction) which is an assembler pseudo-instruction.

For data memory symbol definition, it is necessary to define the bank of the data memory as well as shown in example 2.

However, at this time, if, for example, the data memory symbol-defined in BANK2 as shown in example 2 is used in the range of BANK1 on the program, the data memory of BANK1 is operated.

Example 2:

```
M1      MEM     0.15H     ;┐
M2      MEM     1.15      ;│ Symbol definition pseudo-instruction
M3      MEM     2.15H     ;┘
```

Bank — Row address — Column address

```
BANK1                     ; Assembler incorporation macro instruction BANK ← 1
MOV     M1,     #0000B    ;┐
MOV     M2,     #0000B    ;│ Although M1, M2 and M3 are symbol-defined in different
MOV     M3,     #0000B    ;┘ banks at ①, they are BANK1 on the program. Therefore, all
                            of these three instructions write 0 in the data memory of
                            address 15H of BANK1.
```

7. GENERAL REGISTER (GR)

The general register performs direct operation and transmission with the data memory with the register located in the data memory space.

7.1 GENERAL REGISTER CONFIGURATIONS

The general register configurations are shown in Fig. 7-1.

As shown in Fig. 7-1, the same row addresses 16 words (16 words x 4 bits) can be used as general registers on the data memory space.

Which row address to use can be set by the general register pointer of the system register. As the general register pointer has 3 bits, the data memory space which can be used as the general register becomes row addresses 0H to 7H of BANK0.

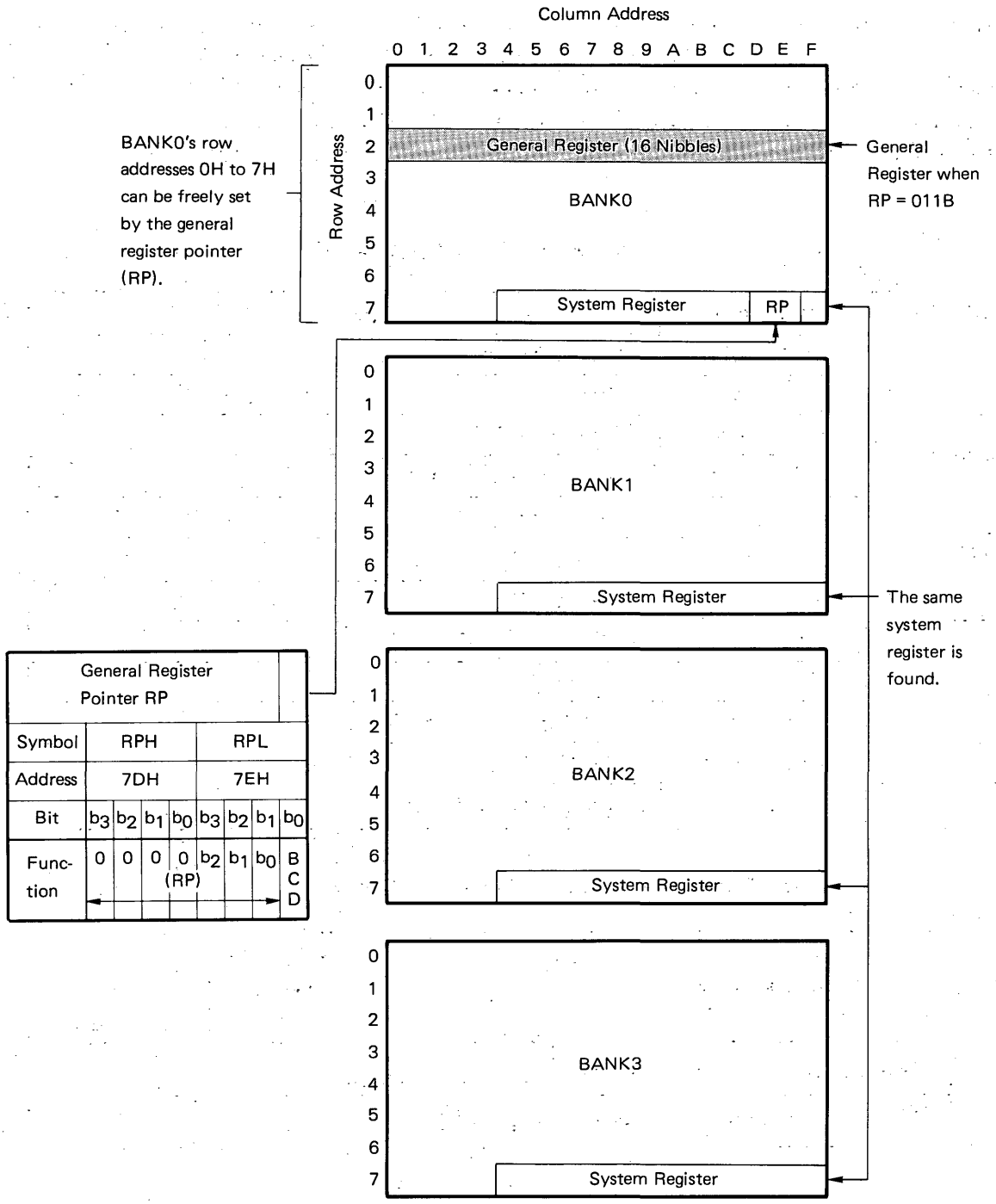
For details, see to **9.6 "General Register Pointer (RP)"**.

7.2 GENERAL REGISTER FUNCTION

By using the general register, arithmetic operations and transmissions between the data memory and the general register are made possible by a single instruction. As the general register is the data memory, arithmetic operations and transmissions between data memories can be realized by a single instruction.

General register belongs to the data memory space, they can be controlled by the data memory operation instruction in the same manner as other data memories.

Fig. 7-1 General Register Configurations



7.3 GENERAL REGISTER AND DATA MEMORY ADDRESS GENERATION IN EACH INSTRUCTION

Arithmetic operation and transmission instructions between the general register and the data memory are shown in Table 7-1.

According to the example ADD r, m (r ← (r) + (m)), as shown in Table 7-2, the address of the general register specified by this instruction is generated by the value specified by the general register pointer and r.

The content of the general register specified by the generated general register address and the content of the m-specified data memory are added together and then the result is stored in the general register.

The general register address generation described above is applied in the same manner with other instructions shown in Table 7-1.

Table 7-1 Operation Instructions between General Register and Data Memory

Instruction Group	Instruction	Operation
Addition	ADD r, m	(R) ← (R) + (M)
	ADDC r, m	(R) ← (R) + (M) + (CY)
Subtraction	SUB r, m	(R) ← (R) − (M)
	SUBC r, m	(R) ← (R) − (M) − (CY)
Logical operation	AND r, m	(R) ← (R) AND (M)
	OR r, m	(R) ← (M) OR (M)
	XOR r, m	(R) ← (R) XOR (M)
Transmission	LD r, m	(R) ← (M)
	ST m, r	(M) ← (R)
	MOV @r, m	[MP, (R)] ← (M) or [m, (R)] ← (M)
	MOV m, @r	M ← [MP, (R)] or M ← [H, (R)]
Shift	RORC r	Right shift including CY

Table 7-2 Address of General Register and Data Memory

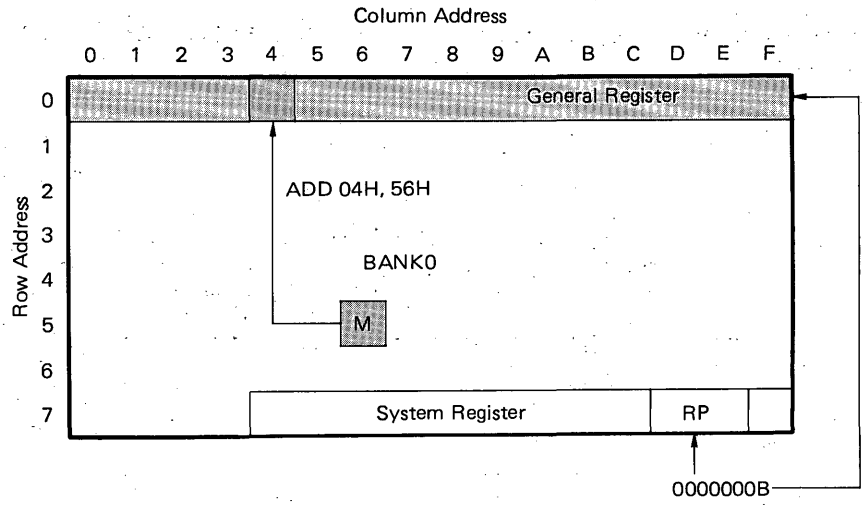
Instruction	Address Content	Generated Address		
		Bank	Row Address	Column Address
ADD r, m	General register address specified by r	00 ←	(RP)	r →
	Data memory address specified by m	00 ← (BANK)		m →

Example 1: BANK0

```
AND   RPL,   #0001B ;  RP ← 0000000B; Sets the general register to BANK0's row address
                       0H.
ADD   04H,   56H    ;
```

When the above instruction is executed, as shown in Fig. 7-2, the content of address 04H of BANK0 which is the general register and the content of address 56H of the data memory are added together and then the result is stored in 04H of the general register.

Fig. 7-2 Execution of Example 1



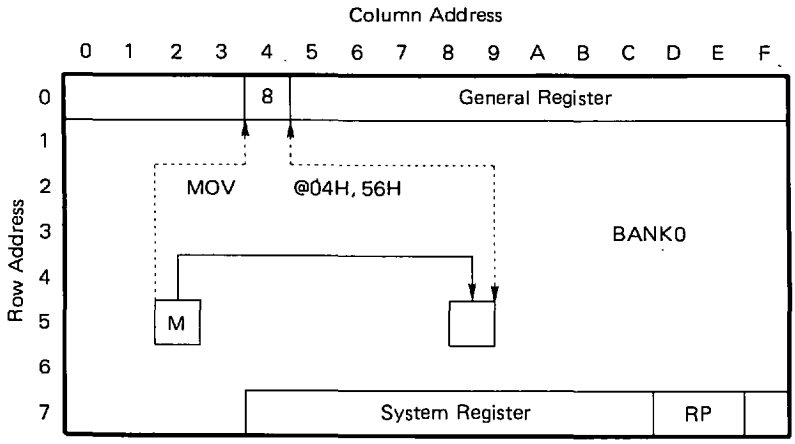
Example 2: When BANK0, and MPE = 0:

```
AND   RPL,   #0001B ;  RP ← 0000000B; Sets the general register to BANK0's row address
                       0H.
MOV   04H,   #8     ;  04H ← 8
MOV   @04H,  52H
```

If the above instructions are executed, the content of data memory address 52H is transmitted to address 58H. In other words, the "MOV @r, m" instruction is called "general register indirect transmission" and becomes the data memory, i.e., address 58H's data memory in which the content (8 in the above description) of the general register specified by r is made the column address and the row address (5 above) specified by m is made the row address (see **Fig. 7-3**).

For details of general register indirect transmission, please refer to **9.5 "Index Register (IX) and Data Memory Row-address Pointer (MP)".**

Fig. 7-3 Execution of Example 2



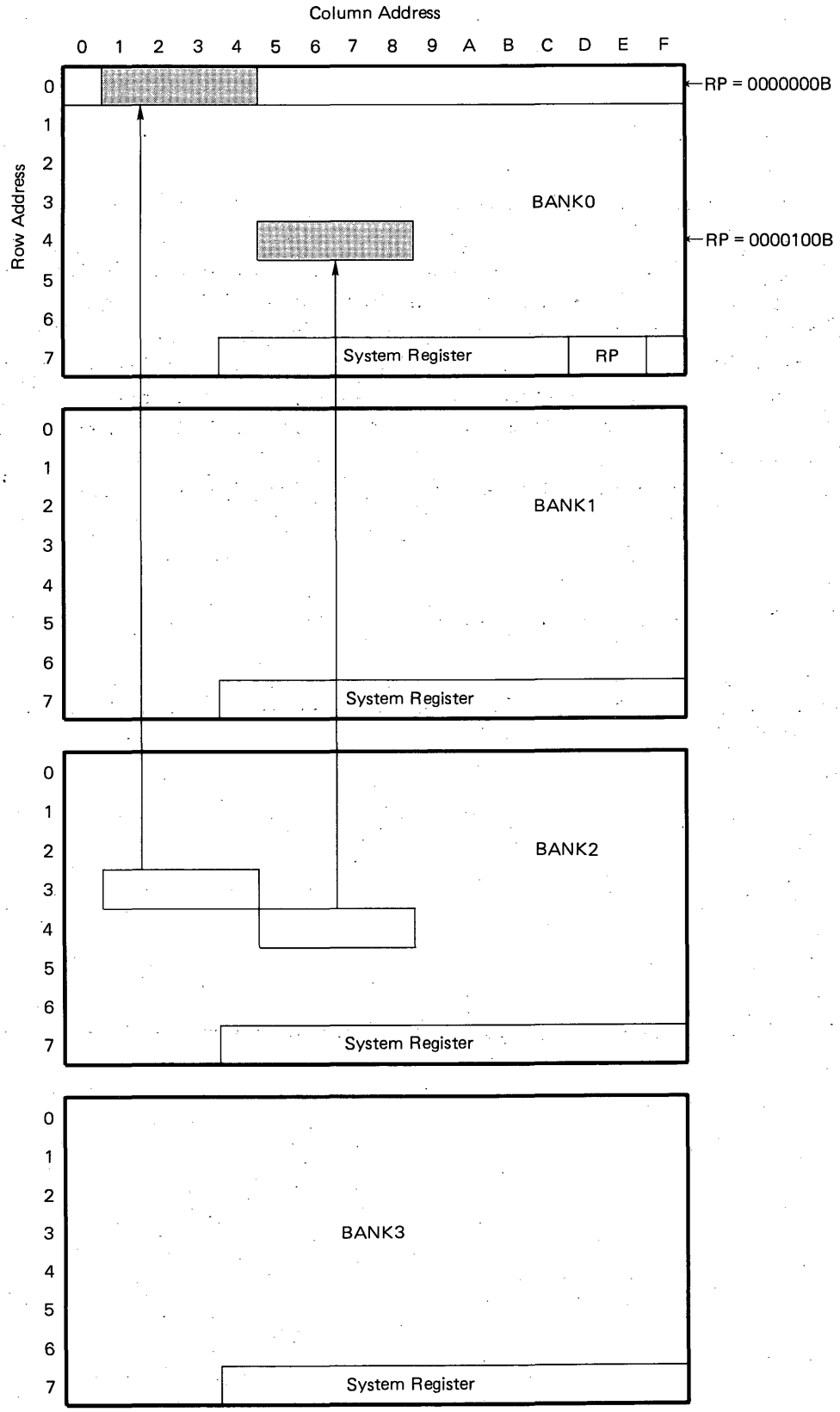
Example 3:

```
AND   RPL,   #0000B ;  RP ← 0000000B; Sets the general register to BANK0's row
                       address.
MOV   BANK,  #0010B ;  BANK2
LD    01H,   31H
LD    02H,   32H
LD    03H,   33H
LD    04H,   34H
OR    RPL,   #1000B ;  RP ← 0000100B; Sets the general register to BANK0's row
                       address.
LD    05H,   45H
LD    06H,   46H
LD    07H,   47H
LD    08H,   48H
```

This example is the program which transmits the 8-word data in BANK2 by 4 words to the data memory of BANK0, as shown in Fig. 7-4.

At this time, if the general register is fixed and exists only in row address 0 of BANK0 for example, the instruction which transmits all 8 words at once to the register and then stores them once again in the data memory is required. However, if the row address of the general register changed by the general register pointer as in this example, the operation can be terminated with the STORE instruction only.

Fig. 7-4 Execution of Example 3



7.4 PRECAUTIONS IN USING GENERAL REGISTER

Precautions when using the general register are explained based on the example below.

```
Example:  AND   RPL,   #0001B   ;  PR ← 0000010B
          OR    RPL,   #0100B   ;
          MOV   BANK,  #0000B   ;  BANK0
          LD    04H,   32H
```

If the above instructions are executed, the content of data memory BANK0's 32H is loaded in address 24H of general register BANK0.

This is because, if the description is as above in the LD r, m instruction, the general register is BANK0's row address 2H and thus the general register address specified by r becomes address 24H of BANK0 and the data memory address specified by m becomes address 32H of BANK0 (see **Fig. 7-5**).

Here, what needs to be careful about is when, in using the assembler, the value specified by r has been described as the actual data memory address, for example, 24H. As r needs only the lower 4 bits even in this case, the row address 2H is ignored in the assembler. Therefore, even when the LD 24H, 32H instruction has been executed, the same result as above is obtained.

When using the assembler, if the general register address is specified directly in the instruction operand as shown below, error occurs.

Error

```
LD   04H,   32H    ; Describes the general register address with 04H.
```

Normally used method

```
R1   MEM   0.04H ;┐
M1   MEM   0.32H ;│
LD   R1,   M1    ;┘
```

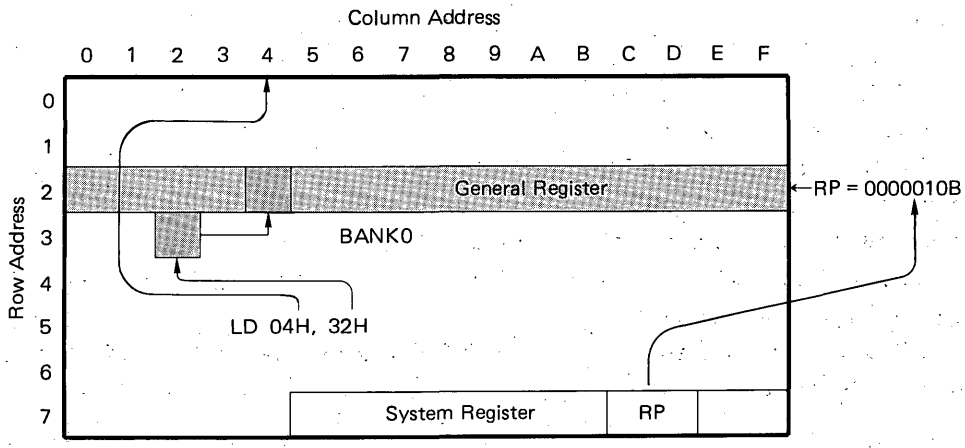
① Allocates BANK0's 04H and 32H to R1 and M1 respectively as the memory type.

At this time, even if

```
R2   MEM   0.34H
M1   MEM   0.32H
LD   R2,   M1
```

is executed, R1 and R2 operate in the same manner as in ① because they are the same column address.

Fig. 7-5 Execution of Example



Another precaution to take when using the general register is that it does not have the arithmetic operation instruction between the general register and the immediate data. In other owrds, to issue an arithmetic operation instruction between the data memory and the immediate data specified in the general register, it is necessary to handle this data memory not as the general register but as the data memory.

8. ALU

ALU processes the arithmetic operation, the logical operation, the bit decision, the comparative decision and the rotation of 4-bit data.

8.1 ALU CONFIGURATIONS

Fig. 8-1 shows ALU block configurations.

As shown in Fig. 8-1, the ALU block is configured with the ALU body, which processes 4-bit data; registers A and B for temporary memory, which are ALU's peripheral circuits; the status flip-flop, which controls the ALU status; and the decimal correction circuit used in decimal operations.

The status flip-flop is configured, as shown in Fig. 8-2, with FF for the zero flag, FF for the carry flag, FF for the compare flag and FF for the BCD flag.

The status flip-flop corresponds to one-to-one with each flag of the program status word (PSWORD: Addresses 7EH and 7FH) in the system register.

Fig. 8-1 ALU Block Configurations

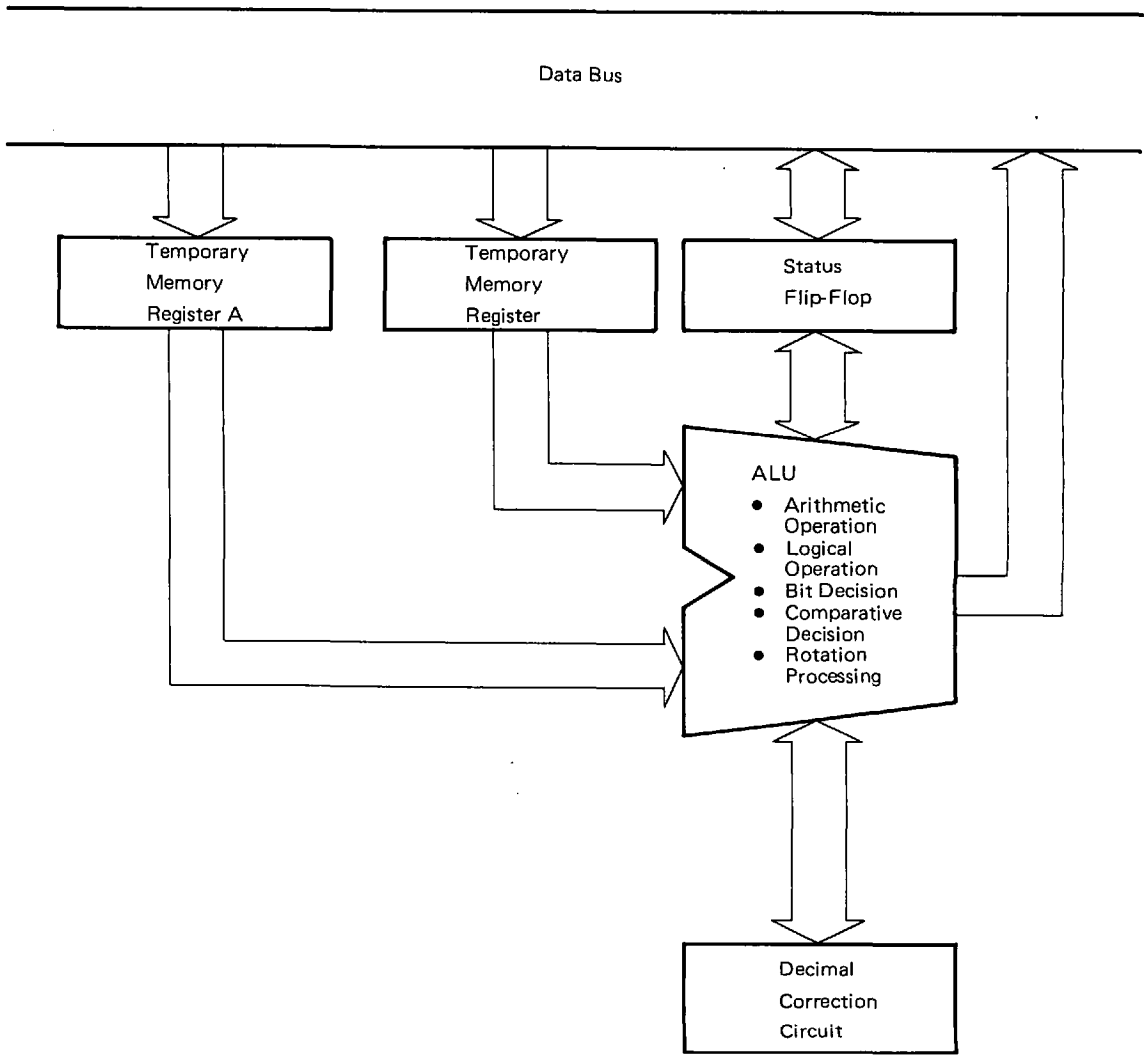
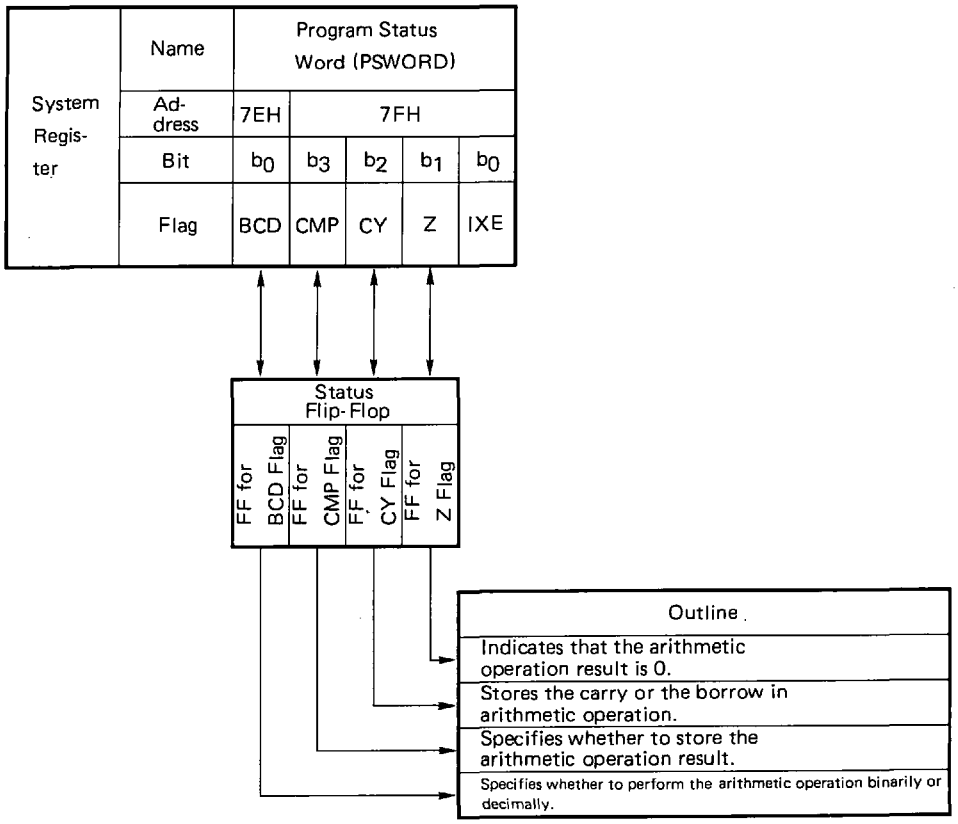


Fig. 8-2 Status Flip-Flop Configurations



8.2 ALU FUNCTION

The ALU block performs the arithmetic operation, the logical operation, the bit decision, the comparative decision and the rotation processing with the instruction specified by the program. Table 8-1 lists arithmetic operations, decisions and rotation processings.

By executing each of the instructions shown in Table 8-1, 4-bit arithmetic operations, decisions and rotation processings, or 1-column decimal arithmetic operations can be executed by a single instruction.

8.2.1 Arithmetic Operation

The arithmetic operation includes addition and subtraction. The arithmetic operation can be performed between the contents of the general register and that of the data memory, or between the contents of the data memory and the immediate data. Also, in the arithmetic operation, the binary 4-bit operation and the decimal 1-column operation are possible.

The logical operation includes the logical AND, the logical OR and the logical exclusive OR. The logical operation can be performed between the contents of the general register and that of the data memory, or between the contents of the data memory and the immediate data.

In the bit decision, the bit which is 0 or 1 among the 4-bit data of the data memory is decided upon.

In the comparison decision, the data memory contents and the immediate data are compared and then the decision is made whether they are "equal", "unequal", "equal or more" or "less".

In this rotation processing, the 4-bit data of the general register is shifted by 1 bit in the lower bit direction (rotated to right).

Table 8-1 ALU Processing Instruction List

ALU Function		Instruction	Operation	Description
Arithmetic operation	Addition	ADD r, m	(R) ← (R) + (M)	Adds together general register and data memory contents. Stores the result in the general register.
		ADD m, #i	(M) ← (M) + i	Adds together the data memory and immediate data contents. Stores the result in the data memory.
		ADDC r, m	(R) ← (R) + (M) + (CY)	Adds general register and data memory contents together with the CY flag. Stores the result in the general register.
		ADDC m, #i	(M) ← (M) + i + (CY)	Adds data memory and immediate data contents together with the CY flag. Stores the result in the data memory.
	Subtraction	SUB r, m	(R) ← (R) − (M)	Subtracts the data memory contents from the general register contents. Stores the results in the general register.
		SUB m #i	(M) ← (M) − i	Subtracts the immediate data from the data memory contents. Stores the result in the data memory.
		SUBC r, m	(M) ← (R) − (M) − (CY)	Subtract the data memory contents and the CY flag from the general register contents. Stores the result in the general register.
		SUBC m, #i	(M) ← (M) − i − (CY)	Subtracts the immediate data and the CY flag from the data memory contents. Stores the result in the data memory.
Logical operation	Logical OR	OR r, m	(R) ← (R) OR (M)	Applies logical OR to the general register and data memory contents. Stores the result in the general register.
		OR m, #i	(M) ← (M) OR i	Applies logical OR to the data memory and immediate data contents. Stores the result in the data memory.
	Logical AND	AND r, m	(R) ← (R) AND (M)	Applies logical AND to the general register and data memory contents. Stores the result in the general register.
		AND m, #i	(M) ← (M) AND i	Applies logical AND to the data memory and immediate data contents. Stores the result in the data memory.
	Logical exclusive OR	XOR r, m	(R) ← (R) XOR (M)	Applies logical XOR to the general register and data memory contents. Stores the result in the general register.
		XOR m, #i	(M) ← (M) XOR i	Applies XOR to the data memory and immediate data contents. Stores the result in the data memory.

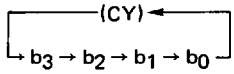
ALU Function		Instruction	Operation	Description
Bit decision	True	SKT m, #n	SKIP if (M) $\geqq$ n	Skips if all the bits in the data memory contents specified by n are true (1). Does not store the result.
	False	SKF m, #n	SKIP if (M) $\leqq$ n XOR 1111B	Skips if all the bits in the data memory contents specified by n are false (0). Does not store the result.
Comparative decision	Equal	SKE m, #i	SKIP if (M) = i	Skips if the data memory contents is equivalent to that of the immediate data. Does not store the result.
	Unequal	SKNE m, #i	SKIP if (M) $\neq$ i	Skips if the data memory contents is not equivalent to that of the immediate data. Does not store the result.
	Equal or more	SKGE m, #i	SKIP if (M) $\geqq$ i	Skips if the data memory contents is equivalent or more than that of the immediate data. Does not store the result.
	Equal or less than	SKLT m, #i	SKIP if (M) < i	Skips if the data memory contents is less than that of the immediate data. Does not store the result.
Rotation	Clockwise rotation	RORC r		Rotates the general register contents clockwise together with the CY flag. Stores the result in the general register.

Table 8-2 Program Status Word Operation

ALU Function	BCD Flag Value	CMP Flag Value	Arithmetic Operation	CY Flag	Z Flag	Modification by IXE = 1
Arithmetic operation	0	0	Stores the binary operation result.	This is set if CARRY/ BORROW occur. It is reset if they do not occur.	This is set if the arithmetic operation result is 0000B. It is reset if the result is other than 0000B.	With
	0	1	Does not store the binary operation result.		If the arithmetic operation operation result is 0000B, the state is held. If the result is other than 0000B, this is reset.	
	1	0	Stores the decimal operation result.		This is set if the arithmetic result is 0000B. It is reset if the result is other than 0000B.	
	1	1	Does not store the decimal operation result.		The state is held if the arithmetic operation result is 0000B. If the result is other than 0000B, this is reset.	
Logical operation	Arbitrary (hold)	Arbitrary (hold)	No change	Previous state held	Previous state held	With
Bit dicision	Arbitrary (hold)	Reset	No change	Previous state held	Previous state held	With
Comparative decision	Arbitrary (hold)	Arbitrary (hold)	No change	Previous state held	Previous state held	With
Rotation	Arbitrary (hold)	Arbitrary (hold)	No change	Value of general register b_0	Previous state held	With

8.2.2 Temporary Memory Registers A and B Function

Temporary memory registers A and B are the registers necessary for processing 4-bit data all at once. Data to be processed and processing data are stored in these registers temporarily.

8.2.3 Status-Flip-Flop Function

The status flip-flop stores the status of ALU's operation control and processed data. The status flip-flop is corresponded one-to-one to each flag of the program status word (PSWORD) of the system register. Therefore, if the system register is operated, the status flip-flop is operated simultaneously. Therefore, each flag of the program status word is explained below.

These flags can change the value by directly operating the program status word. At this time, the status flip-flop corresponded to the changed flag changes in the same manner.

(1) Z flag

This flag is set (1) if the result of the arithmetic operation is 0000B. It is reset (0) if the result is other than 0000B. However, the condition for setting (1) differs as below depending on the CMP flag status.

(a) CMP flag = 0

This is set (1) if the arithmetic operation result is 0000B, and is reset (0) if the result is other than 0000B.

(b) CMP flag = 1

The previous status is held if the arithmetic operation result is 0000B. The flag is reset (0) if the result is other than 0000B.

(2) CY flag

This is set (1) if CARRY or BORROW occurs as a result of the arithmetic operation, and is reset (0) if CARRY or BORROW does not occur.

When performing the arithmetic operation together with CARRY or BORROW, the CY flag content is arithmetically operated in the lowest bit.

When performing the rotation (RORC instruction), the CY flag is shifted to the highest bit (b_3) of data. Thus, the CY flag is reset (0) if the lowest bit (b_0) of the general register is "0", and set (1) if the lowest bit (b_0) of the general register is "1".

The CY flag does not change except in the arithmetic operation or the rotation.

(3) CMP flag

The result of the arithmetic operation executed when the CMP flag is set (1) is not stored in the general register and the data memory.

The CMP flag is reset (0) if the bit decision instruction is executed.

This does not affect the comparative decision, the logical operation or the rotation.

(4) BCD flag

If the BCD flag is set (1), all the arithmetic operations are performed decimally.

They are performed binarily if this flag is reset (0).

It does not affect the logical operation, the bit decision, the comparative decision or the rotation.

8.2.4 Decimal Adjustment Circuit Function

If the BCD flag is set (1) in the arithmetic operation, the arithmetic operation result is converted to decimal figures by the decimal adjustment circuit.

Table 8-3 shows the arithmetic operation output data and the output data after decimal adjustment.

For details of decimal operation, please refer to **8.3 "Arithmetic Operation (Binary and Decimal Addition and Subtraction)"**.

Table 8-3 Decimal Operation Adjustment Data

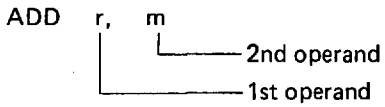
Arithmetic Operation Output Data		Output Data after Decimal Adjustment			
		Addition		Subtraction	
CY Flag	Data	CY Flag	Data	CY Flag	Data
0	0000B	0	0000B	0	0000B
0	0001B	0	0001B	0	0001B
0	0010B	0	0010B	0	0010B
0	0011B	0	0011B	0	0011B
0	0100B	0	0100B	0	0100B
0	0101B	0	0101B	0	0101B
0	0110B	0	0110B	0	0110B
0	0111B	0	0111B	0	0111B
0	1000B	0	1000B	0	1000B
0	1001B	0	1001B	0	1001B
0	1010B	1	0000B	1	1100B
0	1011B	1	0001B	1	1101B
0	1100B	1	0010B	1	1110B
0	1101B	1	0011B	1	1111B
0	1110B	1	0100B	1	1100B
0	1111B	1	0101B	1	1101B
1	0000B	1	0110B	1	1110B
1	0001B	1	0111B	1	1111B
1	0010B	1	1000B	1	1100B
1	0011B	1	1001B	1	1101B
1	0100B	1	1110B	1	1110B
1	0101B	1	1111B	1	1111B
1	0110B	1	1100B	1	0000B
1	0111B	1	1101B	1	0001B
1	1000B	1	1110B	1	0010B
1	1001B	1	1111B	1	0011B
1	1010B	1	1100B	1	0100B
1	1011B	1	1101B	1	0101B
1	1100B	1	1010B	1	0110B
1	1101B	1	1011B	1	0111B
1	1110B	1	1100B	1	1000B
1	1111B	1	1101B	1	1001B

8.2.5 ALU Block Processing Procedure

If an arithmetic operation, logical operation, bit decision, comparative decision or rotation instruction is executed on the program, the arithmetic operation, the decision or the processed data and the processing data are stored respectively in temporary memory registers A and B.

The processed data, which is 4-bit data, is the contents of the general register or data memory address-specified with the first operand of each instruction. The processing data, which is 4-bit data, is the contents of the data memory address-specified with the 2nd operand of each instruction or the immediate data directly specified with the 2nd operand.

For example, in the instruction ADD r, m,



the processed data is the contents of the general register address-specified with r, and the processing data is the contents of the data memory address-specified with m.

In the instruction "ADD m, #i", the processed data is the contents of the data memory address-specified with m, and the processing data is the immediate data specified with #i.

The rotation instruction RORC r, for which the processing method is fixed, needs only the processed data, which is the contents of the general register address-specified with r.

Next, data stored in temporary memory registers A and B execute the arithmetic operation, logical operation, bit decision, comparative decision or rotation in a ALU in accordance with the respective instruction. If the executed instructions are the arithmetic operation, the logical operation and the rotation, data processed in ALU are stored in the general register or data memory specified with the 1st operand of the instruction to terminate the operation. If the executed instructions are the bit decision and the comparative decision, the operation is terminated by skipping the following instruction on the program (the following instruction is executed as the NO OPERATION instruction (NOP)), based on the result processed in ALU.

It is necessary to be careful about the following points in ALU block operations.

(1) The arithmetic operation is affected by the CMP or BCD flag of the program status word.
(2) The logical operation is not affected by the CMP or BCD flag of the program status word. Neither does this operation does affect the Z or CY flag.
(3) The bit decision resets the CMP flag of the program status word.
(4) The arithmetic operation, logical operation, bit decision, comparative decision or rotation receives modification by the index register if the IXE flag of the program status word is set (1). (See **9 "System Register (SYSREG)"**.)

8.3 ARITHMETIC OPERATION (BINARY AND DECIMAL ADDITION AND SUBTRACTION)

As shown in Fig. 8-3, the arithmetic operation is largely divided into addition and subtraction, and these are further subdivided into addition with CARRY and subtraction with BORROW. These four types in the arithmetic operation uses instructions "ADD", "ADDC", "SUB" and "SUBC" respectively.

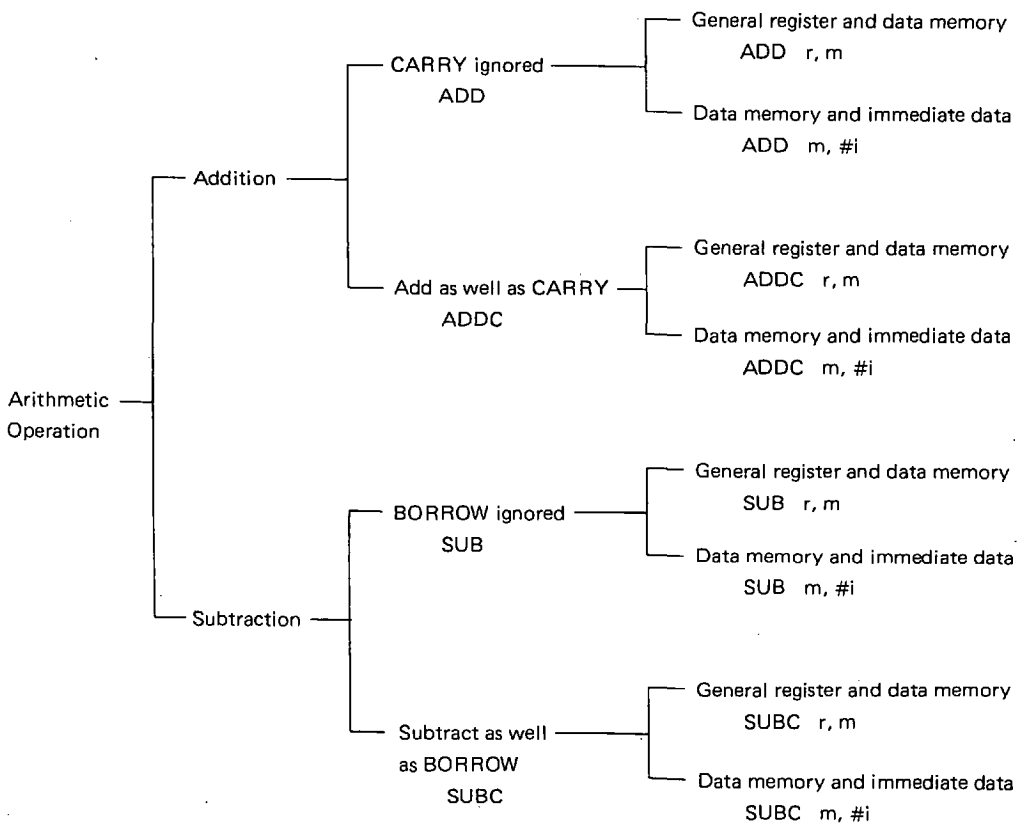
The instructions "ADD", "ADDC", "SUB" and "SUBC" are divided into addition/subtraction of the general register and the data memory; and addition/subtraction of the data memory and the immediate data. This is decided by the value described in the operand of each instruction. In other words, if the operand is "r, m", it will result in addition/subtraction of the general register and the data memory. If "m, #i", it will result in addition/subtraction of the data memory and the immediate data.

The arithmetic operation instruction is affected by the program status word of the system register. Binary or decimal operations can be conducted by the BCD flag of the program status word and it is possible not to store the arithmetic operation result anywhere by means of the CMP flag.

8.3.1 to **8.3.4** describe various arithmetic operation instructions and the program status word. However, for details of the index modification based on the index enable flag (IXE), please refer to **9 "System Register (SYSREG)"**.

And, precautions when using the arithmetic operation are described in **8.3.5.**

Fig. 8-3 Arithmetic Operation Types



8.3.1 Addition/subtraction when CMP Flag = 0, BCD Flag = 0

Binary addition/subtraction is performed and the result is stored in the general register or the data memory.

The CY flag is set (1) if the operation result exceeds 1111B (occurrence of CARRY) or less than 0000B (occurrence of BORROW), and reset (0) if otherwise.

If the operation result becomes 0000B, the Z flag is set (1) regardless of the occurrence of CARRY or BORROW. If it is not 0000B, the Z flag is reset (0).

```
Example 1: MOV  R1,  #1111B ;  Transmits 1111B to general register R.
           MOV  M1,  #0001B ;  Transmits 0001B to data memory M1.
           ADD  R1,  M1     ;  Adds R1 and M1.
```

At this time, R1 + M1 is calculated as below.

```
  1111B ... Contents of R1
+ 0001B ... Contents of M1
-------
1 0000B
~
CARRY
```

Therefore, the addition result 0000B is written in the contents of R1, and the CY flag is set (1). The contents of M1 do not change.

If the data resulting from addition of the contents of R1 with that of M1 is not likely to output CARRY, the CY flag is reset (0).

And, because the arithmetic operation result is 0000B, the Z flag is set (1).

```
Example 2: MOV  M1,  #1010B ;  Transmits 1010B to data memory M1.
           ADD  M1,  #0101B ;  Adds immediate data 0101B to M1.
```

At this time, M1 + 0101B is calculated as below.

```
  1010B ... Contents of M1
+ 0101B ... Immediate data
-------
0 1111B
~
CARRY
```

Thus, 1111B is written in the contents of M1, and the CY and Z flags are reset.

```
3: MOV  R1,  #1000B ;  Writes 1000B in general register R1.
   MOV  M1,  #1111B ;  Writes 0101B in data memory M1.
   ADD  M1,  #0001B ;  ① Adds immediate data 0001B to M1.
   ADDC R1,  M1     ;  ② Adds R1 and M1 together with CARRY.
```

① is calculated as below.

```
  1111B ... Contents of M1
+ 0001B ... Immediate data
-------
1 0000B
~
CARRY
```

Therefore, 0000B is written in M1, and CY and Z flags are set (1).
② is calculated as shown below.

```
  1000B ... Contents of R1
  0000B ... Contents of M1
+    1  ... Contents of CY flag
-------
 0 1001B
 ~
 CARRY
```

In other words, when executing the ADDC instruction, the addition is made together with the contents of the CY flag at this time, and the CY flag is reloaded by the CARRY output of the result.

Example 4:
```
MOV  R1, #0000B ; Writes 0000B in general register R1.
MOV  M1, #1000B ; Writes 10000B in data memory M1.
SUB  R1, M1     ; Subtracts M1 from R1.
```
At this time, R1 − M1 is calculated as below.

```
  0000B ... Contents of R1
− 1000B ... Contents of M1
-------
 1 1000B
 ~
 BORROW
```

Therefore, the subtraction result 1000B is written in R1. At this time, BORROW occurs, the CY flag is set (1). If data does not cause BORROW, the CY flag is reset (0). In other words, CARRY by the addition instruction and BORROW by the subtraction instruction are controlled by the same CY flag.

Example 5:
```
MOV  R1, #0000B ;
MOV  M1, #0000B ;
SUB  M1, #0001B ; ①
SUBC R1, M1     ; ②
```

At this time, ① and ② are calculated as follows.

```
① 0000B ... Contents of M1
 −0001B ... Immediate data
 -------
  1 1111B
  ~
  BORROW
② 0000B ... Contents of R1
  1111B ... Contents of M1
−    1  ... Contents of CY flag
 -------
  1 0000B
  ~
  BORROW
```

Thus, the arithmetic operation results become R1 = 0000B, M1 = 1111B, CY flag = 1 and Z flag = 1.

8.3.2 Addition/subtraction when CMP Flag = 1, BCD Flag = 1

The binary addition/subtraction is performed.

However, as the CMP flag is set (1), the arithmetic result is not stored in the general register or data memory.

If CARRY of BORROW is caused by the operation result, the CY flag is set (1); if not, the CY flag is reset (0).

The Z flag maintains the previous status if the operation result is 0000B; it is reset (0) if the result is not 0000B.

Example 1:

```
MOV   PSW, #1000B ;  Sets the CMP flag (writes in program status word).
MOV   R1,  #1111B ;
MOV   M1,  #1111B ;
ADD   R1,  M1     ;  ①
SUB   R1,  M1     ;  ②
MOV   PSW, #1010B ;  Sets the CMP and Z flags.
SUB   R1,  M1     ;  ③
```

At this time, ① is calculated as follows.

```
  1111B ... Contents of R1
+ 1111B ... Contents of M1
-------
1 1110B
~~
CARRY
```

However, as the CMP flag is set (1), the arithmetic operation result is not stored in R1.

As CARRY occurs, the CY flag is set (1).

As the operation result is not 0000B, the Z flag is reset.

As the contents of R1 and M1 in ② are the same as those in ①, the CY flag is reset (0). As the operation result is 0000B, the Z flag maintains its previous status 0.

Although ③ is operated in the same manner as ②, the Z flag is set (1) in advance. Therefore, the Z flag maintains 1. If the CMP flag is set (1), the arithmetic operation result is not stored changing only the CY and Z flag statuses. Therefore, this is convenient in comparing data of 5 bits or more.

```
Example 2:  MOV   PSW,  #1010B      ;  Sets (1) CMP and Z flags.
            SUB   M1,   #0001B (1H) ;  ①
            SUBC  M2,   #0010B (2H) ;  ②
            SUBC  M3,   #0011B (3H) ;  ③
```

At this time, as the CMP flag is set (1), the operation result is not stored. Thus, even if ①, ② and ③ are executed, the contents of M1, M2 and M3 do not change.

The Z flag is initially set (1). Thus, if the operation results of all ①, ② and ③ are 0000B, this flag keeps the initial setting (1). And, if even one of the operation results is not 0000B, this flag is reset (0).

The CY flag is set when the contents of the 12 bits of M3, M2 and M1 are smaller than 001100100001B (321H). Thus, the 12-bit data of M3, M2 and M1 can be compared with the 12-bit data of 321H by testing the Z flag and the CY flag at the point of terminating ①, ② and ③.

In other words,

If Z = 1, CY = 1: M3, M2, M1 = 321H
↑
Always becomes 0.
If Z = 0, CY = 0: M3, M2, M1 > 321H
If Z = 0, CY = 1: M3, M2, M1 < 321H

In example 2, by using instructions UB r, m and SUBC r, m, it is possible to compare the contents of the general register with that of the data memory.

8.3.3 Addition/subtraction when CMP Flag = 0, BCD Flag = 1

The decimal arithmetic operation is performed.

The operation result is stored in the general register or data memory. The CY flag is set (1) if the operation result is more than 1001B (9D) or less than 0000B (0D), and is reset (0) if 0000B (0D) to 1001B (9D).

The Z flag is set (1) if the operation result becomes 0000B (0D), and is reset (0) if other than 0000B (0D).

The decimal operation uses the method of decimally converting the once-binarily-operated result at the decimal adjustment circuit. For details of this binary-to-decimal conversion, please refer to Table 8-3 in **8.2.4 "Decimal Adjustment Circuit Function"**.

Therefore, to correctly execute the decimal operation, the following restrictions are necessary.

(1) The addition result shall be 0 to 19D.

(2) The subtraction result shall be 0 to 9D or −10D to −1.

0 to 19D are the values which have considered the CY flag. If they are expressed binarily, they become 0, 0000B (CY) to 1, 0011B (CY).

In the same manner, −10D to −1 become 1, 0110B (CY) to 1, 1111B (CY).

If the decimal operation is performed outside the restrictions above, the CY flag is set (1) and data of 1010B (0AH) or more is output as a result of the operation.

Example 1:
```
MOV   M1,   #0111B (7);
MOV   RPL,  #0001B   ; Sets the BCD flag (the BCD flag is allocated to b0 of RPL of the
                       system register).
MOV   PSW,  #0000B   ; Resets CMP, CY and Z flags.
ADD   M1,   #1001B (9); ① 7 + 9
SUB   M1,   #0111B (7); ② 6 − 7
```

At this time, ① is calculated as follows.

```
  0111B ... Contents of M1
+ 1001B ... Immediate data
1 0000B ... Binary addition result
CARRY
   ↓  Table 8-3 "Binary-to-Decimal Conversion" based adjustment.
1 0110B ... M1-stored data
CARRY
```

In other words, the CY flag is set thus storing 0110B (6) in M1. Considering that the CY flag is the weight of 10, it results in the decimal operation of 7 + 9 = 16.

② is calculated as follows.

```
  0110B ... Contents of M1
− 0111B ... Immediate data
1 1111B ... Binary subtraction result
BORROW
   ↓  Binary-to-decimal adjustment
1 1001B ... M1-stored data
```

This means that, as 6 is stored in M1 in ①, 6 − 7 has been performed. The result is 9 and the CY flag is set.

Example 2:

```
MOV   M1,   #0101B (5)  ;
MOV   M2,   #0110B (6)  ;
MOV   M3,   #0111B (7)  ;
MOV   RPL,  #0001B      ;  Sets (1) the BCD flag.
MOV   PSW,  #0000B      ;  Resets (0) CMP, ZY and Z flags.
SUB   M1,   #0111B (7)  ;  ①
SUBC  M2,   #0110B (6)  ;  ②
SUBC  M3,   #0101B (5)  ;  ③
```

At this time, ①, ② and ③ are calculated as follows.

```
①   0101B ... Contents of M1
  −0111B ... Immediate data
  1 1110B
  BORROW
      ↓  Binary → decimal
  1 1000B (8) ... M1-stored data
  BORROW

②   0110B ... Contents of M2
  −0110B ... Immediate data
  1 1111B ... CY flag
  BORROW
      ↓
  1 1001B (9) ... M2-stored data
  BORROW
③   0111B ... Contents of M3
  −0101B ... Immediate data
  0 0001B ... CY flag
  BORROW
      ↓
  0 0001B (1) ... M3-stored data
```

This means that immediate data 567 was subtracted from 765 stored in M3, M2 and M1, thus resulting in 198.

Example 3:
```
MOV   M1,   #1001B ;
MOV   RPL,  #0001B ;  Sets (1) the BCD flag.
MOV   PSW,  #000B  ;  Resets (0) CMP, CY and Z flags.
ADDC  M1,   #1010B ;  ①
ADDC  M1,   #1010B ;  ②
```

At this time, ① is calculated as follows.

```
  1001B (9)   ... Contents of M1
+ 1010B (10)  ... Immediate data
-------------
  1 0011B     ... CY flag
  ~
  CARRY
    ↓   Binary → decimal
  1 1001B     ... Operation result
  ~
  CARRY
```

This means that 9 + 10 = 9. And considering the CY flag, the results that the decimal operation of 9 + 10 = 19 has been performed. On the other hand, ② is calculated as follows.

```
  1001B (9)   ... Contents of M1
+ 1010B (10)  ... Immediate data
-------------
  1 0100B     ... CY flag
  ~
  CARRY
    ↓   Binary → decimal
  1 1110B     ... Operation result
  ~
  CARRY
```

In other words, as the CY flag is set (1), the operation result exceeds 19, thus leading to an incorrect decimal operation.

8.3.4 Addition/subtraction when CMP Flag = 1, BCD Flag = 1

The decimal arithmetic operation is performed.

The operation result is not stored in the general register or data memory.

In other words, operations when CMP flag = 1 and BCD flag = 1 are performed simultaneously.

```
Example:  MOV   RPL,  #0001B ;  Sets (1) the BCD flag.
          MOV   PSW,  #1010B ;  Sets (1) CMP and Z flags and resets (0) the CY flag.
          SUB   M1,   #0001B ;  ①
          SUBC  M2,   #0010B ;  ②
          SUBC  M3,   #0011B ;  ③
```

At this time, it is possible to decimally compare the contents of the 12 bits of M3, M2 and M1 with 321 of the immediate data by means of ①, ② and ③.

8.3.5 Precuations in Using Arithmetic Operation

When applying the arithmetic operation to the program status word, it is necessary to pay attention to the fact that the arithmetic operation result is stored in the program status word.

In other words, the CY and Z flags in the program status word are normally set/reset by the arithmetic operation result. However, if the arithmetic operation is performed on the program status word itself, the arithmetic operation result is stored thus making it impossible to determine CARRY, BORROW or ZERO.

However, when the CMP flag is set (1), the arithmetic operation result is not stored. Therefore, the CY and Z flags are set/reset as usual.

```
Example 1:  MOV   PSW,  #0110B
            ADD   PSW,  #1010B
```

At this time, it is calculated as follows.

```
  0110B  ... Contents of PSW
+ 1010B  ... Immediate data
-------
1 0000B
~~
CARRY
```

Therefore, although the CY and Z flags are supposed to be set, the operation result 0000B is stored in PSW because the CMP flag is "0".

```
Example 2:  MOV   PSW,  #1010B
            ADD   PSW,  #1000B
```

At this time, it is calculated as follows.

```
  1010B  ... Contents of PSW.
+ 1000B  ... Immediate data
-------
1 0010B
~~
CARRY
```

However, as the CMP flag is set (1), the operation result 0010B is not stored in PSW. Therefore, the CY flag of PSW is set (1) and then the Z flag is reset (0). In other words, 1100B is stored in PSW.

8.4 LOGICAL OPERATION

As shown in Fig. 8-4, the logical operation can use the logical OR, the logical AND and the logical XOR.

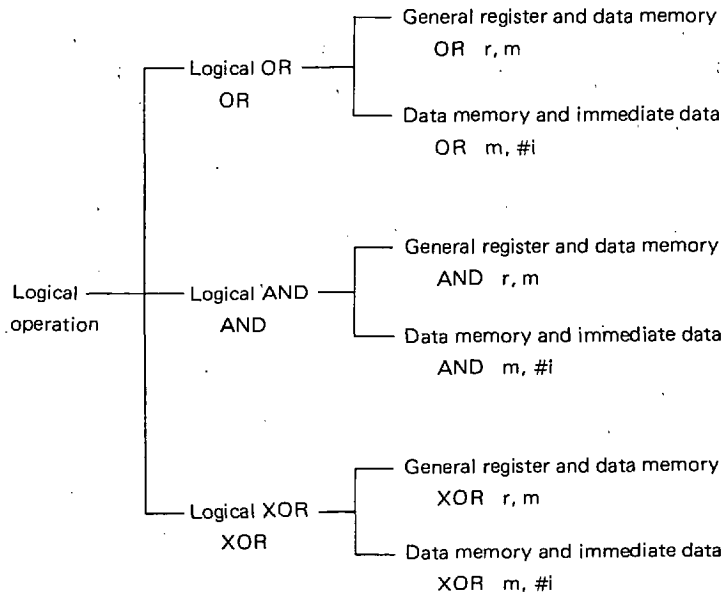
Logical operation instructions are divided into these three types, thus using the "OR", "AND" and "XOR" instructions respectively.

The "OR", "AND" and "XOR" instructions are further divided into the logical operation of the general register and data memory and that of the data memory and immediate data. This is decided by the value "r, m" or "m, #i" described in the operand of the instruction in the same manner as the arithmetic operation.

The logical operation is not affected by the BCD or CMP flag of the program status word. Neither does it affect the CY or Z flag. However, when the index enable flag is set (1), it is modified by the index register. For details of modification by the index register, please refer to **9 "System Register"**.

8.4.1 and **8.4.2** explain the logical OR, the logical AND and the logical XOR.

Fig. 8-4 Logical Operation



8.4.1 Logical OR

Table 8-4 shows the truth value table of the logical OR.

Table 8-4 Truth Value of Logical OR

A OR B = C		
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

The logical OR instruction performs the OR operation of 4-bit data in accordance with the truth value shown in Table 8-4.

```
Example: MOV  R1   #1010B ;
         MOV  M1,  #1001B ;
         OR   R1,  M1     ; ①
         OR   M1,  #1100B ; ②
```

At this time, ① is calculated as follows.

```
   1010B ... Contents of R1
OR 1001B ... Contents of M1
   1011B ... Operation result
```

Thus, 1011B is stored in R1.

2 is calculated as follows.

```
   1001B ... Contents of M1
OR 1100B ... Immediate data
   1101B
```

Thus, 1101B is stored in M1.

The logical OR is convenient if used when setting (1) the contents of data memory in 1-bit, 2-bit, 3-bit or 4-bit unit.

8.4.2 Logical AND

Table 8-5 is the truth value table of the logical AND.

Table 8-5 Truth Value of Logical AND

A AND B = C		
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

The logical AND instruction performs the AND operation of 4-bit data in accordance with the truth value shown in Table 8-5.

```
Example: MOV  R1,  #1010B ;
         MOV  M1,  #1001B ;
         OR   R1,  M1     ; ①
         OR   M1,  #1100B ; ②
```

At this time, ① is calculated as follows.

```
    1010B ... Contents of R1
AND 1001B ... Contents of M1
    ------
    1000B ... Operation result
```

Thus, 1000B is stored in R1.

2 is calculated as follows.

```
    1001B ... Contents of M1
AND 1100B ... Immediate data
    ------
    1000B
```

Thus, 1000B is stored in M1.

The logical AND is convenient if used when resetting (0) the contents of data memory in 1-bit, 2-bit, 3-bit or 4-bit unit.

8.4.3 Logical Exclusive OR

Table 8-6 is the truth value table of the logical exclusive OR.

Table 8-6 Truth Value of Logical Exclusive OR

A XOR B = C		
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

The logical exclusive OR instruction performs the XOR operation of 4-bit data in accordance with the truth value shown in Table 8-6.

```
Example: MOV  R1,  #1010B ;
         MOV  M1,  #1001B ;
         XOR  R1,  M1     ; ①
         XOR  M1,  #1100B ; ②
```

At this time, ① is calculated as follows.

```
    1010B ... Contents of R1
XOR 1001B ... Contents of M1
    0011B ... Operation result
```

Thus, 0011B is stored in R1.

② is calculated as follows.

```
    1001B ... Contents of M1
XOR 1100B ... Immediate data
    0101B
```

Thus, 0101B is stored in M1.

The logical exclusive OR is convenient if used when inverting the contents of data memory in 1-bit, 2-bit, 3-bit or 4-bit unit.

8.5 BIT DECISION

As shown in Fig. 8-5, the bit decision is divided into the true bit (1) decision and the false bit (0) decision.

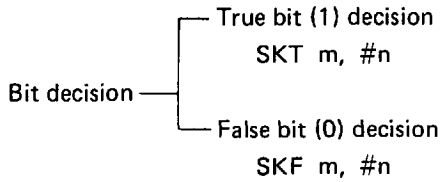
The true bit (1) decision and the false bit (0) decision use the "SKT" and "SKF" instructions respectively.

The "SKT" and "SKF" instructions can be issued only to the data memory.

The bit decision is not affected by the BCD flag of the program status word. Nor does it affect the CY flag and the Z flag. However, the CMP flag is reset (0) if the "SKT" or "SKF" instruction is executed. When the index enable flag has been set (1), it is modified by the index register. For details of modification by the index register, please refer to **9 "System Register (SYSREG)".**

8.5.1 and **8.5.2** describe the true bit (1) decision and the false bit (0) decision.

Fig. 8-5 Bit Decision Instruction



8.5.1 True Bit (1) Decision

The true bit (1) decision instruction "SKT m, #n" decides whether the bit specified by n among the 4 bits of the data memory is "true (1)". When all the bits specified by n are "true (1)", the instruction following this instruction is skipped.

```
Example: MOV  M1,  #1011B ;
         SKT  M1,  #1011B ;  ①
         BR   A
         BR   B
         SKT  M1,  #1101B ;  ②
         BR   C
         BR   D
```

At this time, bits b_3, b_1 and b_0 of M1 are decided in ① and, as all are true (1), are branched to B. In ②, bits b_3, b_2 and b_0 of M1 are decided. However, b_2 of M1 is branched to C as it is false (0).

8.5.2 False Bit (0) Decision

The false bit (0) decision instruction "SKF m, #n" decides whether the bit specified by n among the 4 bits of the data memory is false (0). If all the bits specified by n are "false (0)", the instruction following this instruction is skipped.

```
Example: MOV  M1,  #1001B ;
         SKF  M1,  #0110B ;  ①
         BR   A           ;
         BR   B           ;
         SKF  M1,  #1110B ;  ②
         BR   C           ;
         BR   D           ;
```

At this time, bits b_2 and b_1 of M1 are decided in ① and, as all are false (0), are branched to B. In ②, bits b_3, b_2 and b_0 of M1 are decided. However, b_3 of M1 is branched to C as it is true (1).

8.6 COMPARATIVE DECISION

As shown in Fig. 8-6, the comparative decision is divided into "equal", "unequal", "equal or more" and "less".

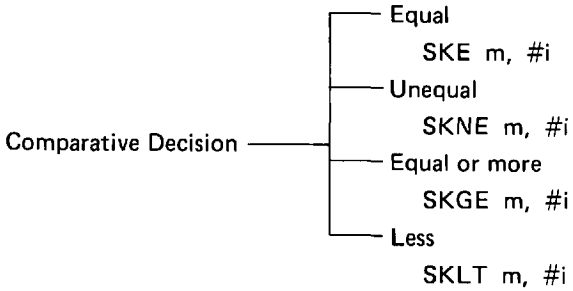
The "equal", "unequal", "equal or more" and "less" decisions use the "SKE", "SKNE", "SKGE" and "SKLT" instructions respectively.

The "SKE", "SKNE", "SKGE" and "SKLT" instructions can perform only the comparative decision between the data memory and the immediate data. The comparative decision between the general register and the data memory is made by the subtraction instruction by means of the CMP and Z flags of the program status word. (See **8.3 "Arithmetic Operation (Binary and Decimal Addition/Subtraction)".**

The comparative decision is not affected by the BCD and CMP flags of the program status word. Nor does it affect the CY and Z flags.

8.6.1 to **8.6.4** describe the "equal", "unequal", "equal or more" and "less" decisions.

Fig. 8-6 Comparative Decision Instruction



8.6.1 "Equal" Decision

The "equal" decision instruction "SKE m, #i" decides whether the contents of the data memory and that of the immediate data are "equal".

If the data memory and immediate data contents are equal to each other, the instruction following this instruction is skipped.

Example:
```
MOV  M1, #1010B
  :
  :
SKE  M1, #1010B ; ①
BR   A
BR   B
  :
  :
SKE  M1, #1000B ; ②
BR   C
BR   D
```

At this time, ① branches to B because the contents of M1 and 1010B of the immediate data are equal.
② branches to C because the contents of M1 and 1000B of the immediate data are not equal.

8.6.2 "Unequal" Decision

The "unequal" decision instruction "SKNE m, #i" decides whether the contents of the data memory and that of the immediate data are "unequal".

If the data memory and immediate data contents are unequal to each other, the instruction following this instruction is skipped.

Example:
```
MOV  M1, #1010B
  :
  :
SKNE M1, #1000B ; ①
BR   A
BR   B
  :
  :
SKNE M1, #1010B ; ②
BR   C
BR   C
```

At this time, ① branches to B because the contents of M1 and 1000B of the immediate data are unequal.
② branches to C because the contents of M1 and 1010B of the immediate data are equal.

8.6.3 "More" Decision

The "more" decision instruction "SKGE m, #i" compares the contents of the data memory and that of the immediate data and skips the following instruction if the contents of the data memory are "bigger" than or "equal" to that of the immediate data.

Example:
```
MOV  M1, #1000B
  :
  :
SKGE M1, #0111B ; ①
BR   A
BR   B
  :
  :
SKGE M1, #1000B ; ②
BR   C
BR   D
  :
  :
SKGE M1, #1001B ; ③
BR   E
BR   F
```

At this time, as the contents of M1 are 1000B, it is decided that ① is "bigger", ② is "equal" and ③ is "smaller" and each is branched to B, D and E respectively.

8.6.4 "Less" Decision

The "less" decision instruction "SKLT m, #i" compares the contents of the data memory and that of the immediate data and skips the following instruction if the contents of the data memory are "smaller" than that of the immediate data.

Example:
```
MOV  M1, #1000B
  :
  :
SKLT M1, #1001B ; ①
BR   A
BR   B
  :
  :
SKLT M1, #1000B ; ②
BR   C
BR   D
  :
  :
SKLT M1, #0111B ; ③
BR   E
BR   F
```

At this time, as the contents of M1 are 1000B, it is decided that ① is "smaller", ② is "equal" and ③ is "bigger" and each is branched to B, C and E respectively.

8.7 ROTATION PROCESSING

The rotation includes the right rotation and the left rotation.

The right rotation is processed with the "RORC" instruction. The left rotation is processed with the "ADDC instruction", which is an addition instruction.

The "RORC" instruction can be issued to the general register only.

The rotation by the "RORC" instruction is not affected by the BCD and CMP flags of the program status word. Nor does it affect the Z flag at all.

The rotation is described below.

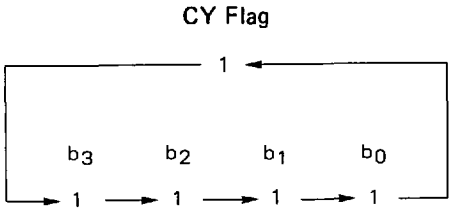
8.7.1 Right Rotation

The right rotation instruction "RORC r" rotates the contents of the general register by one bit in the lower bit directions.

At this time, the contents of the CY flag are written in the highest bit b_3 in the contents of the general register, and the CY flag is written in the lowest bit b_0 and the contents of the lowest bit b_0 are written in the CY flag.

```
Example 1: MOV   PSW,  #0100B ;  Sets (1) the CY flag.
           MOV   R1,   #1001B
           RORC  R1            ;  ①
```

At this time, it is processed as follows.



In other words, the rotation is made to right as in CY flag → b_3, b_3 → b_2, b_2 → b_1, b_1 → b_0, b_0 → CY flag.

```
2:  MOV   PSW,  #0000B ;  Resets (0) the CY flag.
    MOV   R1,   #1000B
    MOV   R2,   #0100B
    MOV   R3,   #0010B
    RORC  R1
    RORC  R2
    RORC  R3
```

At this time, the above program rotates the 12-bit data of R1, R2 and R3 to right.

8.7.2 Left Rotation

The left rotation is made by using the "ADDC r, m" instruction, which is an addition instruction.

```
Example: MOV   PSW,  #0000B ;  Reset (0) the CY flag.
         MOV   R1,   #1000B
         MOV   R2,   #0100B
         MOV   R3,   #0010B
         ADDC  R3,   R3
         ADDC  R2,   R2
         ADDC  R1,   R1
```

At this time, the above program rotates the 12-bit data of R1, R2 and R3 to left.

9. SYSTEM REGISTER (SYSREG)

The system register is the generic name for registers directly related to CPU control. They are located in addresses 74H to 7FH on the data memory.

The system register includes the following.

- Address register
- Window register
- Bank register
- Memory pointer enable flag
- Index register
- Data memory row address pointer
- General register pointer
- Program status word

Fig. 9-1 System Register Configurations

74H				75H				76H				77H				78H				79H			
AR3				AR2				AR1				AR0				WR				BANK			
b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0													0	0		

7AH				7BH				7CH				7DH				7EH				7FH			
MPH, IXH				MPL, IXM				IXL				RPH				RPL				PSW			
b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
MPE	0	0	0	0								0	0	0	0				BCD	CMP	CY	Z	IXE

9.1 ADDRESS REGISTER (AR)

This register specifies the program memory address and is located in 74H to 77H. Instructions which use the address register include the indirect branch instruction (BR @AR, CALL @AR), the table reference instruction (MOVT) and the stack operation instruction (PUSH, POP).

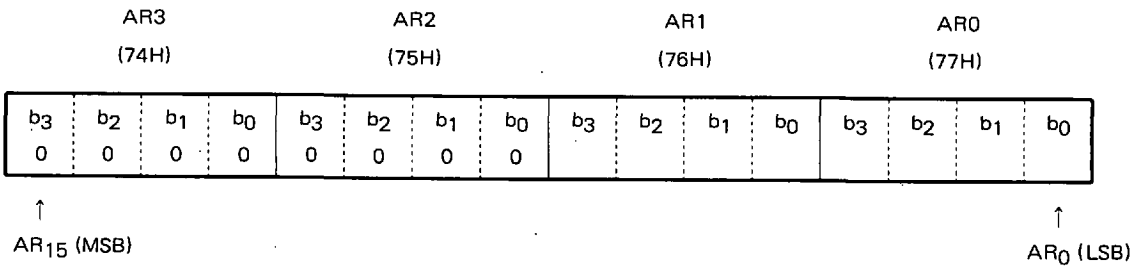
The indirect branch, which is to branch to the address of the program memory specified by the contents of the address register, includes "BR @AR" and "CALL @AR".

The table reference, which is performed with the "MOVT" instruction, transmits the contents of the program memory address specified by the address register to DBF (0DH to 0FH of BANK0) of the data memory.

The stack operation is performed by the "PUSH" or "POP" instruction. The "PUSH" instruction stores the contents of the address register in the stack specified by the stack pointer of the time and then −1 the contents of the stack pointer. The "POP" instruction +1 the contents of the stack pointer and then loads the contents of the stack specified by the stack pointer of the time onto the address register.

μPD17051 is fixed to AR3, and AR2 is fixed to 0. Thus, the program addresses that can be specified by the address register are the 256 words of 0000H to 00FFH.

Fig. 9-2 Address Register Configurations



9.2 WINDOW REGISTER (WR)

The window register is the 4-bit register mapped in address 78H of the system register and is used for data transmission with the register file (RF), which will be described later. Data operations of each register on the register file are all performed via the window register.

Data transmission between the window register and the register file is performed by the specialized "PEEK WR, rf" and "POKE rf, WR" instructions.

9.3 BANK REGISTER (BANK)

The bank register specifies the bank of the data memory.

The bank register becomes BANK0 when resetting. The upper 2 bits of address 79H are always "0".

The data memory, which is divided into four banks by the bank register, operates the data memory in the bank specified by the bank register when the data memory operation instruction is executed.

For example, to operate the data memory of BANK1 when the current bank register is BANK0, it is necessary to switch the bank to BANK1 with the bank register.

However, the system register located in 74H to 7FH of the data memory does not have the concept of the bank, thereby the same system register existing in 74H to 7FH in all the banks. This means that, whether instruction "MOV 78H, #0" is performed at BANK1 or instruction "MOV 78H, #0" is performed at BANK2, the result is 0 is written in address 78H, which is the system register. In other words, as far as operating the system register is concerned, the bank concept is not necessary.

If the interrupt is accepted, BANK is saved.

Table 9-1 Data Memory Bank Specification

Bank Register (BANK)				Data Memory Bank
b_3	b_2	b_1	b_0	
0	0	0	0	BANK0
0	0	0	1	BANK1
0	0	1	0	BANK2
0	0	1	1	BANK3

9.4 MEMORY POINTER ENABLE FLAG (MPE)

This specifies whether to specify the row address which is used to execute the "MOV @r, M" and "MOV M, @r" instructions with MPL or to execute with the same row address. If MPE is set, the row address is specified by MPL; if reset, the same row address is specified.

However, what is specified by MPL is the row address of the same bank currently specified.

9.5 INDEX REGISTER (IX) AND DATA MEMORY ROW ADDRESS POINTER (MP)

9.5.1 Index Register and Data Memory Row Address Pointer Configurations

As shown in Fig. 9-1, the index register consists of 7AH's lower 3 bits (IXH, 7BH and 7CH (IXM, IXL) of the system register — 11 bits in total — and is used for indirectly specifying the address of the data memory.

The data memory row address pointer consists of 7AH (MPH)'s lower 3 bits and 7BH (MPL) — 7 bits in total.

Thus, the upper 7 bits of the index register and the data memory row address pointer are shared by each other.

In μPD17051, the upper 4 bits of the index register, in other the upper 4 bits (7AH's b_2 to b_0, 7BH's b_3) of the data memory row address pointer, are fixed to "0".

9.5.2 Index Register and Data Memory Row Address Pointer Function

If the data memory operation instruction is executed when the index enable flag (IXE) is set (1), the index register OR-operates the bank and address of the data memory specified by the instruction with the contents of the index register and executes the instruction to the data memory specified by this operation result (called real address).

If the general register indirect transmission instruction ("MOV @r, m" and "MOV m, @r") is executed when the memory pointer enable flag is set (1), the data memory row address pointer executes the instruction with the bank and row address of the indirect address specified by the general register as the value of the data memory row address pointer.

Table 9-2 shows the modification of the data memory and the indirect address by the index register and the data memory row address pointer.

The index register and the data memory row address pointer can use all the data memories as objects of modification.

The following instructions do not become the objects of modification by the index register.

```
INC   AR
INC   IX
MOVT  DBF, @AR
PUSH  AR
POP   AR
PEEK  WR, rf
POKE  rf, WR
GET   DBF, p
PUT   p, DBF
BR    addr
BR    @AR
RORC  r
CALL  addr
CALL  @AR
RET
RETSK
RETI
EI
DI
STOP  0
HALT  h
NOP
```

Table 9-2 Data Memory Address Modification by Index Register and Data Memory Row Address Pointer

IXE	MPE	General Register Address Specified by R			Data Memory Address Specified by M			Indirect Transmission Address Specified by @R		
		Bank	Row Address	Column Address	Bank	Row Address	Column Address	Bank	Row Address	Column Address
		b_3 b_2 b_1 b_0	b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_2 b_1 b_0	b_3 b_2 b_1 b_0
0	0	(RP)		r	(BANK)		m	(BANK)	m_H	(R)
0	1		Same as above			Same as above		(MP)		(R)
1	0		Same as above		(BANK)		m	(BANK)	m_H	
					Logical OR			Logical OR		(R)
						(IX)		(IXH)	(IXM)	
1	1		Same as above			Same as above		(MP)		(R)

	Address-modified instruction group			
Addition/subtraction	ADD ADDC SUB SUBC	r	m	
			m, #i	
Comparison	AND OR XOR	r	m	
			m, #i	
Logic	SKE SKGE SKLT SKNE		m, #i	
Decision	SKT SKF		m, #n	
Transmission	LD ST	r	m	
	MOV		m, #i	
		@r	m	Indirect transmission address

M ; Data memory address
(M) ; Data memory address contents
m ; Data memory address excluding the bank
m_H ; Data memory row address
m_L ; Data memory column address
R ; General register address
(R) ; General register address contents
r ; General register column address
RP ; General register pointer
(RP) ; General register pointer contents

BANK ; Bank register
(BANK) ; Bank register contents
IX ; Index register
(IX) ; Index register contents
IXH ; Index register bits b_{10} to b_8
IXM ; Index register bits b_7 to b_4
IXL ; Index register bits b_3 to b_0
MP ; Data memory row address pointer
(MP) ; Data memory row address pointer contents

9.5.3 When MPE = 0 and IXE = 0 (without Data Memory Modification)

As shown in Table 9-2, the data memory address is not affected by the index register and the data memory row address pointer.

Example 1: If the general register in BANK0 is at row address 0:

```
ADD  03H,   11H
```

If the instruction above is executed, the general register 03H and data memory 11H contents are added together and the result is stored in the general register 03H (see **example 1 in Fig. 9-3**).

2: If the general register in BANK0 is at row address 0:

```
MOV  05H,   #8  ;  05H ← 8
MOV  @05H,  34H;  Register indirect transmission
```

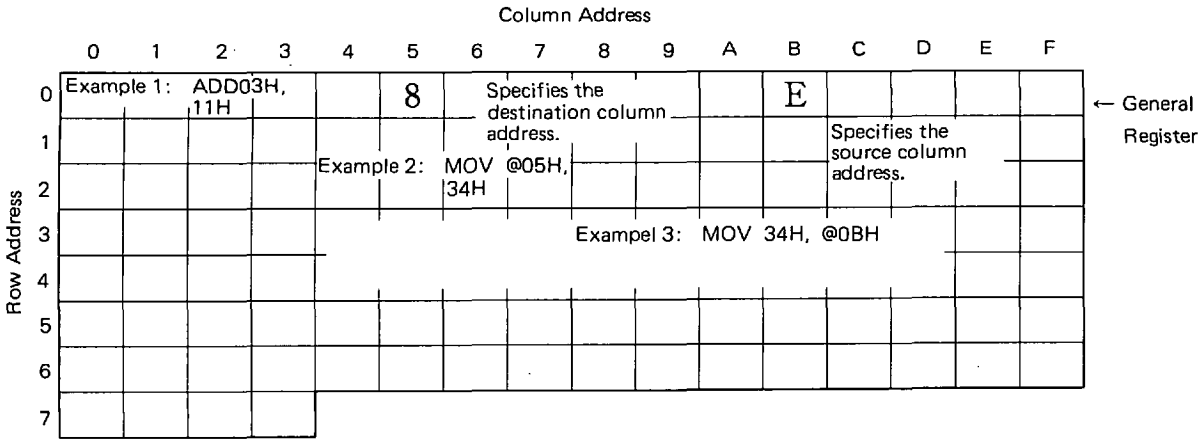
If the instructions above executed, the contents of data memory 34H are transmitted to address 38H. In other words, the "MOV @r, m" instruction transmits the contents of the data memory m to the address (38H above) whose row address (3 above) is the same as m and whose column address is specified by the contents (8 above) of the general register r (see **example 2 in Fig. 9-3**).

Example 3: If the general register in BANK0 is at row address 0:

```
MOV  0BH,   #0EH;  0BH ← 0EH
MOV  34H    @0BH;  Register indirect transmission
```

If the instructions above executed, the contents of data memory 3EH are transmitted to address 34H.
In other words, the "MOV m, @r" instruction transmits the contents of the address (3EH above) whose row address (3 above) is the same as the data memory m and whose column address is specified by the contents (0EH above) of the general register r to m (see **example 3 in Fig. 9-3**).
Compared with example 2, the source of the data memory address to be transmitted and the destination here have replaced each other.

Fig. 9-3 General Register Indirect Transmission when MPE = 0 and IXE = 0



Address Generation of Example 2

MOV @r, m
↓ 05H ↓ 34H

	Bank	Row Address	Column Address
R	0	0	5
M	0	3	4
(@r)	0	3	8
	Same as M		Content of R

9.5.4 When MPE = 1 and IXE = 0 (Diagonal Indirect Transmission)

As shown in Table 9-2, the bank and the row address of the indirect-side data memory address specified by the general register become the values of the data memory row address pointer only when the general register indirect transmission instruction has been issued.

Example 1: When general register in BANK0 is row address 0

```
MOV   MPL,   #0101B ;  MP ← 00101B
MOV   MPH,   #1000B ;  MPE ← 1
MOV   05H,   #8     ;  05H ← 8
MOV   @05H,  34H    ;  Register indirect transmission
```

When the instructions above are executed, the content of the data memory address 34H is transmitted to the data memory address 58H.

In other words, the "MOV @r, m" instruction when MPE = 1 transmits the contents of the data memory m to the data memory whose bank and row address are the values of the data memory row address pointer (BANK0 and row address 5 above) and whose column address is specified (BANK0's 58H above) by the contents (8 above) of the general register r (see **example 1 in Fig. 9-4**).

Compared with when MPE = 0 (**example 2 in 9.5.3**), the bank and the row address of the indirect-side data memory address specified by the general register can be specified by the data memory row address pointer here. (In **example 2 in 9.5.3**, the indirect-side bank and row address are the same as m.)

Thus, by specifying MPE = 1, it is possible to diagonally perform the general register indirect transmission.

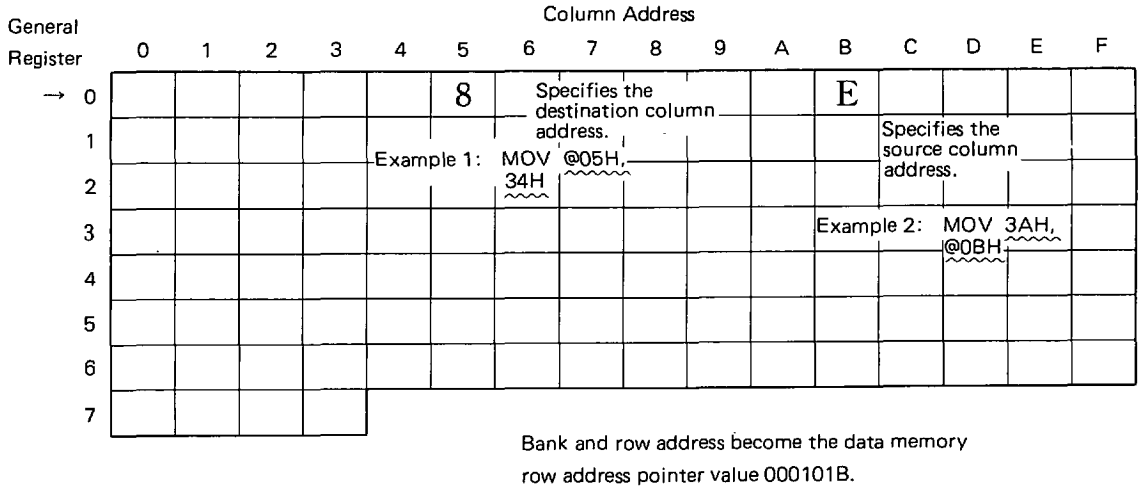
In the same manner, the "MOV m, @r" instruction becomes as shown in example 2 below.

Example 2: When general register in BANK0 is row address 0

```
MOV  MPL,  #0101B ;  MP ← 00101B
MOV  MPH,  #1000B ;  MPE ← 1
MOV  0BH,  #0EH   ;  0BH ← 0EH
MOV  3AH,  @05H
```

(See **example 2 in Fig. 9-4.**)

Fig. 9-4 General Register Indirect Transmission when MPE = 1 and IXE = 0



Address Generation of Example 1

MOV @r, m
↓ ↓
05H 34H
MP = 00101B

	Bank	Row Address	Column Address
R	0	0	5
M	0	3	4
(@r)	0000	101	8
	Value of MP		Content of R

9.5.5 When MPE = 0 and IXE = 1 (Index Modification)

As shown in Table 9-2, when the data memory operation instruction is issued, the bank and address of the data memory directly specified by the instruction is OR-operated with the index register and then the instruction is executed to the data memory address specified by this operation result (called real address).

Example 1: When general register in BANK0 is row address 0

```
MOV   IXL,   #0010B ;  IX ← 000000010B
MOV   IXM,   #0000B ;  MPE ← 0
MOV   IXH,   #0000B ;
OR    PSW,   #0001B ;  IXE ← 1
ADD   03H,   11H
```

If the instructions above are executed, the contents of the data memory address 13H and that of the general register address 03H are added together and then the result is stored in 03H of the general register.

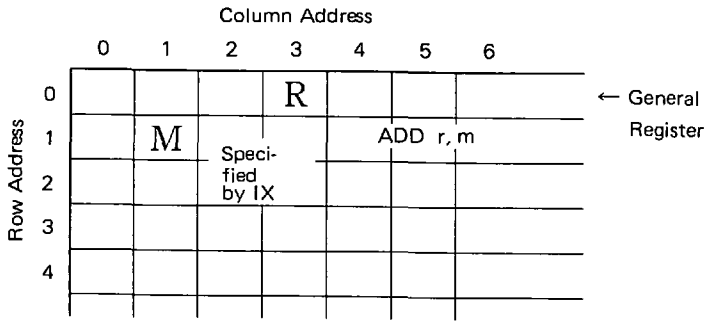
In other words, if the "ADD r, m" instruction is executed, the address (BANK0's 11H above) specified by m and the value (000000010B above) of the index register, and then the instruction is executed to the real address with this result as the real address (BANK0's 13H above) (see **Fig. 9-5**).

Compared with when IXE = 0 (**example 1 in 9.5.3**), the address of the data memory directly specified by the instruction is modified (OR-operation) by the index register here.

Example 2: Clears all bank data memories to 0.

```
        MOV   IXL,   #0      ;
        MOV   IXM,   #0      ;  IX ← 0
        MOV   IXH,   #0      ;
LOOP :
        OR    PSW,   #0001B  ;  IXE ← 1
        MOV   00H,   #0      ;  Clears the data memory specified by IX to 0.
        INC   IX             ;  IX ← IX + 1
        AND   PSW,   #1110B  ;  IXE ← 0: IXE is address 7FH, therefore not modified by
                                IX.
        SKT   IXM,   #0111B  ;  Row address 7?
        BR    LOOP           ;  If not 7, LOOP.
        ADD   IXM,   #1      ;  Specify the next bank without clearing row address 7.
        ADDC  IXH,   #0      ;
        SKF   IXM,   #1000B  ;  Cleared up to BANK2?
        SKT   IXH,   #0001B  ;
        BR    LOOP           ;  If not cleared, LOOP.
```

Fig. 9-5 Data Memory Address Modification when IXE = 1



9.6 GENERAL REGISTER POINTER (RP)

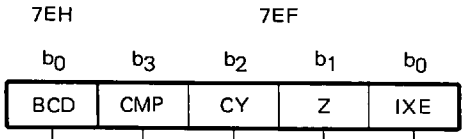
This pointer specifies the bank and the row address of the general register.

However, as RPH is fixed in μPD17052, only RPL (3 bits) can be specified. In other words, 0 to 7 can be specified as the register pointer. Thus, in μPD17052, the row address of the general register can be specified freely in BANK0.

9.7 PROGRAM STATUS WORD (PSWORD)

The program status word is configured with the flag showing the state of the arithmetic operation result by ALU in CPU and the 5-bit flag modifying the ALU function. PSWORD includes the BCD (binary coded decimal) flag, the CMP (compare) flag, the CY (carry) flag, the Z (zero) flag and the IXE (index enable) flag. The functions of these flags are shown in Fig. 9-6.

Fig. 9-6 PSWORD Configurations



Index enable flag	If this flag is set, the index modification is made possible.
Zero flag	This flag is reset when the arithmetic operation result is other than "0". Conditions for setting differ depending on the content of the CMP flag. (1) When CMP = 0 Set when the arithmetic operation result is "0". (2) When CMP = 1 Set when the result of the arithmetic operation executed when Z = 1 is "0".
Carry flag	This is set by the occurrence of CARRY when executing the addition instruction or by the occurrence of BORROW when executing the subtraction instruction. It is reset when either of these does not occur. Also, this flag is set when the lowest bit of the general register for which the RORC instruction is executed is "1". It is reset when this bit is "0".
Compare flag	When this flag is set, the result of the arithmetic operation is not stored in the data memory. The CMP flag is automatically reset when the SKT and SKF instructions are executed.
BCD flag	If this flag is set, all the arithmetic operations are performed decimally. If this is reset, they are performed binarily.

10. REGISTER FILE (RF)

The register file is a collection of registers mainly for controlling CPU's peripheral circuits. It has the capacity of 128 words x 4 bits. However, although the peripheral circuit addresses are actually allocated (control register file) in the upper 64 nibbles (00H to 3FH), addresses 40H to 7FH of the currently selected bank of the data memory are allocated in the lower 64 nibbles (40H to 7FH).

In other words, 40H to 7FH of each bank of the data memory belong to both the data memory address space and the register file address space.

On the assembler, the controller register file is allocated to 80H to BFH.

Fig. 10-1 Control Register File (1/2)

	0	1	2	3	4	5	6	7
0 (8)	0 0 IDCDMAEN 0	SP						0 0 0 CE
1 (9)		HSCGT3 (0) HSCGT2 (0) HSCGT1 HSCGT0	HSCGOPN 0 0 0			RMCSTAT3 (0) RMCSTAT2 RMCSTAT1 RMCSTAT0		0 0 0 TMCY
2 (A)		ADCCH2 ADCCH1 ADCCH0 ADCCMP						0 0 0 P1CGIO
3 (B)	0 0 0 CROMBNK	0 0 0 IDCEN			P2AI/O: P2ABIO3 P2ABIO2 P2ABIO1 P2ABIO0	P1BI/O: P1BBIO3 P1BBIO2 P1BBIO1 P1BBIO0	P0BI/O: P0BBIO3 P0BBIO2 P0BBIO1 P0BBIO0	P0AI/O: P0ABIO3 P0ABIO2 P0ABIO1 P0ABIO0

Remarks: (0) is always "0".

Fig. 10-1 Control Register File (2/2)

	8				9				A	B	C	D	E	F			
0 (8)	Serial Mode Register				TRMODE									0	INTVSYN	0	INT
	SIOCH	SB	SIOMS	SIOTX	ZCROSS	TMMD2	TMMD1	TMMD0									
1 (9)	WAIT													0	IEGVSYN	0	IEG
	SBACK	SIONWT	SIOWRQ1	SIOWRQ0													
2 (A)	STAT													IPSIO	IPVSYN	IPTM	IP
	SIOSF8	SIOSF9	SBSTT(0)	SBBSY													
3 (B)	SINT				SFC									IRQSIO	IRQVSYN	IRQTM	IRQ
	SIOIMD3(0)	SIOIMD2(0)	SIOIMD1	SIOIMD0	SIOCK3(0)	SIOCK2(0)	SIOCK1	SIOCK0									

Remarks: (0) is always "0".

10.1 IDCDMAEN (00H, b_1)

This flag is set to operate the IDC.

If the IDCDMAEN flag is set, the system is changed to the DMA mode, thus making it possible to use IDC. In the DMA mode, the dummy instruction cycle is 12 μs. For details, please refer to **20 "IDC"**.

00H

0	0	IDCDMAEN	0

0	DMA inhibit mode (instruction cycle = 2 μs)
1	DMA mode (instruction cycle = 12 μs)

10.2 SP (01H)

This pointer addresses the stack register.

01H

0	x	x	x

SP (Stack Pointer)

000B Setting Prohibited
001B Level 6
010B Level 5
011B Level 4
100B Level 3
101B Level 2
110B Level 1
111B When Resetting

10.3 CE (07H, b_0)

This flag reads the CE pin level.

When the high level has been input in the CE pin, this flag is "1". When the low level has been input, it becomes "0".

07H

0	0	0	CE

0	CE pin low level
1	CE pin high level

10.4 SERIAL INTERFACE MODE REGISTER (08H)

08H

SIOCH	SB	SIOMS	SIOTX

Setting Transmit/Receive

0	Two-wire bus mode	: RX (receive) mode
	CH0 serial I/O mode	: SI mode
	CH1 serial I/O mode	: Uses $P0A_3$ as general-purpose port.
1	Two-wire bus mode	: TX (transmit) mode
	CH0 serial I/O mode	: SO mode
	CH1 serial I/O mode	: Uses $P0A_3$ as SO pin.

Setting Clock Direction of Serial Interface

0	Two-wire bus mode	: Slave operation
	Serial I/O mode	: External clock operation
1	Two-wire bus mode	: Master operation
	Serial I/O mode	: Internal clock operation

Serial Interface Mode Setting

0	Serial I/O mode
1	Two-wire bus mode

Serial Interface Channel Setting

0	Selects CH0.
1	Selects CH1.

10.5 TMMODE (09H)

09H

ZCROSS	TMMD2	TMMD1	TMDD0

Time Base Setting

			TIMER INT		TIMER CARRY	
0	0	0	5 ms	Internal	100 ms	Internal
0	0	1	100 ms	Internal	5 ms	Internal
0	1	0	20 ms	Internal	100 ms	Internal
0	1	1	20 ms	Internal	5 ms	Internal
1	0	0	5 ms	Internal	$5/f_{TMR}$ S	External
1	0	1	$5/f_{TMR}$ S	External	5 ms	Internal
1	1	0	5 ms	Internal	$6/f_{TMR}$ S	External
1	1	1	$6/f_{TMR}$ S	External	5 ms	Internal

Zero-Cross Setting

0	Zero-cross OFF
1	Zero-cross ON

10.6 INTVSYN (0FH, b_2)

This flag is for reading the level of the vertical synchronous signal. It is set (1) when the signal input in the $\overline{V_{SYNC}}$ pin is at high level and is reset (0) when this signal is at low level.

10.7 INT (0FH, b_0)

This flag is for reading the status of the RMC pin.

It becomes "1" when the high level is input in the RMC pin and becomes "0" when the low level is input.

0FH

0	INTVSYN	0	INT

0	RMC pin is at low level.
1	RMC pin is at high level.

0	$\overline{V_{SYNC}}$ pin is at low level.
1	$\overline{V_{SYNC}}$ pin is in the high level period.

10.8 HORIZONTAL SYNCHRONOUS SIGNAL COUNTER CONTROL (11H, 12H)

11H

HSCGT3	HSCGT2	HSCGT1	HSCGT0

Hsync Counter Setting

0	0	Gate close
0	1	Gate open
1	0	Gate open (for 1.69 ms)
1	1	Setting prohibited

Either is fixed to "0".

12H

HSCGOPN	0	0	0

Gate Open/Close Confirmation Input of Horizontal Synchronous Signal Counter

0	Gate close
1	Gate open

10.9 SETTING THE PULSE WIDTH FOR ACCEPTING THE RMC PIN (15H)

15H

RMCSTAT3	RMCSTAT2	RMCSTAT1	RMCSTAT0

Setting the Pulse Width for Accepting the RMC pin

0	0	0	Edge (without noise cancellation)
0	0	1	200 μs
0	1	0	400 μs
0	1	1	2 ms
1	0	0	4 ms

Fixed to "0".

10.10 TIMER CARRY (17H)

17H

0	0	0	TMCY

Read-Special Flag for Timer Carry
This is set by the selected time base, and is reset by reading.

10.11 SERIAL INTERFACE WAIT CONTROL (18H)

18H

SBACK	SIONWT	SIOWRQ1	SIOWRQ0

Setting the Timing of Waiting

		Two-Wire Bus Mode	Serial I/O Mode
0	0	Does not wait.	Does not wait.
0	1	Waits with the fall of the clock when the clock counter contents are 8.	Waits if the clock counter contents become "8".
1	0	Waits with the fall of the clock when the clock counter contents are 9.	Waits if the clock counter contents become "9".
1	1	Waits with the fall of the clock when the clock counter contents have become 8 after detecting the start condition.	Setting prohibited

Setting Wait

0	Forced wait
1	Wait release

Acknowledge when Two-Wire Bus Mode

10.12 IEG (1FH)

This flag is for selecting the interrupt detection edge of the RMC pin and the V_{SYNC} pin.

When this flag is set to "0", the interrupt is applied at the rising edge. When it is set to "1", the interrupt is applied at the falling edge.

1FH

0	IEGVSYN	0	IEG

0	Interrupt at the rising edge of the RMC pin
1	Interrupt at the falling edge of the RMC pin

0	Interrupt at the rising edge of the $\overline{V_{SYNC}}$ pin
1	Interrupt at the falling edge of the $\overline{V_{SYNC}}$ pin

10.13 A/D CONVERTER CONTROL (21H)

21H

ADCCH2	ADCCH1	ADCCH0	ADCCMP

The result of comparison between V_{IN} and V_{REF} at the time of A/D conversion is input

0	$V_{IN} < V_{REF}$
1	$V_{IN} \geqq V_{REF}$

A/D Converter Input Channel Select

0	0	0	ADC_0 select
0	0	1	ADC_1 Select Shared by $P0D_0$.
0	1	0	ADC_2 Select Shared by $P0D_1$.
0	1	1	ADC_3 select Shared by $P0D_2$.
1	0	0	ADC_4 Select Shared by $P0D_3$.
1	0	1	ADC_5 Select Shared by $P1C_0$.
1	1	0	ADC_6 Select Shared by $P1C_1$.
1	1	1	ADC_7 select Shared by $P1C_2$.

10.14 SETTING PORT 1C INPUT/OUTPUT (27H)

27H

0	0	0	P1CGI0

P1C Port Input/Output Setting

0	P1C input port
1	P1C output port

10.15 SERIAL I/O STATUS REGISTER (28H)

28H

SIOSF8	SIOSF9	SBSTT	SBBSY

Detects BUSY condition.

0	Detects STOP condition.
1	Detects START condition.

Detects START condition.

0	Reset if the contents of the clock counter are 9.
1	Detects START condition.

Detects clock 9.

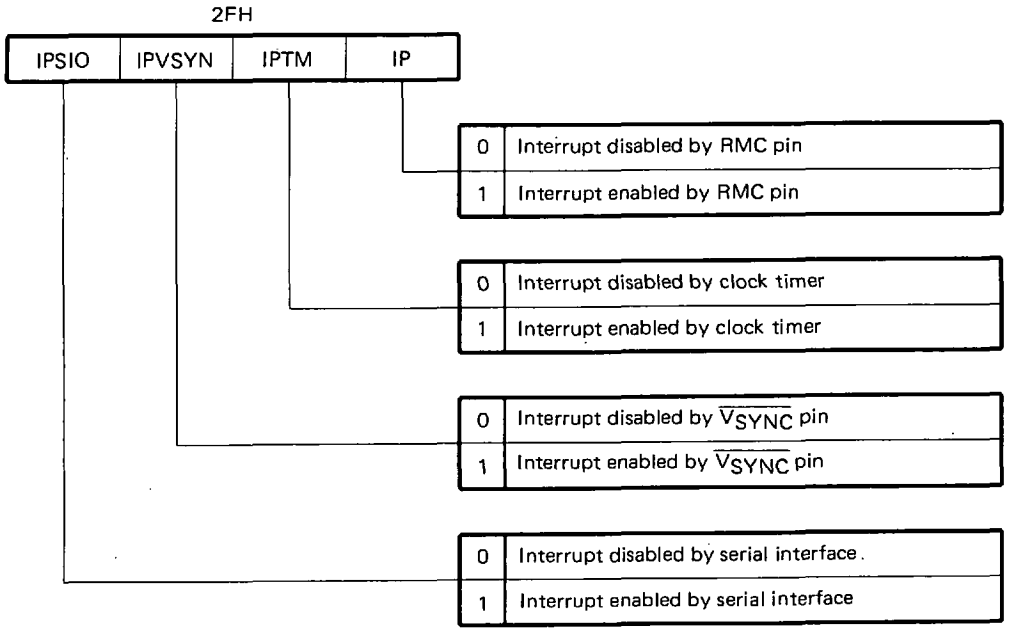
0	Reset if the contents of the clock counter are 0 or 1.
1	Set if the contents of the clock counter are 9.

Detects clock 8.

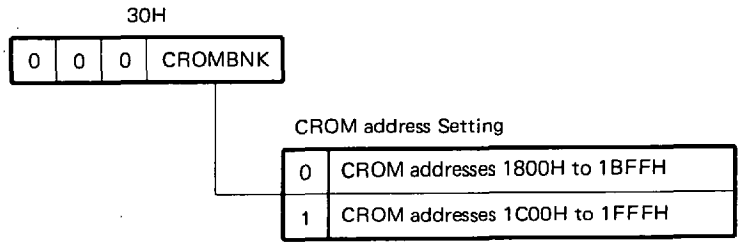
0	Resets if the contents of the clock counter are 0 or 1.
1	Sets if the contents of the clock counter are 8.

10.16 INTERRUPT ENABLE FLAG (2FH)

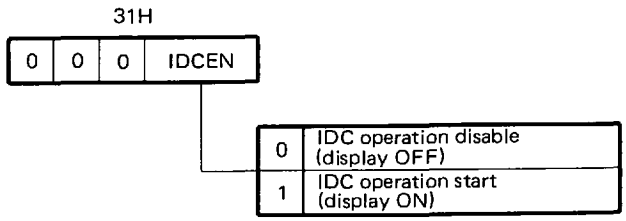
This flag is for enabling the interrupt of each interrupt factor. If this is set to "1", the interrupt is enabled. If "0", the interrupt is disabled.



10.17 CROM BANK SELECTION (30H, b_0)

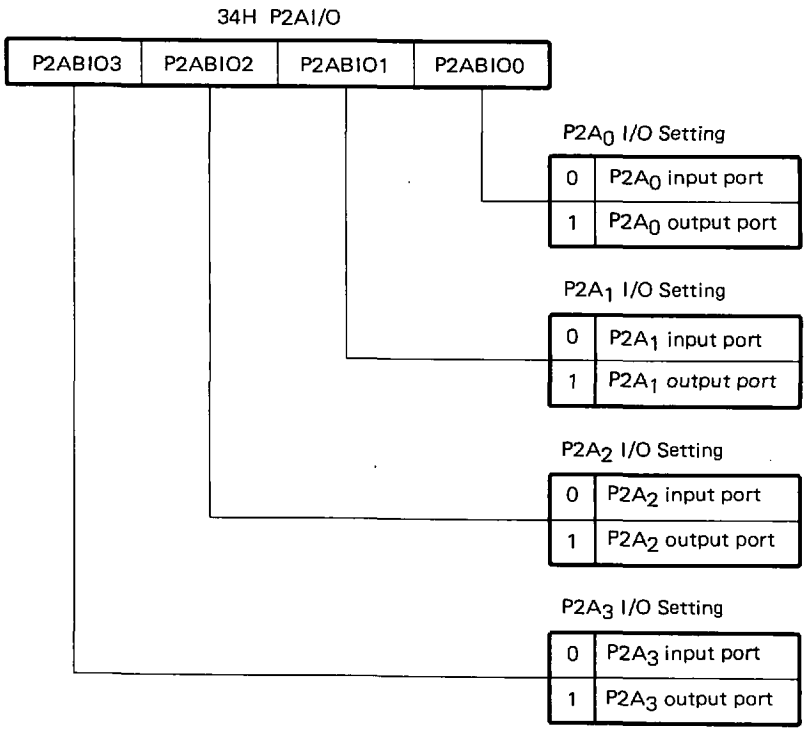


10.18 IDCEN (31H, b_0)



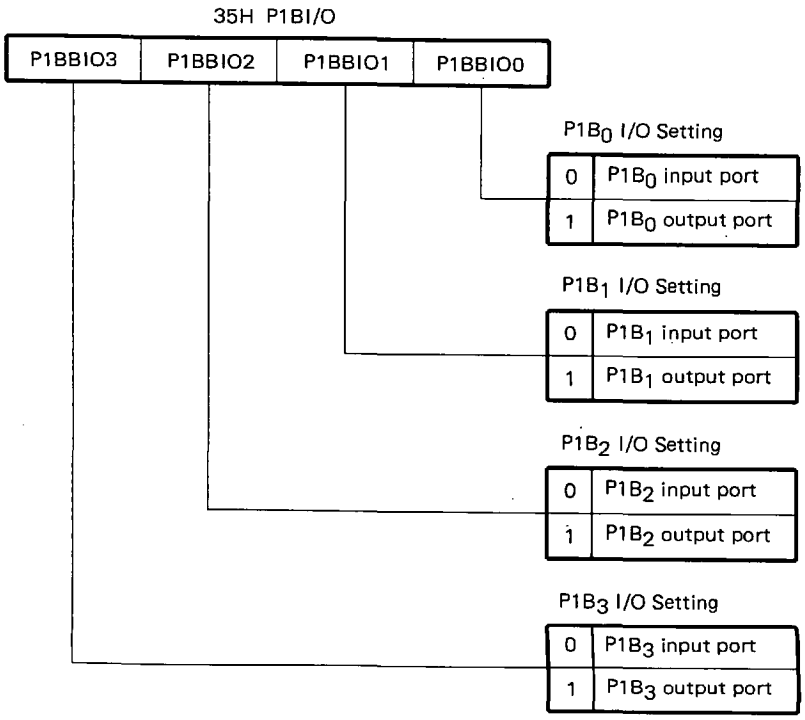
10.19 P2ABIOn (34H)

This specifies input/output of PORT 2A. If this is "0", the input is specified. If "1", the output is specified.



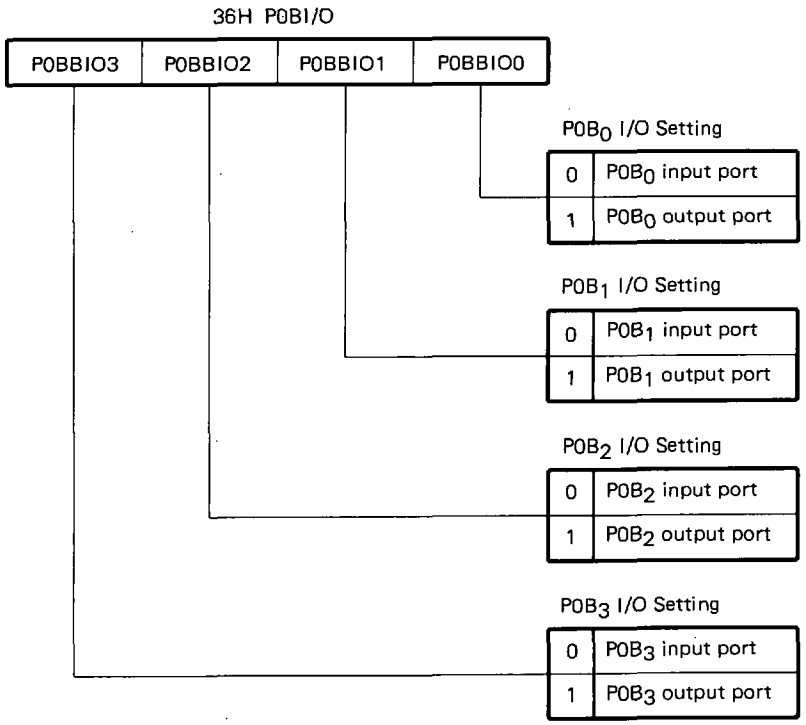
10.20 P1BBIOn (35H)

This specifies the input/output of PORT 1B. If this is "0", the input is specified. If "1", the output is specified.



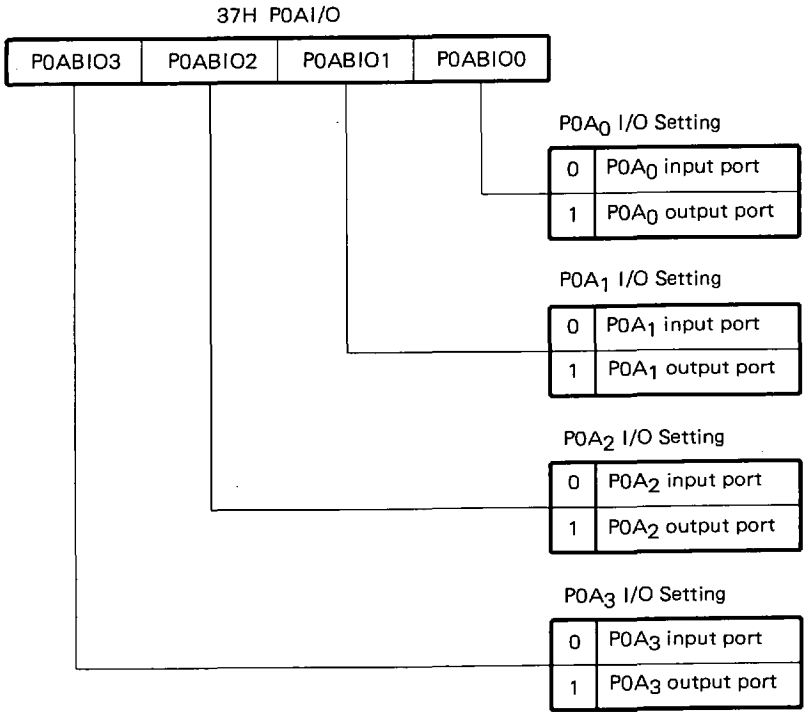
10.21 P0BBIOn (36H)

This specifies the input/output of PORT 0B. If this is "0", the input is specified. If "1", the output is specified.



10.22 P0ABIOn (37H)

This specifies the input/output of PORT 0A. If this is "0", the input is specified. If "1", the output is specified.



10.23 SETTING THE INTERRUPT REQUEST GENERATION TIMING IN SERIAL INTERFACE MODE (38H)

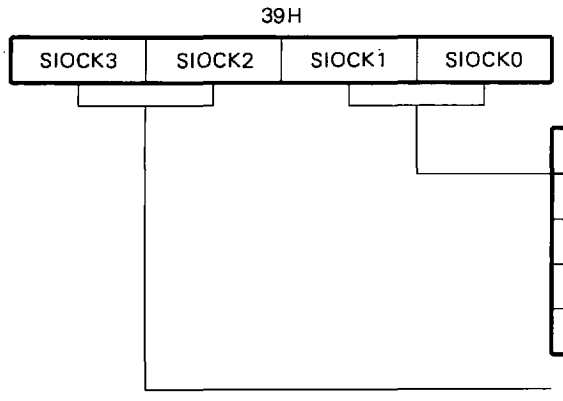
38H

SIOIMD3	SIOIMD2	SIOIMD1	SIOIMD0

Fixed to "0".

SIOIMD1	SIOIMD0	
0	0	Interrupt request occurs at the rise of the 7th bit of the shift clock.
0	1	Interrupt request occurs at the rise of the 8th bit of the shift clock.
1	0	Interrupt request occurs at the rising edge of the 7th bit of the shift clock immediately after the start condition has been detected.
1	1	Interrupt request occurs if the stop condition has been detected.

10.24 SETTING THE SHIFT CLOCK FREQUENCY (39H)



		Internal Clock Frequency
0	0	100 kHz
0	1	200 kHz
1	0	500 kHz
1	1	1 MHz

10.25 IRQ (3FH)

This is the interrupt request flag which shows the interrupt request status.

If the interrupt request occurs, this flag is set to "1". Then, if the interrupt is accepted (interrupt gets started), the interrupt request flag is reset to "0".

The interrupt request flag can be read/written with the program. Therefore, if "1" is written, the interrupt can be made to occur by the software. And, the interrupt hold status can be released by writing "0". The IRQ flag becomes "0" when resetting.

Flag Name	Bit Position	Interrupt Factor
IRQ	b_0	RMC pin
IRQTM	b_1	Clock timer
IRQVSYN	b_2	$\overline{V_{SYNC}}$ pin
IRQSIO	b_3	Serial interface

11. DATA BUFFER (DBF)

The data buffer is used for data transmission with peripheral hardware and data reading for table reference.

11.1 DATA BUFFER CONFIGURATIONS

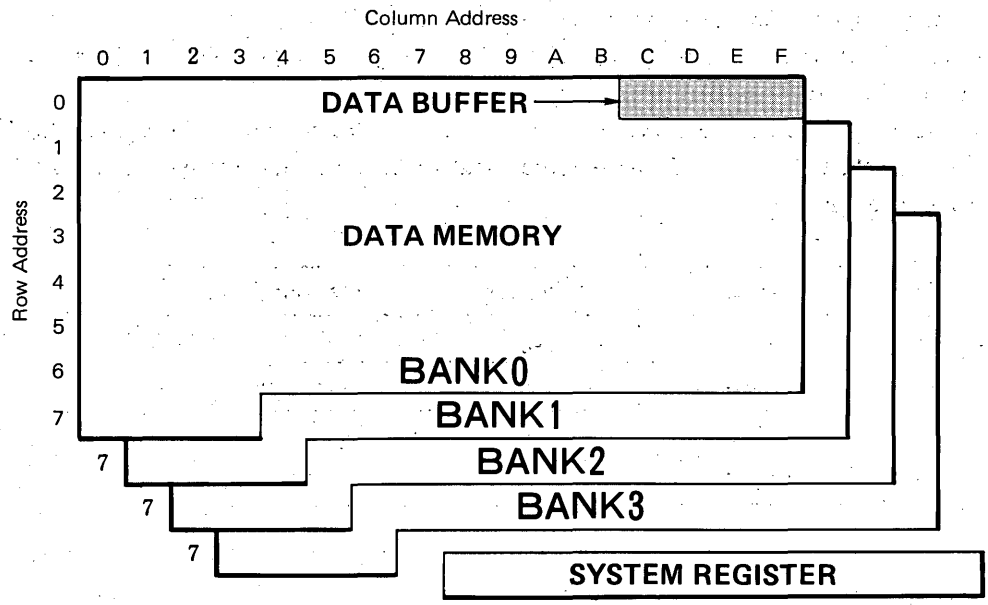
11.1.1 Data Buffer Location on Data Memory

Fig. 11-1 shows the location of the data buffer on the data memory.

As shown in Fig. 11-1, the data buffer is allocated to addresses 0CH to 0FH of BANK0 on the data memory and is configured with 4 words x 4 bits — 16 bits in total.

As the data buffer is located on the data memory, it can be operated by all the data memory operation instructions.

Fig. 11-1 Data Buffer Location



11.1.2 Data Buffer Configurations

Fig. 11-2 shows the data buffer configurations.

As shown in Fig. 11-2, the data buffer is consisted of 16 bits in total – with bit b0 of the data memory address 0FH as LSB and bit b_3 of the address 0CH as MSB.

Fig. 11-2 Data Buffer Configurations

Data Memory	Address	0CH				0DH				0EH				0FH			
	Bit	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Data buffer	Bit	b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
	Symbol	DBF3				DBF2				DBF1				DBF0			
	Data	MSB															LSB
		Data															

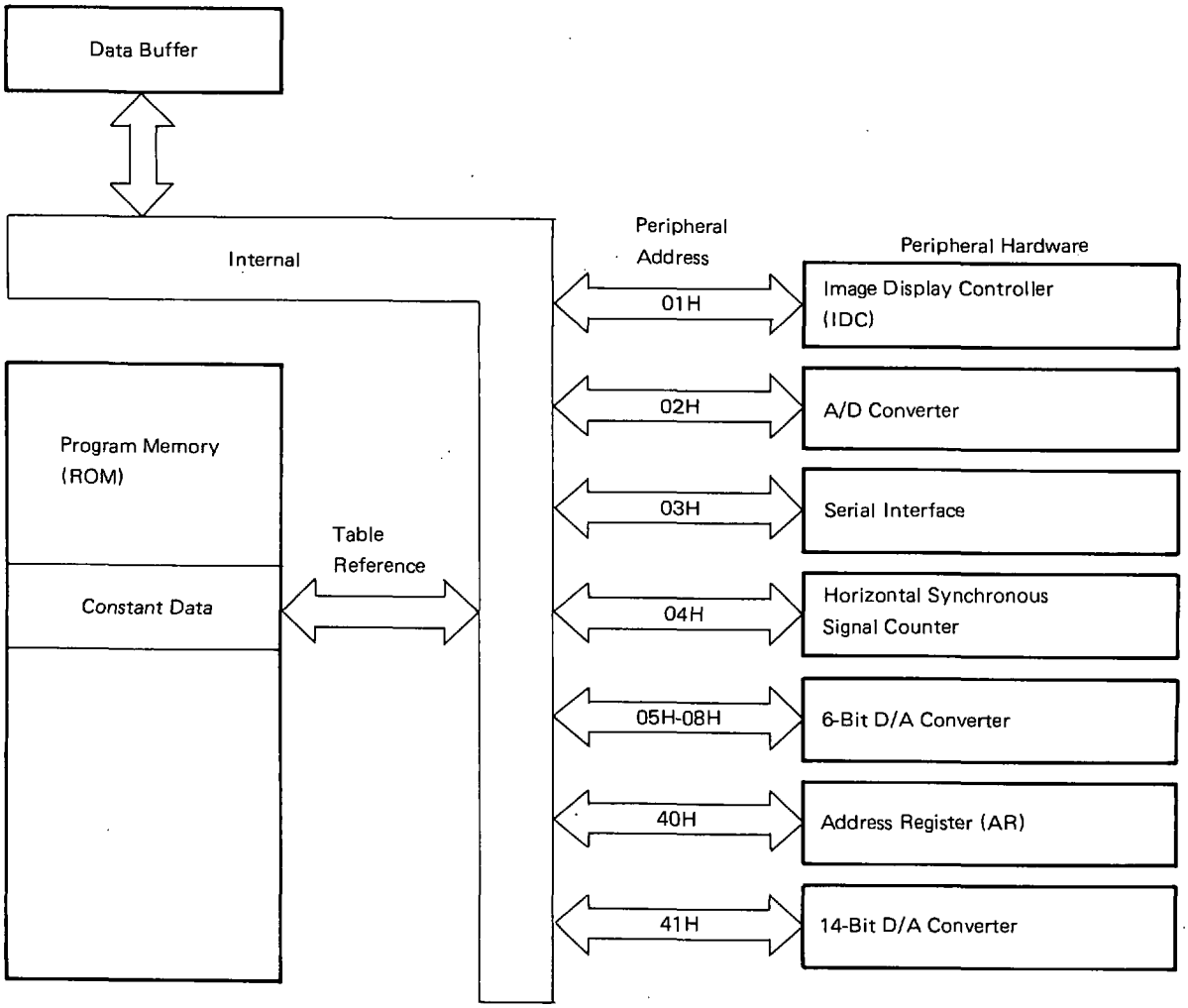
11.2 DATA BUFFER FUNCTION

The data buffer has the following two functions.

(1) Constant data read (table reference) function on the program memory
(2) Data transmission function with peripheral hardware

Fig. 11-3 shows the relationship among data buffer, peripheral hardware and table reference.
Section **11.3** explains the table reference, and sections **11.4** to **11.6** explain the peripheral hardware.

Fig. 11-3 Relationship among Data Buffer, Peripheral Hardware and Table Reference



11.3 DATA BUFFER AND TABLE REFERENCE

11.3.1 Table Reference Operation

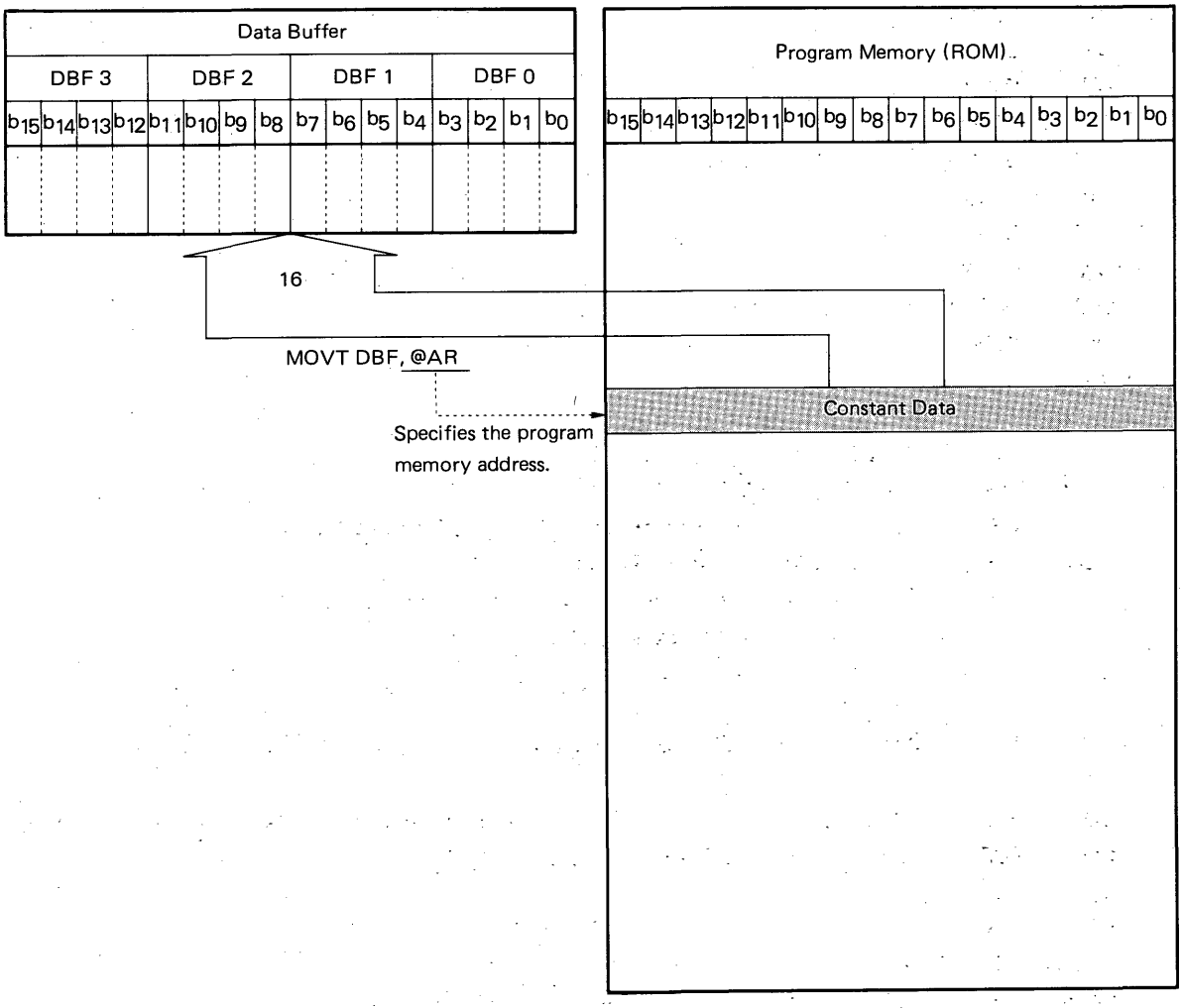
The table reference can read the constant data on the program memory onto the data buffer by means of the "MOVT DBF, @AR" instruction.

Therefore, if, for example, the constant data is written beforehand on the program memory and reference is made to the table when necessary, it becomes unnecessary to prepare the complicated data conversion program.

The "MOVT" instruction is explained below.

And, the program example is shown in **11.3.2.**

MOVT DBF, @AR; As shown in the diagram below, the contents of the program memory address-specified by the contents of the address register are read to the data buffer.



The stack is used 1 level when executing the table reference instruction.

As the valid bits of the address register (AR) are 8 bits, the program memory addresses which can be table-referenced become 256 words of addresses 0000H to 00FFH.

Please refer to **5 "Stack"** and **9.3 "Address Register (AR)"** as well.

11.3.2 Table Reference Program Example

The table reference program example is shown below.

Example:

```
    P0A     MEM     0.70H   ;
    P0B     MEM     0.71H   ;
    P0C     MEM     0.72H   ;
    ORG     0000H
START:
    BR      MAIN
DATA:
    DW      0001H           ;  Constant data
    DW      0002H           ;
    DW      0004H           ;
    DW      0008H           ;
    DW      0010H           ;
    DW      0020H           ;
    DW      0040H           ;
    DW      0080H           ;
    DW      0100H           ;
    DW      0200H           ;
    DW      0400H           ;
    DW      0800H           ;
MAIN:
    BANK0                   ;  Intrinsic macro
    SET4 P0ABIO3, P0ABIO2, P0ABIO1, P0ABIO0
    SET4 P0BBIO3, P0BBIO2, P0BBIO1, P0BBIO0
    MOV     RPL,    #1110B ;  Sets the general register to BANK0's row address 7H.
    MOV     AR1,    #(.DL.DATA SHR 4 AND 0FH)
    MOV     AR0,    #(.DL.DATA SHR 0 AND 0FH)
                            ;  Sets the address register to 0001H.
LOOP:
;   ①
    MOVT    DBF,    @AR     ;  Transmits the ROM value specified by the AR content to the data buffer.
;   ②
    LD      P0A,    DBF2    ;  Transmits the data buffer value to each port data register of Port 0A (70H),
    LD      P0B,    DBF1    ;  Port 0B (71H) and Port 0C (72H).
    LD      P0C,    DBF0    ;
    ADD     AR0,    #1      ;  Increments the address register content by 1.
    ADDC    AR1,    #0
    SKNE    AR0,    #0CH    ;  Writes 0 in AR0 when the AR0 value has become 0CH.
    MOV     AR0,    #0      ;
    BR      LOOP
```

When this program is executed, the constant data stored in addresses 0001H to 000CH of the program memory is written in the data buffer sequentially with ① and then output to Port 0A, Port 0B and Port 0C with ②.

At this time, the constant data outputs the high level sequentially to each pin of Port 0A, Port 0B and Port 0C as a result of the value shifting to left by 1 bit being stored.

11.4 DATA BUFFER AND PERIPHERAL HARDWARE

11.4.1 Peripheral Hardware Control Method

The peripheral hardware performing data transmission via the data buffer is shown below.

- Image display controller
- A/D converter
- Serial interface
- Horizontal synchronous signal counter
- 6-bit D/A converter
- Address register
- 14-bit D/A converter

The peripheral hardware is controlled by setting or reading data in these peripheral hardware devices via the data buffer.

Each of the peripheral hardware devices has the register (called peripheral register) for data transmission. And these peripheral registers are allocated with the address (called peripheral address) respectively.

By issuing the specialized "GET" or "PUT" instruction to these registers, data transmission can be made between the data buffer and each peripheral hardware device.

The "GET" and "PUT" instructions are explained below. And Table 11-1 lists the peripheral hardware and data buffer functions.

GET DBF, p ; Reads the data of the p-addressed peripheral register in the data buffer.
PUT p, DBF ; Sets the data of the data buffer to the p-addressed peripheral register.

Peripheral registers include the register capable of READ/WRITE (PUT/GET), the WRITE-specialized (PUT) register and the READ-specialized register (GET).

If the "PUT" instruction is issued to the WRITE-specialized (PUT only) or READ-specialized (GET only) peripheral register, the device becomes as follows.

- The undefined value is read when the READ (GET) instruction is executed to the WRITE-specialized (PUT only) peripheral register.
- No influence is made when the WRITE (PUT) instruction is executed to the READ-specialized (GET only) peripheral register.

However, precautions are required when using the assembler and the emulator in the 17K series. For details, please refer to **11.6 "Precautions in Using Data Buffer"**.

Table 11-1 Peripheral Hardware and Data Buffer Function List

Peripheral Hardware		Data Buffer and Peripheral Register Performing Data Transmission				Function		
		Name	Symbol	Peripheral Address	Presence of PUT/GET Instructions	Number of Data Buffer Input/ Output Bits	Number of Actually Used Bits	Summary
Image display controller		IDC starrt position set register	IDCORG	01H	PUT/GET	8	7	Sets the display start position of the image display controller.
A/D converter		A/D converter V_{REF} data register	ADCR	02H	PUT/GET	8	4	Sets the comparative voltage V_{REF} data of the A/D converter. $V_{REF} = \frac{x - 0.5}{16} \times V_{DD}$ (V), $1 \leqq x \leqq 15$
Serial interface		Presettable shift register	SIOSFR	03H	PUT/GET	8	8	Sets the serial out data and reads the serial in data.
Horizontal synchronous signal counter		HSYNC counter data register	HSC	04H	GET	8	6	Reads the horizontal synchronous signal counter.
6-bit D/A converter (PWM output)	PWM pin_0	PWM data register 0	PWMR0	05H	PUT/GET	8	7	Sets the output signal duty of the D/A converter. Duty $D = \frac{x + 0.75}{64}$ (%), $0 \leqq x \leqq 63$ Frequency f = 15.625 kHz
	PWM pin_1	PWM data register 1	PWMR1	06H				
	PWM pin_2	PWM data register 2	PWMR2	07H				
	PWM pin_3	PWM data register 3	PWMR3	08H				
Address register		Address register	AR	40H	PUT/GET	16	16	Data transmission with address register
14-bit D/A converter	PWM_{RMP} pin	PWMRMP data register	PWMRMP	41H	PUT/GET	16	15	Sets the output signal duty of the 14-bit D/A converter.

11.4.2 Precautions in Data Transmission with Peripheral Registers

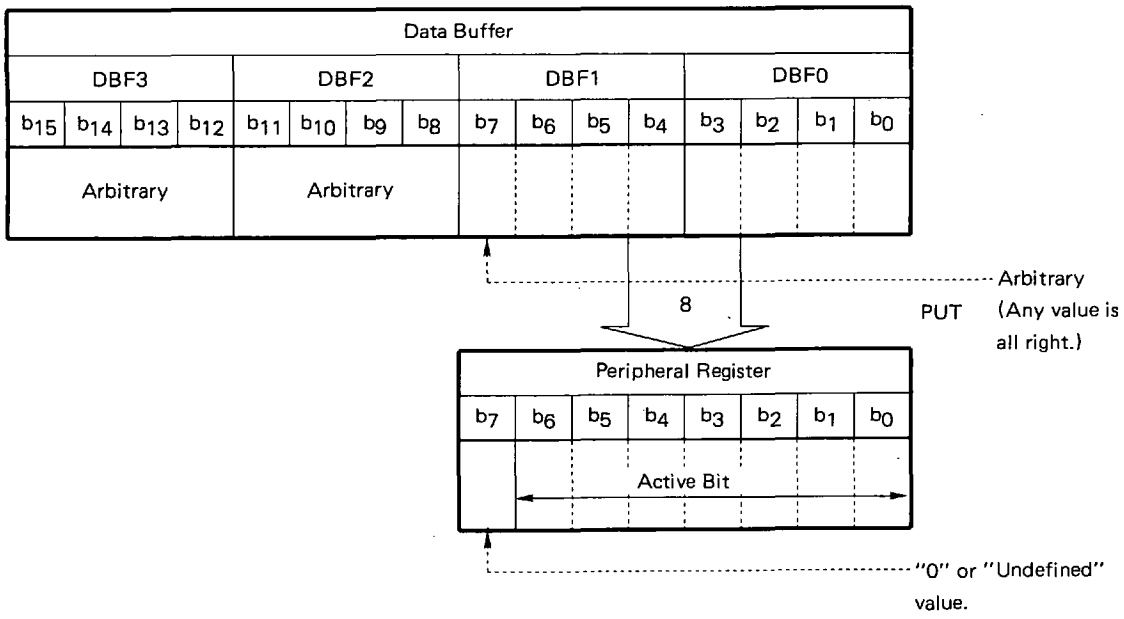
Data transmission between the data buffer and each peripheral register is performed by 8 or 16 bits.

At this time, the "PUT" and "GET" instructions are executed in a single instruction execution time (2 μs) even when the data bits are 16 bits.

If 8-bit data transmission is performed when the data bit length executed by the peripheral register is 7 bits for example, 1 bit becomes redundant.

This redundant data bit becomes the "arbitrary value (any value is all right)" in data write, and the "undefined value" in data read as shown in examples 1 and 2.

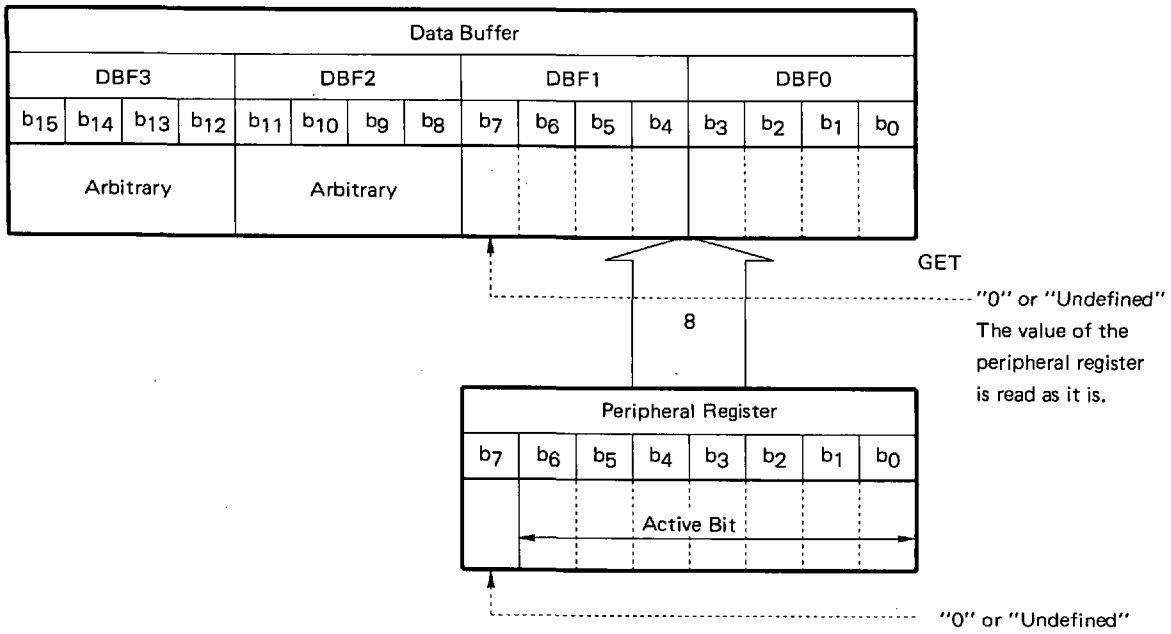
Example 1: When "PUT" instruction (when active bits of the peripheral register are 7 bits from b_0 to b_6)



When writing 8-bit data in the peripheral register, the upper 8 bits (content of DBF3 and DBF2) of the data buffer becomes "arbitrary (any value is all right)"

Among 8-bit data of the data buffer, each bit not corresponding to the active bit of the peripheral register becomes "arbitrary".

Example 2: When "GET" instruction



When reading the 8-bit data of the peripheral register, the value of the upper 8 bits (DBF3 and DBF2) of the data buffer is not changed.

Among the 8-bit data of the data buffer, each bit which is not the active bit of the peripheral register become "0" or "undefined". Whether to become "0" or "undefined" is decided beforehand by each peripheral register.

11.4.3 Status when Resetting the Peripheral Register

The active bit of each peripheral register is set as follows when resetting.

Reset	Active Bit Status
Power ON	Undefined
Clock stop	Previous status hold
CE	Previous status hold

11.5 DATA BUFFER AND EACH PERIPHERAL REGISTER

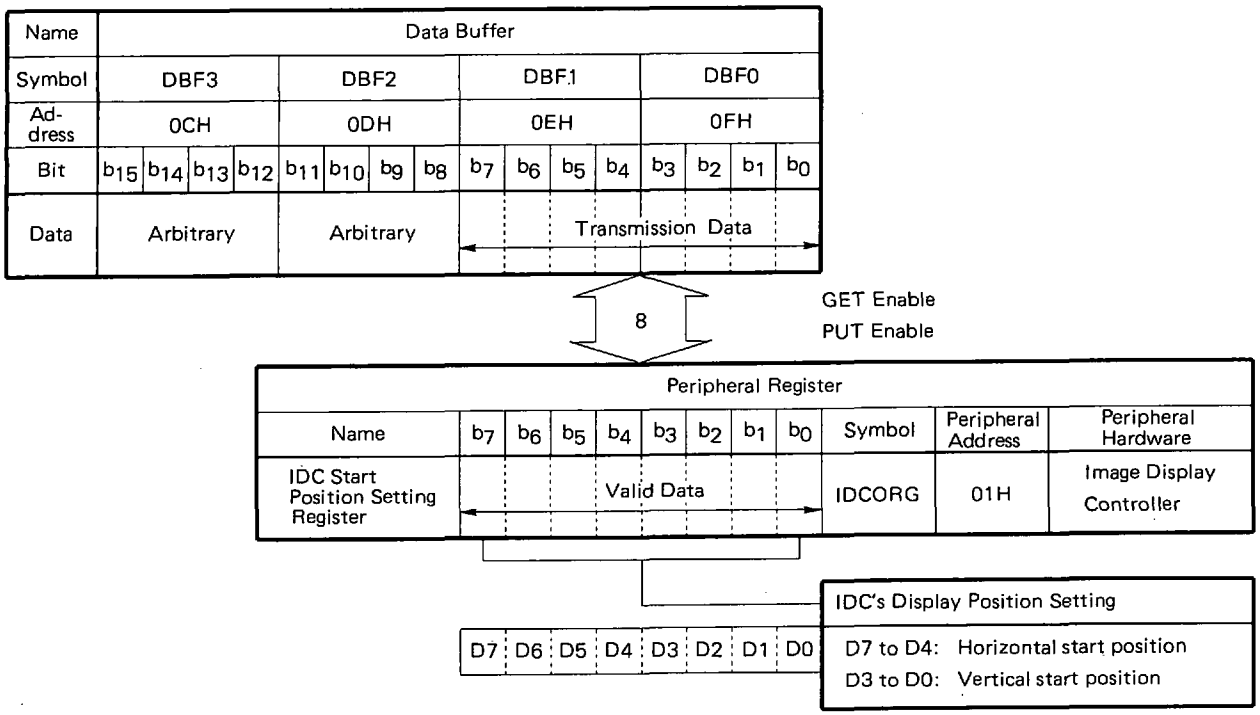
11.5.1 to **11.5.7** explain the data buffer and each peripheral register.

11.5.1 IDC Start Position Setting Register

Fig. 11-4 shows the functions of the IDC start position setting register.

The IDC start position setting register sets IDC's display start position.

Fig. 11-4 IDC Start Position Setting Register Functions



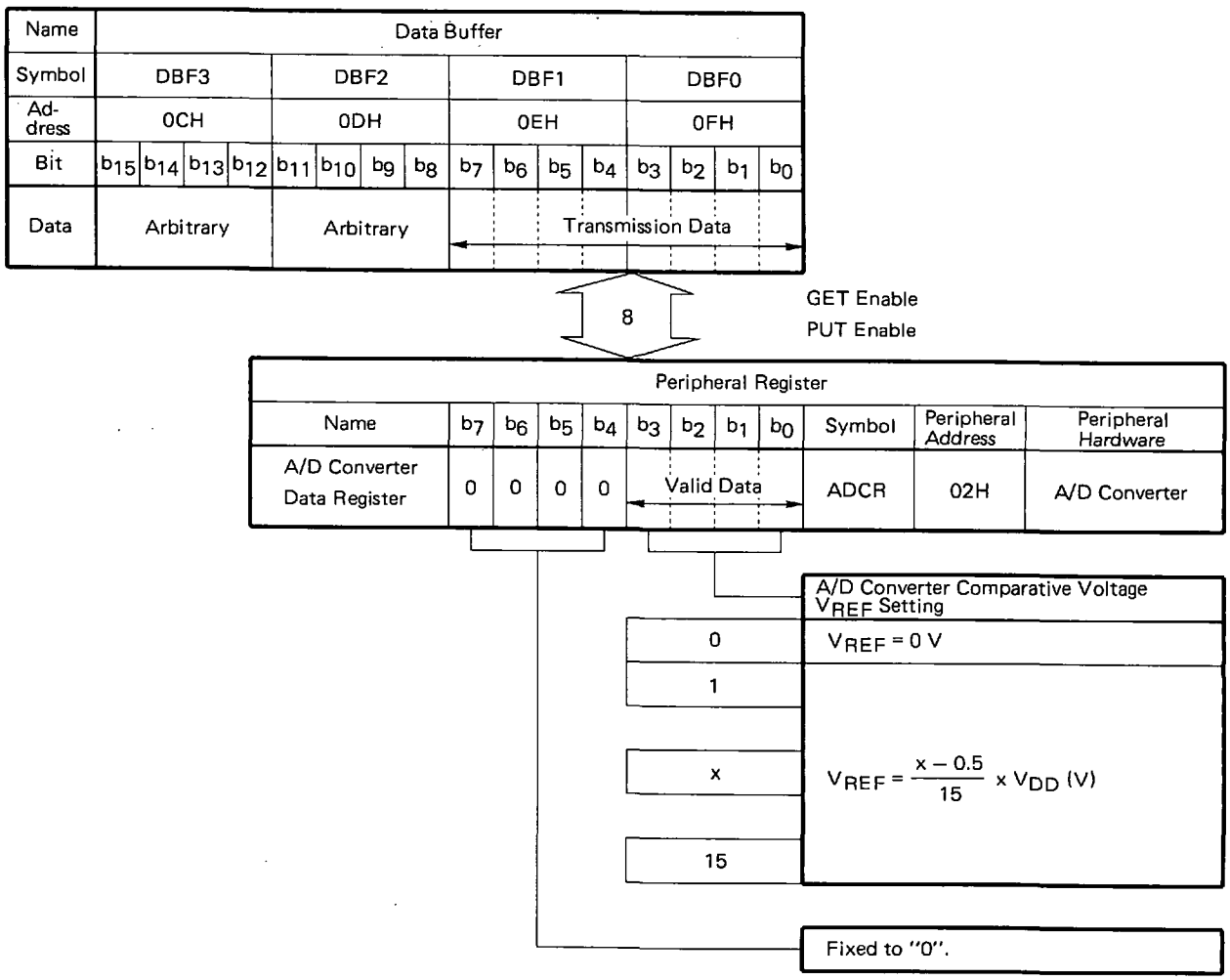
11.5.2 A/D Converter Data Register

Fig. 11-5 shows the functions of the A/D converter data register.

The A/D converter data register sets the comparative voltage of the A/D converter.

As the A/D converter is in 4 bits, the lower 4 bits of the A/D converter data register become valid.

Fig. 11-5 A/D Converter Data Register Functions

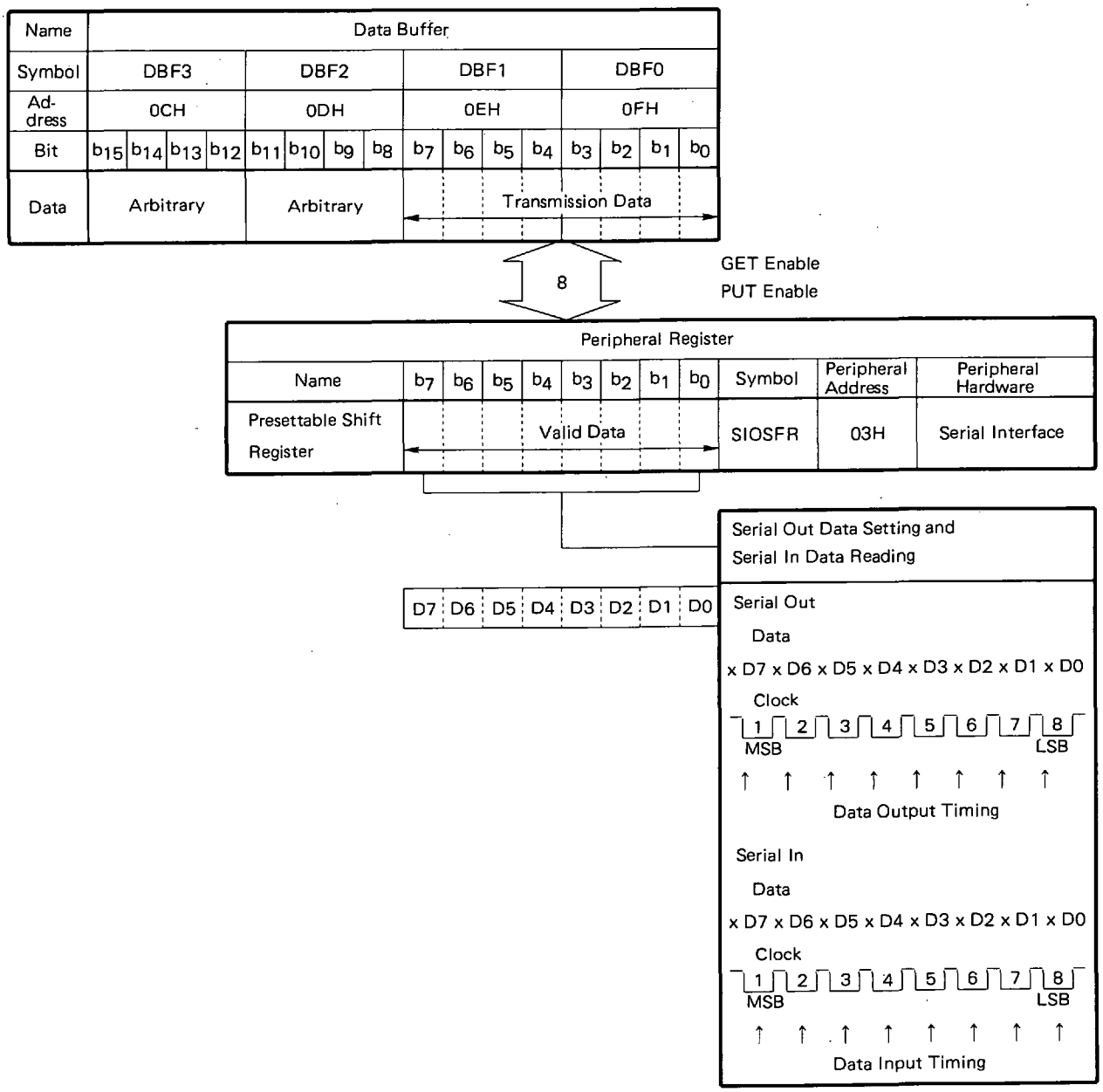


11.5.3 Presettable Shift Register

Fig. 11-6 shows the functions of the presettable shift register.

The presettable shift register sets the serial out data and reads the serial in data of the serial interface.

Fig. 11-6 Relationship between the Presettable Shift Register and the Data Buffer



The serial data output of the serial interface is performed by sequentially shifting from MSB (bit b_7) of the presettable shift register.

The serial data input is performed by sequentially shifting from LSB (bit b_0).

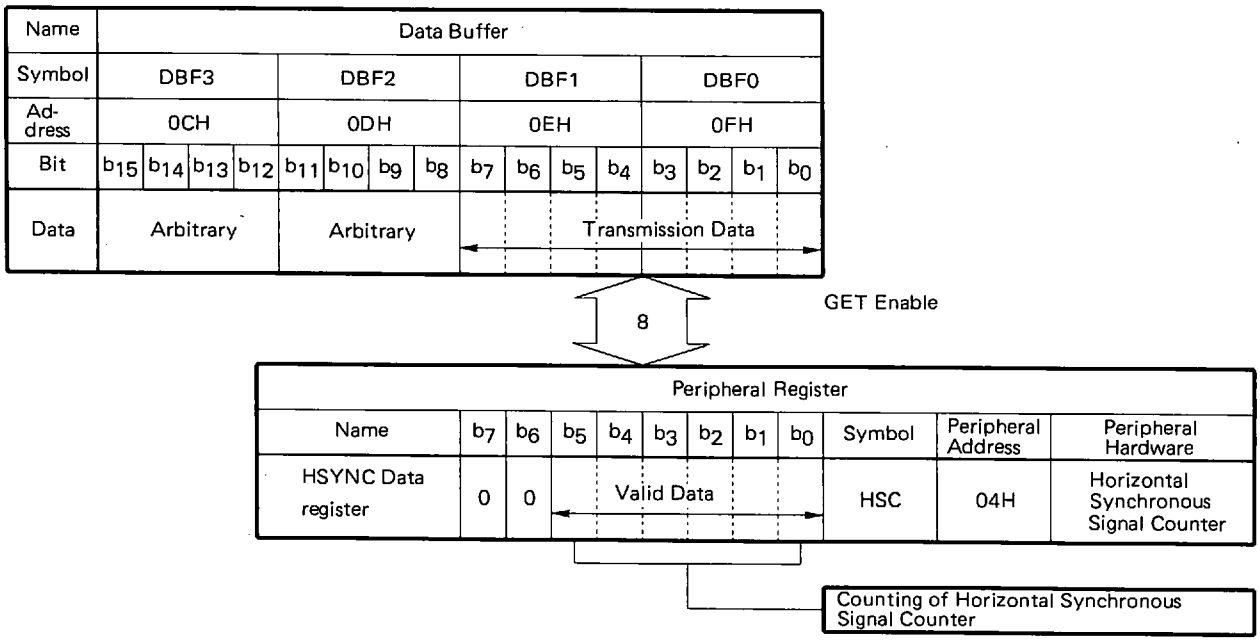
11.5.4 HSYNC Counter Data Register

Fig. 11-7 shows the functions of the HSYNC counter data register.

The HSYNC counter data register reads the counting of the horizontal synchronous signal counter.

If the HSYNC counter data register is counted up to 3FH, the following input becomes 00H.

Fig. 11-7 HSYNC Data Register Functions



11.5.5 PWM Data Register

Fig. 11-8 shows the functions of the PWM data register.

The PWM data register sets the duty of the 6-bit D/A converter (PWM output) output signal.

The 6-bit D/A converter has four channels of PWM_3, PWM_2, PWM_1 and PWM_0 pins. As each of these can set the duty, the PWM data register also has four systems independently.

Fig. 11-8 PWM Data Register Functions

Name	Data Buffer															
Symbol	DBF3				DBF2				DBF1				DBF0			
Address	0CH				0DH				0EH				0FH			
Bit	b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
Data	Arbitrary				Arbitrary				Transmission Data							

8

GET Enable
PUT Enable

Peripheral Register

Name	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	Symbol	Peripheral Address	Peripheral Hardware
PWM0 Data Register	0	Valid Data							PWMR0	05H	PWM_0 Pin
PWM1 Data Register	0								PWMR1	06H	PWM_1 Pin
PWM2 Data Register	0								PWMR2	07H	PWM_2 Pin
PWM3 Data Register	0								PWMR3	08H	PWM_3 Pin

	Sets the PWM output duty of each pin.
0	Duty D = $\frac{x + 0.75}{64}$ (%) Frequency f = 15.625 kHz
x	
63	

0	Uses the PWM pin as the D/A converter.
0	Uses the PWM pin as the 1-bit output pin. Outputs the content of b_5.

11.5.6 Address Register

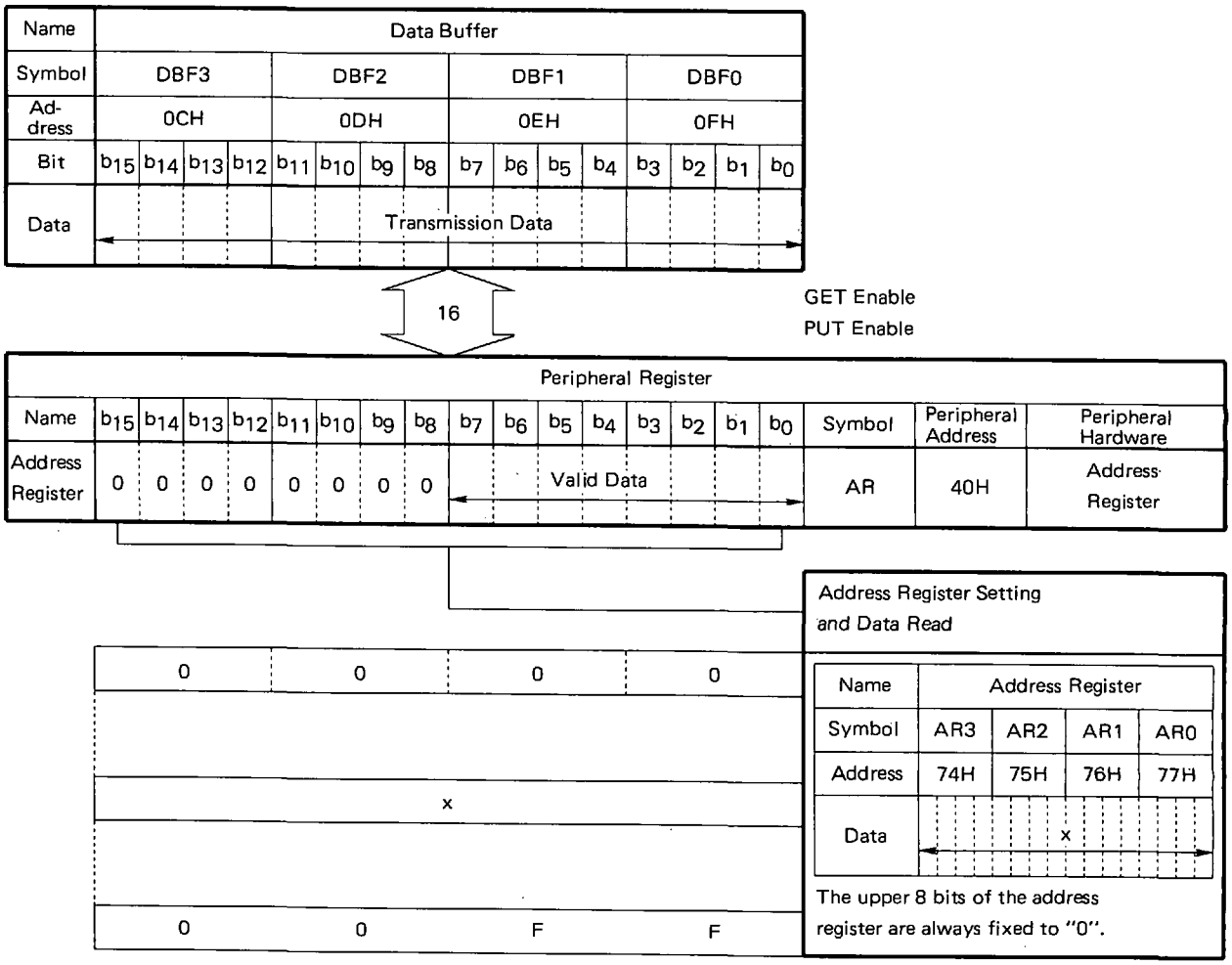
The address register is located in addresses 74H to 77H of the system register (addresses 74H to 7FH on the data memory) to operate the address of the program memory. Please refer to **9 "System Register (SYSREG)".**

Therefore, although data can be operated directly by the data memory operation instruction, data transmission is also possible via the data buffer as a part of the peripheral hardware.

In other words, besides using the data memory operation instruction, the address register can read/write data via the data buffer by using the "PUT" and "GET" instructions.

Fig. 11-9 shows relationship between address register and data buffer.

Fig. 11-9 Relationship between Address Register and Data Buffer



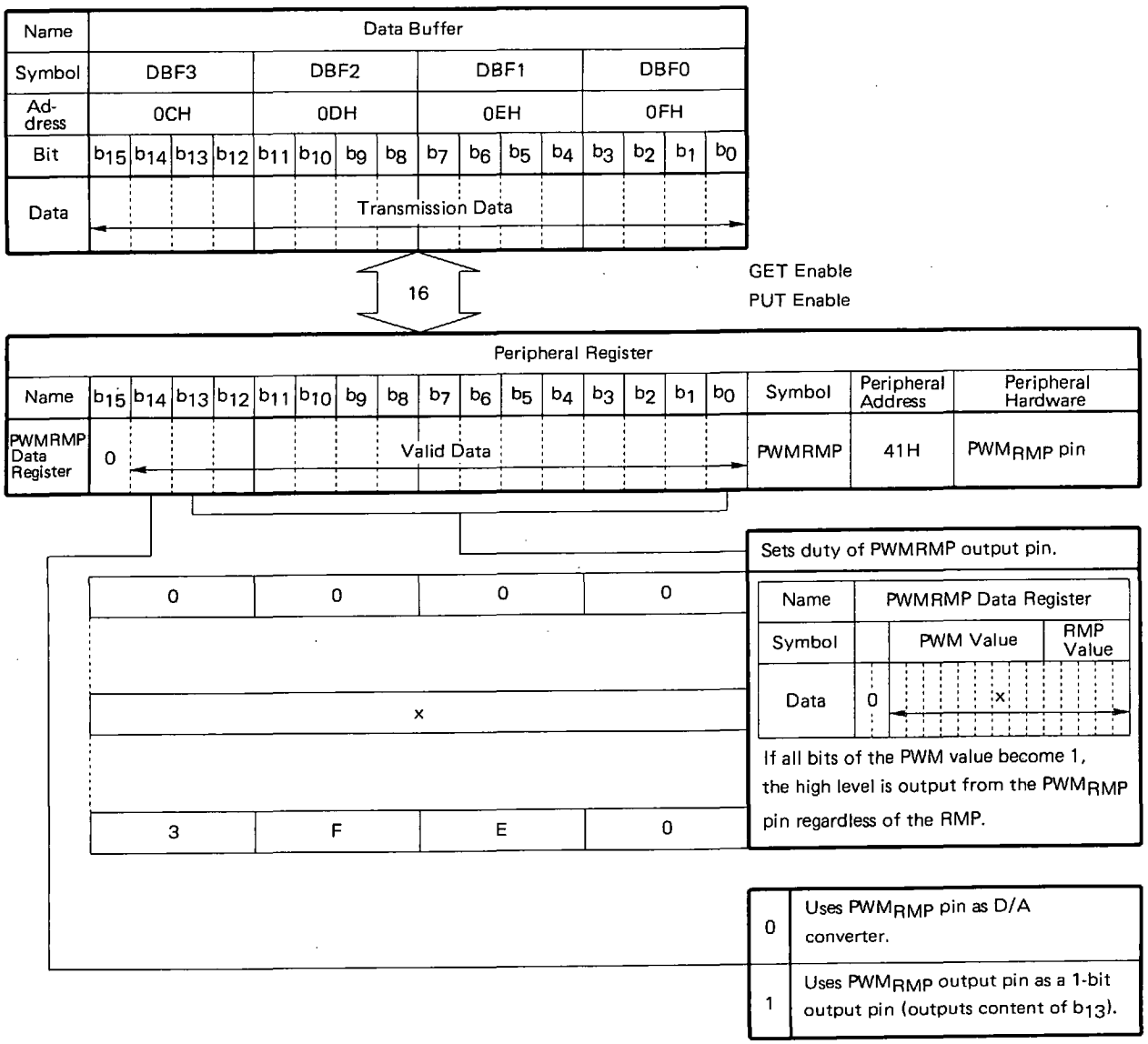
11.5.7 PWMRMP DATA REGISTER

Fig. 11-10 shows the functions of the PWMRMP data register.

The PWMRMP data register sets the duty of the 14-bit D/A converter output signal.

The PWM_{RMP} output is performed by combining the 9-bit PWM and the 5-bit RMP.

Fig. 11-10 PWMRMP Data Register



11.6 PRECAUTIONS IN USING DATA BUFFER

11.6.1 Precautions when Operating the Data Buffer of Write-Specialized, Real-Specialized and Unused Addresses

When performing data transmission between the peripheral hardware via the data buffer, precautions are required as follows in terms of device operation when using the 17K series assembler and emulator regarding the unused peripheral register, the write-specialized peripheral register (PUT only) and the read-specialized peripheral register (GET only).

(1) Device operation

If the write-specialized peripheral register is read, the "undefined value" is read.

Even if the read-specialized peripheral register is written, no change occurs.

If the unused address is read, the "undefined value" is read, and even if it is written, no change occurs.

(2) When using the assembler

"Error" occurs in the instruction reading the write-specialized peripheral register.

"Error" occurs in the instruction writing the read-specialized peripheral register.

"Error" occurs in the instructions reading and writing the unused address.

(3) When using the emulator (when the instruction has been executed in batch processing, etc.)

If the write-specialized peripheral register is read, the "undefined value" is read. "Error" does not occur.

Even if write is performed in the read-specialized peripheral register, no change occurs. "Error" does not occur.

If the unused address is read, the "undefined value" is read, and even if it is written, no change occurs. "Error" does not occur.

11.6.2 Peripheral Register Address and Reserved Word

In using the assembler of the 17K series, even if the peripheral address "p" is directly (in numerals) specified by the "PUT p, DBF" instruction or the "GET DBF, p" instruction as shown in example 1, no error occurs.

However, this method is not desirable in reducing the program bugs.

Therefore, it is necessary to symbol-define the peripheral address beforehand as shown in example 2 with the symbol definition instruction, which is an assembler pseudo-instruction.

Here, the peripheral address is defined beforehand as the "reserved word" in the assembler to simplify the symbol definition.

Therefore, with the reserved word, the program can be prepared without defining the symbol as shown in example 3.

The reserved word of the peripheral register is shown in the "symbol" in Table 11-1 and in Figs. 11-4 to 11-10.

Example 1:
```
PUT     02H,        DBF ;
GET     DBF,        03H ;
```
Even if the peripheral address is directly specified with 02H and 03H, the assembler does not fall in error. However, this is not desirable in reducing program bugs.

2:
```
SIODATA DAT         03H ;
PUT     SIODATA, DBF ;
```
Allocates SIODATA to 03H with the symbol definition instruction.

3:
```
PUT     SIOSFR      ;
```
If the "SIOSFR", which is the reserved word, is used, it is not necessary to define the symbol.

12. INTERRUPT

In accordance with the request from the peripheral hardware (RMC pin, timer, $\overline{V_{SYNC}}$ pin and serial interface), the interrupt temporarily stops the program in current execution and moves the program flow to the address (called vector address) decided beforehand.

12.1 INTERRUPT BLOCK CONFIGURATIONS

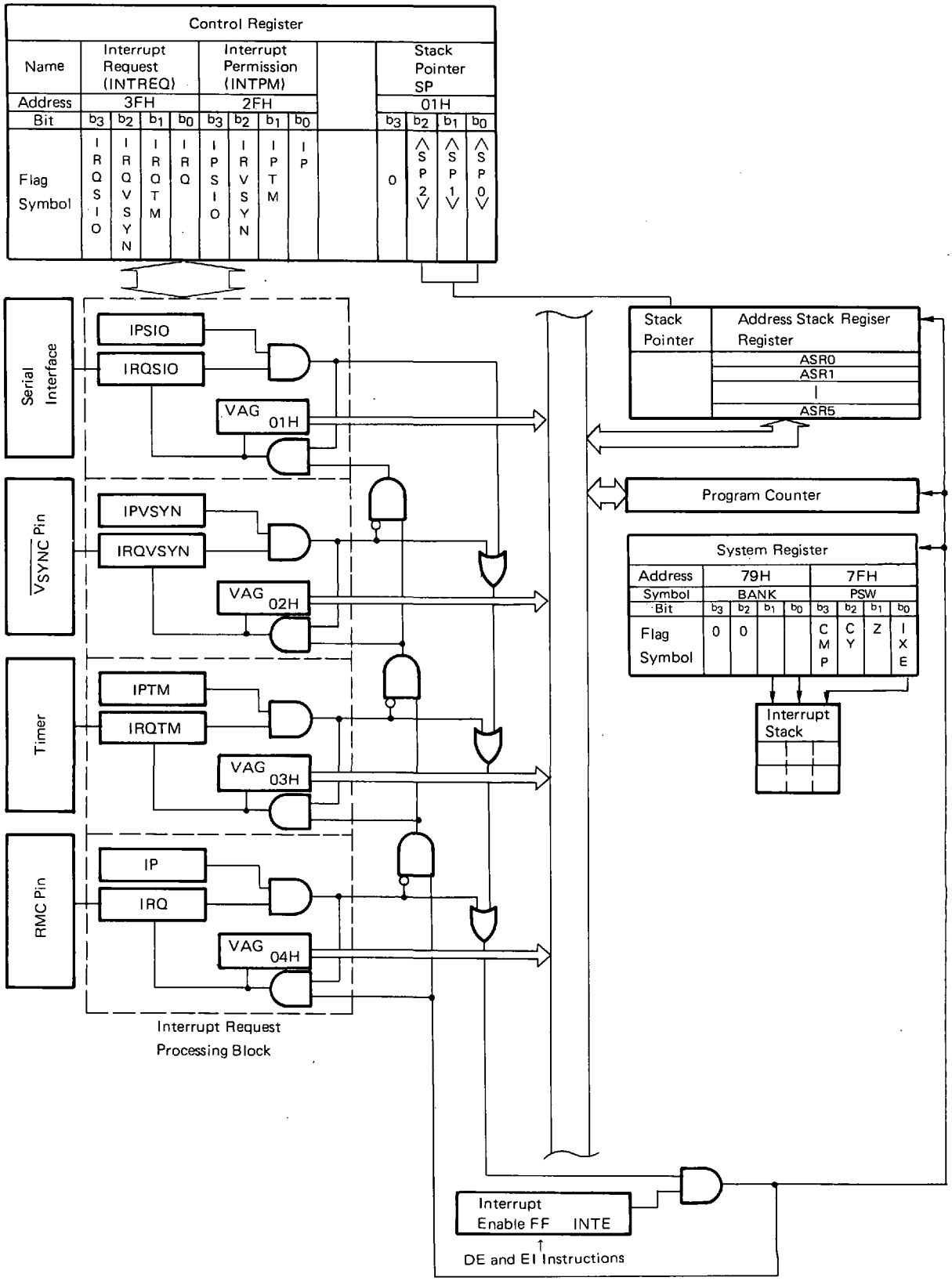
Fig. 12-1 shows the interrupt block configurations.

As shown in Fig. 12-1, the interrupt block consists each "interrupt request control block" which controls the interrupt request output from each peripheral hardware of the RMC pin, the timer, $\overline{V_{SYNC}}$ pin and the serial interface; the "interrupt enable flip-flop (INTE)" which sets the enable of all interrupts; the "stack pointer", "address stack register", "program counter" and "interrupt stack", which are controlled when the interrupt has been accepted.

The "interrupt request processing block" of each peripheral hardware consists of the "flip-flop (IRQxxx) which detects the each interrupt request", the "flip-flop (IPxxx) which sets the enable of each interrupt", and the "vector address generator (VAG)" which specifies the vector address when accepting the interrupt.

The IRQxxx and IPxxx flip-flows correspond one-to-one with each flag of the interrupt request and the interrupt permission of the control register.

Fig. 12-1 Interrupt Block Configurations



12.2 INTERRUPT FUNCTION

The peripheral hardware devices that can use the interrupt function are the RMC pin, the timer, the $\overline{V_{SYNC}}$ pin and the serial interface.

The interrupt function is for these peripheral hardware devices to temporarily stop the program in execution when a certain condition is satisfied (for example, the fall signal is added to the RMC pin) and then to execute the specialized processing program.

At this time, the interrupt signal from the peripheral hardware is called "interrupt request", and the fact that the interrupt signal is output is mentioned that "the interrupt request is issued". The interrupt-special processing program is called "interrupt service routine".

When the interrupt is accepted, it is branched to the program memory address (vector address) decided for each interrupt factor. Therefore, it is all right for each interrupt service routine to be started from this vector address.

The interrupt function is divided into the processing until the interrupt is accepted and that after the interrupt has been accepted. In other words, it is divided into the function until the interrupt is accepted regarding the interrupt request from each peripheral hardware device and the function of branching to the vector address after the interrupt has been accepted and then returning it to the program before the interrupt.

12.2.1 to **12.2.5** show the function of each block shown in Fig. 12-1.

12.2.1 Peripheral Hardware

The peripheral hardware which has the interrupt function includes the four types of the RMC pin, the timer, the $\overline{V_{SYNC}}$ pin and the serial interface.

Each peripheral hardware device can be set with the condition for issuing the interrupt request.

For example, the RMC pin is made to be able to select either to issue the request with the rising edge or with the falling edge of the signal added to the RMC pin.

For details of the interrupt request issuance conditions of each peripheral hardware, please refer to **12.3** to **12.7**.

12.2.2 Interrupt Request Processing Block

The interrupt request processing block, installed on each peripheral hardware, generates vector addresses for the presence/absence of the interrupt request, the interrupt enable and the interrupt acceptance respectively.

12.2.3 to **12.2.5** below explain each flag of the interrupt request processing block.

12.2.3 Interrupt Request Flag (IRQXXX)

This flag is set (1) when the interrupt request is issued from each peripheral hardware device, and reset (0) when the interrupt is accepted.

As it corresponds one-to-one with each flag of the interrupt request register of the control register, it is possible to perform read/write via the window register.

"1" written via the window register is equivalent to issuance of the interrupt request.

Once this flag is set, it is not reset until the corresponding interrupt is accepted or "0" is written via the window register.

Even when multiple interrupt requests have been issued simultaneously, the interrupt request flag to the unaccepted interrupt is not reset.

This flag is reset (0) at the time of power-on reset, executing the clock stop instruction, and CE reset.

12.2.4 Interrupt Permission Flag (IPXXX)

This sets the interrupt enable for each hardware device.

If these flags have been set (1) and the corresponding interrupt request flags are set, the interrupt request is output.

As these flags correspond one-to-one with each flag of the interrupt permission register of the control register, read and write are performed via the window register.

These flags are reset (0) at the time of power-on reset, clock stop and CE reset.

12.2.5 Vector Address Generator (VAG)

When the interrupt of each peripheral hardware device has been accepted, the branch address (vector address) of the program memory to the accepted interrupt factor is generated.

The vector address to each interrupt factor is shown in Table 12-1.

Table 12-1 Interrupt Vector Address

Interrupt Factor	Vector Address
RMC pin	04H
Timer	03H
$\overline{V_{SYNC}}$ pin	02H
Serial interface	01H

12.2.6 Interrupt Enable Flip-Flop (INTE)

The interrupt enable flip-flop sets the enable of all the four interrupt types.

If "1" is output from each interrupt request block when this flip-flop is set (1), "1" is output from this flip-flop, thus accepting the interrupt.

When this flip-flop is reset (0), the interrupt is not accepted even if "1" is output from each interrupt service block.

For setting/resetting of this flip-flop, the specialized "EI" (for setting) and "DI" (for resetting) instructions are used.

When the "EI" instruction is executed, this flip-flop is set at the time the execution of the instruction following the "EI" instruction has been completed, and then reset during the "DI" instruction execution cycle if the "DI" instruction is executed.

If the interrupt is accepted in the state that the interrupt enable flip-flop is set (EI state), this flip-flop is reset at this point of acceptance (DI state).

This flag exerts no influence whether the "DI" instruction is executed during the "DI state" or whether the "EI" instruction is executed during the "EI state".

This flag is reset (DI state) at the time of power-on resetting, clock stop and CE resetting.

12.2.7 Stack Pointer, Address Stack Register and Program Counter

The address stack register saves the return address from the interrupt service routine at the time of its returning.

The stack pointer specifies which register to use among 6 address stack register (ASR0 to ASR5).

In other words, if the interrupt is accepted, the stack pointer value is decremented by −1 and the program counter value at this time is saved in the address stack register specified by the stack pointer. If the "RETI" instruction, which is the specialized return instruction, is executed after executing the processing of the interrupt service routine, the content of the address stack register specified by the stack pointer is saved in the program counter and the stack pointer value is incremented by +1.

For details, please refer to **4 "Stack"** as well.

12.2.8 Interrupt Stack

The interrupt stack saves the contents of the bank register and the index enable flag in the system register when accepting the interrupt.

If the interrupt is accepted and the bank register and the index enable flag are saved, the bank register and the index enable flag on the system register are reset (0).

The interrupt stack can save the contents of the bank register and the index enable flag up to two levels. Thus, multiple interrupts, in which the routine processing an interrupt accepts another interrupt, can be performed up to two levels.

The contents of the interrupt stack are returned to the bank register and the index enable flag of the system register by executing the "RETI" instruction, which is the specialized return instruction from the interrupt service routine.

Please refer to **4 "Stack"** as well.

12.3 INTERRUPT ACCEPTANCE OPERATION

12.3.1 Interrupt Acceptance Operation and Priority

Operations until the interrupt is accepted are as follows.

(1) Each peripheral hardware outputs the interrupt request signal to each interrupt request block when the interrupt conditions are satisfied. (For example, the fall signal is input to the RMC pin.)

(2) Each interrupt request block sets (1) the corresponding IRQXXX flag (for example, IRQ for the RMC pin) when the interrupt request signal from each peripheral hardware is accepted.

(3) When each interrupt request flag has been set, the interrupt permission flag corresponding to each IRQxxx, for example, the IRQ flag outputs "1" from each interrupt request block if the IP flag is set (1).

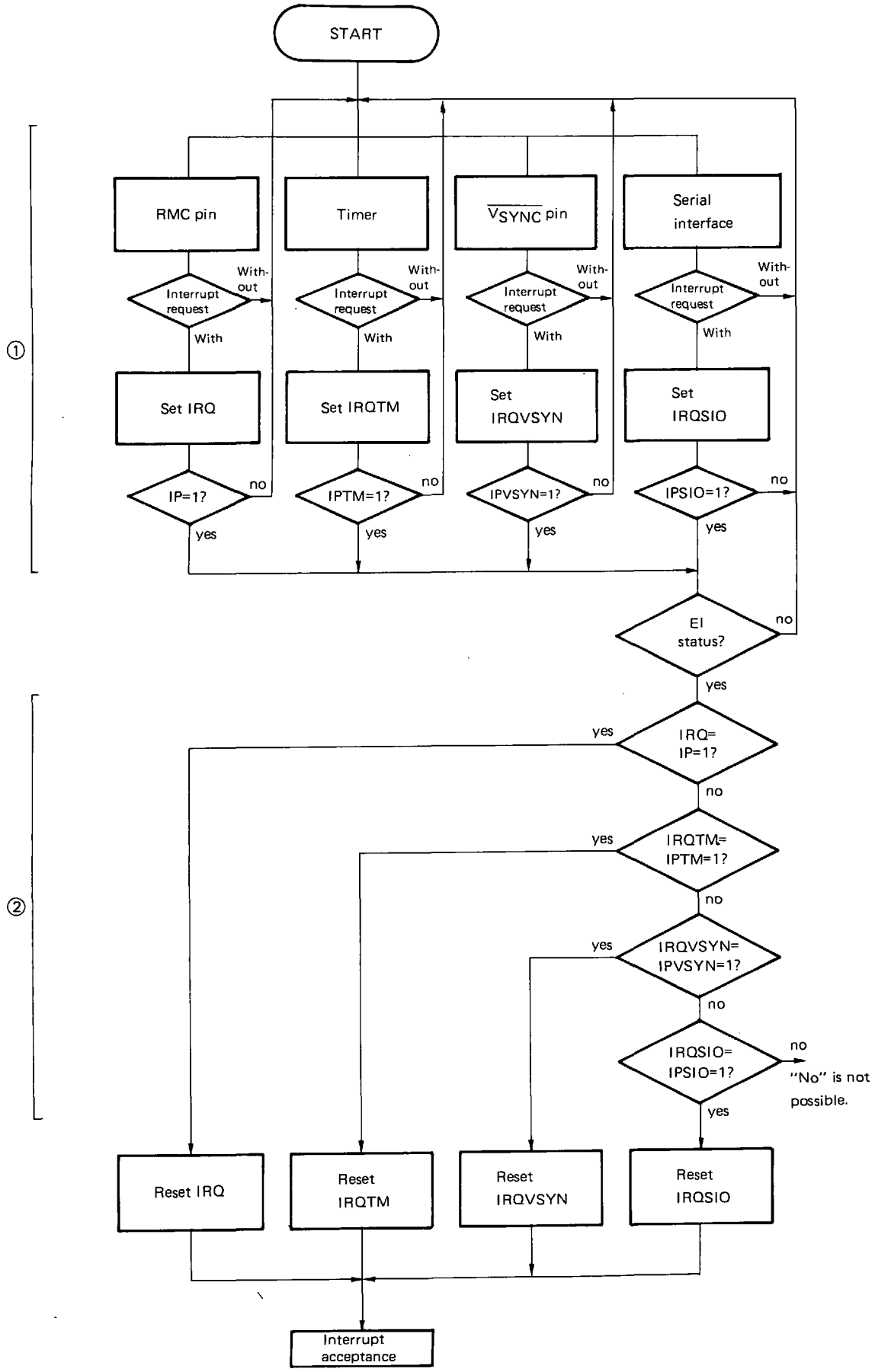
(4) The signal output from each interrupt request block is input to the interrupt enable flip-flop via the OR circuit.
This interrupt enable flip-flop is set (1) by the "EI" instruction and is reset by the "DI" instruction.
If "1" is output from each interrupt request block when the interrupt enable flip-flop has been set, "1" is output from the interrupt enable flip-flop and the interrupt is accepted.

If the interrupt is accepted, the interrupt enable flip-flop output, as shown in Fig. 12-1, is input to each interrupt request block via the AND circuit.
The interrupt request flag is reset by the signal input to each interrupt request block, and the vector address to each interrupt is output.
At this time, if "1" has been output from the interrupt request block, the interrupt acceptance signal is not transferred to the next stage. Therefore, if multiple interrupts have been simultaneously issued, they are accepted in the priority order shown below.

(DMA) > RMC pin > Timer > $\overline{V_{SYNC}}$ pin > Serial interface

This order is called the "hardware priority order".
Fig. 12-2 shows the flowchart of the interrupt acceptance operation.
As ① shown in Fig. 12-2 is always processed in parallel, each interrupt request flag is set simultaneously if multiple interrupt requests have occurred simultaneously.
However, ② is processed according to the priority order by each interrupt permission flag.
In other words, if the interrupt permission flag is not set, the interrupt to its interrupt factor is not accepted.
As the interrupt permission flag can be set/reset by the program, it is possible to inhibit the interrupt of high hardware priority order if the interrupt permission flag reset beforehand.
The interrupt by this interrupt permission flag is called the "maskable interrupt". As the maskable interrupt can inhibit the interrupt factor of high hardware priority with the program, it is also called "software priority order".

Fig. 12-2 Interrupt Acceptance Operation Flowchart



12.3.2 Timing Chart when Accepting Interrupt

Fig. 12-3 shows the timing chart when accepting the interrupt.

(1) in Fig. 12-3 is the timing chart based on interrupts of a single type.

(1) (a) is the timing chart of the interrupt request flag having been set (1) finally; (1) (b) is the timing chart of the interrupt permission flag having been set (1) finally.

In either case, the interrupt is accepted at the point that all of the interrupt request flag, the interrupt enable flip-flop and the interrupt permission flag are set.

If the finally set flag or flip-flop is the first instruction cycle of the "MOVT DBF, @AR" instruction or the instruction which has satisfied the skip condition, the interrupt is accepted after the second instruction cycle of the "MOVT DBF, @AR" instruction and the skipped instruction (becomes NOP) are executed respectively.

The interrupt enable flip-flop is set with the instruction cycle following the execution of the "EI" instruction.

(2) in Fig. 12-3 shows the timing chart when using multiple interrupts.

When using multiple interrupts, the interrupt in higher hardware priority is accepted first if all the interrupt permission flags are set. However, it is possible to change the hardware priority by manipulating the interrupt permission flag with the program.

"Interrupt cycle" shown in Fig. 12-3 is a special cycle to reset the interrupt request flag, specify the vector address and save the program counter after the interrupt has been accepted. It is necessary for the portion of executing one instruction (2 μs, however, 12 μs in IDC operation). For details, please refer to **12.2.7 "Operation after Accepting Interrupt".**

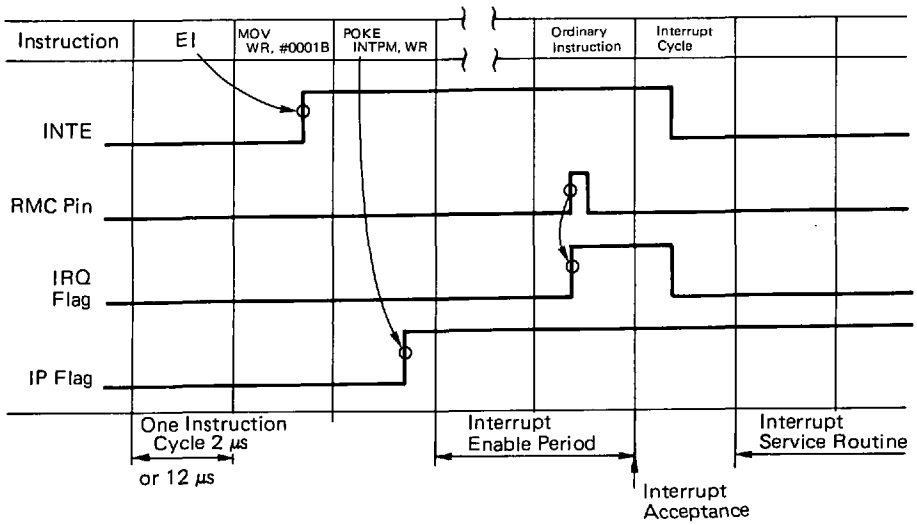
The interrupt request flag is set (1) by the interrupt request of the peripheral hardware regardless of the "EI" instruction or the interrupt permission flag. Therefore, it is possible to know the presence/absence of the interrupt request by detecting the interrupt request flag with the program.

Fig. 12-3 Interrupt Acceptance Timing Chart

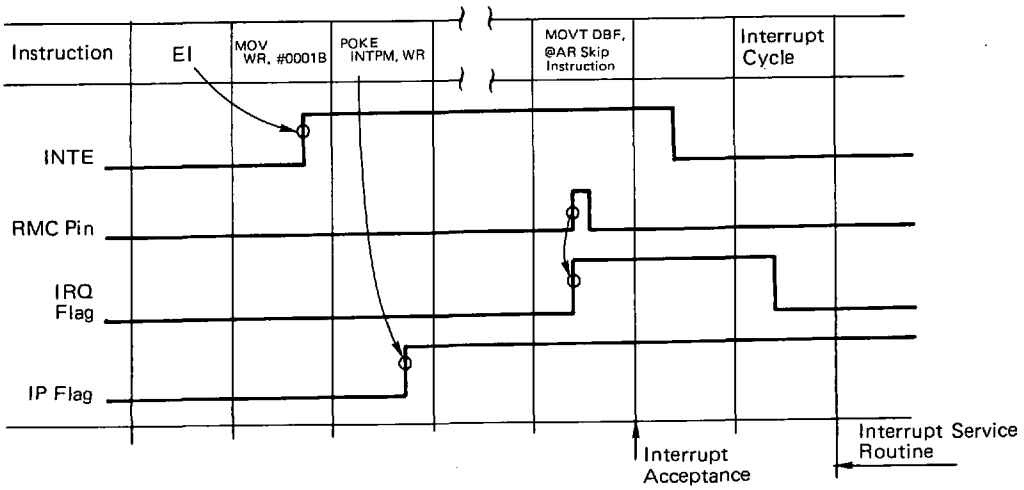
(1) When using interrupts of a single type (Example: Rise of the RMC pin)

(a) When there is no interrupt mask time by the interrupt permission flag

① If when the interrupt is accepted is with the "MOVT" instruction and the ordinary instruction which has not satisfied the skip condition



② If when the interrupt is accepted is with the "MOVT" instruction or the instruction which has satisfied the skip condition.



(b) If there is the interrupt hold period by the interrupt permission flag

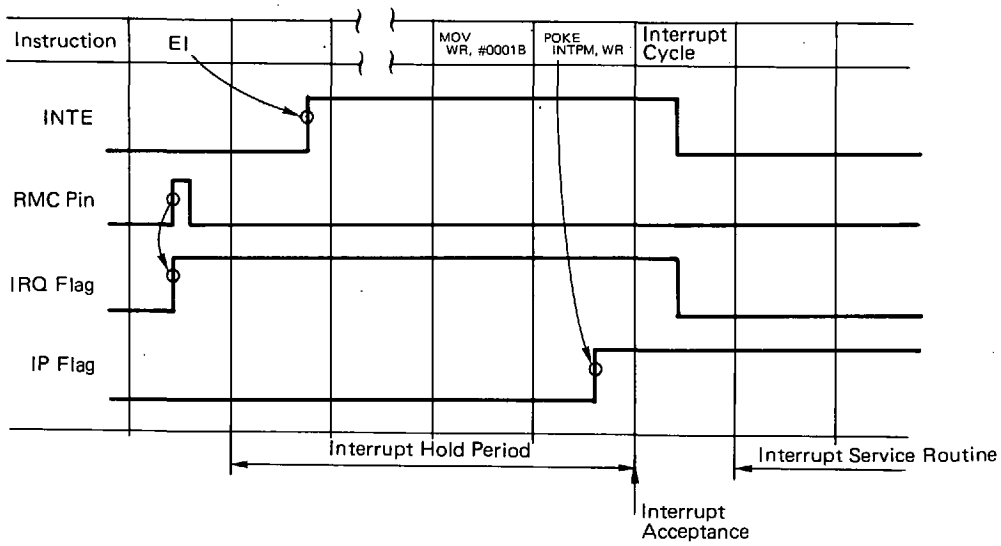
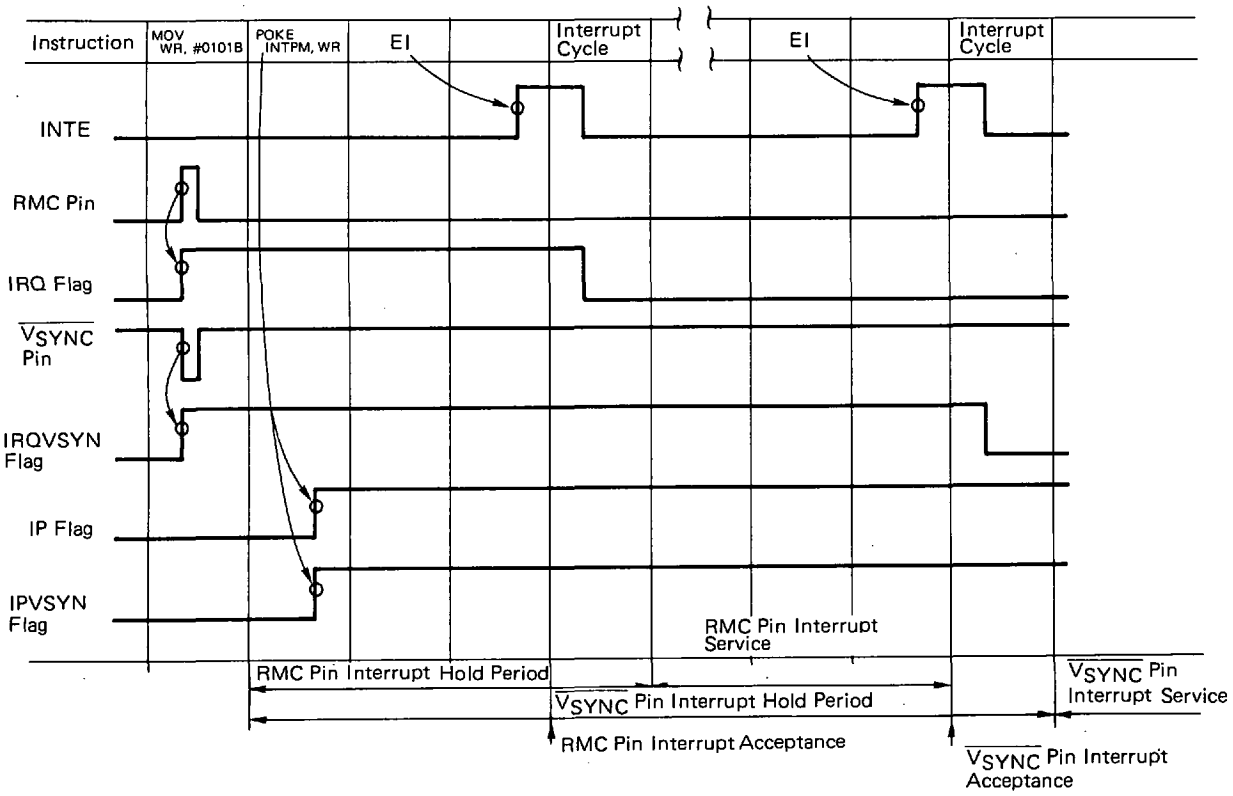


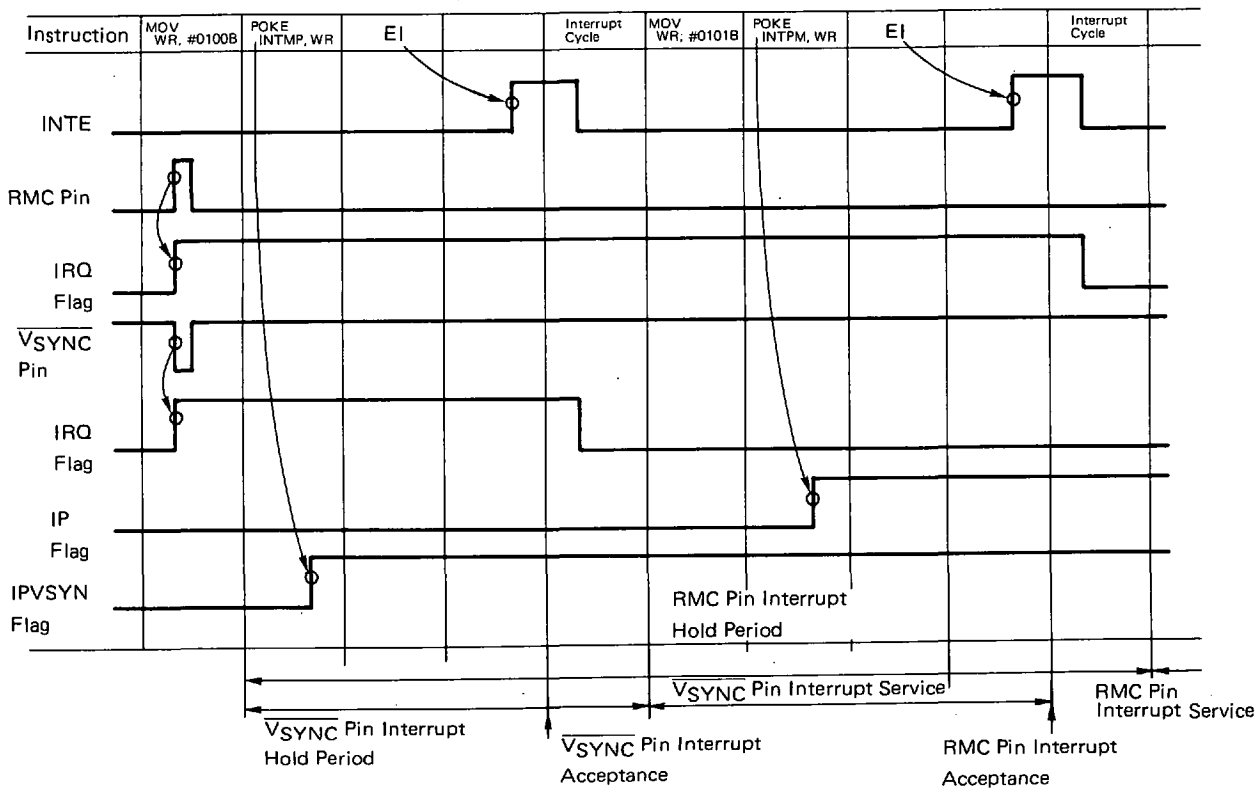
Fig. 12-3 Interrupt Acceptance Timing Chart

(2) When using multiple interrupts (Example: Two types of RMC pin rise and $\overline{V_{SYNC}}$ pin fall)

(a) Hardware priority



(b) Software priority



12.4 OPERATION AFTER INTERRUPT ACCEPTANCE

If the interrupt is accepted, the following processings are automatically executed sequentially.

(1) Resets the interrupt request flag which corresponds to the interrupt enable flip-flop and the accepted interrupt request. In other words, the system is put in the interrupt inhibit state.

(2) Decrements the contents of the stack pointer by −1.

(3) Saves the contents of the program counter to the address stack register specified by the stack pointer. At this time, the program counter contents become the program memory address following the interrupt acceptance. For example, if the instruction is the branch, this is the address of the branch destination; if the instruction is the subroutine call, this is the address called. When the skip condition has been satisfied by the skip instruction, the interrupt is accepted after the next instruction is executed as the "NOP" instruction. Therefore, the program counter contents become the skipped address.

(4) Saves the lower 2 bits of the bank register (BANK; address 79H) and the index enable flag (IXE, address 7FH's bit b_0 in the interrupt stack).

(5) Transmits the contents of the vector address generator corresponding to the accepted interrupt to the program counter, that is, branches it to the interrupt service routine.

The processings (1) to (5) above are executed within the special one-instruction cycle (2 μs, however 12 us in IDC operation) which is not accompanied by execution of the ordinary instruction.

This instruction cycle is called "interrupt cycle". In other words, the time of one-instruction cycle is necessary from the time the interrupt is accepted till it is branched to the corresponding vector address.

12.5 RETURNING FROM INTERRUPT SERVICE ROUTINE

The specialized "RETI" instruction is used to return from the interrupt service routine to the processing when the interrupt was accepted. If the "RETI" instruction is executed, the following processings are automatically executed sequentially.

(1) Returns the contents of the address stack register specified by the stack pointer to the program counter.

(2) Returns the contents of the interrupt stack to the lower 2 bits of the bank register or bit b_0 of the index enable flag.

(3) Increments the contents of the stack pointer by +1.

Processings (1) to (3) above are executed within the one-instruction cycle in which the "RETI" instruction is executed. The only difference between the "RETI" instruction and the "RET" and "RETSK" instructions, which are subroutine return commands, is the difference of the return operation of the bank register and the index enable flag in (2) above.

12.6 INTERRUPT SERVICE ROUTINE

The interrupt is accepted regardless of the program being executed at the time the interrupt request is issued if the interrupt is enabled in the program area.

Therefore, when returning the interrupt to the program even after executing the interrupt service, it is necessary to return it to the state as if the interrupt service has not been executed.

For example, if arithmetic operation is executed while processing the interrupt, it is possible that the contents of the CARRY flag might be changed to the state before the interrupt is accepted, thus resulting in the program after returning making erroneous decisions.

For this reason, it is necessary to perform the save or return operation in the interrupt service routine at least for the system register and the control register which are capable of this operation in the interrupt service routine.

For processing required when enabling another interrupt while processing an interrupt (multiple interrupts), please refer to **12.9 "Multiple Interrupts".**

12.6.1 Save Processing

The example shows the save processing in the interrupt routine.

As it is only the bank register and the index enable flag that are automatically saved by the hardware in the system register, another system register is saved by the program if necessary as shown in the example.

As shown in the example, the "POKE" and "PEEK" instructions are convenient for the save and return processings of the system register.

Besides the "PEEK" and "POKE" instructions, there is also the method of using transmission instructions (LD r, m and ST m, r, etc.). However, if the transmission instruction is used for saving when the row address of the general register is not definite at the time the interrupt has been accepted, it is difficult to specify the data memory address for saving.

This is because the indefinite address of the general register when using the transmission instruction to save the general register itself will result in the address being indefinite as well. Thus, at least the general register must be fixed to be used during the interrupt enable routine.

However, the register file, which is controlled by the "PEEK" or "POKE" instruction, is address-specified regardless of the contents of the general register, and address 40H to 7FH of the register file are overlapping with the data memory of the bank selected at the time. Thus, it is possible to save each system register merely by specifying the bank.

In the example, the window register and the general register pointer are saved with the "PEEK" and "POKE" instructions, the general register is re-specified to BANK0's row address 07H, and then other system registers are saved with the "ST" instruction.

Fig. 12-4 shows the save operation example based on the "PEEK" and "POKE" instructions.

12.6.2 Return Processing

The return processing is shown in the example.

The return processing is the reverse operation of the save processing explained in **12.6.1.**

The interrupt is supposed to have been accepted in the interrupt enable state (EI state). Therefore, if the interrupt has been accepted, it is necessary to execute the "EI" instruction before executing the "RETI" instruction.

The "EI" instruction sets (1) the interrupt enable flip-flop after the "RETI" instruction is executed. Therefore, it is put in the interrupt enable state after returning in the program before the interrupt is accepted.

12.6.3 Precautions in Interrupt Service Routine

Be careful about the following points in the interrupt service routine.

(1) Data saved by the hardware

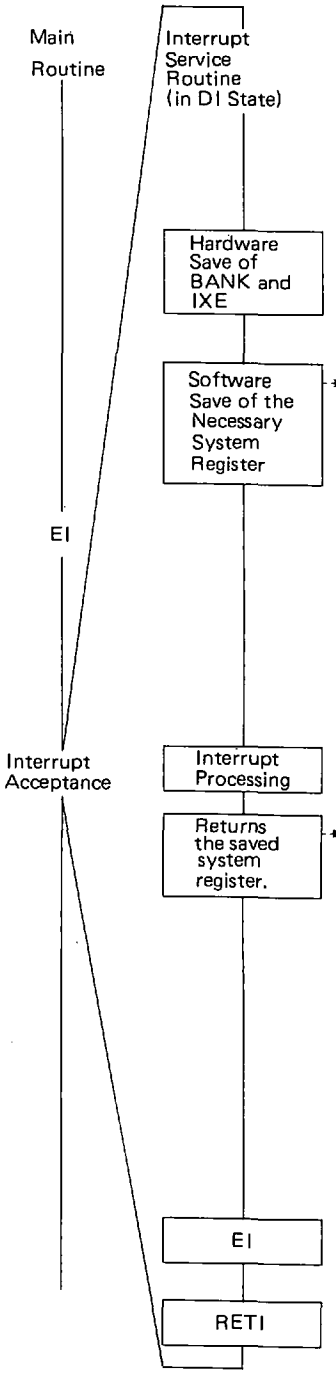
The bank register and the index enable flag are all reset to "0" after being saved in the interrupt stack.

(2) Data saved by the software

Data saved by the software is not reset even after being saved.

Especially, the program status words BCD flag, compare flag, CARRY flag, zero flag and memory pointer enable flag maintain the value before the interrupt is accepted. Therefore, these require initialize.

Example: Status save method in the interrupt service routine

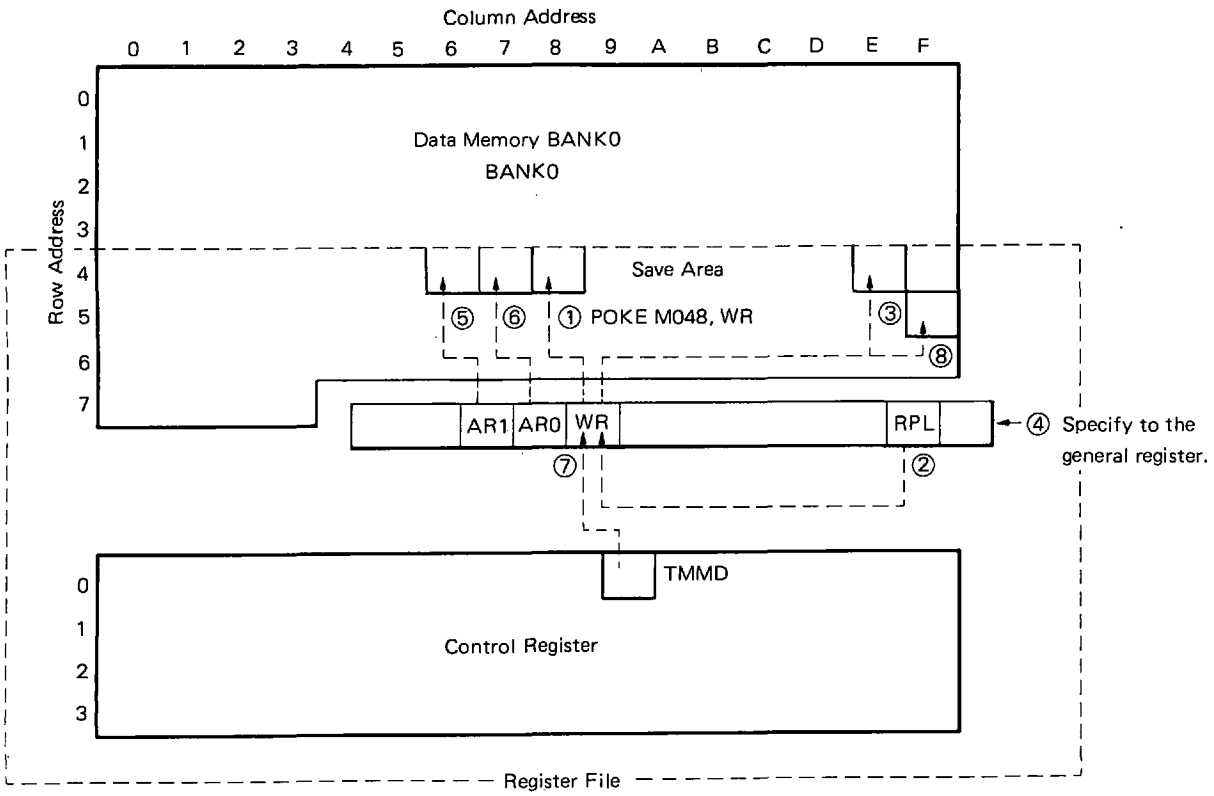


Program Example

```
M046    MEM   0.46H
M047    MEM   0.47H
M048    MEM   0.48H
M04D    MEM   0.4DH

M04E    MEM   0.4EH
M05F    MEM   0.5FH
TMMD    MEM   0.89H

① POKE  M048,  WR    ;  Saves the window register content in M048.
② PEEK  WR,    RPL   ;  Saves the general register pointer content in M04E.
③ POKE  M04E,  WR    ;
④ MOV   RPL,   #0EH  ;  Sets the general register to BANK0's row address 7.
⑤ ST    M046,  AR1   ;
⑥ ST    M047,  AR0   ;  Saves the necessary system register.
        :
⑦ PEEK  WR,    TMMD  ;
⑧ ST    M05F,  WR    ;  Saves the necessary control register.
        :

BANK0                ;  Sets the bank to 0.
MOV     RPL,   #0EH  ;  Sets the general register to BANK0's row address 7.
LD      AR1,   M046  ;  Returns the saved system register.
LD      AR0,   M047  ;
        :
LD      WR,    M05F  ;  Returns the saved control register.
POKE    TMMD,  WR    ;
        :
PEEK    WR,    M04E  ;  Returns the general register pointer.
POKE    RPL,   WR    ;
PEEK    WR,    M048  ;  Returns the saved window register.

                     ;  The interrupt enable (INTE setting) by the EI instruction
                        is executed after the following "RETI"
                        instruction is executed.
                     ;  Returns BANK and IXE, based on the hardware.
```

Fig. 12-4 Save Operation of the System Register and the Control Register which Use the Window Register

① to ⑧ are the numbers of the program examples.



12.7 EXTERNAL INTERRUPT (RMC PIN AND $\overline{V_{SYNC}}$ PIN)

The external interrupt includes two systems, based on the RMC pin (pin No. 15) and the $\overline{V_{SYNC}}$ pin (pin No. 36).

The interrupt request is issued by the rise or fall of the signal added to these pins.

12.7.1 Configurations

Fig. 12-5 shows the configurations of RMC and $\overline{V_{SYNC}}$ pins.

As shown in Fig. 12-5, the signals input from the RMC and $\overline{V_{SYNC}}$ pins are input to the INT latch and the INTVSYN latch respectively as well as to each edge detection circuit.

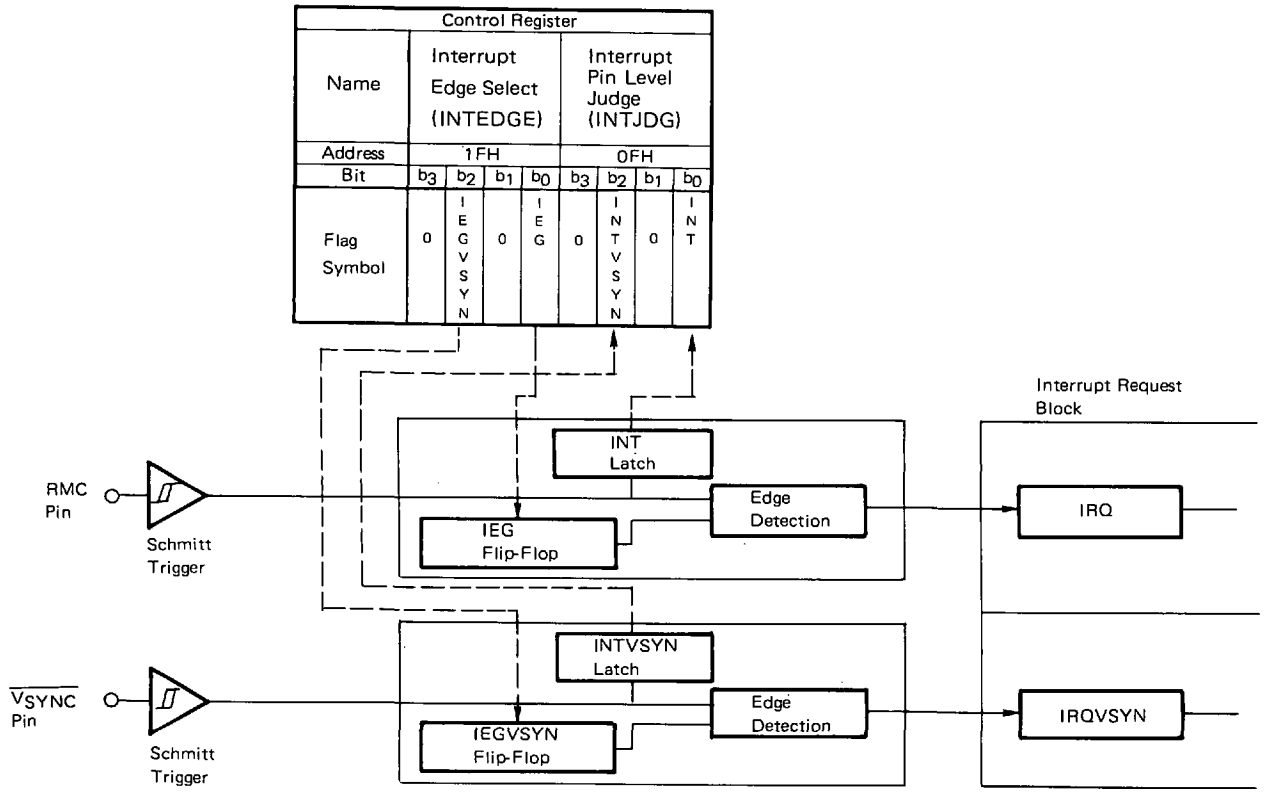
The edge detection circuit outputs the signals input from each pin and the interrupt request signals based on the input from IEG and IEGVSYN flip-flops.

IEG and IEGVSYN flip-flops correspond one-to-one to the IEG flag and IEGVSYN flag of control register's interrupt edge select register (INTEDGE: Address 1FH).

INT and INTVSYN latches correspond one-to-one to the INT flag and the INTVSYN flag of control register's interrupt pin level judge register (INTJDG: Address 0FH).

The RMC pin and the $\overline{V_{SYNC}}$ pin is based on the Schmitt-trigger to avoid erroneous operations caused by noise and does not accept the pulse input below 1 μs.

Fig. 12-5 INT_0 Pin and INT_1 Pin Configurations



12.7.2 Function

The RMC and $\overline{V_{SYNC}}$ pins issue the interrupt request, based on the falling or rising edge aded to each pin.

Selection of the rising edge or the falling edge is made by the IEG and IEGVSYN flags of control register's interrupt edge select register.

Table 13-2 shows the relationship between the IEG and IEGVSYN flags and the interrupt request issuance edge.

Here, be careful about the following points.

If the interrupt request issuance edge is switched by the IEG and IEGVSYN flags, the interrupt request signal is issued sometimes immediately after the switching.

Suppose, as shown in Table 12-3, that the IEG flag is currently set to "0" (falling edge) and the high level is input from the RMC pin. At this time, ensure that the IEG flag is not set (1) because the edge detection circuit decides that the rising edge has been input thus issuing the interrupt request.

For operations after the interrupt request is issued, please refer to **12.2 "Interrupt Function".**

As the signals input to the RMC pin and the $\overline{V_{SYNC}}$ pin are, as shown in Fig. 12-5, input to the INT latch and the INTVSYN latch respectively, it is possible to detect the input signal level by reading the INT and INTVSYN flags.

As these INT and INTVSYN flags are set/reset regardless of the interrupt, they can be used as 2-bit general-purpose input ports when not using the interrupt function.

If the interrupt is not enabled, they can be used as the general-purpose ports which can detect the rising or falling edge by reading the interrupt request flag (IRQ, IRQVSYN). However, the interrupt request flag is not reset automatically here. Therefore, it is necessary to reset it with the program.

Table 12-2 IEG and IEGVSYN Flags and Interrupt Request Issuance Edge

Each Flag Value		Interrupt Request Issuance Edge of Each Pin	
IEG	INTVSYN	RMC Pin	$\overline{V_{SYNC}}$ Pin
0	0	Rise	Rise
0	1	Rise	Fall
1	0	Fall	Rise
1	1	Fall	Fall

Table 12-3 Issuance of the Interrupt Request by Change of the IEG Flag

Change of IEG and IEGVSYN Flags	Status of RMC and $\overline{V_{SYNC}}$ Pins	Presence/Absence of Interrupt Request Issuance	Status of IRQ Flag
1 → 0 (Fall) (Rise)	Low level	Not issued	Maintains the status.
	High level	Issued	The status is set.
0 → 1 (Rise) (Fall)	Low level	Not issued	The status is set.
	High level	Issued	Maintains the status.

12.8 INTERNAL INTERRUPT (TIMER AND SERIAL INTERFACE)

The internal interrupt includes two systems – the timer and the serial interface.

12.8.1 Timer Interrupt

The timer interrupt can issue the interrupt request periodically.

The timer that can be selected is in three types of 100 ms, 20 ms and 5 ms.

For details, see **13 "Timer Function"**.

12.8.2 Serial Interface Interrupt

The serial interface interrupt can issue the interrupt request when the serial out or serial in operation is ended.

The interrupt request is issued mainly by the serial clock.

For details, please refer to **17 "Serial Interface"**.

12.9 MULTIPLE INTERRUPT

As shown in Fig. 12-6, the multiple interrupts are the interrupt method of processing other C and D interrrupts while processing the interrupt factors A and B.

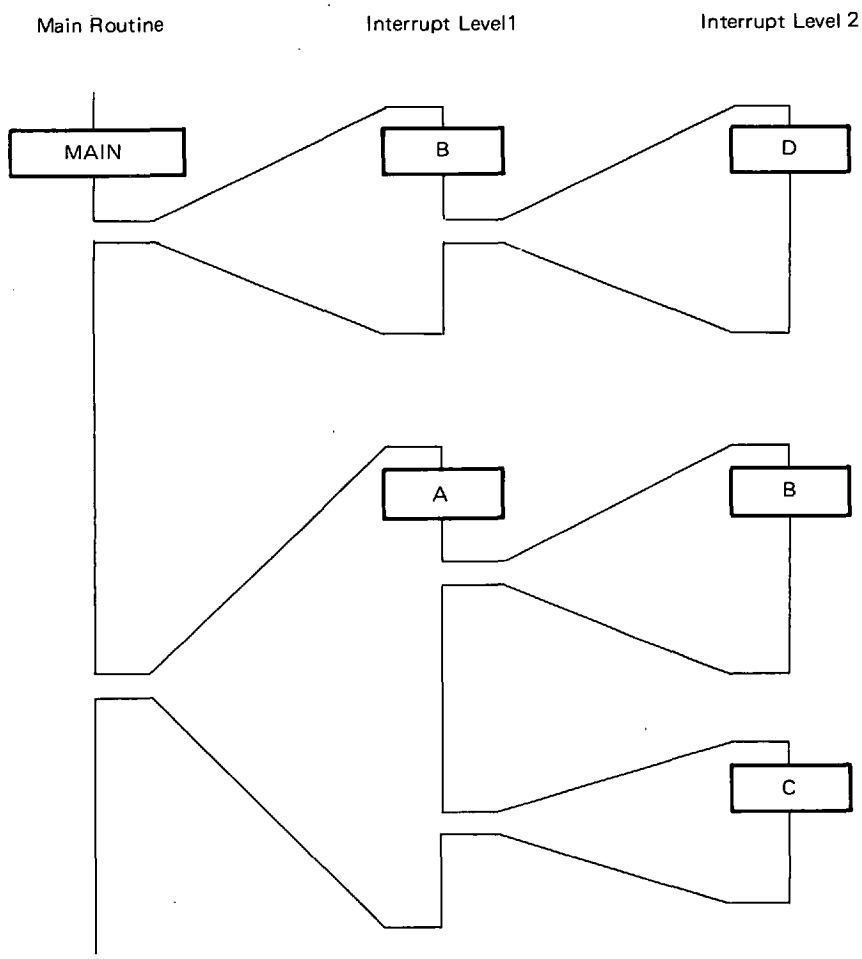
At this time, the interrupt depth is called the interrupt level.

When using the multiple interrupts, attention is required to the following points.

(1) Priority of the interrupt factors
(2) Interrupt level restriction by the interrupt stack
(3) Interrupt level restriction by the address stack register
(4) Saving the system register and the control register

Details of (1), (2), (3) and (4) above are explained in **12.9.1** to **12.9.4**.

Fig. 12-6 Multiple Interrupt Example



12.9.1 Priority Order of the Interrupt Factor

When using multiple interrupts, it is necessary to decide beforehand the priority order of the interrupt factors.

For example, when interrupt factors are A, B, C and D, the priority order is either A = B = C = D or A < B < C < D.

However, if A = B = C = D, the main routine always accepts A, B, C and D interrupts. However, if the C interrupt is accepted for example, the other A, B and D interrupts are inhibited, thus making multiple interrupts meaningless.

If the priority order is A < B < C < D, C must be processed earlier even while the A or B interrupt is being processed. And D interrupt must be processed earlier even while the C interrupt is being processed.

The priority order described previously may well be in the priority order by the hardware or be in the priority order by the software using the interrupt permission flag, as was explained in **12.3 "Interrupt Acceptance Operation"**.

The necessity of decide beforehand on the priority order in the multiple interrupts arises when, for example, in A and B interrupt factors, the A factor issues the request every 10 ms and its interrupt service time is 4 ms and the B factor issues the request every 2 ms and its processing time is 1 ms.

At this time, suppose that there is not priority order between A and B. And, if, by accident, the A interrupt is processed based on the A interrupt request while B interrupt is being processed, the result is that the B interrupt is not processed for several times.

In general, as the interrupt is in many cases used for highly emergent processings, a program is required in which the A < B priority order is applied so that processing of the A interrupt is inhibited while the B interrupt is being processed or the B interrupt is accepted even while the A interrupt is being processed.

When using the interrupt with a non-emergency purpose, it is not always necessary to apply the priority order. However, if the number of interrupt factors exceeds the restriction of the multiple-interrupt level as shown in **12.9.2** and **12.9.3**, it is necessary to decide the priority order so that the interrupt level is not exceeded.

12.9.2 Restriction of the Interrupt Level by the Interrupt Stack

The contents of the system register's bank register and index enable flag are automatically saved in the interrupt stack.

The interrupt stack operation is shown in (a) in Fig. 12-7.

The bank register and the index enable flag are saved in the interrupt stack and at the same time all of them are reset.

As the interrupt stack is in two levels, the bank register and the index enable flag cannot be normally returned as shown in (b) in Fig. 12-7 if multiple interrupts exceeding the two levels are performed.

In other words, the multiple interrupts exceeding the two levels cannot be used.

However, if the bank register and the index enable flag are always fixed in the main routine where the interrupt is enabled and if the multiple interrupts have a clear priority order as shown in Fig. 12-8, it is possible to make multiple interrupts of two levels or more by using the "RET" instruction, which is the subroutine return instruction.

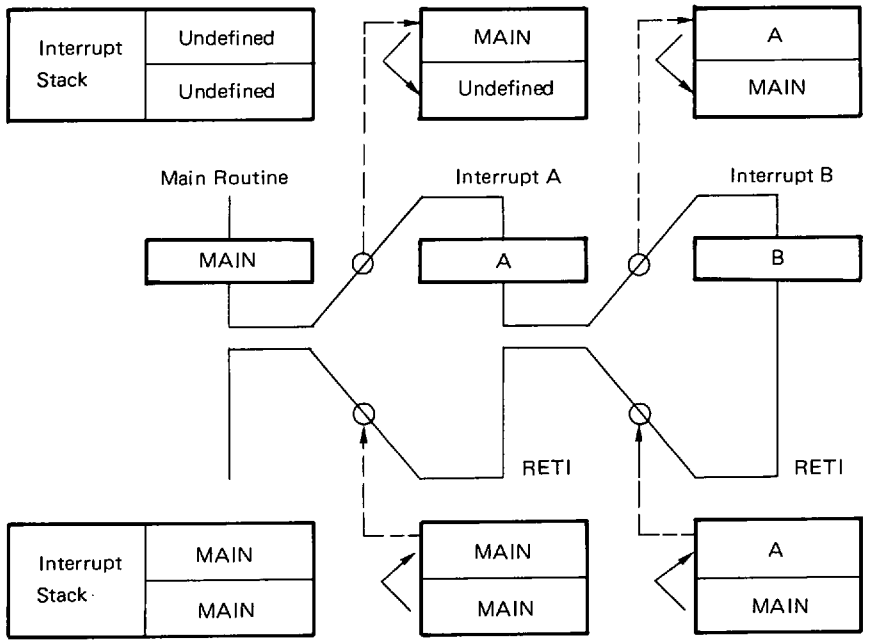
If the multiple interrupts exceed two levels, attention is required because the device operation and the emulator operation differ as shown in Fig. 12-8 and Fig. 12-9.

In other words, the operation in the device interrupt stack is the "discharge type", and that of the emulator is the "rotation type".

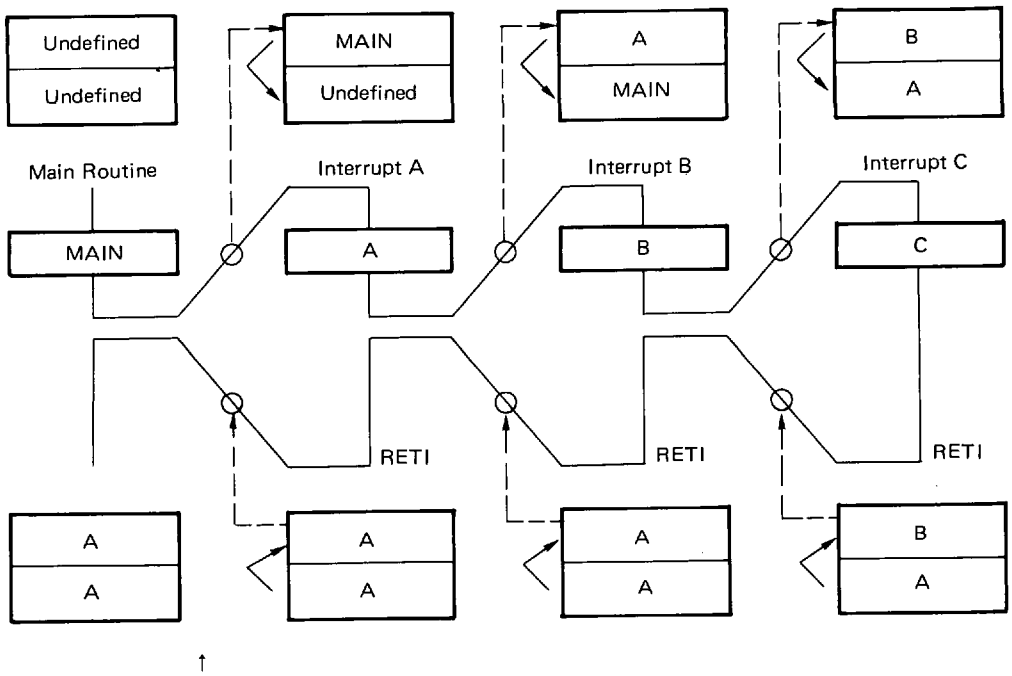
Therefore, please use the "RET" instruction for the final return instruction when using multiple interrupts exceeding the two levels. And, the "RETI" instruction and the "RET" instruction perform the same processings other than the processing of returning the interrupt stack.

Fig. 12-7 Interrupt Stack Operation in Multiple Interrupts

(a) Two-level multiple interrupts



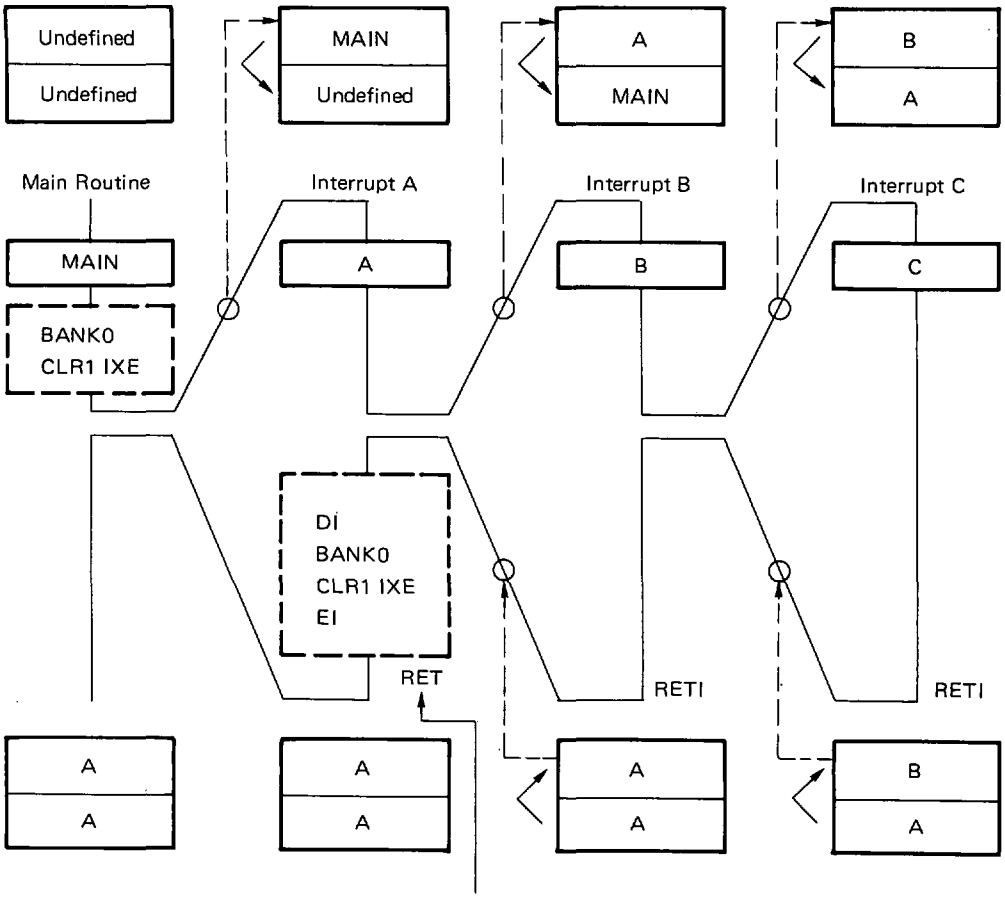
(b) Three-level multiple interrupts



↑

If returned to the main routine at this point, interrupt A's BANK and IXE are returned, thus preventing a normal operation of the main routine.

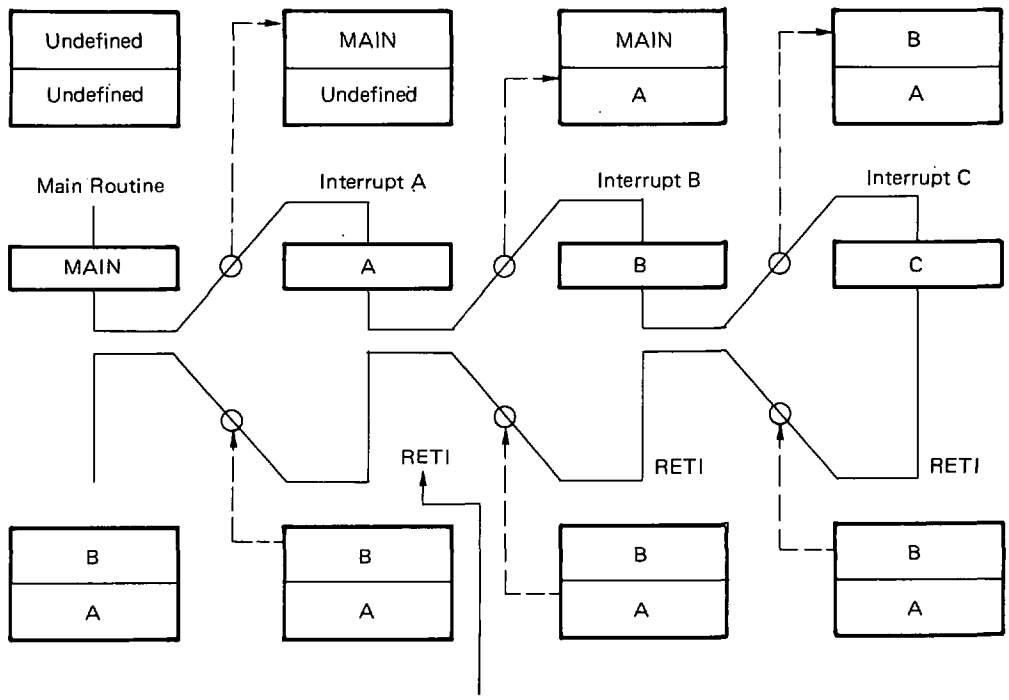
Fig. 12-8 Example of Using the Three-Level Multiple Interrupts



If the priority order of interrupt A is ensured to be lower than that of interrupt B and C, and if the bank register and the index enable flag are always fixed (BANK0 and IXE = 0 in this example) in the main routine which enables interrupt A, the three-level multiple interrupts are possible by using the "RET" instruction after specifying the bank register and the index enable flag of the main routine at the point that the processing of interrupt A is ended.

If the bank register and the index enable flag are exactly the same as the main routine at interrupt A, the "RETI" instruction can be used. However, debugging cannot be made by the "RETI" instruction because the operation at the emulator of the 17K series differs as shown in Fig. 12-9.

Fig. 12-9 Interrupt Stack Operation when Using the 17K Series Emulator



If the "RETI" instruction is used on the emulator, the content of interrupt B's bank register and index enable flag is returned.

12.9.3 Restriction of the Interrupt Level by the Address Stack Register

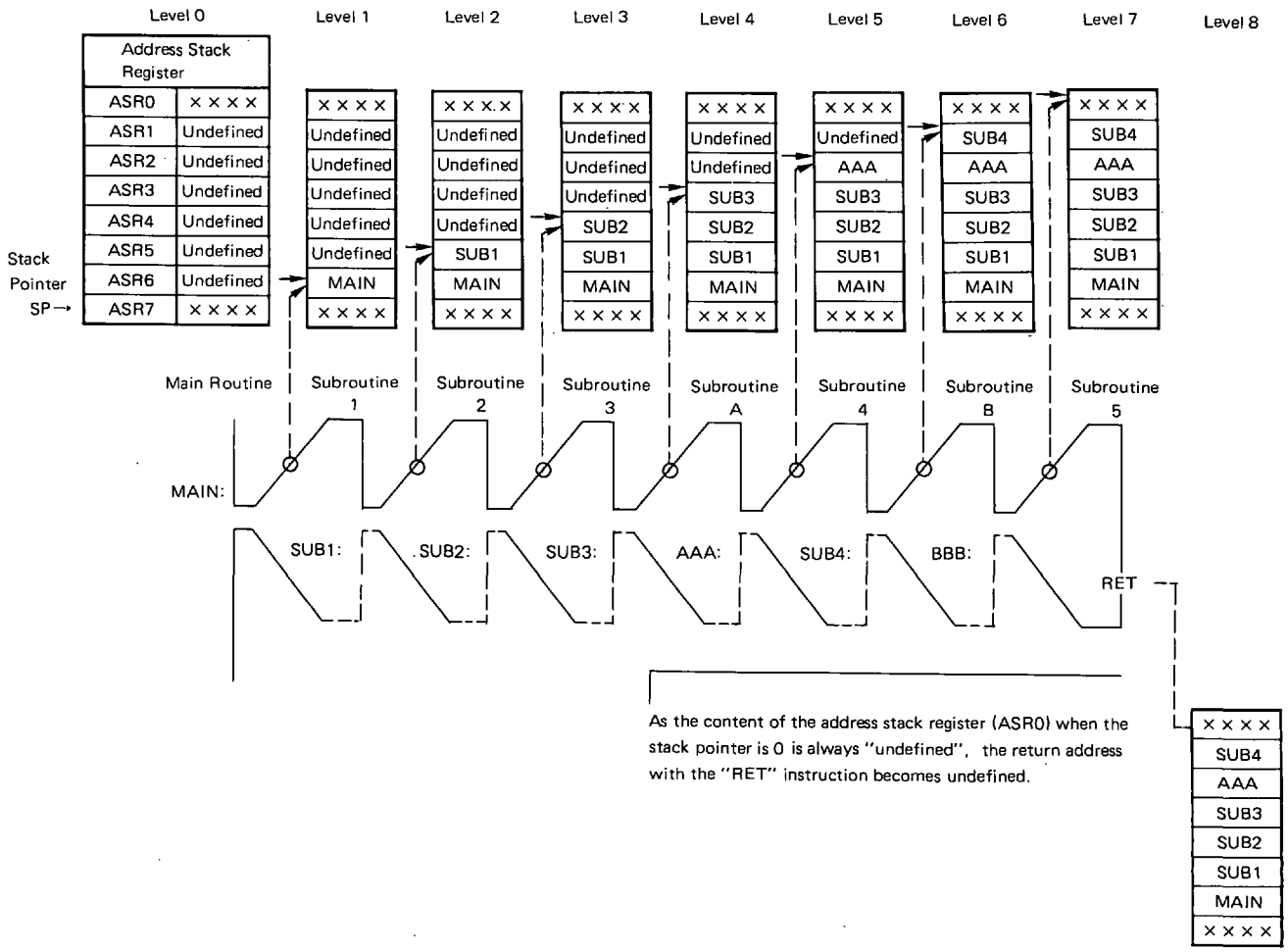
The return address when returning from the interrupt service is automatically saved in the address stack register.

The address stack register can be used in six levels of ASR0 to ASR5 as explained in **5 "Stack"**. As the interrupt factors include the RMC pin, the timer, the $\overline{V_{SYNC}}$ pin and the serial interface, it can be said that there is no restriction on the multiple-interrupt level in using the address stack register only with the interrupt.

However, as the address stack register is used for saving the return address at the time of the subroutine call as well, the multiple-interrupt level is restricted only to the portion of the address stack register level used for the subroutine call.

For example, when four levels are used for the subroutine call as shown in Fig. 12-10, the multiple interrupts can be used only in two levels.

Fig. 12-10 Address Stack Register Operation



12.9.4 Saving the System Register and the Control Register

When using the multiple interrupts, it is necessary to save beforehand the contents of the system register and the control register, which change while the interrupt is being processed.

At this time, it is necessary to secure the save areas for these contents separately for each interrupt factor.

It is also necessary to inhibit the interrupt currently accepted and the interrupt lower in the priority order than this interrupt, and to enable the interrupt with higher priority.

At this time, the interrupt with higher priority must be enabled with higher priority because it is higher in emergency.

Therefore, it is desirable to save the contents of the system register and the control register after "processing enabling the interrupt of higher priority".

Examples of processing the "enabling of the interrupt of higher priority" and "saving the contents of the system register and the control system, etc." in the interrupt service routine are shown below.

Example: Interrupt enable and save processing examples in multiple interrupts

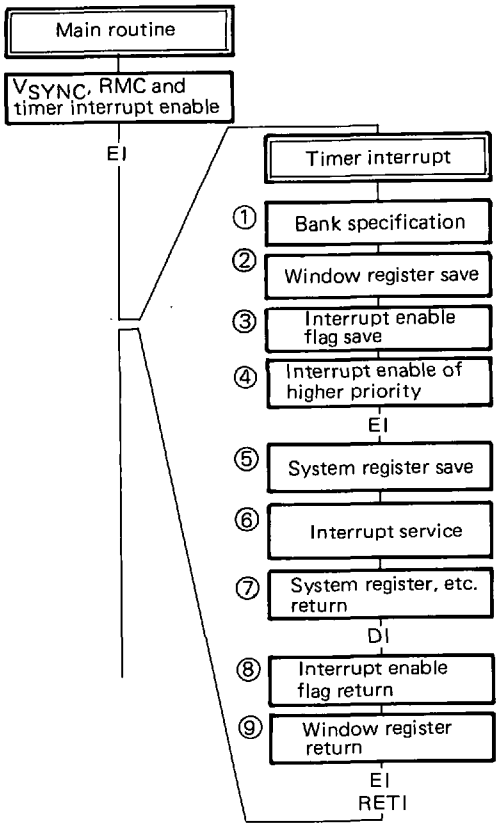
Suppose that the RMC pin, the $\overline{V_{SYNC}}$ pin and the timer interrupt are used in the following priority order (software priority):

$\overline{V_{SYNC}}$ pin > Timer > RMC pin

Besides the timer interrupt was accepted at level 1.

The program example and the flowchart at this time are shown below.

Flowchart **Program Example**




```
MOV   WR,     #0111B ;  Interrupts of the timer, the RMC pin and VSYNC
POKE  INTPM,  WR     ;  pin are enabled by the interrupt permission register.
                     ;  The contents of the bank register, the index enable
                     .  flag and the program counter are automatically.
BANK1 (Intrinsic Macro) ;  The memory bank which saves data shall be BANK1.
POKE  M1,     WR     ;  Saves the window register.
PEEK  WR,     INTPM  ;  Saves the interrupt permission register in M2 of
POKE  M2,     WR        the data memory.
MOV   WR,     #0100B ;  Interrupt enable of the VSYNC pin by the interrupt
POKE  INTPM,  WR     ;  permission register.
PEEK  WR,     RPL    ;  Saves the system register and the control register
POKE  M3,     WR     ;  other than the window register and the interrupt
                        permission register.
                     ;  Returns the system register and the control register
PEEK  WR;     M3     ;  other than the window register and the interrupt
POKE  INTPM,  WR     ;  permission register.
PEEK  WR,     M2     ;  Returns the interrupt permission register.
POKE  INTPM,  WR     ;
PEEK  WR,     M1     ;  Returns the window register.
```

The data memory bank in which the system register content is saved is specified in ①.

At this time, if the interrupt is accepted, the bank becomes BANK0. Therefore, if the memory in which data is to be saved is BANK0, this instruction is not necessary.

In ②, the window register content is saved in M1 of the data memory.

At this time, as the "POKE" instruction is used, the address of data memory M1 needs to be 40H or more. As the window register is used as the work area of the data save following this one, it needs to be saved initially.

In ③, each of the interrupt enable flags (IP, IPTM, IPVSYN) at the time the interrupt has been accepted is saved. This is because, although, in the case of the example, it is necessary to enable the interrupts of all the RMC pin, $\overline{V_{SYNC}}$ pin and the timer when returning to the main routine, as the timer interrupt is higher in the priority order than the RMC pin, it must be returned by inhibiting the interrupt of the RMC pin if accepted while the interrupt of the RMC pin is being processed.

In ④, the interrupt of the $\overline{V_{SYNC}}$ pin which has a higher priority than the timer interrupt is enabled. After this, all the interrupts are enabled by the "EI" instruction.

As processings need to be performed with all the interrupts inhibited in ①, ②, ③ and ④, even the interrupt by the $\overline{V_{SYNC}}$ pin, which has the highest priority, is inhibited during this time.

The system register and the control register are saved and returned in ⑤ and ⑥. However, here, it is all right for the interrupt of a high priority to be enabled.

This is because, if the save is processed in the same manner when the interrupt of the high-priority $\overline{V_{SYNC}}$ pin has been accepted, the contents of the system register and the control register do not change when returned from processing the $\overline{V_{SYNC}}$ pin interrupt.

In ⑦ and ⑧, the interrupt enable flag and the window register are returned.

At this time, all the interrupts must be inhibited.

This is because, if the instruction of ⑦ which enables the interrupt is executed in the "EI" state, the window register return in ⑧ is not performed but rather the window register is saved once again in ② when the timer interrupt request has been issued accidentally, thus making the return of the window register content impossible.

13. TIMER FUNCTIONS

The timer functions are used for time management in program creation.

13.1 TIMER CONFIGURATION

Fig. 13-1 shows the timer configuration.

As shown in this figure, the timer consists of a timer carry flip-flop (FF) block and a timer interrupt block.

The timer carry FF and a clock generator for timer interrupt time setting consist of an 8 MHz frequency divider, selector A, selector B, a bias circuit and a timer mode select register (TMMD: Address 09H) which is a control register.

13.1.1 Timer Carry FF Block Configuration

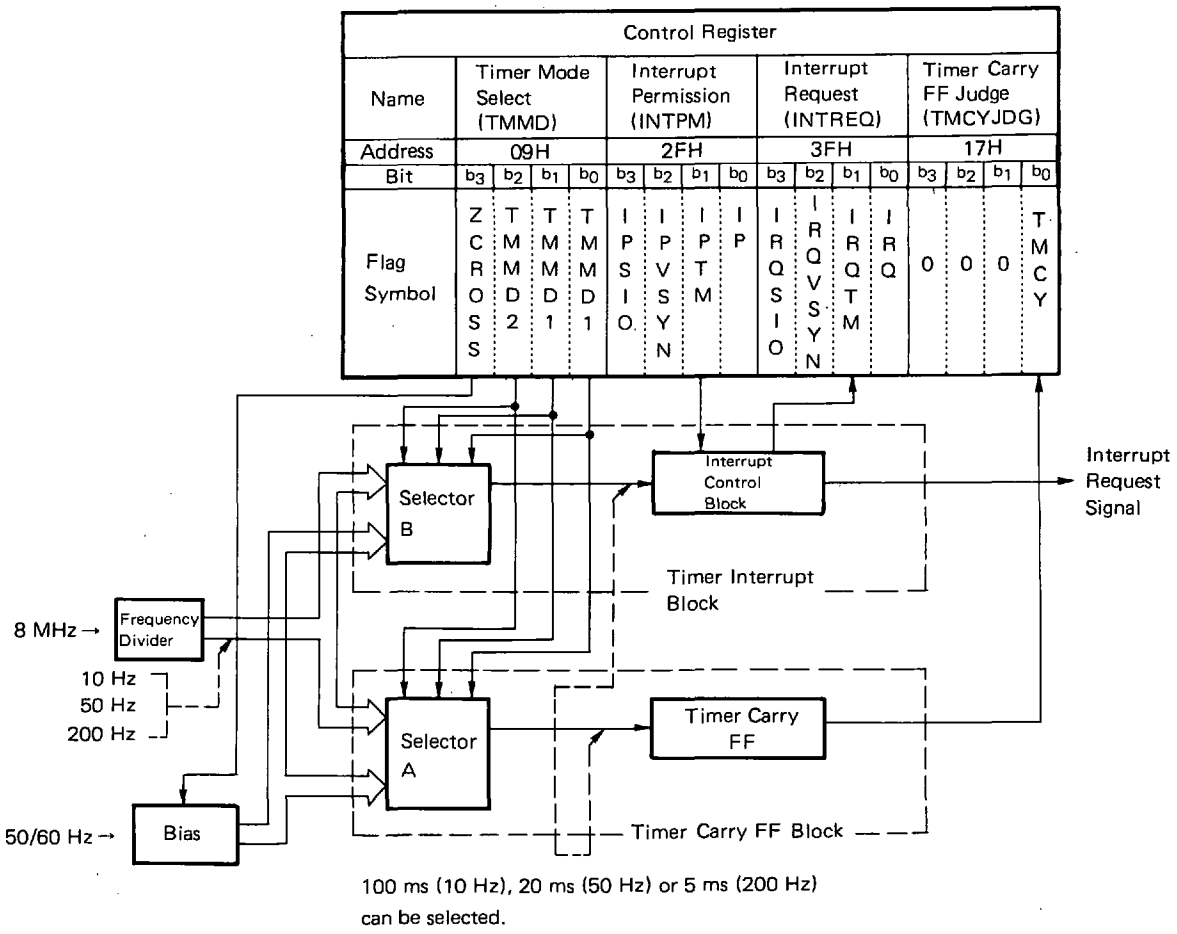
As shown in Fig. 13-1, the timer carry FF block consists of selector A, a timer carry FF and a timer carry FF judge register (TMCYJDG: Address 17H) which is a control register.

13.1.2 Timer Interrupt Block Configuration

As shown in Fig. 13-1, the timer interrupt block consists of selector B, an interrupt control block, an interrupt permission register (INTPM: Address 2FH) and an interrupt request register (INTREQ: Address 3FH). Both INTPM and INTREQ are control registers.

Fig. 13-1 Timer Configuration

Control Register															
Name	Timer Mode Select (TMMD)				Interrupt Permission (INTPM)				Interrupt Request (INTREQ)				Timer Carry FF Judge (TMCYJDG)			
Address	09H				2FH				3FH				17H			
Bit	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Flag Symbol	ZCROSS	TMMD2	TMMD1	TMMD1	IPSIO	IPVSYN	IPTM	IP	IRQSIO	IRQVSYN	IRQTM	IRQ	0	0	0	TMCY



100 ms (10 Hz), 20 ms (50 Hz) or 5 ms (200 Hz) can be selected.

13.2 TIMER FUNCTIONS

The timer carries out two functions, timer carry FF detection and timer interrupt.

With the first function, the timer carries out time management by detecting by a program the timer carry FF status to be set at regular intervals. With the second function, the timer carries out time management by applying an interrupt at regular intervals.

The timer carry FF set (1) timing and the timer interrupt issue timing are controlled by timer time set pulses output from selectors A and B, respectively.

10 Hz (100 ms), 50 Hz (20 ms) and 200 Hz (5 ms) timer time set pulses are available by setting data to the timer mode select register.

The timer mode select register is also used to select the internal timer mode or the external timer mode. In the internal timer mode the pulse obtained by dividing the 8 MHz device operation frequency is used as the time base for input to selectors A and B. In the external timer mode 50 Hz or 60 Hz input to the $P1B_3$/TMIN pin (23 pin) is used.

The timer mode select register is further used to determine whether the pulse input to the $P1B_3$/TMIN pin should be divided by 5 or 6.

The timer time set pulse is used for combinations of the timer carry FF and the timer interrupt.

Fig. 13-2 shows the relations between the timer mode select register and the timer time set pulse.

In the internal timer mode the timer time set pulse is created by dividing the 8 MHz device operation frequency. Thus, if 8 MHz is shifted, the timer time set pulse will be shifted by the same proportion.

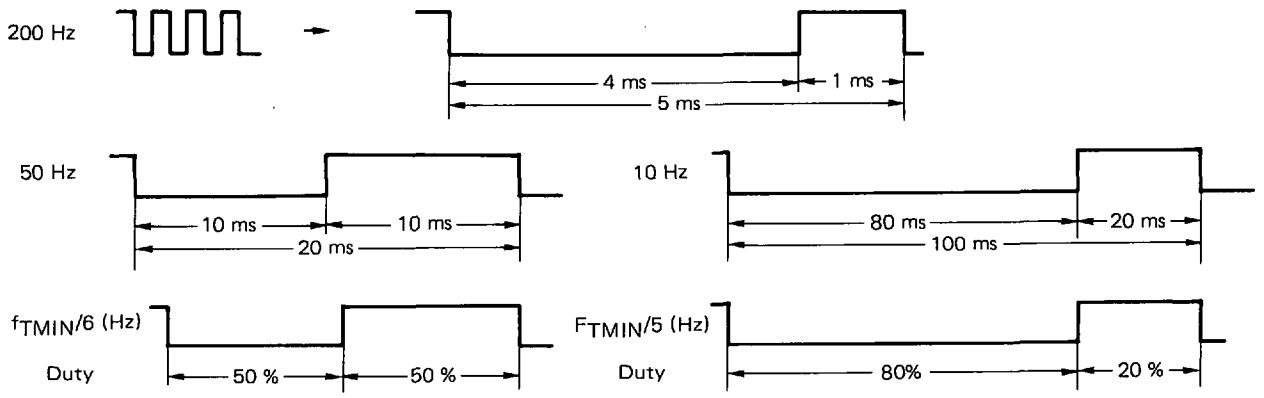
Fig. 13-2 Relations between Timer Mode Select Register and Timer Time Set Pulse

Control Register				
Name	Timer Mode Select (TMMD)			
Address	09H			
Read/Write	R/W			
Bit	b_3	b_2	b_1	b_0
Flag symbol	ZCROSS	TMMD2	TMMD1	TMMD0

			Timer Carry FF Set Pulse Frequency (Time) Select		Timer Interrupt Pulse Frequency (Time) Select	
0	0	0	10 Hz (100 ms)	Internal timer	200 Hz (5 ms)	Internal timer
0	0	1	200 Hz (5 ms)	Internal timer	10 Hz (100 ms)	Internal timer
0	1	0	10 Hz (100 ms)	Internal timer	50 Hz (20 ms)	Internal timer
0	1	1	200 Hz (5 ms)	Internal timer	50 Hz (20 ms)	Internal timer
1	0	0	f_{TMIN}/5 Hz (5/f_{TMIN} s)	External timer	200 Hz (5 ms)	Internal timer
1	0	1	200 Hz (5 ms)	Internal timer	f_{TMIN}/5 Hz (5/f_{TMIN} s)	External timer
1	1	0	f_{TMIN}/6 Hz (6/f_{TMIN} s)	External timer	200 Hz (5 ms)	Internal timer
1	1	1	200 Hz (5 ms)	Internal timer	f_{TMIN}/6 Hz (6/f_{TMIN} s)	External timer

f_{TMIN} is the P1B3/TMIN pin input frequency (50 Hz or 60 Hz).

0	Bias circuit stop
1	Bias circuit in operation



13.3 TIMER CARRY FLIP-FLOP (TIMER CARRY FF)

The timer carry FF is set (1) at the rising edge of the timer carry FF set pulse which is set by the timer mode select register.

The timer carry FF contents correspond to the least significant bit (TMCY flag) of the timer carry FF judge register bitwise. When the timer carry FF is set (1), the TMCY flag is simultaneously set (1).

When the contents are read to the window register by "PEEK" instruction the TMCY flag is reset (0) (Read & Reset).

When the TMCY flag is reset (0), the timer carry FF is simultaneously reset (0).

In other words, a timer with the time set by the timer mode select register can be created by reading the TMCY flag using a program.

A program example is described in **13.3.1.**

When using the timer carry FF, note the following.

Upon power-on reset, the timer carry FF is disabled for set. It is not set until the TMCY flag contents are read by "PEEK" instruction.

In other words, "0" is read when the TMCY flag is first read after power-on reset. After that, "1" is set at intervals set by the timer mode select register.

The timer carry FF controls the CE reset timing.

This means that when the CE pin changes from the low level to the high level, CE reset is applied at the timing when the timer carry FF is set next time.

Thus, power failure can be detected by reading the TMCY flag contents upon system reset (power-on reset and CE reset). For details, refer to **13.4 "Timer Carry FF Operating Precautions"** and **15 "Reset Functions".**

Because the TMCY flag is a read dedicated flag, device operations are not affected if a write operation is carried out by "POKE" instruction. However, if a 17K series assembler is used, an "error" occurs.

13.3.1 An Example of Timer Operation by TMCY Flag

A program example is shown below.

Example:

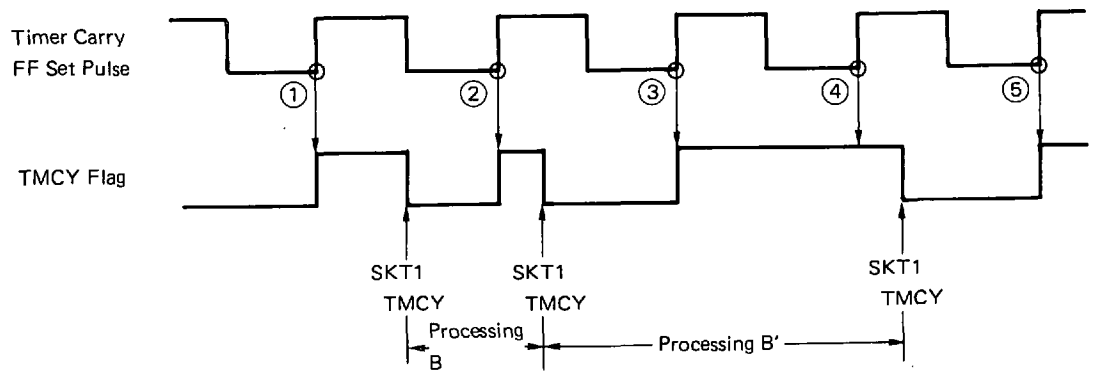
```
INITFLG NOT ZCROSS, NOT TMMD2, NOT TMMD1, NOT TMMD0
                        ; Intrinsic macro
                        ; Timer carry FF set time is set to 100 ms.
LOOP1:
  MOV   M1,   #0110B
LOOP2:
  SKT1  TMCY            ; Intrinsic macro
                        ; TMCY flag is tested. If "0", branch to NEXT.
  BR    NEXT
  ADD   M1,   #0100B
                        ; 4 is added to data memory M1.
  SKT1  CY              ; Intrinsic macro
                        ; CY flag is tested.
  BR    NEXT            ; If "0", branch to NEXT.
  [Processing A]        ; If "1", processing A is executed.
  MOV   M1,   #0110B
NEXT:
  [Processing B]        ; Processing B is executed and branch to LOOP.
  BR    LOOP
```

In the above program, processing A is executed every second.
When creating a program, note the following.

(1) The TMCY flag detection time must be shorter than time carry FF set (1) time.

This is because, according to an example described as above, as shown in Fig. 13-3, the timer carry FF will not be set (1).

Fig. 13-3 TMCY Flag Detection and Timer Carry FF



Because processing B' takes time after the TMCY flag set in ② has been detected, the TMCY flag status set in ③ cannot be detected.

13.3.2 Timer Errors by TMCY Flag

Timer errors by the TMCY flag consist of an error due to TMCY flag detection time and an error due to timer carry FF set time change.

These are described in the following (1) and (2) respectively.

(1) Error due to TMCY flag detection time

As described in **13.3.1**, the TMCY flag detection time interval must be shorter than the timer carry FF set (1) time.

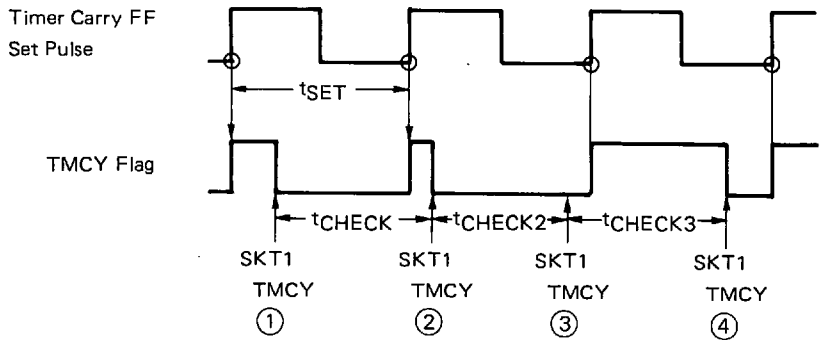
In other words, when TMCY flag detection time interval is t_{CHECK} and the timer carry FF set time interval (100 ms or 5 ms) is t_{SET}, the following relationship must be maintained.

$$t_{CHECK} < t_{SET}$$

Then, the timer error upon TMCY flag detection is the following as shown in Fig. 13-4.

$$0 < \text{Error} < t_{CHECK}$$

Fig. 13-4 Error Due to TMCY Flag Detection Time



As shown in Fig. 13-4, if the TMCY flag is detected to be "1" in (2), the timer is updated.

If the TMCY flag is detected to be "0" in ③, the timer is not updated until another detection is made in ④.

That is, the timer has a longer time by t_{CHECK3}.

(2) Error due to timer carry FF set time change

Timer carry FF set time is set by the timer mode select register flags TMMD2, TMMD1 and TMMD0.

As shown in Figs. 13-1 and 13-2, three timer time set pulses, 200 Hz, 10 Hz and the external timer, are available.

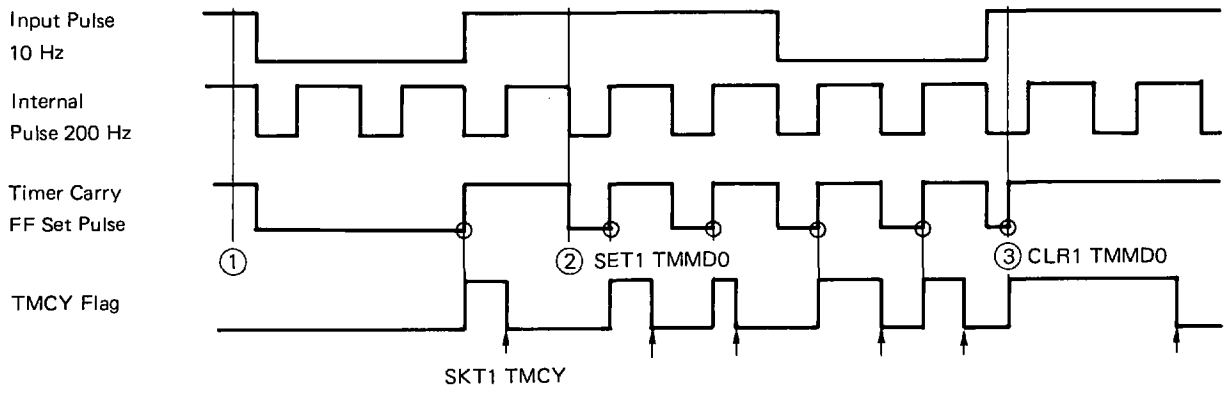
Each pulse operates independently.

Thus, if the timer time set pulse is changed by the TMMD2, TMMD1 and TMMD0 flags, an error will occur as in the following example.

Example:

```
; ①
INITFLG NOT ZCROSS, NOT TMMD2, TMMD1, NOT TMMD0
                        ; Intrinsic macro
Processing A            ; Timer carry FF set pulse is set to 10 Hz (100 ms)
; ②
SET1    TMMD0           ; Intrinsic macro
                        ; Timer carry FF set pulse is set to 200 Hz (5 ms).
Processing A
; ③
CLR1    TMMD0           ; Intrinsic macro
                        ; Timer carry FF set pulse is set to 10 Hz (100 ms).
```

As a result, the timer carry FF set pulse is changed as follows.



As shown above, if the changed pulse rises when the timer carry FF set time is changed, the TMCY flag holds the previous status (② in the figure). If the changed pulse falls, the TMCY flag is set (1) (③ in the figure).

The error remains as follows until the first TMCY flag is set after the timer carry FF set time is changed as shown in Fig. 13-5.

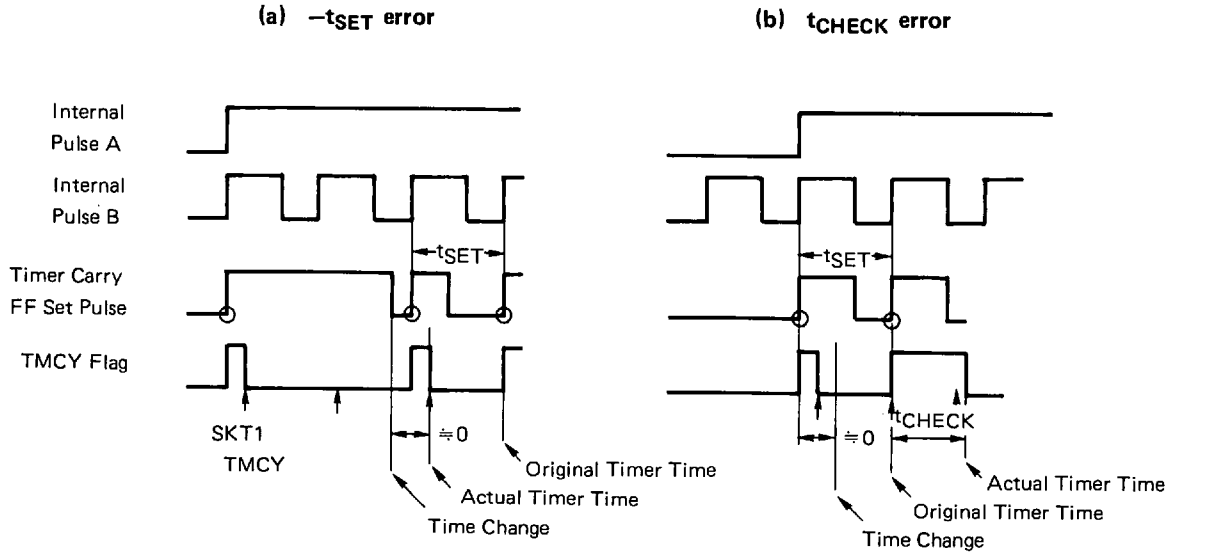
$-t_{SET} < \text{Error} < t_{CHECK}$

t_{SET} : The changed timer carry FF set time

t_{CHECK}: TMCY flag detection time

The 4 Hz, 10 Hz, 200 Hz and 1 KHz internal pulses each have a phase difference. Because this phase difference is shorter than the changed pulse time, it is included in the above error.
For details of each pulse phase difference, refer to **13.6 "Timer Interrupt Applying Precautions"**.

Fig. 13-5 Error when Timer Carry FF Set Time is Changed from A to B



If TMCY flag is detected to be "1" just after the timer time is changed, $-t_{SET}$ error results.

If the timer time is changed just after TMCY flag is detected, t_{CHECK} error results because TMCY flag is reset once.

13.4 TIMER CARRY FF OPERATING PRECAUTIONS

The timer carry FF is used not only for the timer functions but also for the reset synchronous signal upon CE reset.

That is, if the next timer carry FF set pulse rises after the CE pin has changed from the low level to the high level, CE reset is applied.

In this case, note the following.

(1) The sum of the timer update time and the TMCY flag detection time interval must be shorter than the timer carry FF set time.

(2) If a program is created to cause the selected timer to operate irrespective of CE reset after power-on reset, it is necessary to correct upon each CE reset.

(3) With the reset synchronous signal for TMCY flag detection and CE reset, TMCY flag detection is given priority. Thus, if the two operations overlap, CE reset is delayed by one operation.

(1) to (3) above will be described in **13.4.1** to **13.4.3** respectively.

13.4.1 Timer Update Time and TMCY Flag Detection Time Interval

As described in **13.3.1**, TMCY flag detection time interval t_{SET} must be shorter than the timer carry FF set time.

If the timer update time is long when the TMCY flag detection time interval is short, timer processing may not be executed normally if CE reset is applied.

Therefore, it is necessary to meet the following condition.

$$t_{CHECK} + t_{TIMER} < t_{SET}$$

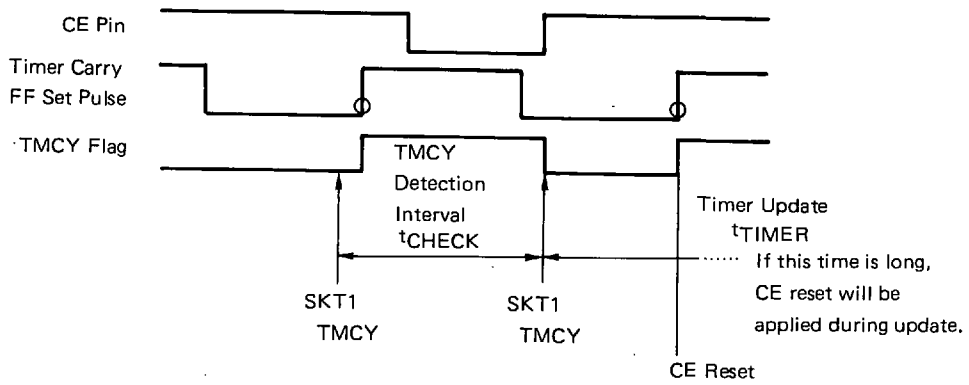
t_{CHECK} : TMCY flag detection time interval
t_{TIMER} : Timer update time
t_{SET} : Timer carry FF set time

An example is shown below.

Example: Timer update and TMCY flag detection time interval example

```
START:                  ; Program address 0000H
  INITFLG NOT TMMD3, NOT TMMD2, NOT TMMD1, NOT TMMD0
                        ; Intrinsic macro
                        ; Timer carry FF set time is set to 100 ms.
TIMER:
  ; ①
  SKT1   TMCY           ; Intrinsic macro
                        ; TMCY flag is tested.
  BR     AAA            ; If "0", branch to AAA.
  [Timer Update]
  BR     TIMER
AAA:
  [Processing A]
  BR     TIMER
```

The timing chart of the above programming example is shown below.



13.4.2 Timer Carry FF Correction in CE Reset

This section describes a timer correction example in CE reset.

As shown in the example, timer correction in CE reset may be necessary when "the timer carry FF is simultaneously used for power failure detection and as a clock timer".

Upon power-on reset, the timer carry FF is reset (0) and remains disabled for set until the TMCY flag is read by the "PEEK" instruction.

When the CE pin changes from the low level to the high level, CE reset is applied at the rising edge of the timer carry FF set pulse. At this point the system starts with the TMCY flag set (1).

That is, power-on reset or CE reset can be judged by detecting the TMCY flag status upon system reset (power-on reset and CE reset). When the TMCY flag status is "0" or "1", power-on reset or CE reset (power failure detection) can be identified, respectively.

Upon CE reset, the clock timer must continue operating.

However, because the TMCY flag is reset (0) by TMCY flag read due to power failure detection, the TMCY flag set (1) status is missed by one time.

Thus, if CE reset is judged because of power failure detection, it is necessary to update the clock timer.

An example is shown below.

For details of power failure detection, refer to **15.6 "Power Failure Detection"** as well.

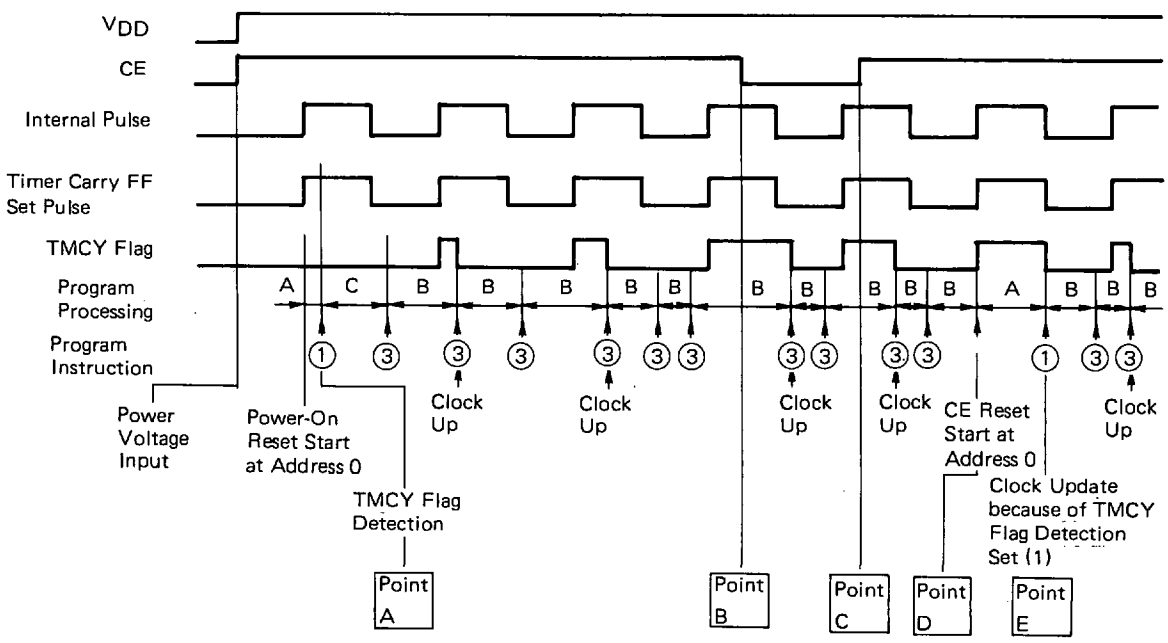
Example: Example of timer correction upon CE reset

When detecting power failure and updating the clock using the timer carry FF

```
START:                  ; Program address 0000H
  [Processing A]
  ; ①
  SKT1   TMCY           ; Intrinsic macro
                        ; TMCY flag is tested.
  BR     INITIAL        ; If "0", branch to INITIAL (power failure detection).
BACKUP:
  ; ②
  [100 ms clock update] ; Clock correction because of backup (CE reset)
LOOP:
  ; ③
  [Processing B]        ; While executing processing B
  SKF1   TMCY           ; TMCY flag is tested and the clock is updated.
  BR     BACKUP
  BR     LOOP
INITIAL:
  INITFLG NOT TMMD3, NOT TMMD2, NOT TMMD1, NOT TMMD0
                        ; Intrinsic macro
                        ; Because of power failure (power-on reset), the timer carry FF set time is set to
                          100 ms and processing C is executed.
  [Processing C]
  BR     LOOP
```

The timing chart of the above program is shown in Fig. 13-6.

Fig. 13-6 Timing Chart



As shown in Fig. 13-6, upon power-on reset, the program starts at address 0000H at the rising edge of the internal 10 Hz pulse.

Next, when the TMCY flag is detected at point A, the TMCY flag has been reset (0) after power-on, the state is judged to be power failure (power-on reset).

Thus, execute "Processing C" and set the timer carry FF set pulse to 100 ms.

Because the TMCY flag contents have been read once, the TMCY flag will be set (1) every 100 ms.

If the CE pin becomes low level at point B and high level at point C, the program will count up the clock while executing "Processing B" unless the clock stop instruction has not been executed.

Because the CE pin has risen from the low to the high level at point C, CE reset will be applied at point D at the rising edge of the next timer carry FF set pulse and the program will start at address 0000H.

If the TMCY flag is detected at point E, the state is judged to be backup (CE reset) because the TMCY flag has been set (1).

As is clear from the above figure, if the clock is not updated by 100 ms at point E, the clock will be delayed by 100 ms each time CE reset is applied.

If processing A takes 100 ms or more for power failure detection at point E, TMCY flag will be missed twice. To prevent that from occurring, processing A must be carried out within 100 ms.

Thus, TMCY flag detection for power failure detection must be carried out within the timer carry FF set time after the program has started at address 0000H.

13.4.3 If TMCY Flag Detection and CE Reset Occur Simultaneously

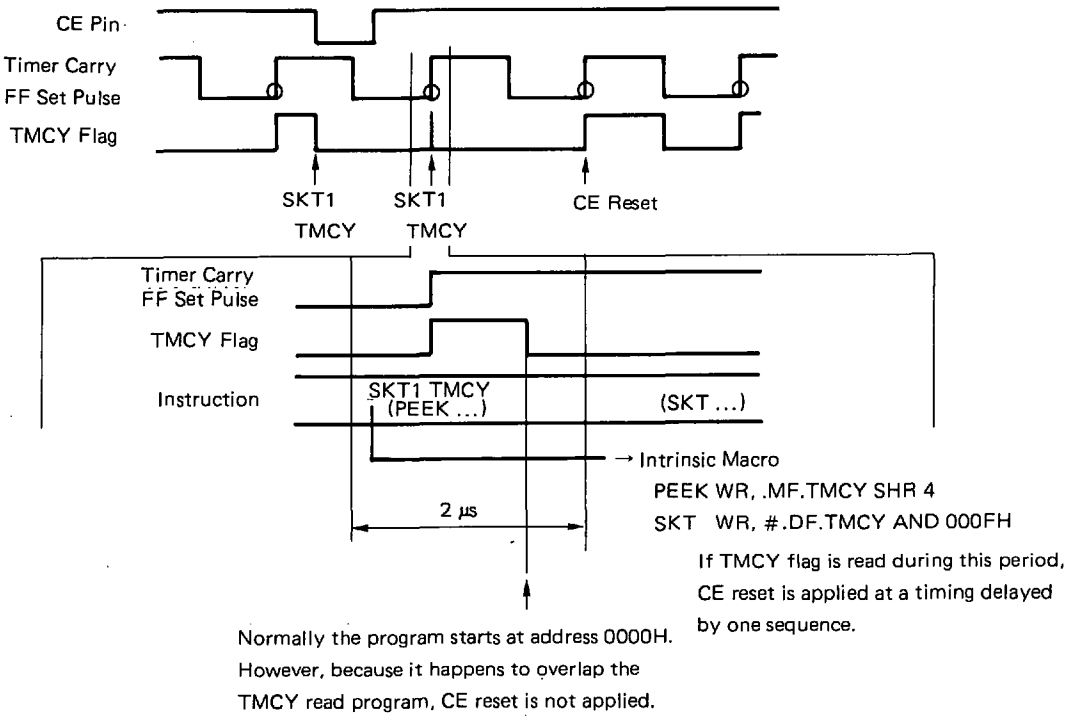
As described in **13.4.2**, CE reset is applied simultaneously upon TMCY flag set (1).

If the TMCY flag read instruction happens to be generated upon CE reset, it is given priority.

Thus, if the TMCY flag which is generated after the CE pin changes from the low level to the high level is set (at the timer carry FF set pulse rising edge) simultaneously when the TMCY flag read instruction is generated, CE reset is applied to the "timing when the next TMCY flag is set".

This operation is shown in Fig. 13-7.

Fig. 13-7 Operation when CE Reset is Applied Simultaneously when TMCY Flag Read Instruction is Generated



Thus, CE reset can never be applied if the program is one that detects the TMCY flag cyclically and in which the TMCY flag detection time interval matches the TMCY flag set time.

In other words, the following must be noted.

Because one instruction cycle is 2 μs (1/500 KHz), a program which detects the TMCY flag every 500 instructions, for example, will read the TMCY flag every 2 μs x 500 = 1 ms.

Whether the timer time set pulse 5 ms or 100 ms is selected, CE reset will not be applied forever once TMCY flag set and detection occur simultaneously.

That is, do not create any cyclic program with n given as

$$\frac{t_{SET} \times X}{500} = n \text{ (n: Natural number)}$$

where t_{SET} : TMCY flag set time

X : No. of steps (TMCY flag read instruction cycle)

The following is an unrecommendable programming example which satisfies the above conditions.

Example:

```
    Processing A
    INITFLG NOT TMMD3, NOT TMMD2, NOT TMMD1, TMMD0
                        ; Intrinsic macro
                        ; Timer carry FF set pulse is set to 5 ms.
LOOP:
    ; ①
    SKT1   TMCY         ; Intrinsic macro
    BR     BBB
AAA:
    496 steps
    BR     LOOP
BBB:
    496 steps
    BR     LOOP
```

In this example ① TMCY flag read instruction is repeated every 500 instructions. Thus, if the TMCY flag is set at the timing of ① instruction, CE reset can no longer be applied.

Also, because the instruction execution time is 12 μs (1/83.33 kHz) during IDC operation, do not create a cyclic program which satisfies the following conditions.

$$\frac{t_{SET} \times X}{83.33} = n \text{ (n: Natural number)}$$

where t_{SET} : TMCY flag set time

X : No. of steps (TMCY flag read instruction cycle)

13.5 TIMER INTERRUPT

A timer interrupt request is issued at the falling edge of the timer interrrupt pulse set by the timer mode select register.

The timer interrupt request has a 1:1 relationship with the IRQTM flag of the interrupt request register. When a timer interrupt request is issued, the IRQTM flag is set (1).

That is, when the timer interrupt pulse falls, the IRQTM flag is set (1).

As described in **12 "Interrupt"**, not only the generation of an interrupt request but also the execution of "EI" instruction enabling all interrupts and timer interrupt enable must be set for timer interrupt.

Timer interrupt enable is set by setting (1) the IPTM flag of the interrupt permission register.

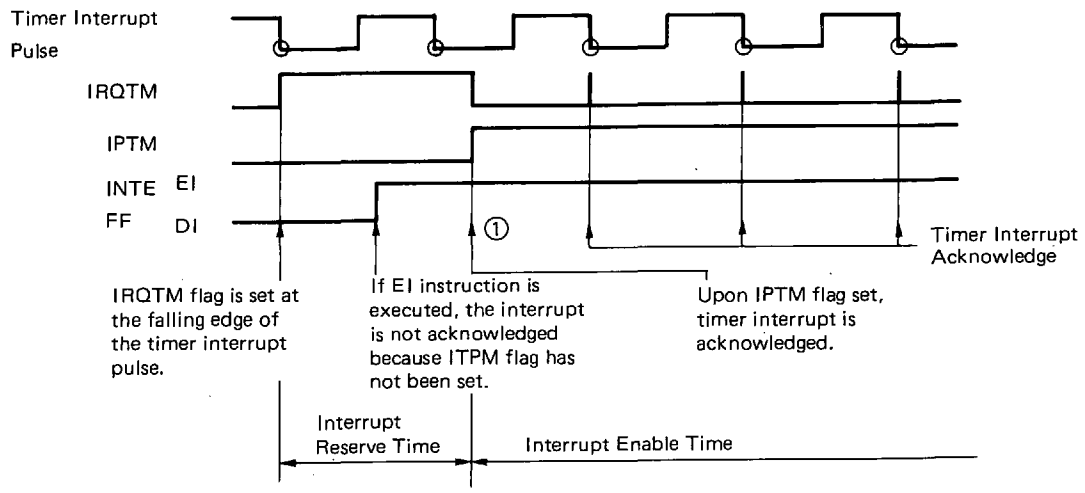
Thus, a timer interrupt is acknowledged when the IRQTM flag is set (1) after the "EI" instruction has been executed and the IPTM flag has been set (1).

When a timer interrupt is acknowledged, the program flow will proceed to program memory address 0003H.

Upon acknowledgment of the interrupt, the IRQTM flag is reset (0).

Fig. 13-8 shows the relations between the timer interrupt pulse and the IRQTM flag.

Fig. 13-8 Relationship between Timer Interrupt Pulse and IRQTM Flag



Note point ① in Fig. 13-8. Once the IRQTM flag is set when time interrupt is disabled by the "DI" instruction or the IPTM flag, timer interrupt is acknowledged when the "EI" instruction is executed and the IPTM flag is set next time.

In this case, the interrupt request can be released by writing "0" to the IRQTM flag.

Conversely, writing "1" to the IRQTM flag is equal to an interrupt request issuance.

When a timer interrupt is acknowledged, one level of stack is used.

At this point the bank register contents and the index enable flag contents are automatically saved.

To return from the interrupt processing routine, the "RETI" instruction, a dedicated instruction, is used.

For details, refer to **5 "Stack"** and **12 "Interrupt"**.

A timer interrupt usage example and a timer interrupt error are described in **13.5.1** and **13.5.2**, respectively.

Refer to **12 "Interrupts"** for details of relations with other interrupts (RMS pin, $\overline{V_{SYNC}}$ pin and serial interface).

13.5.1 Example of Operating a Timer Using Timer Interrupt

An example is shown below.

Example:

```
    BR      AAA             ;  Branch to AAA
TIMER:                      ;  Program address 0003H
    ADD     M1,   #0001B    :  1 is added to M1.
    SKT1    CY              :  CY flag is tested.
    BR      BBB             :  Returns if no carry out is generated.
    [Processing A]
BBB:
    EI
    RETI
AAA:
    INITFLG NOT TMMD3, NOT TMMD2, NOT TMMD1, NOT TMMD0
                            ;  Intrinsic macro
                            ;  Timer interrupt pulse is set to 5 ms.
    MOV     M1,   #0000B    :  M1 contents are cleared to 0.
    SET1    IPTM            :  Timer interrupt enable is set.
    EI                      :  All interrupt enable is set.
LOOP:
    [Processing B]
    BR      LOOP
```

The above program executes processing A every 80 ms.

Note that if an interrupt is acknowledged, the DI status is automatically set and that if the DI status has previously been set, the IRQTM flag is set (1).

That is, when processing A takes 5 ms or more, an interrupt is immediately acknowledged if returned by the "RETI" instruction and processing B is not executed.

13.5.2 Timer Interrupt Error

As described in **13.4**, if timer interrupt has been enabled, an interrupt is acknowledged at each rising edge of the timer interrupt pulse.

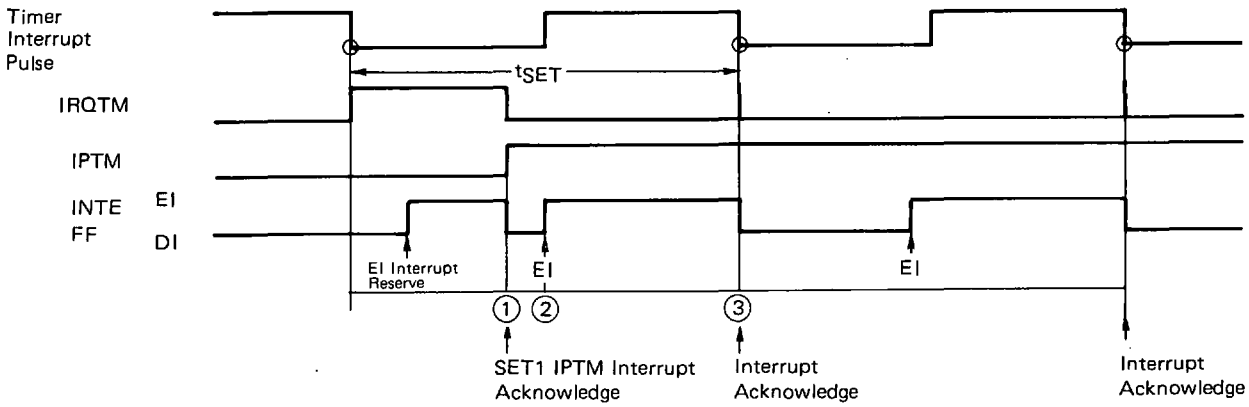
Thus, a timer error in timer interrupt use occurs only when the following operation is carried out.

(1) When the first interrupt is acknowledged after timer interrupt has been enabled
(2) When the first interrupt is acknowledged after the timer interrupt pulse time has been changed
(3) When a data write operation is carried out to the IRQTM flag

Fig. 13-9 shows an error in each operation.

Fig. 13-9 Timer Interrupt Error

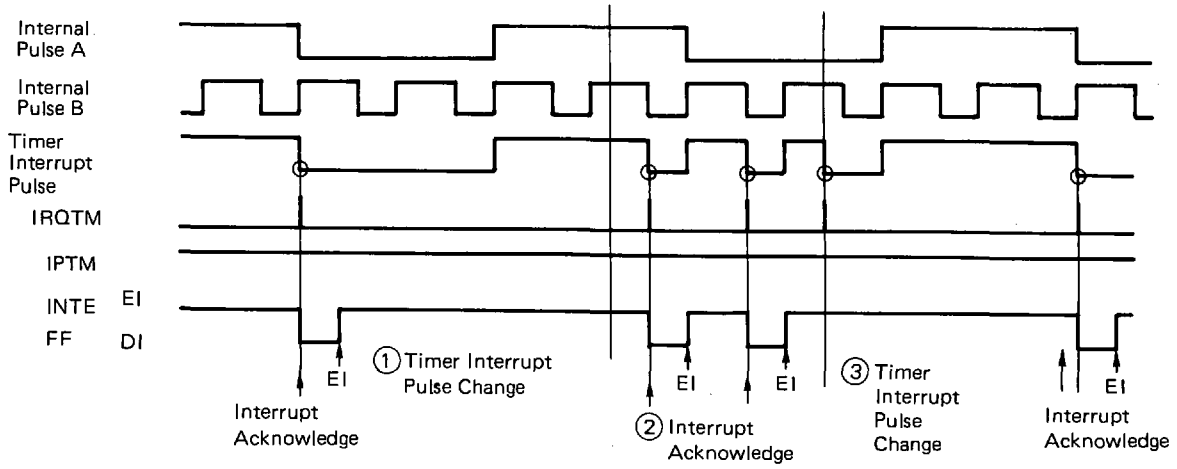
(a) When timer interrupt is enabled



If the IPTM flag is set and timer interrupt is enabled at point ① above, an interrupt is immediately acknowledged.
In this case the error is $-t_{SET}$.
If interrupt is enabled by the "EI" instruction at point ②, an interrupt is generated at the falling edge of the timer interrupt pulse at point ③.
In this case the error is between $-t_{SET}$ and 0.

Fig. 13-9 Timer Interrupt Error

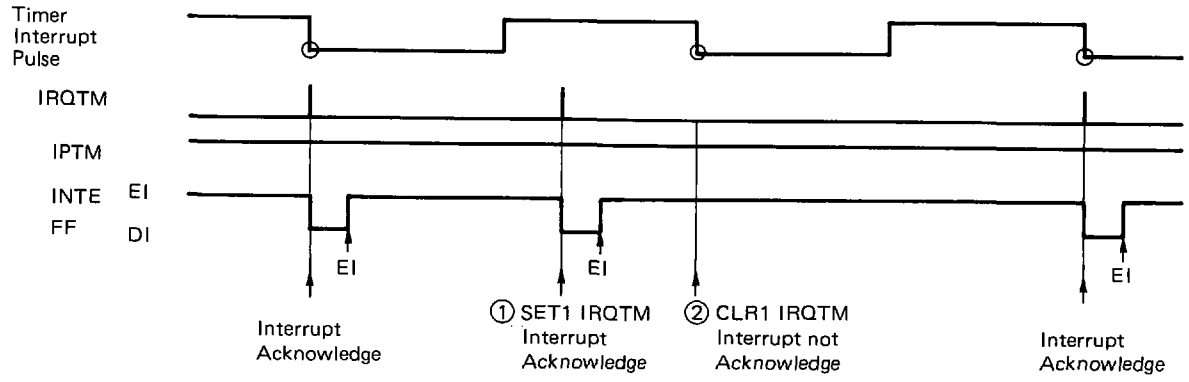
(b) When timer interrupt pulse is changed



Because the timer interrupt pulse does not fall if it is changed to B in ①, an interrupt is acknowledged in the next ②.

Because the timer interrupt pulse falls if it is changed to A in ③, an interrupt is immediately acknowledged.

(c) When IRQTM flag is operated



If the IRQTM flag is set in ①, an interrupt is immediately acknowledged.

If IRQTM flag reset and interrupt pulse fall overlap in ②, no interrupt is acknowledged.

13.6 PRECAUTIONS TIMER INTERRUPT USING

When creating a program, such as a clock program, which causes the specified timer to always operate after power-on reset using a timer interrupt, it is necessary to terminate the timer interrupt processing time within the specified period.

Such a program is described using an example.

Example:

```
    BR      AAA             ; Branch to AAA after reset.
TIMER:                      ; Program address 0003H
    ADD     M1,   #0100B    ; 0100B is added to M1 contents.
    SKT1    CY              ; If a carry out is generated, clock processing is executed.
    BR      AAA
    ; ①
    [Clock processing]
    EI
    RETI
AAA:
    INITFLG NOT TMMD3, NOT TMMD2, NOT TMMD1, NOT TMMD0
                            ; Intrinsic macro
                            ; Timer interrupt time is set to 250 ms and timer carry FF set time is set to 100
                              ms.
    SET1    IPTM            ; Intrinsic macro
    EI
    [Processing A]
    BR      AAA
```

In this example, clock processing ① is executed every second while processing A is executed.

As shown in Fig. 13-10 (a), when the CE pin changes from the low to the high level, CE reset is applied at the falling edge of the timer carry FF set pulse.

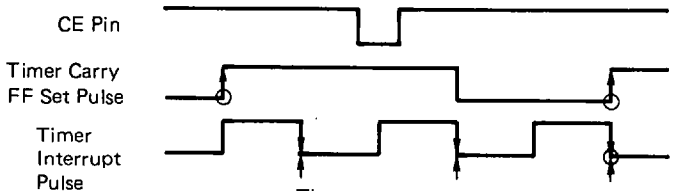
If timer interrupt request generation happens to overlap timer carry FF set, CE reset is given priority. When CE reset is applied, one timer processing operation is messed because the timer interrupt request (IRQTM flag) is reset.

To prevent timer interrupt from being missed, the "falling edge of the timer carry FF set pulse" and the "falling edge of the timer interrupt pulse" are both delayed, as shown in Fig. 13-10 (b).

Thus, as shown in Fig. 13-10 (2), timer interrupt fetch error upon CE reset can be prevented by executing clock processing within 10 ms as in this example.

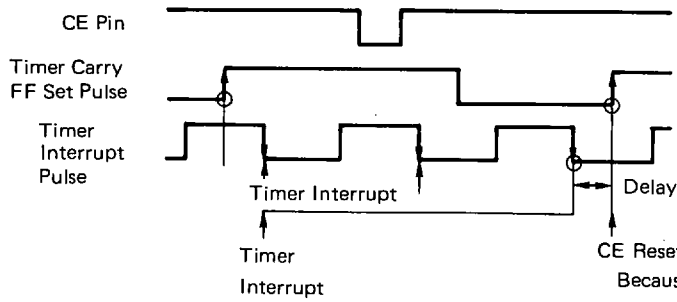
Fig. 13-10 Timing Chart

(a)



Because the timer carry FF set pulse rises, CE reset is applied and one timer interrupt is missed.

(b)



Because there is a 10 ms delay between the falling edge of the timer interrupt pulse and the rising edge of the timer carry FF set pulse, executing timer processing within 10 ms will enable timer processing to be executed normally if CE reset is applied.

14. STANDBY FUNCTION

The standby function is used to decrease the device current consumption during backup.

14.1 STANDBY BLOCK CONFIGURATION

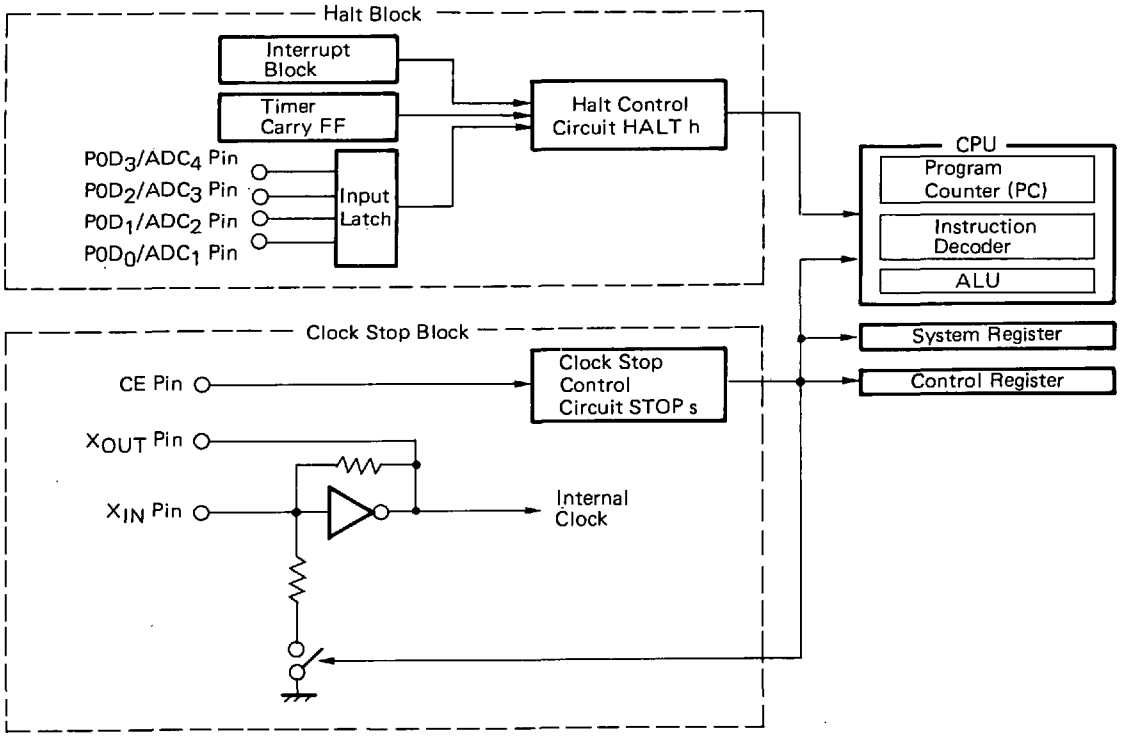
Fig. 14-1 shows the standby block configuration.

As shown in the figure, the standby block is divided into a halt control block and a clock stop control block.

The halt control block consists of a halt control circuit, an interrupt control block, a timer carry FF and key-input pins, P0D/ADC pins (pin No. 62) through P0D/ADC pin (pin No. 59). It controls the operations of the CPU (consisting of a program counter, an instruction decoder and an ALU block).

The clock stop control block controls the clock oscillator, the CPU and the system register and the control register with the clock stop control circuit.

Fig. 14-1 Standby Block Configuration



14.2 STANDBY FUNCTION

The standby function is intended to decrease device current consumption by stopping the device operation partially or totally.

It is divided into a halt function and a clock stop function.

The halt function decreases device current consumption by stopping the CPU operation using the "HALT h" dedicated instruction.

The clock stop function decreases device current consumption by stopping the 8 MHz crystal resonator circuit using the "STOP s" dedicated instruction.

In addition to the halt and clock stop functions, a function to set the device operation mode with the CE pin is available.

The device operation mode setting function with the CE pin is described in **14.3.**

The halt function and the clock stop function are described in **14.4** and **14.5**, respectively.

14.3 DEVICE OPERATION MODE SETTING WITH CE PIN

The CE pin is used to control the following functions at the external input signal level and the rising edge of the input level.

(1) Clock stop instruction valid/invalid control
(2) Device reset

These functions are described in **14.3.1** and **14.3.2.**

14.3.1 Clock Stop Instruction Validate/Invalidate Control

The "STOP s" clock stop instruction is validated only when the CE pin is at the low level.

The "STOP s" instruction executed when the CE pin is at the high level is executed as a no-operation instruction (NOP).

14.3.2 Device Reset

The device can be reset (CE reset) by raising the CE pin from the low level to the high level.

In addition to CE reset, reset includes power-on reset upon power voltage V_{DD} input.

For details, refer to **15 "Reset Functions".**

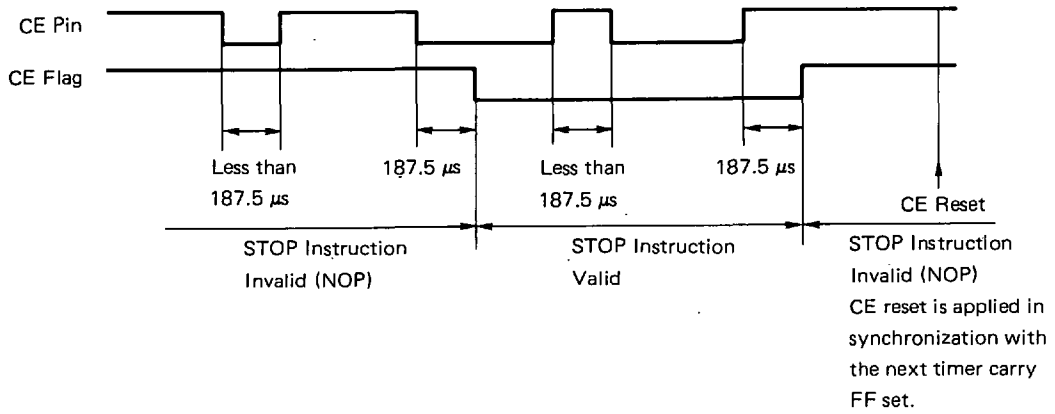
14.3.3 Signal Input to CE Pin

The CE pin acknowledges neither signals at a low level of less than 187.5 μs nor those at a high level to prevent malfunctioning due to noise.

The input level of the signal input to the CE pin can be detected from the CE flag of the control register (bit b_0 of address 07H).

Fig. 14-2 shows the relationship between the input signal and the CE flag.

Fig. 14-2 Relationship between CE Pin Input Signal and CE Flag



14.4 HALT FUNCTION

The halt function is intended to stop the CPU operation clock by executing the "HALT h" instruction.

That is, when the "HALT h" instruction is executed, the program stops with the "HALT h" instruction and remains halted until the halt status is released.

Thus, device current consumption during the clock halt is reduced by the CPU operating current.

The halt status is released using the timer carry FF, an interrupt or key-input.

The timer carry FF, interrupt and key-input cancel conditions are specified by the "HALT h" instruction operand "h".

The "HALT h" instruction is valid regardless of the CE pin input level.

Halt status, halt release conditions, and individual halt conditions are described in **14.4.1** to **14.4.5.**

14.4.1 Halt Status

The halt status stops all CPU operations.

That is, program execution stops with the "HALT h" instruction.

However, the peripheral hardware continues to carry out the operation set before the "HALT h" instruction.

14.4.2 Halt Release Condition

Fig. 14-3 shows the halt release conditions.

As shown in the figure, the halt conditions are set by the 4-bit data specified by the "HALT h" instruction operand "h".

The halt status is canceled when the conditions set for operand "h" by "1" are satisfied.

When the halt status is released, execution starts with the instruction after the "HALT h" instruction.

If two or more release conditions are set at one time, the halt status is released if any of those conditions is satisfied.

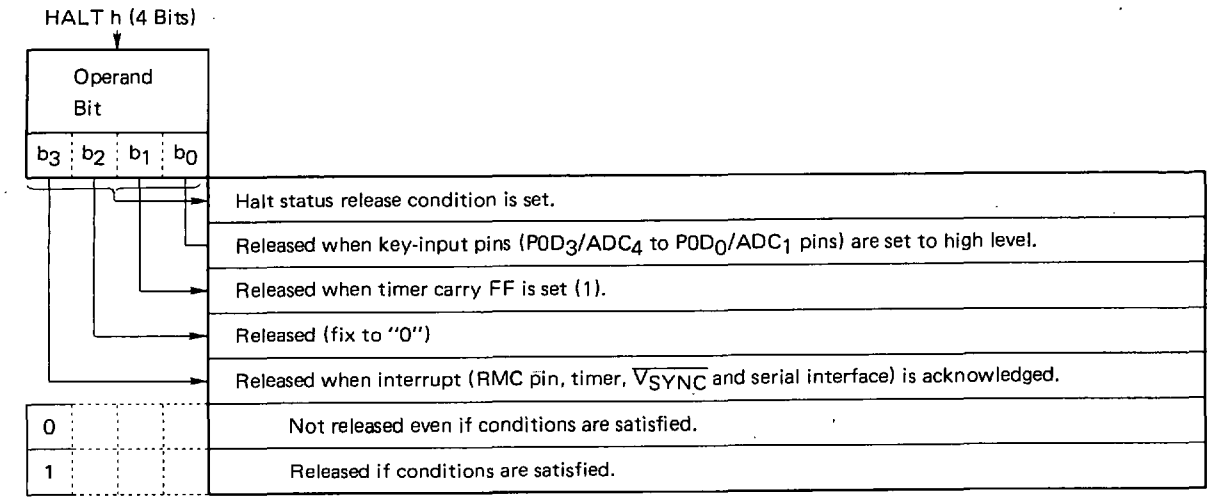
When the device is reset (power-on reset or CE reset), the halt status is release and reset operation is carried out.

If 0000B is set for halt release condition "h", none of the release conditions will be set.

In this case, the halt status is released when the device is reset (power-on reset or CE reset).

The halt release conditions for the timer carry FF, interrupt and key-input are described in **14.4.3** to **14.4.6.**

Fig. 14-3 Halt Release Conditions



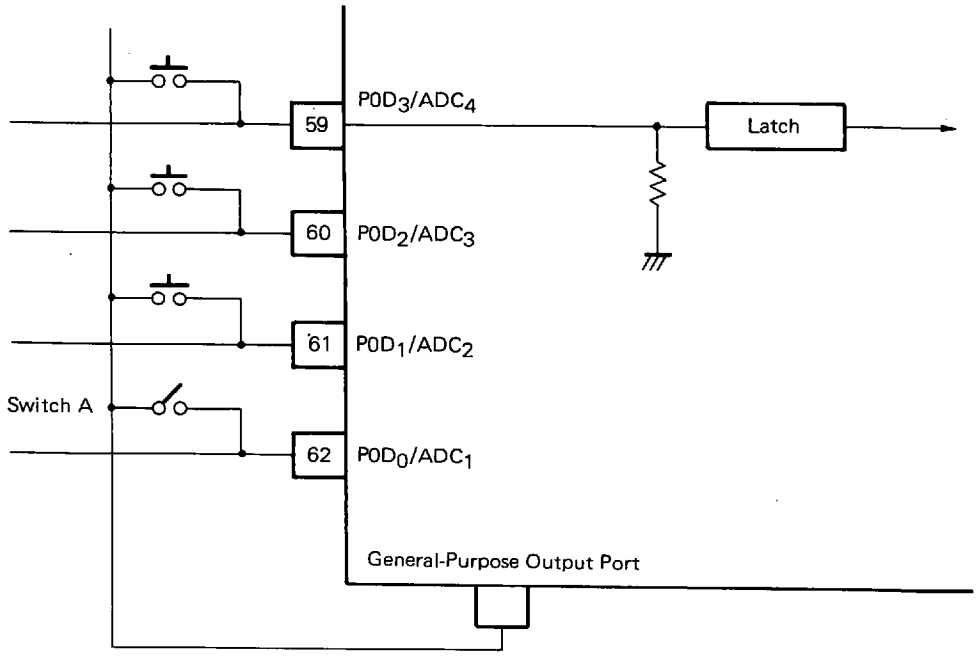
14.4.3 Halt Release by Key-Input

The key-input halt release conditions are set by the "HALT 0001B" instruction.

If the key-input halt release conditions are set, the halt status is released when any one of the four pins, $P0D_0/ADC_1$ to $P0D_3/ADC_4$, is set to the high level.

Note the following remarks when using the key-input pins as an A/D converter.

(1) When using the general-purpose port as the key source signal



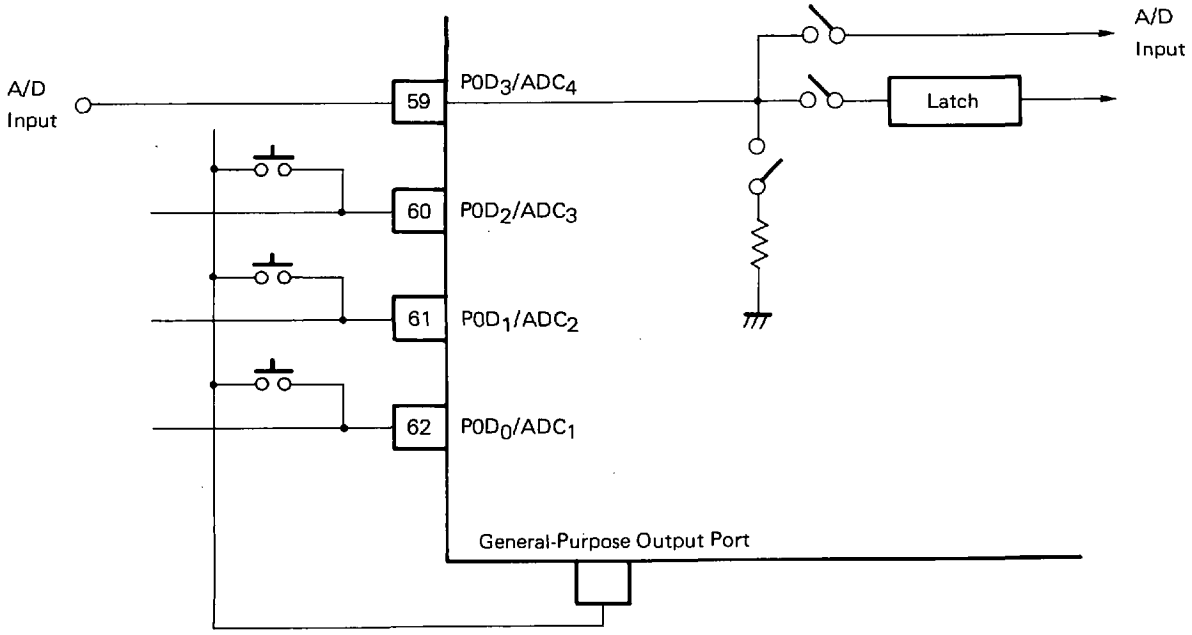
After the key source signal general-purpose port is set to the high level, the "HALT 0001B" instruction is executed.

If an alternate switch such as switch A in the above figure is in use, the halt status is immediately released because a high level is always applied to the $P0D_0/ADC_1$ pins while switch A remains closed.

Thus, take extra care when using an alternate switch.

The $P0D_0/ADC_1$ to $P0D_3/ADC_4$ pins are automatically pulled down internally.

(2) When using $P0D_0/ADC_1$ to $P0D_3/ADC_4$ pins as an A/D converters



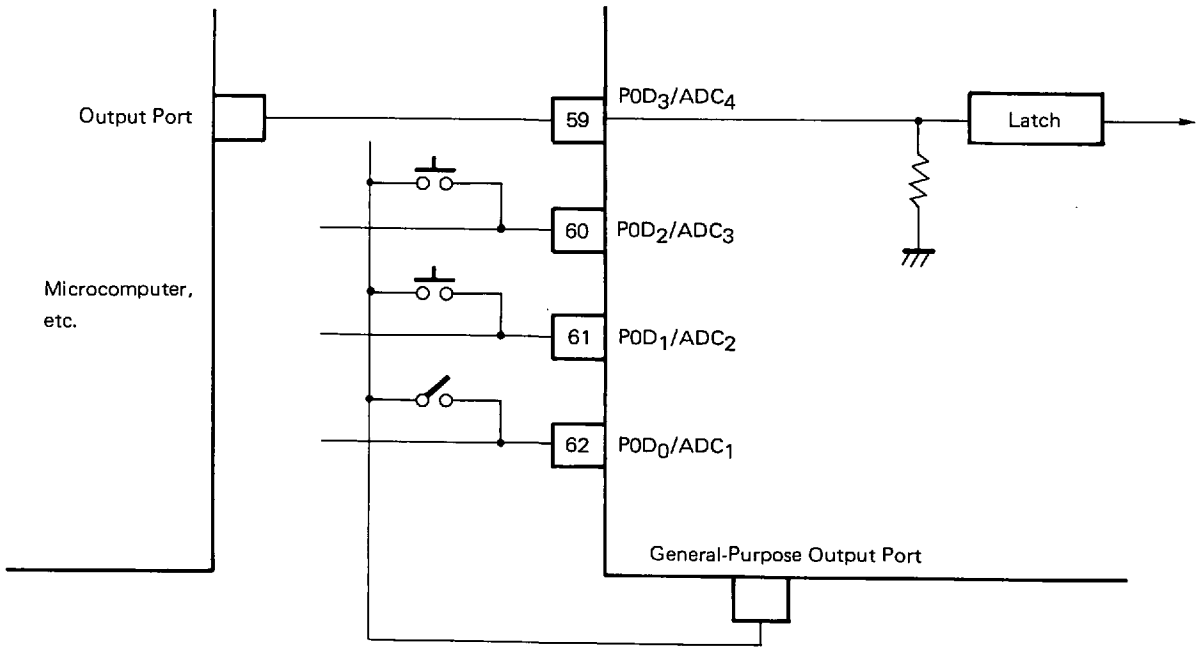
When one of the $P0D_0/ADC_1$ to $P0D_3/ADC_4$ pins is selected as A/D converters, the selected pin (only one pin can be selected at a time) is released from the input latch and connected to the internal A/D converter input.

If a high level happens to have been input to the pin when selected as an A/D converter, the latch circuit is held at the high level.

If the "HALT 0001B" instruction is executed at this point, the halt status is immediately released because the input latch is at the high level.

Thus, do not use the pins as an A/D converter.

(3) Others



The $P0D_0/ADC_1$ to $P0D_3/ADC_4$ pins can also be used as general-purpose input ports with pull-down resistors. Thus, the halt status can also be released using another microcomputer, etc. as shown above.

14.4.4 Halt Release by Timer Carry FF

Halt status release by the timer carry FF is set by the "HALT 0010B" instruction.

If halt release by the timer carry FF is set, the halt status is released simultaneously when the timer carry FF is set (1).

The timer carry FF has a 1:1 relation with the TMCY flag of the control register (bit b_0 of address 17H) and is set (1) periodically (5 ms, 100 ms).

Thus, the halt status can be released periodically.

A usage example is shown below.

Example:

```
HLTTMR  DAT  0010B  ;  Symbol definition
INITFLG  NOT  ZCROSS, NOT  TMMD2, NOT  TMMD1, NOT  TMMD0
                     ;  Intrinsic macro
                     ;  Timer carry FF set time is set to 100 ms.
LOOP1:
  MOV   M1,   #0110B
LOOP2:
  HALT  HLTTMR       ;  Timer carry FF halt release condition is set.
  SKT1  TMCY         ;  Intrinsic macro
  BR    LOOP         ;  Branch to LOOP2 if TMCY flag is not set.
  ADD   M1,   #0001B ;  ADD 0001B to contents of M1.
  SKT1  CY           ;  Intrinsic macro
  BR    LOOP2        ;  Processing A is executed if a carry out is generated.
  [Processing A]
  BR    LOOP1
```

In the above example, the halt status is released every 100 ms and processing A is executed every second.

14.4.5 Halt Status Release by Interrupt

Halt status release by interrupt is set by the "HALT 1000B" instruction.

If halt status release by interrupt is set, the halt status is released when an interrupt is acknowledged.

There are four interrupt sources. They are an RMC pin, a timer, $\overline{V_{SYNC}}$ and a serial interface.

Thus, the interrupt source to be used for halt release must be prespecified by a program.

For an interrupt to be acknowledged, all interrupt enable (EI instruction) and each interrupt enable (with the interrupt permission flag set) must also be satisfied in addition to the generation of an interrupt request from each interrupt resource.

Thus, if an interrupt request is generated, the interrupt is not acknowledged unless enabled and the halt status is not released.

When the halt status is released by interrupt acknowledgment, the program flow proceeds to each interrupt vectored address. When the "RETI" instruction is executed after interrupt service, the program flow returns to the interruption following the HALT" instruction.

A usage example is shown below.

Example:

```
    HLTINT  DAT  1000B   ;  Symbol definition
START:                   ;  Program address 0000H
    BR      MAIN         ;
    NOP
INTTM:                   ;  Timer interrupt vector address
    BR      INTTIMER     ;  Branch to timer interrupt service INTTIMER
INT0:                    ;  RMC pin interrupt vector address
    [Processing A]       ;  Interrupt service by RMC pin
    EI
    RETI
INTTIMER:
    [Processing B]       ;  Interrupt service by timer
    EI
    RETI
MAIN:
    SET2    IPTM, IP     ;  Intrinsic macro
                         ;  RMC pin and timer interrupt enable
    SET1    TMMD2        ;  Intrinsic macro
LOOP:                    ;  Timer interrupt time interval is set to 5 ms.
    [Processing C]       ;  Main routine service
    EI                   ;  All interrupts enable
    HALT    HLTINT       ;  Halt release by interrupt is set.
    ; ①
    BR      LOOP
```

In the above example, when a timer interrupt is acknowledged, the halt status is released and processing B is executed. When an RMC pin interrupt is acknowledged, processing A is carried out.

Each time the halt status is released, processing C is executed.

If an RMC pin interrupt request and an timer interrupt request are simultaneously generated in the halt status, the RMC pin processing having higher hardware priority is executed.

When "RETI" is executed after processing A execution, the program returns to ① "BR LOOP" instruction. As soon as the "BR LOOP" instruction is executed, a timer interrupt is acknowledged.

When the "RETI! instruction is executed after execution of processing B (timer interrupt service), the "BR LOOP" instruction is executed.

14.5 CLOCK STOP FUNCTION

The clock stop function is intended to stop the clock oscillator by executing the "STOP s" instruction.

Thus, the device current consumption is decreased to 10 μA max.

"0000B" is specified for operand "s" of the "STOP s" instruction.

The "STOP s" instruction is only valid when the CE pin (pin No. 14) is at the low level. While the CE pin is at the high level, the "STOP s" instruction is executed as a no-operation instruction (NOP).

In other words, it is necessary to execute the "STOP s" instruction while the CE pin is at the low level.

The clock stop status is canceled by CE reset.

Points to be noted when setting, using and setting the clock stop instruction are given in paragraphs **14.5.1** to **14.5.3.**

14.5.1 Clock Stop Status

In the clock stop status the clock oscillator stops and thus all operations of devices such as the CPU and peripheral hardware stop.

In this status if the device power voltage V_{DD} is decreased to about 2.2 V, the power failure detector does not operate. Thus, data memory backup with a low voltage is enabled.

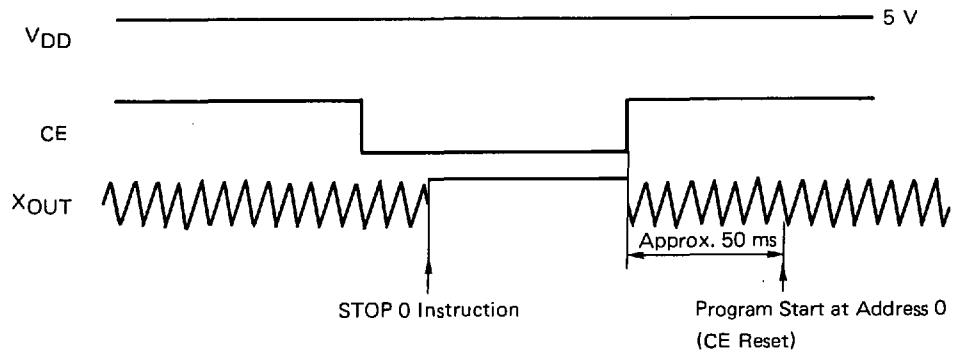
14.5.2 Clock Stop Status Release

The clock stop status is released by raising the CE pin from the low level to the high level (CE reset) or first decreasing the device power voltage V_{DD} to 2.2 V or less and then increasing it to 4.5 V (power-on reset).

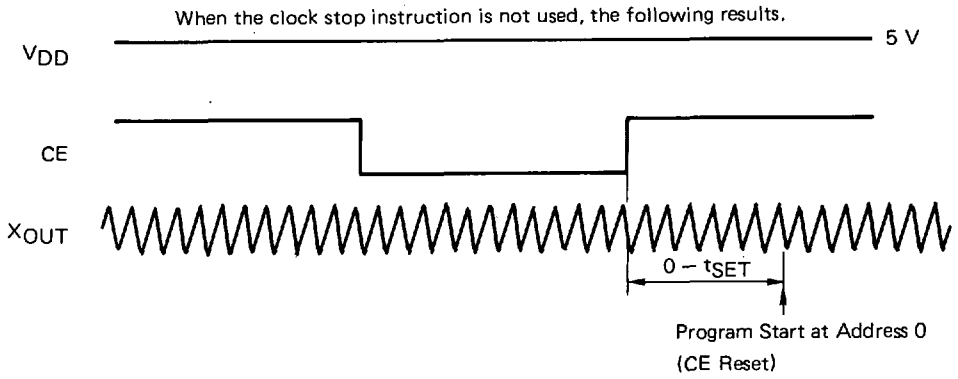
Released operations upon CE reset and power-on reset are shown in Figs. 14-4 and 14-5, respectively.

When the clock stop status is released upon power-on reset, the power failure detection circuit operates.

Fig. 14-4 Clock Stop Status Cancel upon CE Reset

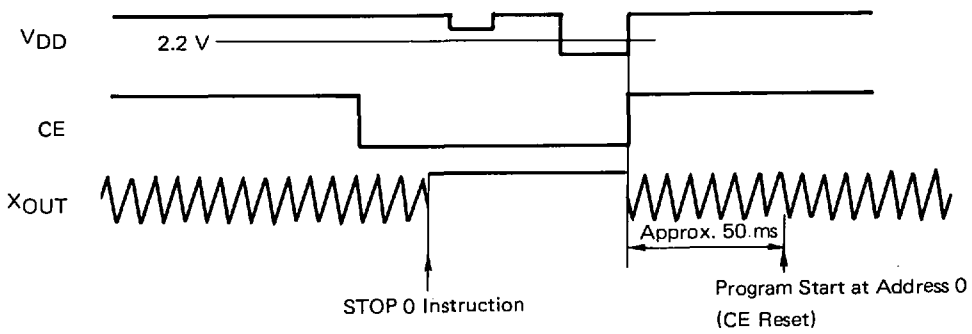


When the clock stop instruction is not used, the following results.

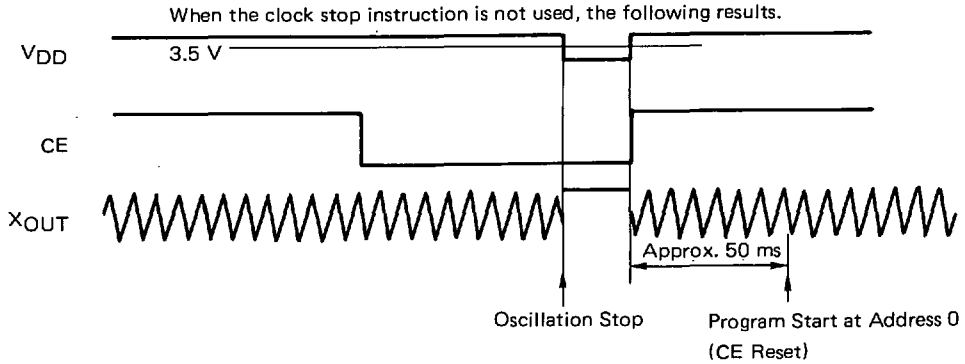


CE reset is applied upon the timer carry FF set after the rising edge of CE pin at high level.

Fig. 14-5 Clock Stop Release upon Power-On Reset



When the clock stop instruction is not used, the following results.



14.5.3 Precautions Using Clock Stop Instruction

The clock stop instruction (STOP s instruction) is valid only when the CE pin is at the low level.

Thus, it is necessary to consider the processing in the program when the CE pin happens to be at the high level.

The following program illustrates an example.

Example:

```
    XTAL  DAT  0000B  ; Clock stop condition symbol definition
CEJDG:
    ; ①
    SKF1  CE          ; Intrinsic macro
                      ; CE pin input level detection
    BR    MAIN        ; Branch to main processing when CE = high level.
    [Processing A]    ; CE = low level processing
    ; ②
    STOP  XTAL        ; Clock stop
    ; ③
    BR    $ – 1
MAIN:
    [Main processing]
    BR    CEJDG
```

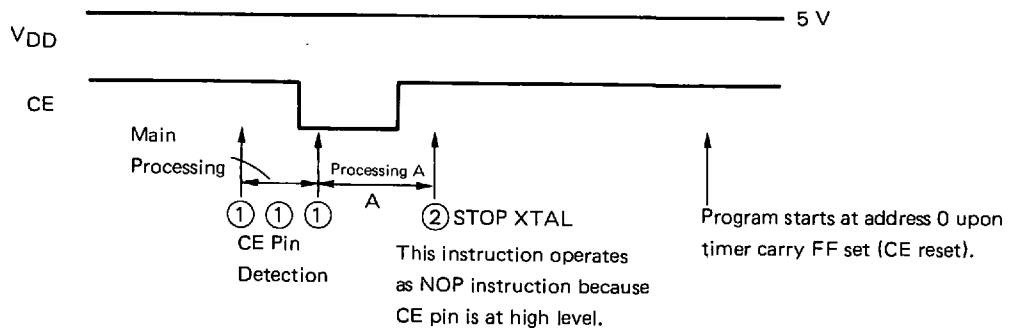
In the above example, the CE pin status is detected in ①. If the CE pin is at the low level, processing A is executed and then ② "STOP XTAL" clock stop instruction is executed.

However, if the CE pin becomes high while ② "STOP XTAL" instruction is being executed as shown below, the "STOP XTAL" instruction is being executed as shown below, the "STOP XTAL" instruction operates as a no-operation instruction (NOP).

If ③ "BR $–1" branch instruction were not reset at this point, the program would shift to main processing and operate incorrectly.

Thus, a branch instruction should be inserted as in ③ or the program should be able to operate correctly for main processing.

When a branch instruction is used as in ②, CE reset is applied upon the next timer carry FF set if the CE pin remains at the high level.



14.6 DEVICE OPERATIONS UPON HALT AND CLOCK STOP

14.6.1 Individual Pin Status upon Halt and Clock Stop

Table 14-1 shows CPU and peripheral hardware operations in the halt and clock stop statuses.

As shown in the table, instruction execution stops but all peripheral hardware continue to carry out normal operations in the clock stop status.

The control register which controls peripheral hardware operation statuses operates normally (not initialized) in the halt status but it is initialized to the specified value in the clock stop status (when the STOP s instruction is executed).

In other words, each peripheral hardware continues to carry out the operation set in the control register in the halt status. In the clock stop status the operation of each peripheral hardware is determined by the control register initialized to the specified value.

Refer to **10 "Register File (RF)"** for details of the values to which the control register is initialized.

An example is shown below.

Example: When $P0A_0$/SDA and $P0A_1$/SCL pins of port0A are set as the output port and the $P0A_2/\overline{SCK}$ and $P0A_3$/SO pins are used as serial interfaces

```
HLTINT   DAT   1000B  ; Symbol definition
XTAL     DAT   0000B  ;
INITFLG  P0ABIO3, P0ABIO2, P0ABIO1, P0ABIO0
                      ; Intrinsic macro
; ①
SET2     P0A0, P0A1   ;
INITFLG  SIOCH, NOT SB, SIOMS, SIOTX
                      ;
SET2     SIOCK1, SIOCK0
; ②
SET2     SIOIMD1, SIOIMD0
CLR1     IRQSIO
SET1     IPSIO
EI
; ③
SET1     SIONWT
; ④
HALT     HLTINT
; ⑤
STOP     XTAL
```

In the above example, the high level is output from the $P0A_0$ and $P0A_1$ pins in ① ; the serial interface conditions are set in ② , and serial communication is started in ③ .

When ④ "HALT" instruction is executed, serial communication carries on and the halt status is released upon acknowledgment of a serial interface interrupt.

If ⑤ "STOP" instruction is executed in place of ④ "HALT" instruction, all of the control flags set in ①, ② and ③ are initialized. Thus, serial communication is discontinued and all pins at port0A are set to the general-purpose input port.

Table 14-1 Device Operations in Halt Status and Clock Stop Status

Peripheral Hardware	Status			
	CE Pin = High Level		CE Pin = Low Level	
	Halt	Clock Stop	Halt	Clock Stop
Program counter	Stop at HALT instruction address	STOP instruction invalid (NOP)	Stop at HALT instruction address	Initialized to 0000H and stopped
System register	Hold		Hold	Initialize*
Peripheral register	Hold		Hold	Hold
Control register	Hold		Hold	Initialize*
Timer	Normal operation		Normal operation	Operation stop
A/D converter	Normal operation		Normal operation	Operation stop
D/A converter	Normal operation		Normal operation	Operation stop
Serial interface	Operation stop		Operation stop	Operation stop
General-purpose input/output dual-function port	Normal operation		Normal operation	Input port
General-purpose input port	Normal operation		Normal operation	Input port
General-purpose output port	Normal operation		Normal operation	Hold
IDC	HALT instruction execution status hold	Stop instruction invalid (NOP)	Operation stop	Operation stop

*: Refer to 9 "System Register (SYSREG)" and 10 "Register File (RF)" for details of the initialize values.

14.6.2 Pin Processing Precautions in Halt and Clock Stop

The halt function is used to decrease current consumption when, for example, operating only the clock.

The clock stop function is used to decrease current consumption to hold only the data memory.

Thus, it is necessary to minimize current consumption in the hold status and the clock stop status.

In this case, the precautions listed in Table 14-2 must be taken because current consumption considerably varies depending on each pin status.

Table 14-2 Pin Statuses and Precautions in Halt and Clock Stop Status

Pin Function		Pin Symbol	Each Pin Status and Processing Precautions	
			Halt Status	Clock Stop Status
General-purpose input/output port	Port 0A	$P0A_3$/SO $P0A_2$/$\overline{SCK}$ $P0A_1$/SCL $P0A_0$/SDA	The status before halt is held. **(1) When specified for output pin** If the pin is externally pulled down during high-level output or externally pull up during low-level output, current consumption increases. Take extra note of N-ch open-drain output ($P0A_1$, $P0A_0$, $P1A_3$ to $P1A_0$, $P2B_3$ to $P2B_0$, $P2C_3$ to $P2C_0$). **(2) When specified for input pin** If the pin is in the floating status, current consumption increases due to noise. **(3) Port0D ($P0D_3$/ADC_4 to $P0D_0$/ADC_1)** Because a pull-down resistor is incorporated, current consumption increases if the pin is externally pulled up. The pull-down resistor is turned OFF in the case of the pin selected for the A/D converter. **(4) Port0B ($P0B_3$/HSCNT to $P0B_0$/SI)** **Port1B ($P1B_3$/TMIN to $P1B_0$)** When the $P0B_3$/HSCNT pin operates as a HSYNC counter and the $P1B_3$/TMIN pin operates as an external input timer, the on-chip self-bias circuit operates and current consumption increases.	All of these pins are specified for general-purpose input ports. All input ports except $P0A_1$/SCL and $P0A_0$/SDA pins have circuit configurations with no current consumption increase due to noise if they are externally in a floating status. $P0A_1$/SCL and $P0A_0$/SDA pins must be externally pulled down or up to prevent current consumption for increase due to noise. Port0D ($P0D_3$/ADC_4 to $P0D_0$/ADC_1) are internally pulled down.
	Port 0B	$P0B_3$/HSCNT $P0B_2$ $P0B_1$ $P0B_0$/SI		
	Port 1B	$P1B_3$/TMIN $P1B_2$ $P1B_1$ $P1B_0$		
	Port 1C	$P1C_3$ $P1C_2$/ADC_7 $P1C_1$/ADC_6 $P1C_0$/ADC_5		
	Port 1A	$P2A_3$ $P2A_2$ $P2A_1$ $P2A_0$		
General-purpose input port	Port 0D	$P0D_3$/ADC_4 $P0D_2$/ADC_3 $P0D_1$/ADC_2 $P0D_0$/ADC_1		
General-purpose output port	Port 0C	$P0C_3$ $P0C_2$ $P0C_1$ $P0C_0$		These pins are specified for general-purpose output port. The output contents are held. Thus, if they are externally pulled down during high-level output or pulled up during low-level output, current consumption increases.
	Port 1A	$P1A_3$ $P1A_2$ $P1A_1$ $P1A_0$		
	Port 1D	$P1D_3$ $P1D_2$ $P1D_1$ $P1D_0$		

Pin Function		Pin Symbol	Each Pin Status and Processing Precautions	
			Halt Status	Clock Stop Status
General-purpose output port	Port 2B	$P2B_3$ $P2B_2$ $P2B_1$ $P2B_0$		These pins are specified for general-purpose output port. The output contents are held. Thus, if they are externally pulled down during high-level output or pulled up during low-level output, current consumption increases.
	Port 2C	$P2C_3$ $P2C_2$ $P2C_1$ $P2C_0$		
A/D converter		ADC_0	In the floating status, current consumption increases due to external noise.	
Interrupt		RMC		
IDC		RED GREEN BLUE BLANK $\overline{H_{SYNC}}$ $\overline{V_{SYNC}}$	The output pin holds the HALT instruction execution status. When the IDCEN flag is set, current consumption increases.	IDC disable. Each pin is set as follows. RED, GREEN and BLUE pins and $\overline{H_{SYNC}}$ and $\overline{V_{SYNC}}$ pins have circuit configurations where current consumption does not increase if they become low-level output or floating, respectively.
D/A converter		PWM_{RMP} PWM_2 PWM_1 PWM_0	Same precautions as with the general-purpose output port are necessary.	All pins generate low-level output.
Clock oscillator		X_{IN} X_{OUT}	Current consumption varies depending on the clock oscillator waveform. As the amplitude increases, current consumption decreases. Oscillation amplitude must be checked because it is determined by the ceramic oscillator, crystal resonator or load capacitor.	X_{IN} pin is internally pulled down and X_{OUT} pin generates high-level output.

15. RESET FUNCTION

The reset function is used to initialize device operations.

15.1 RESET BLOCK CONFIGURATION

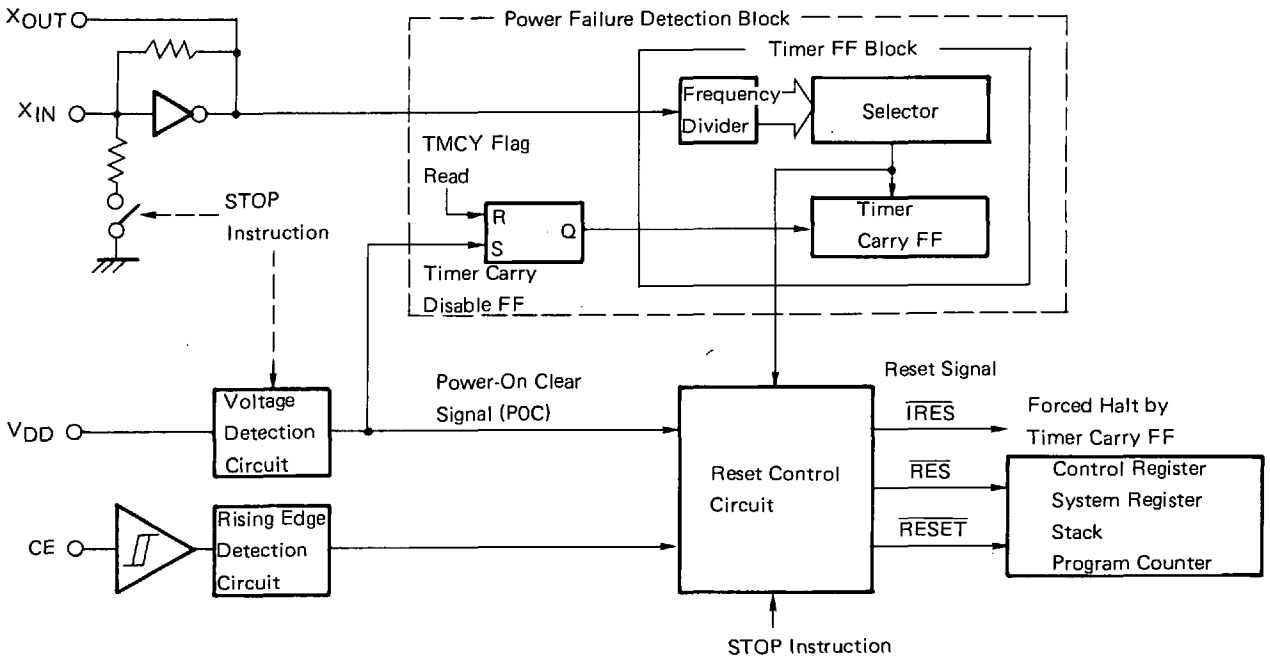
Fig. 15-1 shows the reset block configuration.

Device reset is divided into power voltage V_{DD} input reset (power-on reset or V_{DD} reset) and CE pin reset (CE reset).

The power-on reset block consists of a voltage detection circuit which detects the voltage input from the V_{DD} pin, a power failure detection circuit and a reset control circuit.

The CE reset block consists of a circuit which detects the rising edge of the signal input to the CE pin and a reset control circuit.

Fig. 15-1 Reset Block Configuration



15.2 RESET FUNCTION

Power-on reset is applied when the power voltage V_{DD} rises from the specified voltage and CE reset is applied when the CE pin rises from the low level to the high level.

Power-on reset initializes the program counter, the stack, the system register and the control register and executes the program at address 0000H.

CE reset partly initializes the program counter, the stack, the system register and the control register and executes the program at address 0000H.

Power-on reset and CE reset differ mainly in the control register to be initialized and the operations of the power failure detection circuit to be described in 15.6.

Power-on reset and CE reset are controlled by the $\overline{IRES}$, $\overline{RES}$ and $\overline{RESET}$ reset signals output from the reset control circuit shown in Fig. 15-1.

Table 15-1 shows the relationship between $\overline{IRES}$, $\overline{RES}$ and $\overline{RESET}$ signals and power-on reset and CE reset.

The reset control circuit also operates when the clock stop instruction (STOP) described in **14 "Standby"** is executed.

CE reset and power-on reset are described in **15.3** and **15.4**, respectively.

The relations between CE reset and power-on reset are described in **15.5.**

Table 15-1 Relations between Internal Reset Signal and Each Reset

Internal Reset Signal	Output Signal			Contents Controlled by Each Reset Signal
	Upon CE Reset	Upon Power-On Reset	Upon Clock Stop	
$\overline{IRES}$	X	O	O	The device is forcible set to halt status. Halt status is canceled by setting the timer carry FF.
$\overline{RES}$	X	O	O	Control register is partly initialized.
$\overline{RESET}$	O	O	O	The program counter, stack, system register and control register are partly initialized.

15.3 CE RESET

CE reset is applied by raising the CE pin from the low level to the high level.

When the CE pin rises to the high level, the $\overline{\text{RESET}}$ signal is output at the rising edge of the next timer carry FF set pulse and the device is reset.

When CE reset is applied, the program counter, the stack, the system register and the control resister are partly initialized to the initial values by the $\overline{\text{RESET}}$ signal and the program is executed at address 0000H.

For details of the initial values, refer to the relevant description.

CE reset operations vary depending on whether clock stop is used or not.

This is described in **15.3.1** and **15.3.2.**

CE reset precautions are described in **15.3.3.**

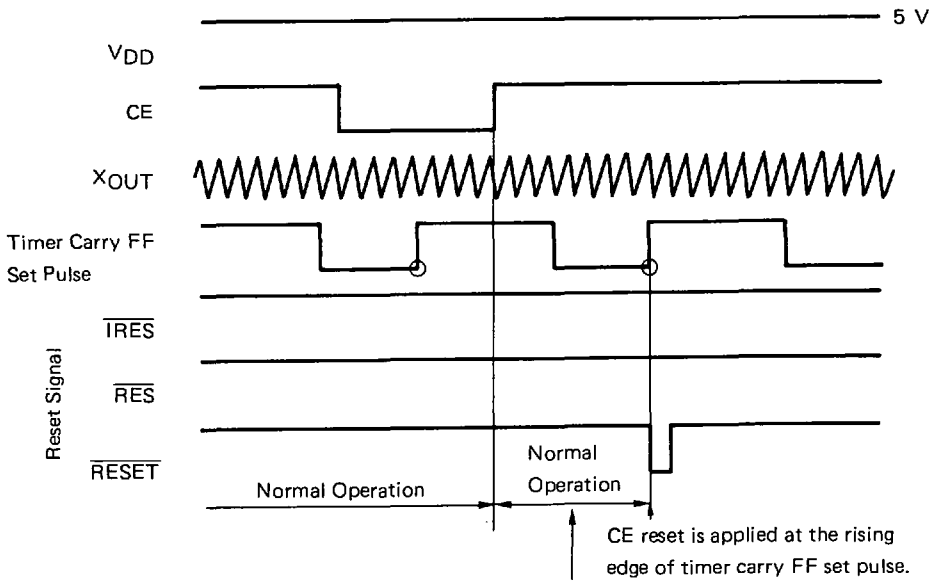
15.3.1 CE Reset without Clock Stop (Stop Instruction)

CE reset operation is shown in Fig. 15-2.

When clock stop (STOP instruction) is not used, the timer mode select control register is not initialized.

Thus, after the CE pin has become high, the $\overline{\text{RESET}}$ signal is output at the rising edge of the selected timer carry FF set pulse (5 ms, 100 ms) and CE reset is applied.

Fig. 15-2 CE Reset Operation without Clock Stop



When the selected timer carry FF set time is t_{SET}, $0 < t < t_{SET}$ because of the CE pin rising timing.
During this period the program continues to operate.

15.3.2 CE Reset with Clock Stop (Stop Instruction)

CE reset operation is shown in Fig. 15-3.

When clock stop is used, the $\overline{IRES}$, $\overline{RES}$ and $\overline{RESET}$ signals are output when the "STOP s" instruction is executed.

Because the timer mode select control register is initialized to 0000B by the $\overline{RES}$ signal at this point, the timer carry FF set signal is specified for 100 ms.

While the CE pin is at the low level, timer carry FF cancel is forcibly halted because the $\overline{IRES}$ signal continues to be output.

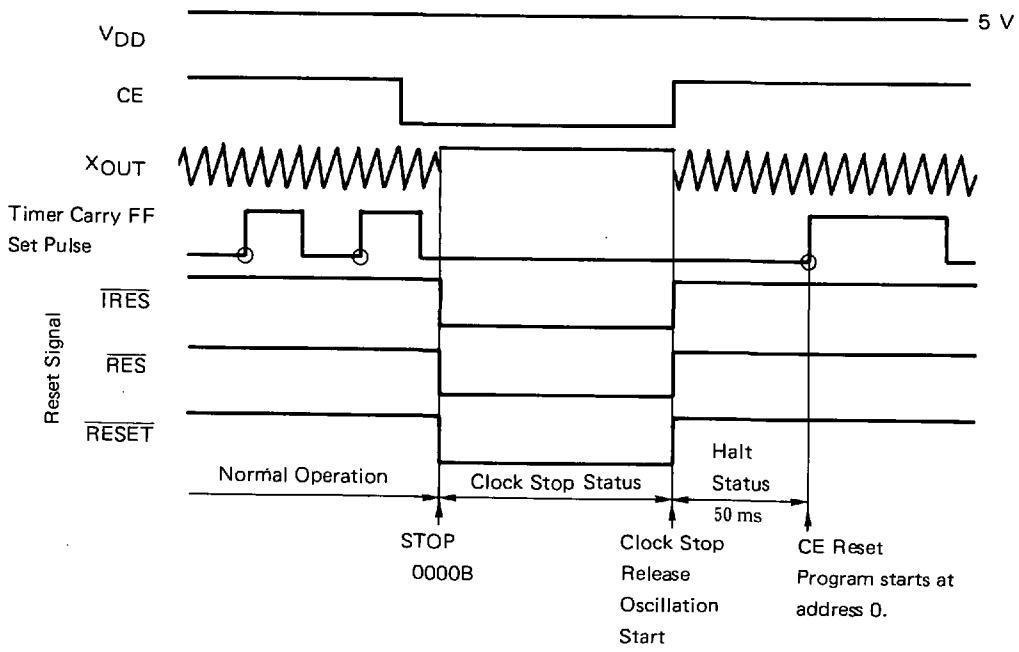
However, during this period, the device remains at reset because the clock itself is stopped.

When the CE pin rises to the high level, the clock stop status is released and oscillation starts.

Because timer carry FF cancel has been halted by the $\overline{IRES}$ signal at this point, the timer carry FF set pulse rises after the CE pin has risen. After that, the halt status is released and the program starts at address 0.

Because the timer carry FF set pulse has been initialized to 100 ms, CE reset is applied 50 ms after the CE pin has risen to the high level.

Fig. 15-3 CE Reset Operation with Clock Stop



15.3.3 CE Reset Precautions

Note (1) and (2) because CE reset is applied irrespective of the instruction being executed.

(1) Time required for clock timer processing

When creating a clock program using the timer carry FF or timer interrupt, it is necessary to terminate program processing within the specified time.

For details, refer to **13.4 "Timer Carry FF Using Precautions"** and **13.6 "Timer Interrupt Using Precautions"**.

(2) Program data and flag processing

Among data and flags which cannot be processed by one instruction, those with contents which should not change if CE reset is applied, the last channel, for example, must be carefully handled when its contents are rewritten.

This is described according to the following example.

Example 1:

```
; ①
LCTUNE:
  Initial receive processing
  Channel receive with M1
  and M2 contents          ; Last channel receive
MAIN:                      ; Main processing
  Channel change           ; When the channel is changed, the new channel is substituted for general
                             registers R1 and R2.
; ②
  ST    M1,   R1           ; Last channel rewrite
; ③
  ST    M2,   R2
  BR    MAIN
```

In the above example 1, if the last channel is "12H", the contents of data memories M1 and M2 will be "1H" and "2H", respectively.

If CE reset is applied at this point, the last channel 12 ch is received in ①.

If the channel is changed in main processing, the new channel is rewritten to M1 and M2 in ② and ③.

If the channel is changed to "04H", "0H" and "4H" are written to M1 and M2 in ② and ③, respectively.

However, if CE reset happens to be applied when ② is executed, CE reset results without executing ③.

Thus, the last channel becomes "02H" and 02 ch is received in ①.

In this case, a program in the following Example 2 will be run.

Example 2:

```
; ④                          ; When FLG1 flag is "1",
  SKT1   FLG1
  BR     LCTUNE
  ST     M1,   R1            ; Rewritten to M1 and M2.
  ST     M2,   R2
  CLR1   FLG1
; ①
LCTUNE:
  Initial receive processing
  Channel with M1 and
  M2 contents is received.   ; Last channel receive
MAIN:                        ; Main processing
  Channel change             ; If the channel is changed, the new channel is substituted for general registers
                               R1 and R2.
; ⑤
  SET1   FLG                 ; While the last channel is rewritten, FLG1 flag is set.
; ②
  ST     M1,   R1            ; Last channel rewrite
; ③
  ST     M2,   R2
  CLR1   FLG1
  BR     MAIN
```

In the above Example 2, the last channel is rewritten in ② and ③, the FLG1 flag is set.
Thus, if CE reset happens to be applied in ③, rewrite is carried out in ④.

15.4 POWER-ON RESET

Power-on reset is applied by raising the device power voltage V_{DD} from the specified voltage (called a power-on clear voltage).

If power voltage V_{DD} is less than the power-on clear voltage, a power-on clear signal (POC) is generated from the voltage detection circuit shown in Fig. 15-1.

When the POC signal is output, device operation is stopped by stopping the clock oscillator.

While generating the POC signal, the $\overline{IRES}$, $\overline{RES}$ and $\overline{RESET}$ signals are output.

If power voltage V_{DD} becomes greater than the power-on clear voltage, the power-on clear signal is discontinued and as soon as crystal oscillation starts, the $\overline{IRES}$, $\overline{RES}$ and $\overline{RESET}$ signals are also discontinued.

Because the timer carry FF cancel has been halted by the $\overline{IRES}$ signal, power-on reset is applied at the rising edge of the next timer carry FF set signal.

Because the timer carry FF set signal has been initialized to 100 ms by the $\overline{RESET}$ signal, power-on reset is applied 50 ms after the power voltage V_{DD} has exceeded the power-on clear voltage and the program starts at address 0.

This operation is shown in Fig. 15-4.

Initialization of the program counter, stack, system register and control register upon power-on reset is carried out when the power-on clear signal is generated.

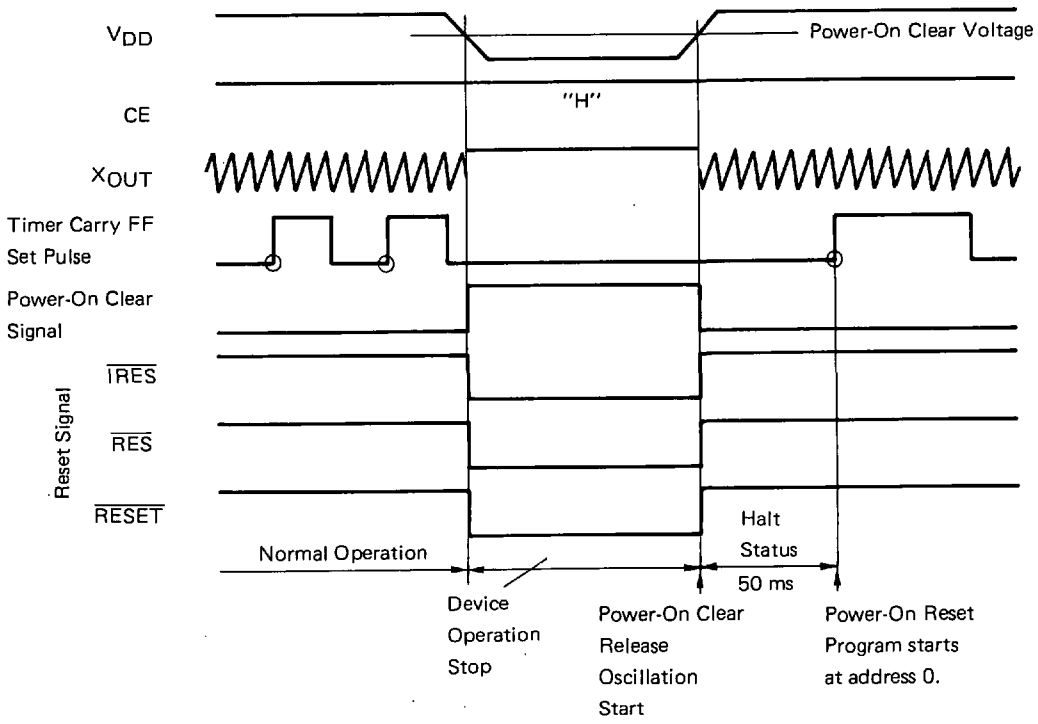
For details of the initial values, refer to the relevant description.

The power-on clear voltage is 3.5 V (specified value) in normal operation and 2.2 V (specified value) in the clock stop status.

Operations in these cases are shown in **15.4.1** and **15.4.2.**

Operation when the power voltage V_{DD} is increased from 0 V is described in **15.4.3.**

Fig. 15-4 Power-On Reset Operation



15.4.1 Power-On Reset in Normal Operation

Power-on reset operation is shown in Fig. 15-5 (a).

As shown in the figure, if the power voltage V_{DD} becomes lower than 3.5 V irrespective of the CE pin input level, the power-on clear signal is output and device operation stops.

Next, when the power voltage V_{DD} becomes 3.5 V or more again, the program will start at address 0000H after a 50 ms halt.

Normal operation means the time when the clock stop instruction is not in use. It includes the halt status by the HALT instruction.

15.4.2 Power-On Reset in Clock Stop Status

Power-on reset operation is shown in Fig. 15-5 (b).

As shown in the figure, if the power voltage V_{DD} becomes lower than 2.2 V, the power-on clear signal is output and the device operation stops.

However, because the clock stop status has been set, the device operation apparently remains unchanged.

Next, when the power voltage V_{DD} becomes 3.5 V or more again, the program will start at address 0000H after a 50 ms halt.

15.4.3 Power-On Reset when Power Voltage V_{DD} Rises from 0 V

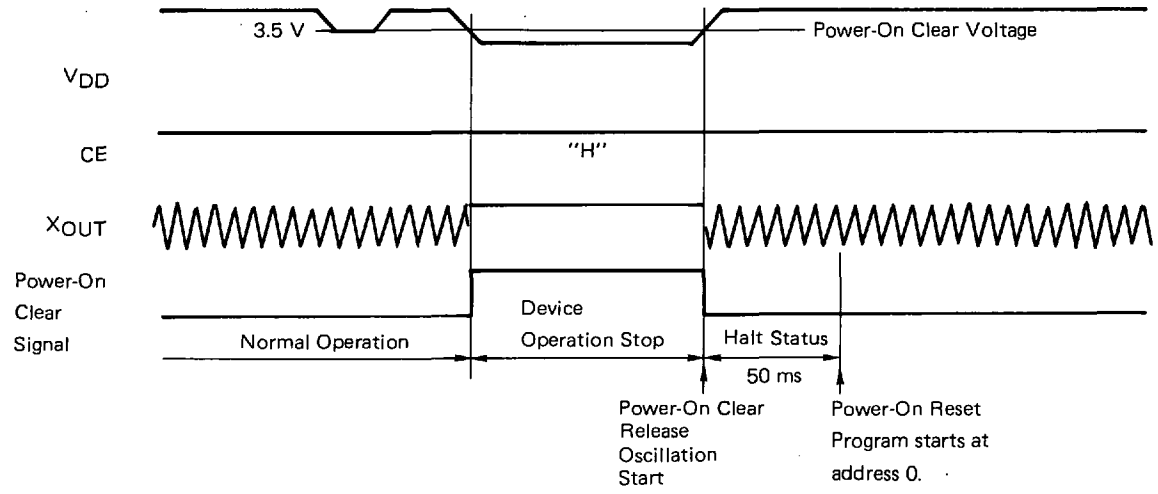
Power-on reset operation is shown in Fig. 15-5 (c).

As shown in the figure, the power-on clear signal continues to be output until the power voltage V_{DD} increases from 0 V to 3.5 V.

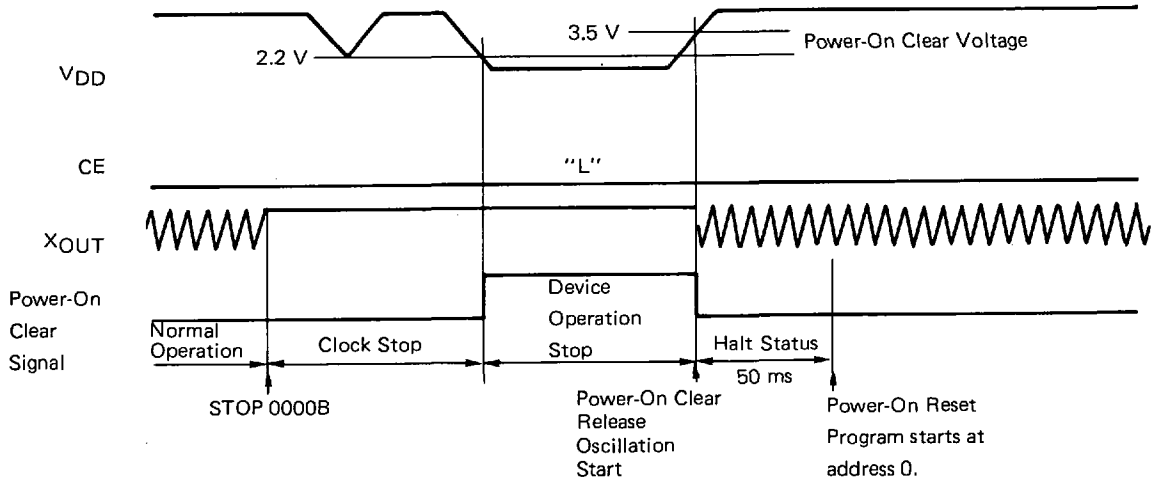
When the power voltage exceeds the power-on clear voltage, the clock oscillator starts operating and the program starts at address 0000H after a 50 ms halt.

Fig. 15-5 Power-On Reset and Power Voltage V_{DD}

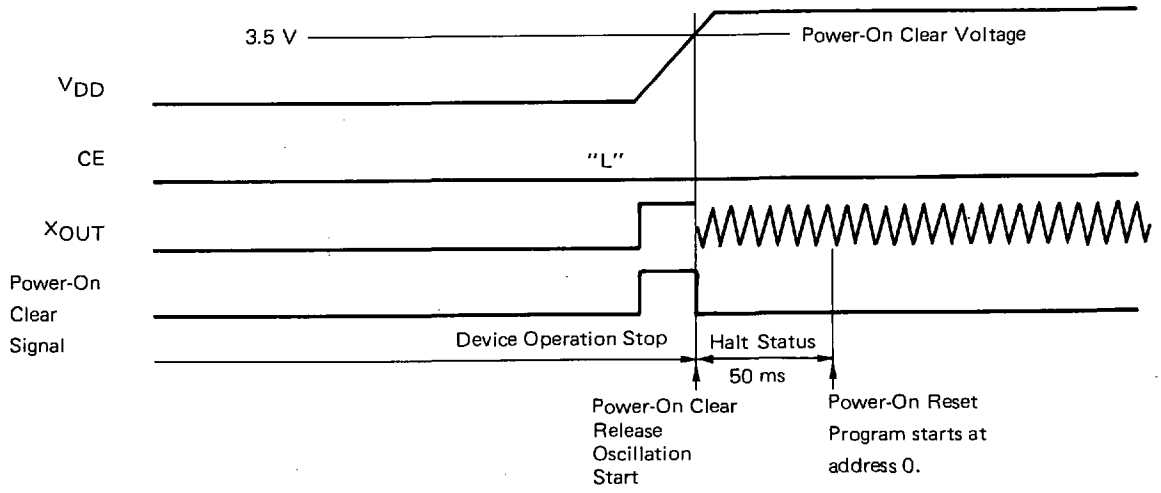
(a) Normal operation (including halt status)



(b) Clock stop



(c) When power voltage V_{DD} is increased from 0 V



15.5 RELATIONS BETWEEN CE RESET AND POWER-ON RESET

When the power voltage is turned on for the first time, power-on reset and CE reset may be applied at the same time.

Reset operation at that time is described in **15.5.1** to **15.5.3.**

Power voltage raising precautions are described in **15.5.4.**

15.5.1 When V_{DD} Pin and CE Pin Start at the Same Time

Operation is shown in Fig. 15-6 (a).

In this case, the program starts at address 0000H by power-on reset.

15.5.2 When CE Pin Starts in Power-On Reset Forced Halt

Operation is shown in Fig. 15-6 (b).

As is the case in **15.5.1**, the program starts at address 0000H by power-on reset.

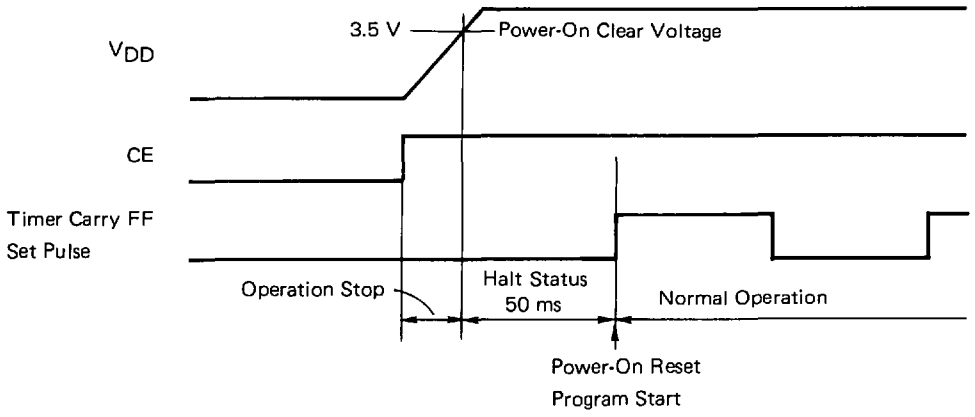
15.5.3 When CE Pin Starts After Power-On Reset

Operation is shown in Fig. 15-6 (c).

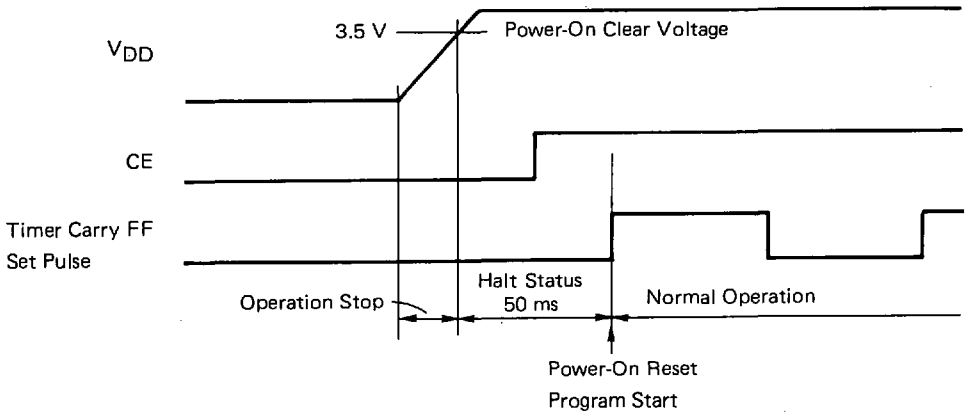
In this case, the program starts at address 0000H by power-on reset and restarts at address 0000H at the rising edge of the next timer carry FF set signal by CE reset.

Fig. 15-6 Relations between Power-On Reset and CE Reset

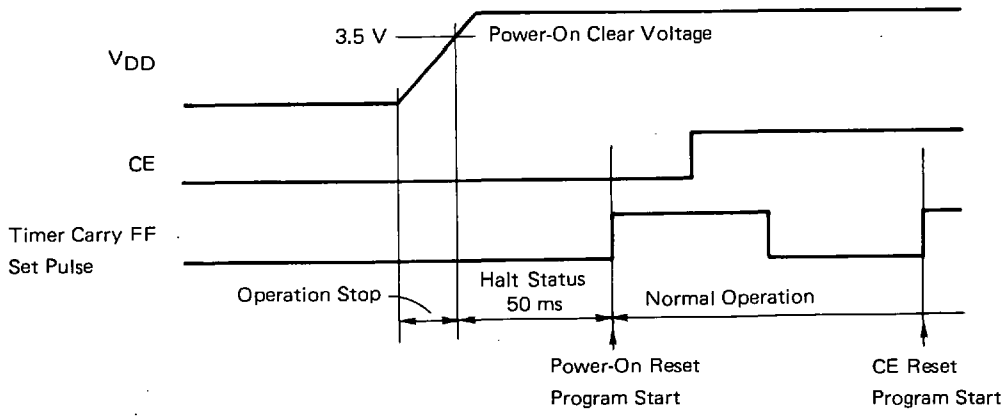
(a) When V_{DD} and CE pin rise at the same time



(b) When CE pin rises in halt status



(c) When CE pin rises after power-on reset



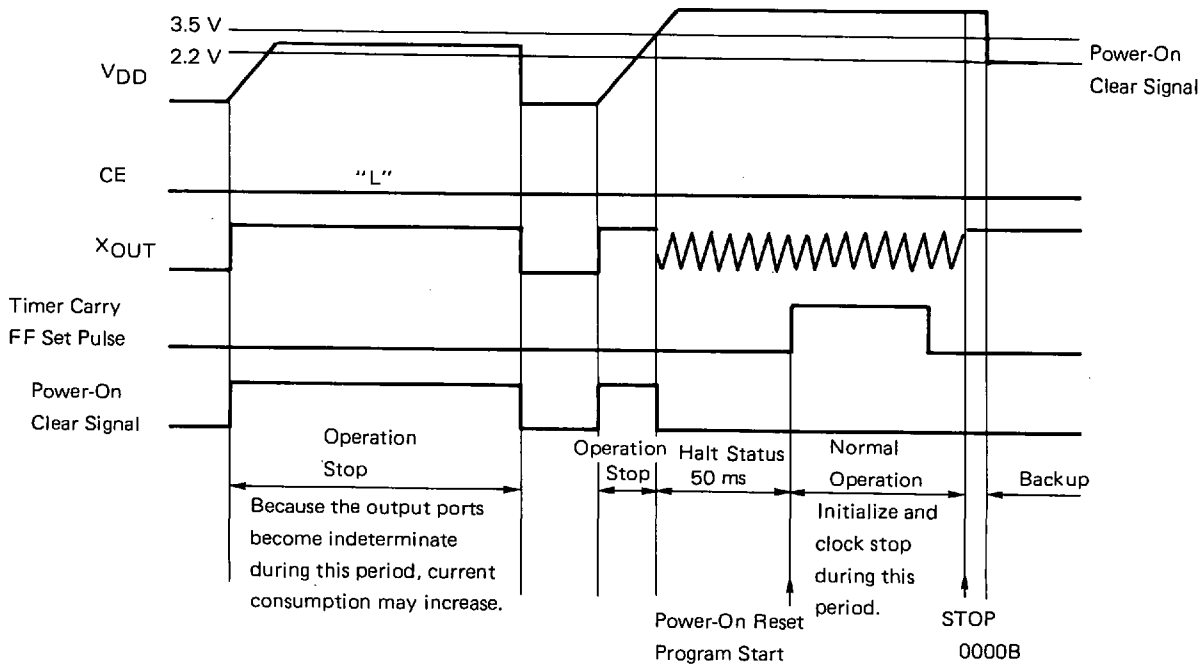
15.5.4 Power Voltage Rising Precautions

When rising the power voltage, note (1) and (2) below.

(1) When rising the power voltage from below the power-on clear voltage

When rising the power voltage, it is necessary to rise above 3.5 V.
This is shown in Fig. 15-7.
As shown in the figure, if only a voltage of less than 3.5 V is applied for V_{DD} input in a program for backup with V_{DD} = 2.2 V using clock stop, a power-on clear signal is output but the program does not operate.
In this case, because the device output ports generate indeterminate values, current consumption rises.
That is, during backup using a battery, the backup time reduces considerably.

Fig. 15-7 V_{DD} Increasing Precautions



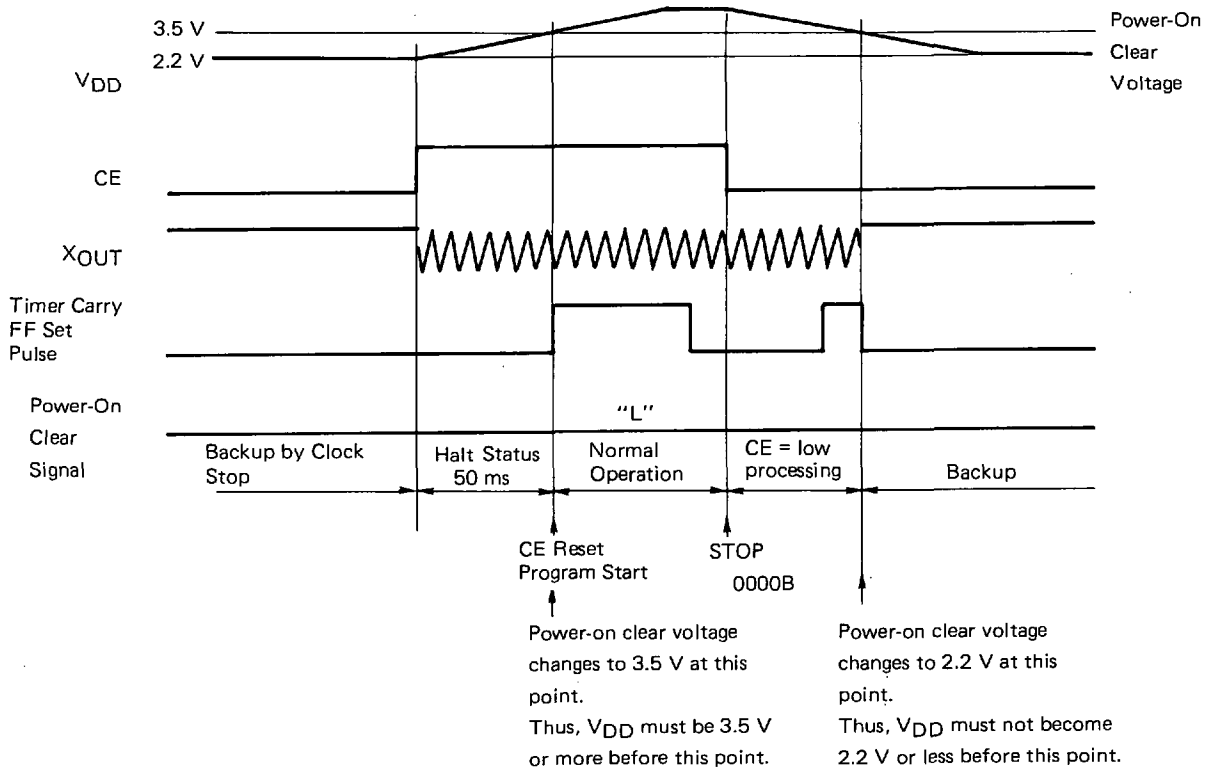
(2) When resetting from clock stop status

When resetting from the backup of power voltage with 2.2 V using clock stop, it is necessary to rise V_{DD} to 3.5 V or more within 50 ms after the CE pin has become high.

As shown in Fig. 15-8, resetting from the clock stop status is carried out by CE reset. However, because the power-on clear voltage changes to 3.5 V 50 ms after CE pin startup, power-on reset in applied if V_{DD} is not 3.5 V or more.

Also, take extra care when reducing V_{DD}.

Fig. 15-8 Resetting from Clock Stop Status

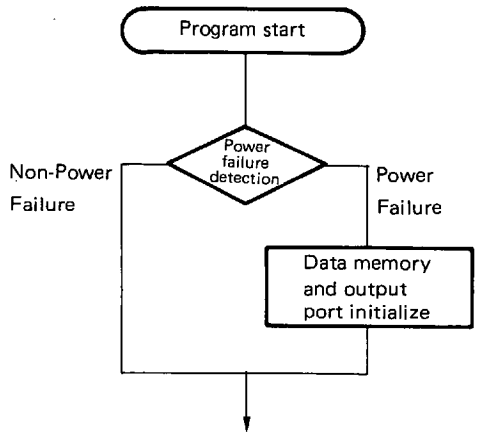


15.6 POWER FAILURE DETECTION

As shown in Fig. 15-9, power failure detection is executed to judge whether device reset has been applied by power voltage V_{DD} input or CE pin reset.

Because the data memory or output port contents are "indeterminate" upon power voltage input, they are initialized by power failure detection.

Fig. 15-9 Power Failure Detection Flowchart



15.6.1 Power Failure Detection Circuit

As shown in Fig. 15-1, the power failure detection circuit consists of a voltage detection circuit, a timer carry disable flip-flop to be reset by voltage detection circuit output (power-on clear signal) and a timer carry flip-flop.

The timer carry disable FF is set (1) by the power-on clear signal and reset (0) when the TMCY flag (bit b_0 of address 17H) read instruction is executed.

When the timer carry disable FF is set (1), the TMCY flag is not set (1).

That is, when the power-on clear signal is output (upon power-on reset), the program starts with the TMCY flag reset and remains disabled for set until the TMCY flag read instruction is executed.

Once the TMCY flag read instruction is executed, the TMCY flag is set at each rising edge of the timer carry FF set pulse. Thus, when the device is reset, the TMCY flag contents are detected. If the TMCY flag is reset (0) or set (1), power-on reset (power failure) or CE reset (non-power failure) is identified, respectively.

Because the voltage from which power failure can be detected is equal to the power-on reset voltage, it is about 3.5 V in clock oscillation and about 2.2 V upon clock stop.

Fig. 15-10 shows the TMCY flag status change.

Fig. 15-11 shows the timing chart of Fig. 15-10 and the TMCY flag operation.

Fig. 15-10 TMCY Flag Status Change

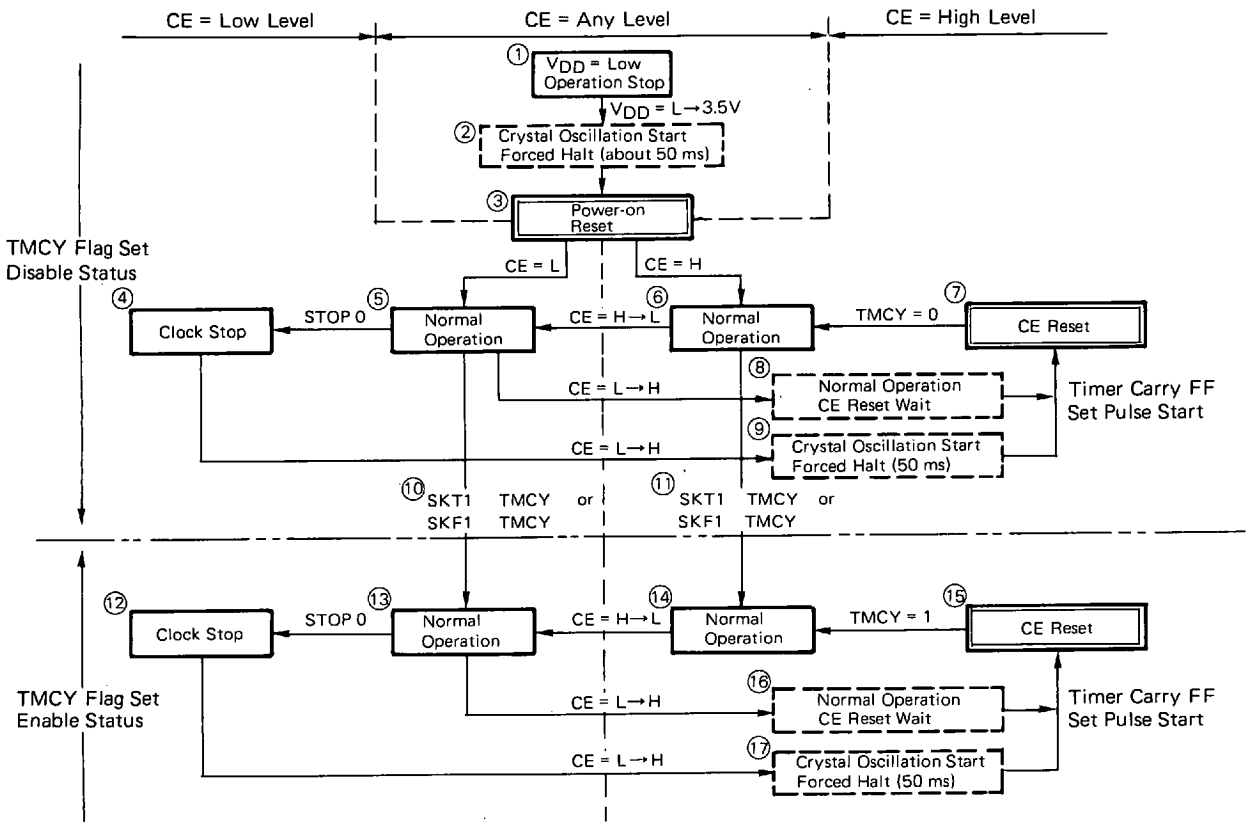
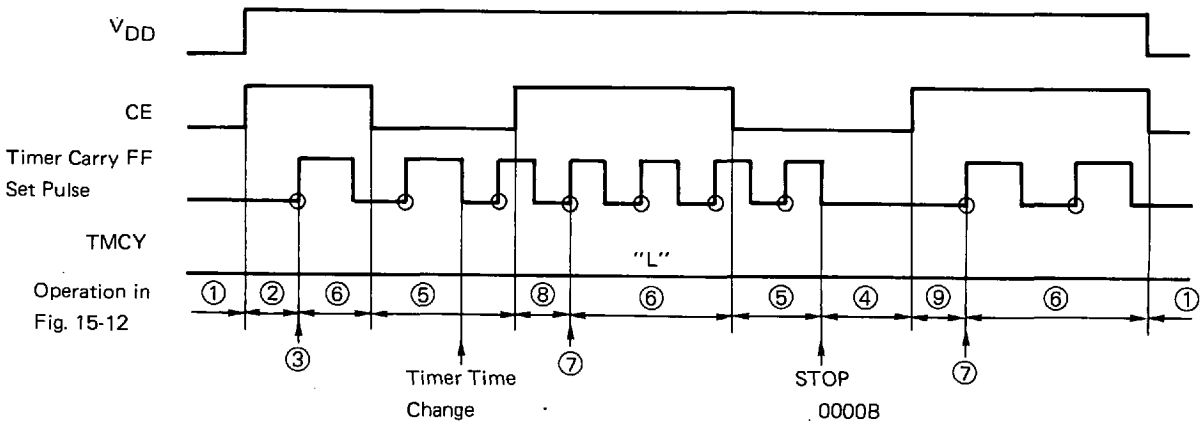
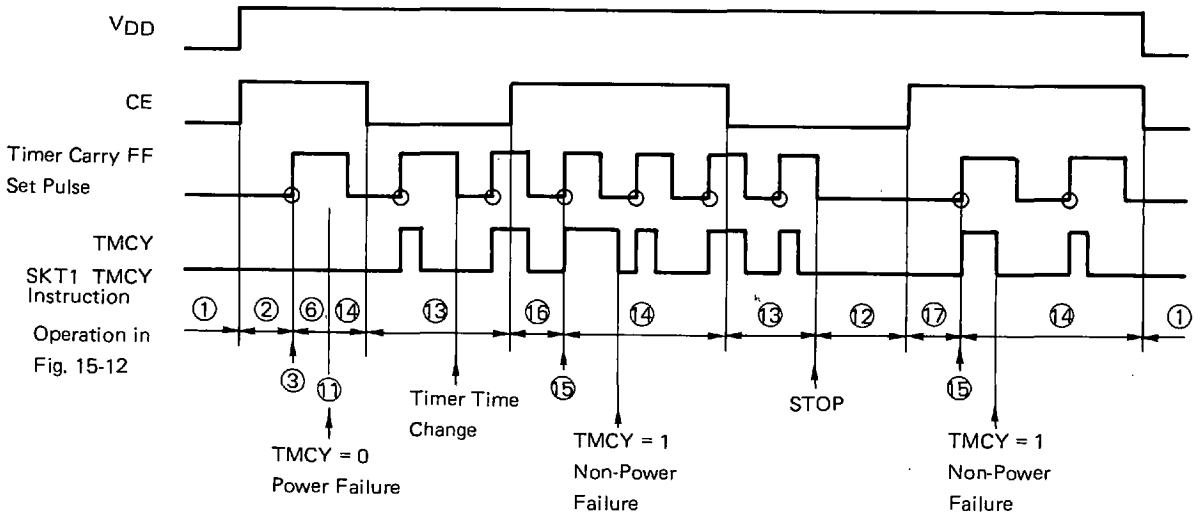


Fig. 15-11 TMCY Flag Operation

(a) When no TMCY flag is detected (SKT1 TMCY or SKF1 TMCY is not executed)



(b) When power failure is detected by TMCY flag



15.6.2 Precautions Concerning Power Failure Detection by TMCY Flag

Note the following when counting the clock time by the TMCY flag.

(1) Clock update

When making a clock program using a timer carry FF, it is necessary to update the clock after power failure detection.

This is because TMCY flag is reset (0) to read the TMCY flag upon power failure detection and one clock count is missed.

(2) Clock update time

Clock update must be terminated before the next timer carry FF set pulse starts.

This is because if the CE pin becomes high level during clock update, CE reset is applied before clock update is executed completely.

For details of (1) and (2), refer to **13.4.2 "Timer Carry FF Correction upon CE Reset".**

Note the following for processing upon power failure.

(3) Power failure detection timing

When counting the clock with the TMCY flag, the TMCY flag must be read for power failure detection before the next timer carry FF set pulse starts after the program starts at address 0000H.

This is because if, for example, the timer carry FF set time is set to 5 ms and power failure detection is done 6 ms after the program has started, one TMCY flag will be missed.

For details, refer to **13.4.2 "Timer Carry FF Correction upon CE Reset".**

As shown in the following example, power failure detection and initialization must be carried out within the timer carry FF set time.

This is because if the CE pin starts and CE set is applied during power failure processing or initialization, this operation is discontinued and a fault may occur.

When changing the timer carry FF set time by initialization, one instruction at the end of initialization must be used.

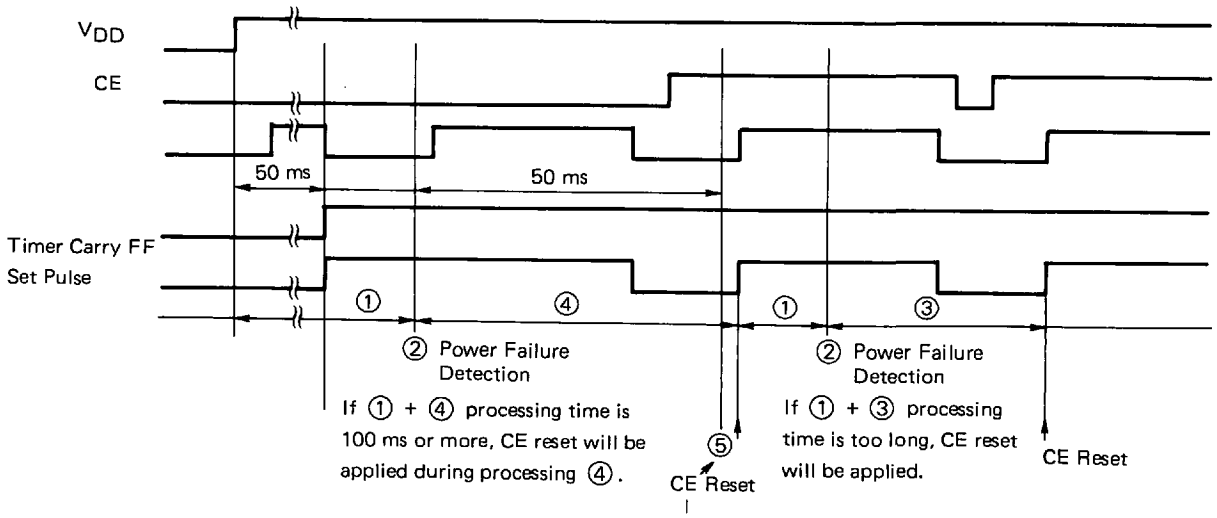
This is also because, as shown in the following example, initialization may not be executed completely due to CE reset if the timer carry FF set time is changed before initialization.

Example:

Program example

```
START :                    ; Program address 0000H
  ; ①
  Processing upon reset ;
  ; ②
  SKT1   TMCY              ; Power failure detection
  BR     INITIAL
BACKUP:
  ; ③
  Clock update
  BR     MAIN
INITIAL:
  ; ④
  Initialization
  ; ⑤
  INITFLG  NOT ZCROSS, NOT TMMD2, NOT TMMD1, TMMD0
                           ; Intrinsic macro
                           ; Timer carry FF set time is set to 5 ms.
MAIN :
  SKT1   TMCY
  BR     MAIN
  Clock update
```

Operation example



CE reset may be immediately applied depending on the timer carry FF set time change timing. Thus, if ⑤ is executed before ④, power failure processing ④ will not be executed completely.

16. GENERAL-PURPOSE PORTS

The general-purpose pors are intended to generate high-level, low-level and floating signals to the external circuit and read the high-level and low-level signals of the external circuit.

16.1 GENERAL-PURPOSE PORT CONFIGURATION AND CLASSIFICATION

Fig. 16-1 is a block diagram of the general-purpose port.

Also, Table 16-1 gives the general-purpose classification.

As shown in the figure, the general-purpose port system consists of port 0A (P0A) to port 1D (P1D) which set data using addresses 70H to 73H (port register) of each bank of the data memory. A pair of BANK0 and BANK2 and a pair of BANK1 and BANK3 are assigned the same port register.

Each port consists of general-purpose port pins. For example, port 0A consists of $P0A_3$ pin to P0A, pin.

As shown in Table 16-1, the general-purpose ports are classified into input/output dual-function ports (input/output ports), input dedicated ports (input ports) and output dedicated ports (output ports).

The inpurt/output ports are classified into a bit I/O port which can specify input/output in 1-bit units (1-pin units) and a group I/O port which can specify input/output in 3-bit units (3-pin units).

Fig. 16-1 General-Purpose Port Block Diagram

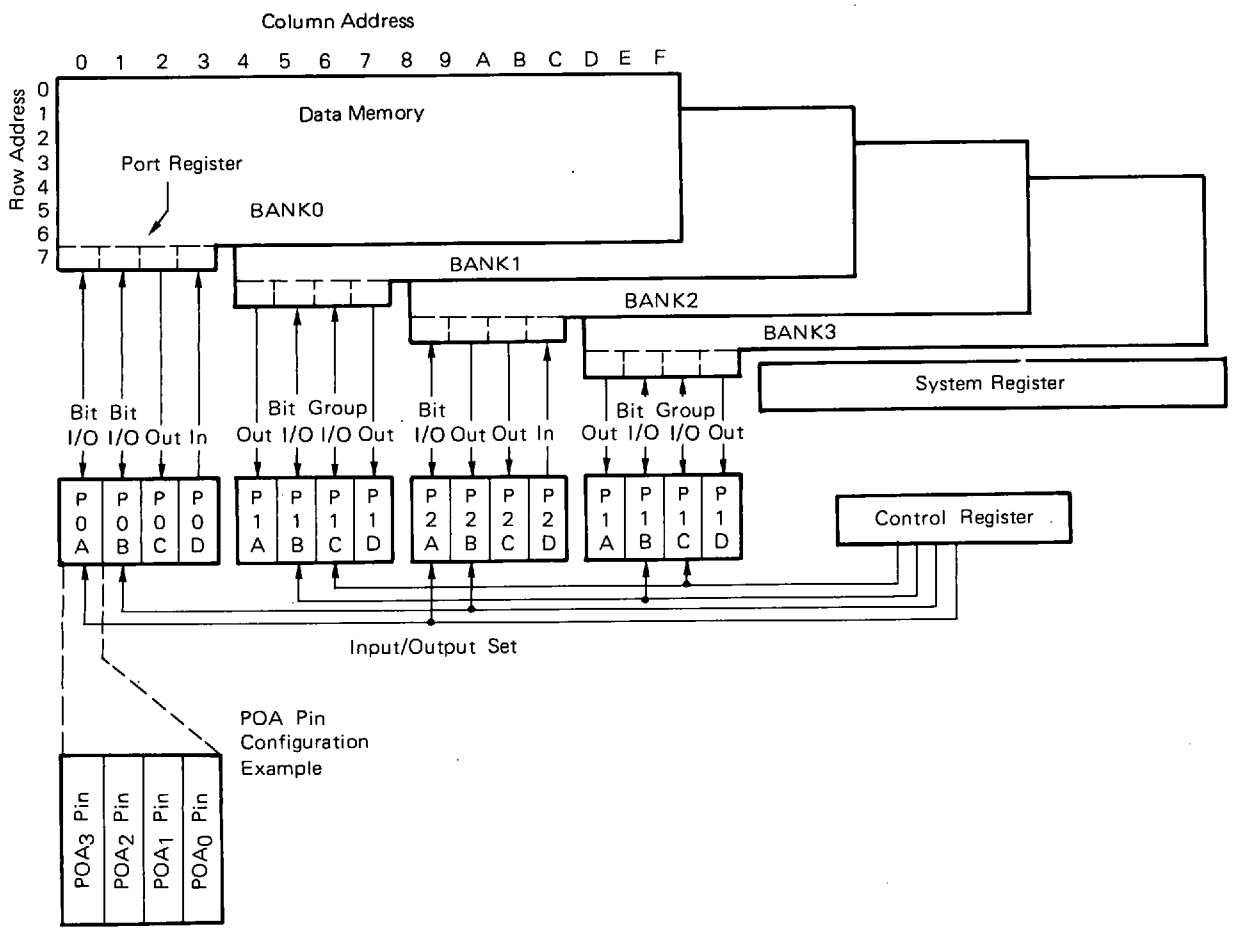


Table 16-1 General-Purpose Port Classification

General-Purpose Port Classification			Target Port	Data Setting Method
General-purpose port	Input/output dual-function port	Bit I/O	Port0A Port0B Port1B Port2A	Port register
		Group I/O	Port1C	Port register
	Dedicated input port		Port0D	Port register
	Dedicated output port		Port0C Port1A Port1D Port2B POrt2C	Port register

16.2 OUTLINE OF GENERAL-PURPOSE PORT FUNCTIONS

The general-purpose input/output port set for general-purpose output port or output port generates high or low level from the corresponding pin by setting data to the port register.

The general-purpose input/output port set for general-purpose input port or input port can detect the input signal level applied to the corresponding pin by reading the port register contents.

The general-purpose input/output port is switched to the input or output port with the control register corresponding to each port.

That is, input/output can be switched by program.

Because P0A to P0D, P1A to P1D and P2A to P2C are set to general-purpose ports upon power-on reset, pins which also serve as another peripheral hardware are independently set with the corresponding control register.

The port register functions and outline of each port function are described in **16.2.1** to **16.2.4.**

16.2.1. General-Purpose Port Register (Port Register)

The port register sets each general-purpose port output data and reads input data.

Because the port register is located in the data memory, it can be operated by all data memory manipulate instructions.

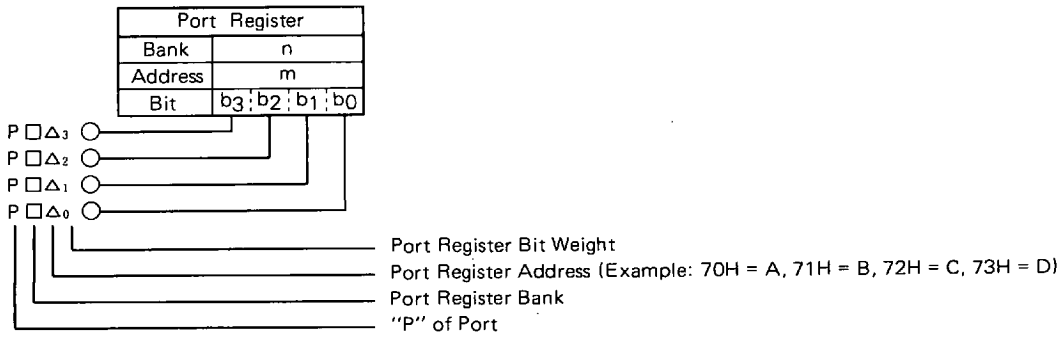
Fig. 16-2 shows the relations between the port register and the corresponding pins.

Each pin output is set by setting data to the port register corresponding to the pin set for the general-purpose output port.

Each pin input status is detected by reading data from the port register corresponding to the pin set for the general-purpose input port.

Table 16-2 shows the relations between each port (pin) and the port register.

Fig. 16-2 Relations between Port Register and Each Pin



A reserved word is defined in the assembler for the port register.

Because this reserved word is defined in flag (bit) units, as assembler assemble macro instruction can be used.

Note that data memory type reserved words are not defined for the port register.

16.2.2 General-Purpose Input-Output Ports (P0A, P0B, P1B, P1C, P2A)

P0A, P0B, P1B, P1C and P2A input/output are selected using the P0A bit I/O select register (RF address 37H), the P0B bit I/O select register (RF address 36H), the P1B bit I/O select register (RF address 35H), the P1C group I/O select register (RF address 27H), and the P2A bit I/O select register (RF address 34H), respectively.

P0A, P0B, P1B, P1C and P2A input/output data are set at P0A (data memory address 70H of BANK0), P0B (data memory address 71H of BANK0), P1B (data memory address 71H of BANK1 or BANK3), P1C (data memory address 72H of BANK1 or BANK3) and P2A (data memory address 70H of BANK2) of the port register, respectively.

Refer to Table 16-2.

Refer to **16.3** for details.

16.2.3 General-Purpose Input Port (P0D)

P0D input data is read at P0D (data memory address 73H of BANK0 or BANK2) of the port register.

Refer to Table 16-2.

Refer to **16.4** for datails.

16.2.4 General-Purpose Output Ports (P0C, P1A, P1D, P2B, P2C)

P0C, P1A, P1D, P2B and P2C output data are set at P0C (data memory address 72H of BANK0), P1A (data memory address 70H of BANK1 or BANK3), P0C (data memory address 73H of BANK1 or BANK3), P2B (data memory address 71H of BANK2) and P2C (data memory address 72H of BANK2), respectively.

Refer to Table 16-2.

Refer to **16.5** for details.

Table 16-2 Relations of Each Port (Pin) and Port Register

Port	Pin			Data Setting Method				
				Port Register (Data Memory)				
	No.	Symbol	Input/Output	Bank	Address	Symbol	Bit Symbol (Reserved Word)	
Port 0A (P0A)	39	$P0A_3$	Input/Output (Bit I/O)	BANK0	70H	P0A	b_3	P0A3
	40	$P0A_2$					b_2	P0A2
	41	$P0A_1$					b_1	P0A1
	42	$P0A_0$					b_0	P0A0
Port 0B (P0B)	35	$P0B_3$	Input/Output (Bit I/O)		71H	P0B	b_3	P0B3
	36	$P0B_2$					b_2	P0B2
	37	$P0B_1$					b_1	P0B1
	38	$P0B_0$					b_0	P0B0
Port 0C (P0C)	51	$P0C_3$	Output		72H	P0C	b_3	P0C3
	52	$P0C_2$					b_2	P0C2
	53	$P0C_1$					b_1	P0C1
	54	$P0C_0$					b_0	P0C0
Port 0D (P0D)	59	$P0D_3$	Input		73H	P0D	b_3	P0D3
	60	$P0D_2$					b_2	P0D2
	61	$P0D_1$					b_1	P0D1
	62	$P0D_0$					b_0	P0D0
Port 1A (P1A)	18	$P1A_3$	Output	BANK1 BANK3	70H	P1A	b_3	P1A3
	19	$P1A_2$					b_2	P1A2
	20	$P1A_1$					b_1	P1A1
	21	$P1A_0$					b_0	P1A0
Port 1B (P1B)	9	$P1B_3$	Input/Output (Bit I/O)		71H	P1B	b_3	P1B3
	10	$P1B_2$					b_2	P1B2
	11	$P1B_1$					b_1	P1B1
	12	$P1B_0$					b_0	P1B0
Port 1C (P1C)	55	$P1C_3$	Input/Output (Group I/O)		72H	P1C	b_3	P1C3
	56	$P1C_2$					b_2	P1C2
	57	$P1C_1$					b_1	P1C1
	58	$P1C_0$					b_0	P1C0
Port 1D (P1D)	1	$P1D_3$	Output		73H	P1D	b_3	P1D3
	2	$P1D_2$					b_2	P1D2
	3	$P1D_1$					b_1	P1D1
	4	$P1D_0$					b_0	P1D0

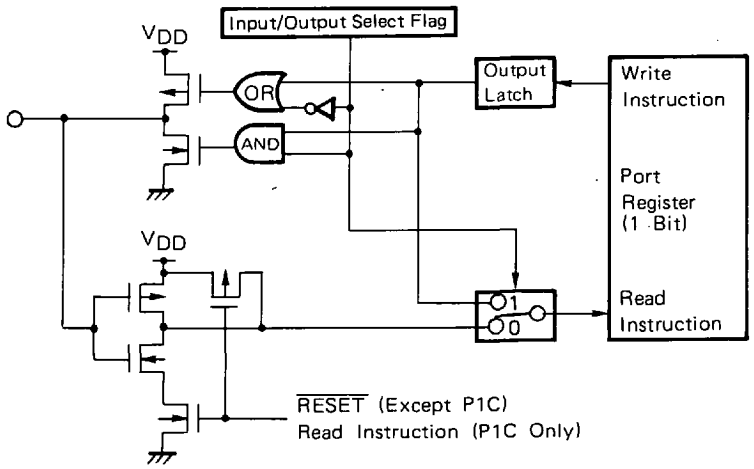
Port	Pin			Data Setting Method				
				Port Register (Data Memory)				
	No.	Symbol	Input/Output	Bank	Address	Symbol	Bit Symbol (Reserved Word)	
Port 2A (P2A)	5	$P2A_3$	Input/Output (Bit I/O)	BANK2	70H	P2A	b_3	P2A3
	6	$P2A_2$					b_2	P2A2
	7	$P2A_1$					b_1	P2A1
	8	$P2A_0$					b_0	P2A0
Port 2B (P2B)	43	$P2B_3$	Output		71H	P2B	b_3	P2B3
	44	$P2B_2$					b_2	P2B2
	45	$P2B_1$					b_1	P2B1
	46	$P2B_0$					b_0	P2B0
Port 2C (P2C)	47	$P2C_3$	Output		72H	P2C	b_3	P2C3
	48	$P2C_2$					b_2	P2C2
	49	$P2C_1$					b_1	P2C1
	50	$P2C_0$					b_0	P2C0
Port 0D (P0D)	59	$P0D_3$	Input		73H	P0D	b_3	P0D3
	60	$P0D_2$					b_2	P0D2
	61	$P0D_1$					b_1	P0D1
	62	$P0D_0$					b_0	P0D0

16.3 GENERAL-PURPOSE INPUT/OUTPUT PORTS (P0A, P0B, P1B, P1C, P2A)

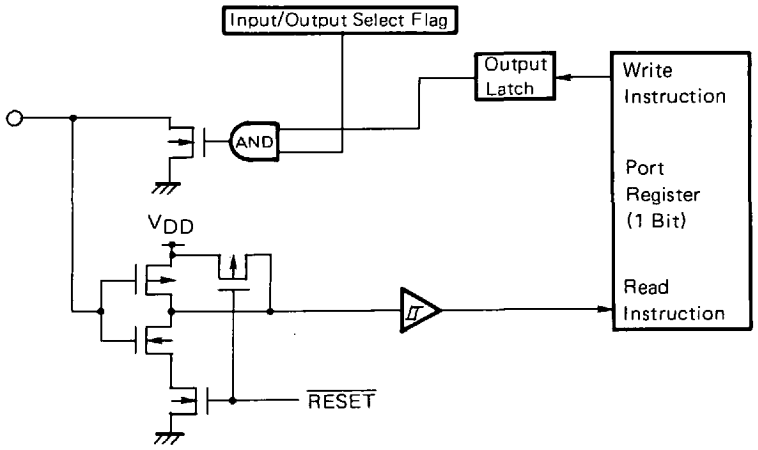
16.3.1. Input-Output Port Configurations

The input/output port configurations are shown in (1) to (3).

(1) P0A, ($P0A_3$ and $P0A_2$ pins)
P0B ($P0B_3$, $P0B_2$, $P0B_1$ and $P0B_0$ pins)
P1B ($P1B_3$, $P1B_2$, $P1B_1$ and $P1B_0$ pins)
P1C ($P1C_3$, $P1C_2$, $P1C_1$ and $P1C_0$ pins)
P2A ($P2A_3$, $P2A_2$, $P2A_1$ and $P2A_0$ pins)



(2) P0A ($P0A_1$ and $P0A_0$ pins)



16.3.2. Input/Output Port Operating Method

Input or output is set for the input/output port using the I/O select registers, P0A, P0B, P1B, P1C and P2A, of the control register.

Input/output can be set for the bit I/O ports (P0A, P0B, P1B and P2A) in 1-bit units (1-pin units) and for the group I/O port (P0C) in 4-bit units (4-pin units).

Output data is set or input data is read by executing an instruction for writing or reading data to the corresponding port register, respectively.

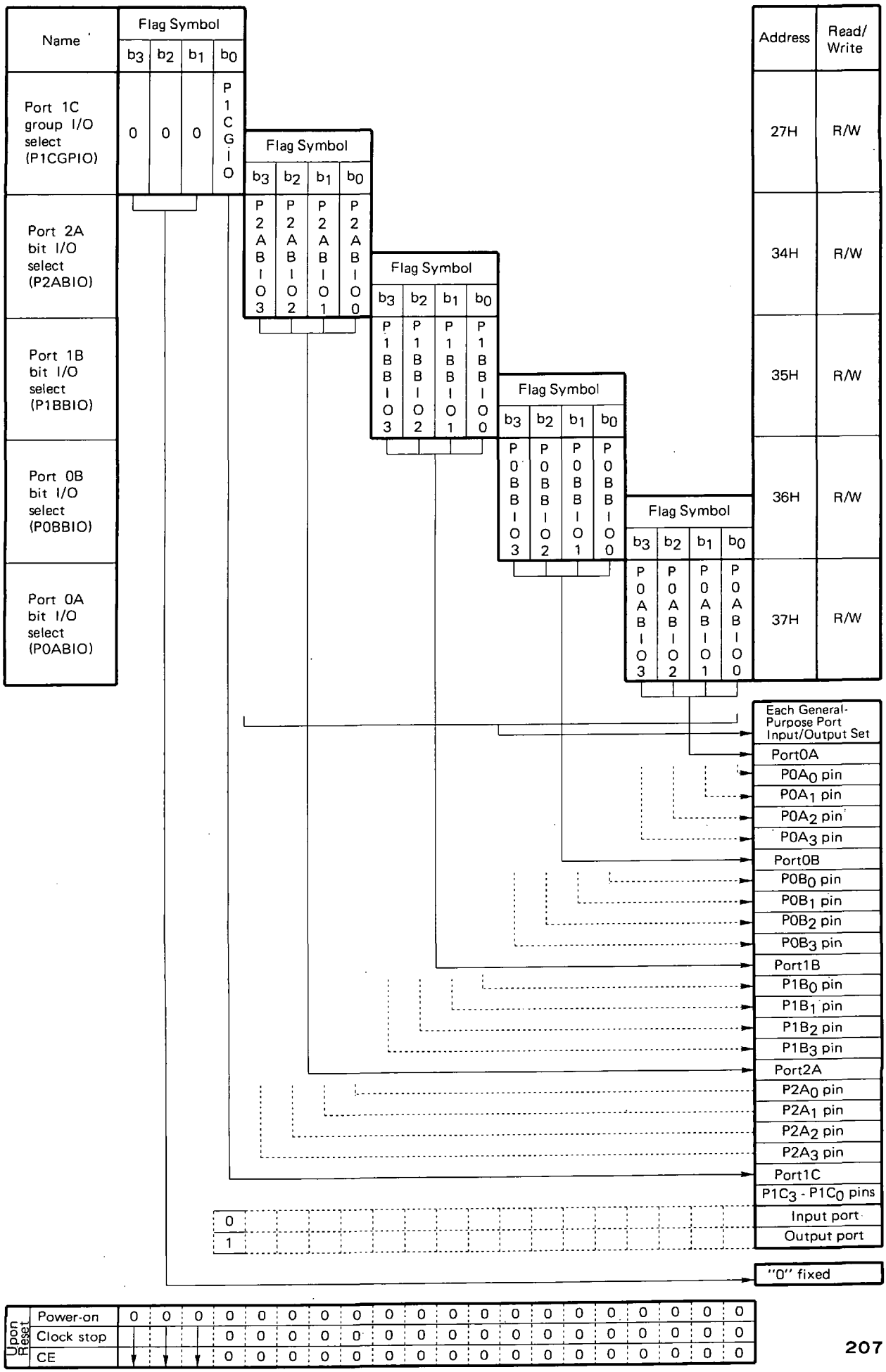
The I/O select register of each port is described in **16.3.3.**

Input or output port operating method is described in **16.3.4** and **16.3.5.**

16.3.3 Port 0A BIT I/O SELECT REGISTER (P0ABIO)
Port 0B BIT I/O SELECT REGISTER (P0BBIO)
Port 1B BIT I/O SELECT REGISTER (P1BBIO)
Port 1C GROUP I/O SELECT REGISTER (P1CGPIO)
Port 2A BIT I/O SELECT REGISTER (P2ABIO)

Port 0A bit I/O, Port 0B bit I/O, Port 1B bit I/O, Port 1C group I/O select registers and Port 2A bit I/O select register set input/output for the P0A, P0B, P1B, P1C and P2A pins, respectively.

The configurations and functions are shown below.


Name
Flag Symbol
b3 b2 b1 b0
Address
Read/Write
Port 1C group I/O select (P1CGPIO)
0 0 0 P1CGIO
27H R/W
Port 2A bit I/O select (P2ABIO)
P2ABIO3 P2ABIO2 P2ABIO1 P2ABIO0
34H R/W
Port 1B bit I/O select (P1BBIO)
P1BBIO3 P1BBIO2 P1BBIO1 P1BBIO0
35H R/W
Port 0B bit I/O select (P0BBIO)
P0BBIO3 P0BBIO2 P0BBIO1 P0BBIO0
36H R/W
Port 0A bit I/O select (P0ABIO)
P0ABIO3 P0ABIO2 P0ABIO1 P0ABIO0
37H R/W
Each General-Purpose Port Input/Output Set
Port0A
P0A0 pin
P0A1 pin
P0A2 pin
P0A3 pin
Port0B
P0B0 pin
P0B1 pin
P0B2 pin
P0B3 pin
Port1B
P1B0 pin
P1B1 pin
P1B2 pin
P1B3 pin
Port2A
P2A0 pin
P2A1 pin
P2A2 pin
P2A3 pin
Port1C
P1C3 - P1C0 pins
0 Input port
1 Output port
"0" fixed
Upon Reset
Power-on 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Clock stop 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CE 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

16.3.4 When Using Input/Output Ports (P0A, P0B, P1B, P1C, P2A) as Input Ports

Select the pins to be used as input ports with the I/O select register at each port.

The pins specified for the input ports are set to the floating status (Hi-Z) waiting for an external signal input.

Input data can be read by executing an instruction for reading the contents of the port register corresponding to each pin such as the "SKT" instruction.

If a high level has been input to each pin, "1" is read to the port register. If a low level has been input, "0" is read.

When a write instruction such as the "MOV" instruction is executed for the port register specified for an input port, the contents of the output latch are rewritten.

16.3.5 When Using Input/Output Ports (P0A, P0B, P1B, P1C, P2A) as Output Ports

Select the pins to be used as output ports with the I/O select register at each port.

The pins specified for the output ports generate the output latch contents.

Output data can be set by executing an instruction for writing the contents of the port register corresponding to each pin such as the "MOV" instruction.

Write "1" to output a high level to each pin and "0" to output a low level.

Each pin can be set to the floating status (Hi-Z) by being specified for an input port.

When an instruction such as the "SKT" instruction for reading the port register specified for an output port is executed, the contents of the output latch are read.

In this case, however, the output latch contents may differ from what are read because the $P0A_1$ and $P0A_0$ pin statuses are read what they are.

Refer to **16.3.6** for details.

16.3.6 Input/output Port ($P0A_1$ and $P0A_0$ pins) Operating Precautions

As described in the following example, when using the $P0A_1$ and $P0A_0$ pins as output ports, the output latch contents may be rewritten.

Example:

```
INITFLG     NOT P0ABIO3, NOT P0ABIO2, P0ABIO1, P0ABIO0
                ; P0A1 and P0A0 pins are specified as output ports.
INITFLG     NOT P0A3, NOT P0A2, P0A1, P0A0
; ①              ; High level is output to P0A1 and P0A0 pins.
CLR1        P0A1; Low level is output to P0A1 pin.
;  Macro development
   AND  .MF.P0A1 SHR 4, #.DF. (NOT P0A1 and 0FH)
```

If the $P0A_0$ pin happens to be pulled at the low level when ① instruction is executed, the $P0A_0$ pin output latch contents are rewritten to "0" by the "CLR1" instruction.

16.3.7 Input/output Port (P0A, P0B, P1B, P1C, P2A) Reset Statuses

(1) Upon power-on reset

All ports are specified for input ports.

Because the output latch contents are "indeterminate", initialize by program before switching to the output ports when necessary.

(2) Upon CE reset

All ports are specified for input ports.

The output latch contents are held.

(3) Upon clock stop

All ports are specified for input ports.

The output latch contents are held.

All input/output ports except P1C prevent current consumption from increasing due to input buffer noise by $\overline{\text{RESET}}$ signal output upon clock stop as shown in Fig. 17-3.

If P1C is in the floating status upon clock stop, current consumption may increase due to external noise. Thus, pull it down or up externally when necessary.

(4) In halt status

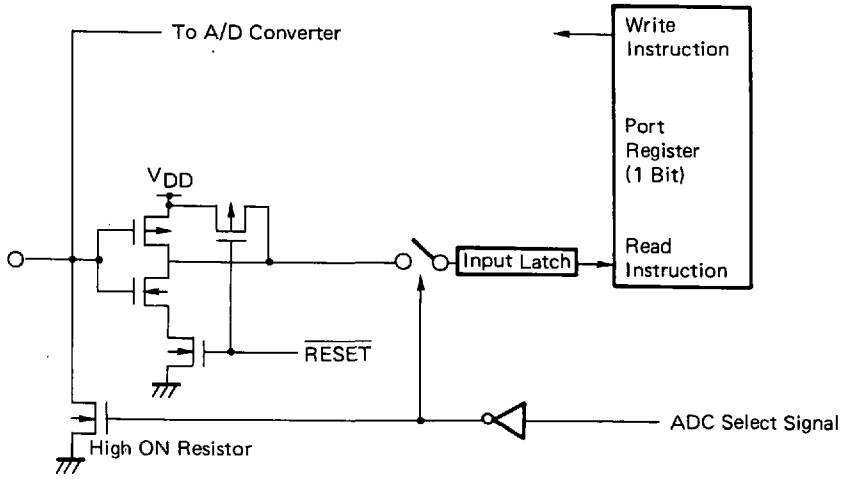
The previous status is held.

16.4 GENERAL-PURPOSE INPUT PORT (P0D)

16.4.1 Input Port Configuration

The input port configuration is shown below.

(1) P0D ($P0D_3$, $P0D_2$, $P0D_1$, $P0D_0$)



16.4.2 Input Port (P0D) Usage Example

Input data can be read by executing an instruction such as the "SKT" instruction for reading the contents of the port register corresponding to each pin.

When a high or low level is input to each pin, "1" or "0" is read to the port register, respectively.

If a write instruction such as the "MOV" instruction is executed for the port register, nothing changes.

16.4.3 Input Port (P0D) Operating Precautions

When using P0D as a general-purpose port, it is internally pulled down.

16.4.4 Input Port (P0D) Reset Status

(1) Upon Power-on reset

All ports are specified for general-purpose input ports.

(2) Upon CE reset

All ports are specified for general-purpose input ports.

(3) Upon clock stop

All ports are specified for general-purpose input ports.

Because the $\overline{RESET}$ signal is output upon clock stop, current consumption due to input buffer noise is prevented from increasing as shown in Fig. 16-4.

P0D is internally pulled down.

(4) In the halt status

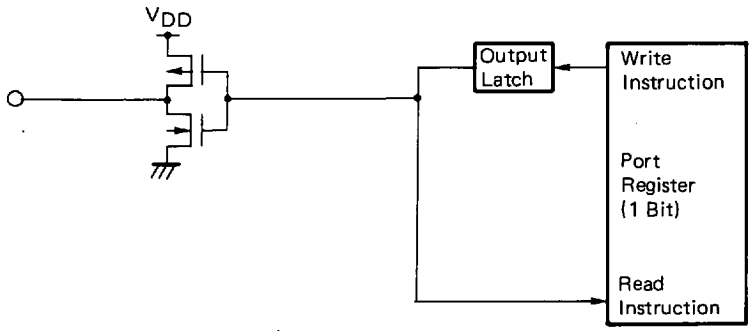
The previous status is held.

16.5 GENERAL-PURPOSE OUTPUT PORTS (P0C, P1A, P1D, P2B, P2C)

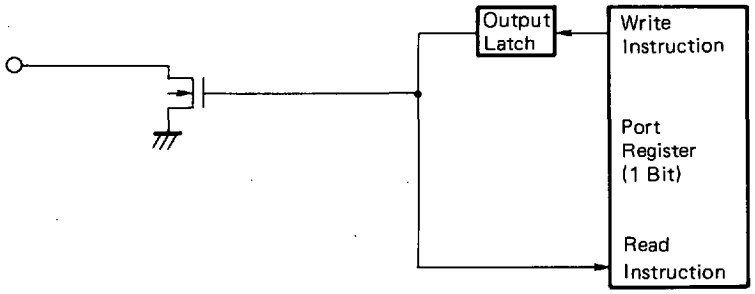
16.5.1 Output Port (P0C, P1A, P1D, P2B, P2C) Configurations

The output port configurations are shown in (1) and (2) below.

(1) P0C ($P0C_3$, $P0C_2$, $P0C_1$ and $P0C_0$ pins)
P1D ($P1D_3$, $P1D_2$, $P1D_1$ and $P1D_0$ pins)



(2) P1A ($P1A_3$, $P1A_2$, $P1A_1$ and $P1A_0$ pins)
P2B ($P2B_3$, $P2B_2$, $P2B_1$ and $P2B_0$ pins)
P2C ($P2C_3$, $P2C_2$, $P2C_1$ and $P2C_0$ pins)



16.5.2 Output Port (P0C, P1A, P1D, P2B, P2C) Usage Example

The output ports generate output latch contents from each pin.

Output data is set by executing an instruction such as the "MOV" instruction for writing the contents of the port register corresponding to each pin.

Write "1" or "0" to generate a high or low level to each pin, respectively.

Because P1A, P2B and P2C and N-ch generate open-drain output, they are set to floating status (Hi-Z) for high-level output.

When a port register read instruction such as the "SKT" instruction is executed, the output latch contents are read.

16.5.3 Output Port (P0C, P1A, P1D, P2B, P2C) Reset Statuses

(1) Upon power-on reset

Output latch contents are generated.

Because the output latch contents are "indeterminate", "indeterminate" value are output for the specified time (until initialized by program).

(2) Upon CE reset

Output latch contents are generated.

Because the output latch contents are held, output data remains unchanged upon CE reset.

(3) Upon clock stop

Output latch contents are output.

Because the output latch contents are held, output data remains unchanged upon clock stop.

Thus, initialized by program when necessary.

(4) In halt status

The output latch contents are output.

Because the output latch contents are held, output data remains unchanged in the halt status.

17. SERIAL INTERFACE

The μPD17051 has two serial interface pins, channel 0 (CH0) and channel 1 (CH1), for data communication.

The CH0 pin consists of SDA and SCL. It can be operated in three modes, two clock synchronize modes (2-wire serial input mode and 2-wire serial output mode) and 2-wire bus mode. The SDA and SCL pins can be used as general-purpose ports when not used as serial interface.

The CH1 pin consists of $\overline{SCK}$, SO and SI. It can be operated in the 2-wire serial I/O input, 2-wire serial I/O output and 3-wire serial I/O input/output modes.

CH0 and CH1 cannot be operated simultaneously. CH0 or CH1 is specified by the SIOCH flag (register file: 08H, b_3) of SMODE (serial interface mode register).

The 2-wire bus mode supported by hardware is used for the single master. Thus, because the arbitration function is not available, use software;

Table 17-1 List of External Pins for Serial Interface

CH	Pin Name	Function		
		2-Wire Bus Mode	Serial I/O Mode	Port I/O Set Register
0	$P0A_0$/SDA	Serial data input/output	Serial data input/output	P0ABIO0
	$P0A_1$/SCL	Shift clock input/output	Shift clock input/output	P0ABIO1
1	$P0A_2$/SCK	Use inhibited	Shift clock input/output	P0ABIO2
	$P0A_3$/SO		Serial data output	P0ABIO3
	$P0B_0$/SI		Serial data input	P0BBIO0

17.1 SERIAL INTERFACE MODE REGISTER

The serial interface mode register is intended to specify the serial interface operation mode. It sets the channel, protocol and clock to be used and send/receive operations.

The serial interface mode register is located at address 08H of the register file.

All flags of the serial interface mode register are set to "0" upon power-on reset.

Fig. 17-1 Serial Interface Mode Register Configuration

Bit position	b_3	b_2	b_1	b_0
Flag name	SIOCH	SB	SIOMS	SIOTX

Table 17-2 CH0 Operation Mode List

Serial Interface Mode Register			Port 0A I/O Specification		SDA Pin	SCL Pin	Operation Mode
SB	SIOMS	SIOTX	P0ABIO0	P0ABIO1			
0	0	0	0	0	SD-IN	CK-IN	Serial I/O-SI, EXT-CLK
0	0	0	0	1	SD-IN	OUT-PORT	Serial I/O-SI, INT-CLK (SOFT-CLK)
0	0	0	1	0	OUT-PORT	IN-PORT	1OUT-PORT+1IN-PORT
0	0	0	1	1	OUT-PORT	OUT-PORT	2OUT-PORT
0	0	1	x	0	SD-OUT	CK-IN	Serial I/O-SO, EXT-CLK
0	0	1	x	1	SD-OUT	OUT-PORT	Serial I/O-SO, INT-CLK (SOFT-CLK)
0	1	0	0	x	SD-IN	CK-OUT	Serial I/O-SI, INT-CLK
0	1	0	1	x	OUT-PORT	CK-OUT	CLK-OUT+1OUT-PORT
0	1	1	x	x	SD-OUT	CK-OUT	Serial I/O-SO, INT-CLK
1	0	0	0	0	SD-IN	CK-IN	BUS-SLAVE-RX
1	0	0	0	1	SD-IN	OUT-PORT	BUS-MASTER-RX(SOFT-CLK)
1	0	0	1	0	OUT-PORT	IN-PORT	1OUT-PORT+1IN-PORT
1	0	0	1	1	OUT-PORT	OUT-PORT	2OUT-PORT
1	0	1	x	0	SD-OUT	CK-IN	BUS-SLAVE-TX
1	0	1	x	1	SD-OUT	OUT-PORT	BUS-MASTER-TX(SOFT-CLK)
1	1	0	0	x	SD-IN	CK-OUT	BUS-MASTER-RX
1	1	0	1	x	OUT-PORT	CK-OUT	CLK-OUT+1OUT-PORT
1	1	1	x	x	SD-OUT	CK-OUT	BUS-MASTER-TX

Remarks: x: 0/1

Table 17-3 CH1 Operation Mode List

Serial Interface Mode Register			Port I/O Specification			SI Pin	$\overline{SCK}$ Pin	SO Pin	Operation Mode
SB	SIOMS	SIOTX	P0ABIO2	P0ABIO3	P0BBIO0				
0	0	0	0	0	0	SD-IN	CK-IN	IN-PORT	Serial I/O-SI, EXT-CLK, 1IN-PORT
0	0	0	0	0	1	SD-IN	CK-IN	OUT-PORT	Serial I/O-SI, EXT-CLK, 1OUT-PORT
0	0	0	0	1	0	OUT-PORT	IN-PORT	IN-PORT	1OUT-PORT+2IN-PORT
0	0	0	0	1	1	OUT-PORT	IN-PORT	OUT-PORT	2OUT-PORT+1IN-PORT
0	0	0	1	0	0	SD-IN	OUT-PORT	IN-PORT	Serial I/O-SI, INT-CLK (SOFT-CLK), 1 IN-PORT
0	0	0	1	0	1	SD-IN	OUT-PORT	OUT-PORT	Serial I/O-SI, INT-CLK (SOFT-CLK), 1OUT-PORT
0	0	0	1	1	0	OUT-PORT	OUT-PORT	IN-PORT	2OUT-PORT+1IN-PORT
0	0	0	1	1	1	OUT-PORT	OUT-PORT	OUT-PORT	3OUT-PORT
0	0	1	0	0	x	SD-IN	CK-IN	SD-OUT	Serial I/O-SI/SO, EXT-CLK
0	0	1	0	1	x	OUT-PORT	CK-IN	SD-OUT	Serial I/O-SO, EXT-CLK, 1OUT-PORT
0	0	1	1	0	x	SD-IN	OUT-PORT	SD-OUT	Serial I/O-SI/SO, INT-CLK (SOFT-CLK)
0	0	1	1	1	x	OUT-PORT	OUT-PORT	SD-OUT	Serial I/O-SO, INT-CLK (SOFT-CLK), 1OUT-PORT
0	1	0	x	0	0	SD-IN	CK-OUT	IN-PORT	Serial I/O-SI, INT-CLK, 1IN-PORT
0	1	0	x	0	1	SD-IN	CK-OUT	OUT-PORT	Serial I/O-SI, INT-CLK, 1OUT-PORT
0	1	0	x	1	0	OUT-PORT	CK-OUT	IN-PORT	CLK-OUT, 1OUT-PORT, 1IN-PORT
0	1	0	x	1	1	OUT-PORT	CK-OUT	OUT-PORT	CLK-OUT, 2OUT-PORT
0	1	1	x	0	x	SD-IN	CK-OUT	SD-OUT	Serial I/O-SI/SO, INT-CLK
0	1	1	x	1	x	OUT-PORT	CK-OUT	SD-OUT	Serial I/O-SO, INT-CLK 1OUT-PORT
0	x	x	x	x	x	–	–	–	Operation Mode

Remarks: x: 0/1

17.1.1 SIOCH

This flag is used to select the interface channel.

When the SIOCH flag is "0" or "1", the serial interface hadware is conected to CH0 or CH1, respectively.

The external pin of a channel not selected can be used as general-purpose port.

Table 17-4 Serial Interface Channel Setting

SIOCH	Channel to be Selected
0	CH0
1	CH1

17.1.2 SB

This flag is intended to specify the serial interface protocol.

When the SB flag is "0" or "1", the serial I/O mode or the 2-wire bus mode is specified, respectively.

Because CH1 does not support the 2-wire bus mode, be sure to set the SB flag to "0" when using CH1.

Table 17-5 Serial Interface Protocol Specification

SB	Protocol
0	Serial I/O mode
1	2-wire bus mode

17.1.3 SIOMS

SIOMS is a flag to specify the serial interface clock direction.

When "0", the external clock is selected and when "1", the internal clock is selected. When the internal clock is selected, the frequency is set by the shift clock frequency register (RF: 39H).

When SIOMS is "0" or "1" in the 2-wire bus mode, the slave or master operation is specified, respectively.

Table 17-6 SIOMS Flag Function List

SIOMS	Function	
0	2-wire bus mode:	Slave operation
	Serial I/O mode:	External clock operation
1	2-wire bus mode:	Master operation
	Serial I/O mode:	Internal clock operation

17.1.4 SIOTX

When SIOTX is "0" or "1" in the 2-wire bus mode, the receive or transmit mode is specified, respectively.

When the SIOTX flag is set to "0" or "1" in the CH0 serial I/O mode, the SI mode (with SDA pin set to the input mode) or the SO mode (with SDA pin set to the output mode) is specified, respectively.

When the CH1 serial I/O mode is specified, the SIOTX flag is used to specify whether the SO pin should be used as a serial interface. When the SIOTX flag is "1" or "0", the SO pin is used as SO pin or general-purpose port, respectively.

Table 17-7 SIOTX Function List

SIOTX	Function	
0	2-wire bus mode :	RX (receive) mode
	CH0 serial I/O mode:	SI mode
	CH1 serial I/O mode:	$P0A_3$ to be used as general-purpose port
1	2-wire bus mode :	TX (transmit) mode
	CH0 serial I/O mode:	SO mode
	CH1 serial I/O mode:	$P0A_3$ to be used as SO pin

17.2 CLOCK COUNTER

The clock counter is a wraparound counter to count clocks in the shift clock pin of the selected serial interface ($P0A_1$/SCL pin in the case of CH0 or $P0A_2/\overline{SCK}$ pin in the case of CH1). It counts shift clocks repeatedly with count numbers 1 to 9. That is, the count starts with 0 and is incremented by 1 each time the clock start is detected. When the count reaches 9, it returns to 1 and repeats the same sequence.

The clock counter is reset to "0" in the following cases.

(1) In the 2-wire bus mode

(a) Upon power-on reset
(b) When STOP instruction is executed and the system clock stops
(c) When start condition is detected
(d) When the serial interface operation mode is changed from the 2-wire bus mode to the serial I/O mode

(2) In the serial I/O mode

(a) Upon power-on reset
(b) When STOP instruction is executed and the system clock stops
(c) When data is writen to the wait register
(d) When the serial interface operation mode is changed from the serial I/O mode to the 2-wire bus mode

Whether the clock counter has reached "8" or "9" can be checked by software using the status register. A request for clock stop in the transmit mode or clock stop in the receive mode in the 2-wire bus mode is generated by operating the wait register.

17.3 STATUS REGISTER

The status register is a 4-bit read only register. It holds the 2-wire bus mode start and stop statuses and the current clock counter contents.

Fig. 17-1 Status Register Configurations

Bit position	b_3	b_2	b_1	b_0
Flag name	SIOSF8	SIOSF9	SBSTT	SBBSY

17.3.1 SBBSY (Serial Bus Busy) Flag

The SBBSY flag is located at b_0 (LSB) of the status register (RF: 28H). It is a busy signal detection flag in the 2-wire bus mode.

The SBBSY flag is only valid when the 2-wire bus mode is selected by the SB flag of the serial mode register. When the start condition is detected, the SBBSY flag is set to "1" and when the stop condition is detected, it is reset to "0".

When the serial I/O mode is selected by the serial mode register, SBBSY flag is reset to "0" and remains "0" until the 2-wire bus mode is selected.

That is, the SBBSY flag remains unchanged in the serial I/O mode.

If neither transmission non reception is carried out in the 2-wire bus mode, whether another device is engaged in communication can be checked by testing the SBBSY flag.

17.3.2 SBSTT (Serial Bus Start Test) Flag

The SBSTT flag is located at b_1 of the status register. It is a start condition flag in the 2-wire bus mode.

The SBSTT flag is only valid when the 2-wire bus mode is selected by the SB flag of the serial mode register. When the start condition is detected, the SBSTT flag is set to "1" and when the clock counter reads "9", it is reset to "0".

The SBSTT flag is only is valid when the 2-wire bus mode is selected by the SB flag of the serial mode register. When the start condition is detected, the SBSTT Flag is set to "1" and when the clock counter reads "9", it is reset to "0"

17.3.3 SIOSF9 (Serial I/O Shift 9 Clock) Flag

The SIOSF9 flag is located at b_2 of the status register. When the clock counter reaches "9", it is set to "1". When the clock counter reads "0" or "1", this flag is reset to "0".

In the master mode in the 2-wire bus mode, the contents of the flag which hold whether the slave has returned acknowledge must be read before the SIOSF9 flag becomes "1" again after it has become '1".

The SIOSF9 flag is not affected by the contents of the serial mode register. In other words, the SIOSF9 flag is set when the clock counter becomes "9" in the serial I/O mode.

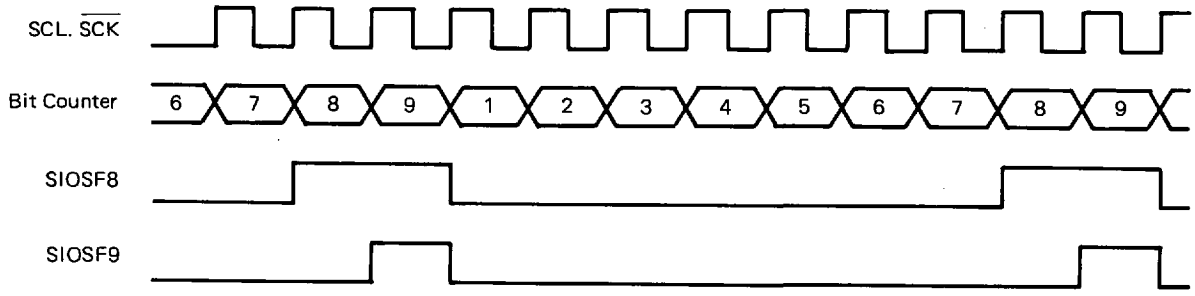
17.3.4 SIOSF8 (Serial I/O Shift 8 Clock) Flag

The SIOSF8 flag is located at b_3 of the status register. When the clock counter reaches "8", it is set to "1". When the clock counter reads "0" or "1", the SIOSF0 flag is rest to "0".

Read the presettable shift register when the SIOSF8 flag is "1".

The SIOSF8 flag is not affected by the contents of the serial mode register.

Fig. 17-2 SIOSF8 and SIOSF9 Operations



17.4 WAIT REGISTER

The μPD17051 can set a mode in which the serial interface hardware is not operated if the shift clock is input. This mode is called "wait mode" and is set using the wait register.

The wait register consists of four bits, SIOWRQ0 and SIOWRQ1 flags which specify the timing when communication should be stopped (waited) with a serial interface, SIONWT flag which indicates whether communication is currently waited for and SBACK flag which indicates whether acknowledge should be returned or has been returned in the 2-wire bus mode.

The wait register is located in the register file and is operated by the "PEEK" and "POKE" instructions via the window register.

Upon power-on reset or when the system clock stops as a result of execution of the "STOP" instruction, all flags are reset to "0".

Fig. 17-3 Wait Register Configuration

Bit position	b_3	b_2	b_1	b_0
Flag name	SBACK	SIONWT	SIOWRQ1	SIOWRQ0

17.4.1 SIOWRQ1 and SIOWRQ0 (Serial I/O Wait Request) Flag

These are flags to reserve (specify) the timing of waiting for the serial interface hardware. In the μPD17051, the concept of wait has been extended from the 2-wire bus mode slave to the 2-bus mode transmit side and serial I/O mode internal clock operations.

While in the wait mode, shift clock to the clock counter and the presettable shift register is disabled. In other words, if the shift clock level changes, the clock counter is not updated and the presettable shift register contents are not shifted.

Table 17-8 Wait Timing List

SIOWRQ1	SIOWRQ0	Wait Mode	2-Wire Bus Mode	Serial I/O Mode
0	0	No wait	No wait	No wait
0	1	Data wait	Wait at the falling edge of the shift clock when clock counter reads "8".	Wait at high level for shift clock when clock counter reads "8".
1	0	Acknowledge wait	Wait at the falling edge of shift clock when clock counter reads "9".	Wait at high level for shift clock when clock counter reads "9".
1	1	Address wait	Wait at the falling edge of shift clock when clock counter reads "8" after start condition detection.	Setting prohibited

(1) Wait in 2-wire bus mode slave operation

The SCL pin is changed to the output mode at the timing soecified by SIOWRQ1 and SIOWRQ0 and outputs a low level.

If no wait (SIOWRQ1 = SIOWRQ0 = 0) is specified, this operation is not carried out.

Wait is released by writing "1" to the SIONWT flag of the wait register.

For example, the wait status set by specifying the data wait mode (SIOWRQ1 = 0, SIOWRQ0 = 1: Wait at the falling edge of the shift clock with the shift clock clock counter reading "8") is released by writing "1" to the SIONWT flag. When the clock counter becomes "8" again, wait mode is reset at the falling edge of the shift clock.

If communication has not started in slave operation, the address wait mode (SIOWRQ1 = SIOWRQ0 = 1) is normally specified. In the address wait mode, wait is carried out at the falling edge of the shift clock when the clock counter becomes "8" for the first time after detection of the start condition. That is, this mode is for wait before the (9th) acknowledge clock starts for the transmitted salve address. In the wait mode, the pre-settable shift register (PSR) contents are read to check whether the address has been allocated for the local station.

Whether in wait or not can be checked by testing the SIONWT flag.

(2) Wait in 2-wire bus mode master operation

Wait in 2-wire bus mode master operation means the suspension of transmission. In this mode the shift clock is fixed to the low level at the timing specified for the SIOWRQ1 and SIOWRQ0 flags.

For example, in the wait status set by specifying the acknowledge wait mode (SIOWRQ1 = 1, SIOWRQ0 = 0: Wait at the falling edge of the shift clock with the shift clock counter reading "9") whether the receiver has returned acknowledge can be tested by testing the flag. Also in this status, the next data to be transmitted can be set to the presettable shift register.

As is the case with slave operation, the wait status is canceled by writing "1" to the SIONWT flag. If no wait is specified, wait operation is not carried out.

(3) Wait in serial I/O mode internal clock operation

In this case, wait is almost the same as wait in the 2-wire bus mode master operation. That is, this wait also means the suspension of transmission. The only difference is that wait is set with the shift clock set to the low level in the case of 2-wire bus mode master operation and wait is set with the shift clock set to the high level in the case of the serial I/O mode.

If the serial I/O mode has been specified, the clock counter is reset to "0" by writing data to the wait register. Thus, if, for example, the data wait mode is first specified and then changed to the acknowledge wait mode, the clock counter is reset to "0" and starts counting. When the clock counter reaches "8" and stops with the shift clock set to the high level.

Namely, in the serial I/O mode internal clock operation mode, the clock counter is first reset by writing data to the wait register and transmission starts.

If no wait is specified, the shift clock is continuously output.

(4) Wait in serial I/O mode external clock operation

In the case of serial I/O mode external clock operation, clock counter update and presettable shift register shift operations are disable at the timings specified by SIOWRQ1 and SIOWRQ0. For example, if the data wait mode has been specified, wait is applied at the falling edge of the clock with the clock counter set to "8" and the clock counter will not be updated by the shift clock to be input afterward. Also, data is never shifted in the presettable shift register.

To enable data input after wait, write "1" on the SIONWT flag as in all other cases. In other words, when data is written to this wait register, the clock counter is reset to "0" and wait is released.

17.4.2 SIONWT (Serial I/O No Wait) Flag

Wait can be released or wait can be forcibly applied by writing data to the SIONWT flag.

(1) When "0" is written to SIONWT

In this case, forced wait is applied. That is, clock supply to the clock counter and the presettable shift register is disabled.

If the SIOMS flag of the serial interface mode register has been set to "1", shift clock operation is also disabled at the same time.

(2) When "1" is written to SIONWT

In this case, wait is released. That is, clocks are supplied to the clock counter and the presettable shift register. If the SIOMS flag of the serial interface mode register has been set to "1", shift clock operation is continuously carried out in the status just before wait setting.

17.4.3 SBACK (Serial Bus Acknowledge)

SBACK depends on the serial interface operation mode.

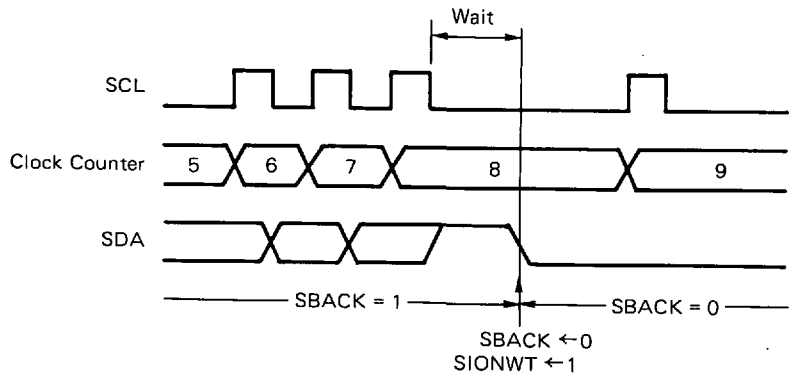
SBACK operation is described below.

(1) 2-wire bus mode receive operation (SIOTX = 0)

In this case, data set to the SBACK flag is output automatically to the SDA pin at the timing of acknowledge output. Also in this case, the SBACK flag contents are only changed by the "POKE" instruction executed for the wait register. Thus, to return acknowledge consecutively, writing "0" to the SBACK flag will cause "0" to be automatically transmitted as acknowledge. Consequently, it is not necessary to manipulate the SBACK flag each time one byte is received.

Data write to the SBACK flag must be carried out before the 9th bit of the next data is set after the 9th bit of the previous data is set. In the normal operation, data is written to the SBACK flag in the wait status to be set at the falling edge of the 8th or 9th bit.

Fig. 17-4 Timing Chart when SBACK is Rewritten in Wait



(2) 2-wire bus mode transmit operation (SIOTX = 1)

In this case the acknowledge contents from the receive side are set to the SBACK flag. That is, the receive side acknowledge status can be checked by reading the SBACK flag contents.

SBACK flag judge must be done before the 9th bit of the next data is set after the 9th bit of 1-byte data is set. In the normal operation, wait is applied at the falling edge of the 9th bit and the SBACK flag is read in the applied status.

Data can be written to the SBACK flag by the POKE instruction even during transmission.

(3) Serial I/O mode

In this case, the SBACK flag contents are not affected by the shift clock. That is, the SBACK is released completely from the serial interface. Thus, in this mode, the SBACK can also be used as a 1-bit flag for data storage.

17.5 PRESETTABLE SHIFT REGISTER (PSR)

The PSR is a presettable shift register. It generates the contents of its most significant bit to the serial data output pin (POA_0/SDA pin on CH0 and POA_3/SO pin on CH1) in synchronization with the falling edge of the clock signal on the shift clock pin (POA_1/SCL pin on CH0 and $POA_2/\overline{SCK}$ pin on CH1), and reads the data on the serial data input pin (POA_0/SDA pin on CH0 and POB_0/SI pin on CH1) into its least significant bit in synchronization with the rising edge of the clock.

In the wait status, shift clocks are not supplied to the PSR. That is, if an internal or external clock is supplied to the shift clock pin in the wait status, the PSR does not shift data.

PSR operations not in the wait status differ depending on the 2-wire bus mode or the serial I/O mode.

Data write and read to the PSR is executed by the PUT and GET instructions via the least significant 8 bits (data memory address: 0EH, 0FH) of the DBF in the data memory.

(1) PSR operation in the 2-wire bus mode

In the 2-wire bus mode shift clocks are supplied to the PSR only while the clock counter increments from "1" to "8". For example, when data of 9 bits (8-bit data + 1-bit acknowledge) is received in the 2-wire bus mode, only the first 8-bit data is read to the PSR. The 9th bit data vis read to the SBACK flag of the wait register. When PSR contents are transmitted in the 2-wire bus mode, PSR contents are output to the serial data pin while the clock counter increments from "1" to "8" and SBACK flag contents are output while the clock counter reads "9" (from the falling edge of the 8th bit clock to the rising edge of the 9th bit clock to be precise).

The above operation is carried out not only when the hardware of the serial interface built in the μPD17052 is used (including cases when an internal or external clock is used) but also when clocks are generated by software using as an output port the port (POA_0) which also serves as a shift clock pin.

In transmission, data output to the SDA pin is read again to the PSR in synchronization with the rising edge of the next shift clock. That is, when 8 shift clocks are generated in transmission operation, data of the pin engaged in transmission is stored in the PSR. If no data collision occurs during transmission, the same data as that before is stored in the PSR. Thus, whether data has been transmitted correctly can be checked by comparing data before transmission to that after transmission.

The above operation is PSR operation not in the wait status. In the wait status the PSR does not carry out shift operation.

(2) PSR operation in the serial I/O mode

Shift clock supply to the PSR in the serial I/O mode has nothing to do with the clock counter contents. Unless in the wait status, shift operation is carried out by the clock on the shift clock pin.

The PSR does not carry out shift operation in the wait status. Thus, when the PSR is not used as a serial interface and it is only used for data storage, the wait status must be set.

In the serial I/O mode, execute data write and read to the PSR when the shift clock is at the high level or in the wait status. If data write and read is carried out in any other cases, the PSR does not operate correctly. When using an internal clock, set the wait status at the rising edge of the 8th bit clock and operate the PSR in the wait status. In the case of external clock operation, operate the PSR during the period when the transmit side is guaranteed to maintain the shift clock at the hgih level.

17.6 SERIAL INTERFACE INTERRUPT SOURCE REGISTER (SIOIMD)

The interrupt source register (SIOIMD) is a 4-bit register which specifies the timing to apply an interrupt to the CPU in communication using a serial interface.

SIOIMD is located at address 38H of the register file.

The SIOIMD register configuration is shown in Fig. 17-5. The register is not allocated to the most significant 2 bits of SIOIMD. When the most significant 2 bits of SIOIMD are read, "0" is read from each bit.

Fig. 17-5 Serial Interface Interrupt Source Register Configuration (RF: 38H)

Bit position	b_3	b_2	b_1	b_0
Flag name	SIOIMD3 (0)	SIOIMD2 (0)	SIOIMD1	SIOIMD0

Table 17-9 List of Serial Interface Interrupt Source Register Functions

SIOCK1	SIOCK0	Function
0	0	Interrupt request generated at the rising edge of the 7th bit of shift clock
0	1	Interrupt request generated at the rising rdge pf the 8th bit of shift clock
1	0	Interrupt request generated at the rising edge of the 7th bit of shift clock just after start condition is detected
1	1	Interrupt request generated upon detection of stop condition

17.7 SHIFT CLOCK FREQUENCY REGISTER (SIOCK)

The shift clock frequency register is a 4-bit register to set the internal clock frequency of the serial interface.

The SIOCK register is located at address 39H of the register file.

The shift clock frequency register configuration is shown in Fig. 17-6. The register is not allocated to the most significant 2 bits of the shift clock frequency register. When the most significant 2 bits of SIOCK and read, "0" is read from each bit.

Fig. 17-6 Shift Clock Frequency Register Configuration (RF: 39H)

Bit position	b3	b2	b1	b0
Flag name	SIOCK3 (0)	SIOCK2 (0)	SIOCK1	SIOCK0

Table 17-10 List of Internal Clock Frequencies of Serial Interface

SIOCK1	SIOCK0	Internal Clock Frequency
0	0	100 kHz
0	1	200 kHz
1	0	500 kHz
1	1	1 MHz

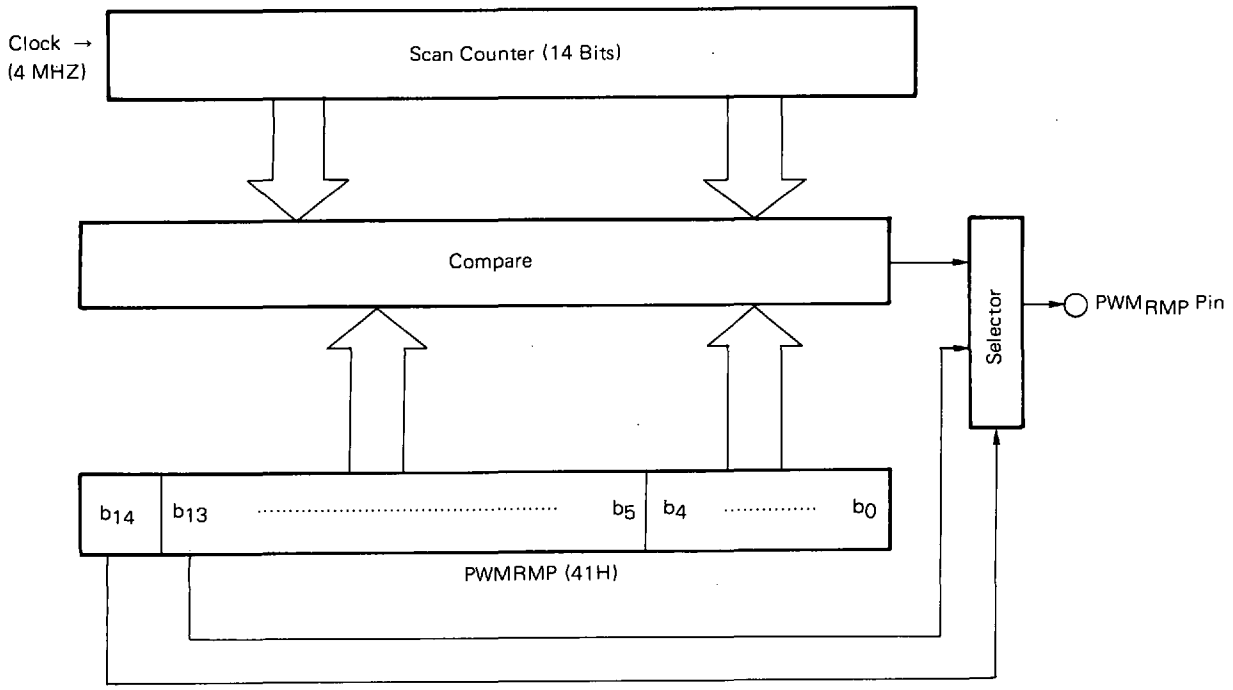
18. D/A CONVERTER

18.1 PWM_{RMP} PIN

This is a 14-bit D/A converter for voltage synthesizer. Its conversion mode is based on a combination of 9-bit PWM and 5-bit RMP (rate multiplier). The 14-bit D/A converter configuration is shown in Fig. 18-1. Since a pulse signal is output from the PWM_{RMP} pin, D/A conversion can be executed by externally adding a low pass filter.

This PWM_{RMP} pin can also be used as a 1-bit output port.

Fig. 18-1 14-Bit D/A Converter Configuration



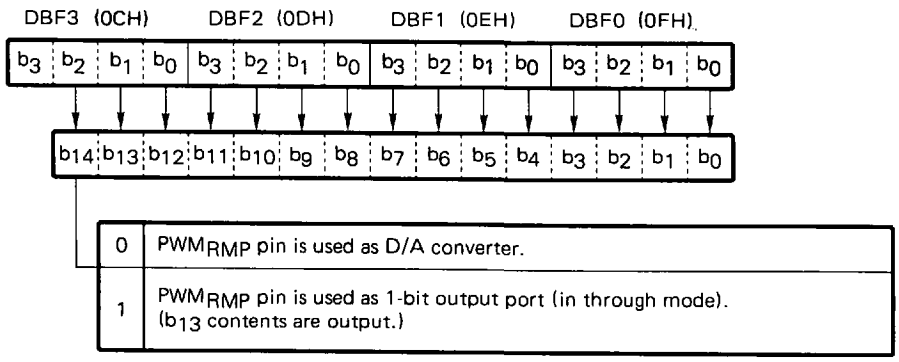
The D/A converter output value is set to the PWMRMP at peripheral address 41H. PWMRMP data is set and read via DBF. The PWMRMP configuration and the relations between DBF and PWMRMP are shown in Fig. 18-2. The most significant bit (b_3) of DBF3 has no corresponding PWMRMP. Thus, when setting data to the PWMRMP, this bit can be "1" or "0". When PWMRMP is read to DBF, this bit is set to "0".

The most significant bit (b_{14}) of PWMRMP is a bit to select the PWM_{RMP} pin for use as a D/A converter or a 1-bit output port.

Upon power-on, the PWMRMP pin is indeterminate. However, the PWM_{RMP} pin generates a low level upon power-up. That is, the PWMRMP and PWM_{RMP} pin outputs unmatch upon power-on. Once the "PUT PWMRMP, DBF" instruction is executed for the PWMRMP, the PWMRMP and PWM_{RMP} pin outputs will match.

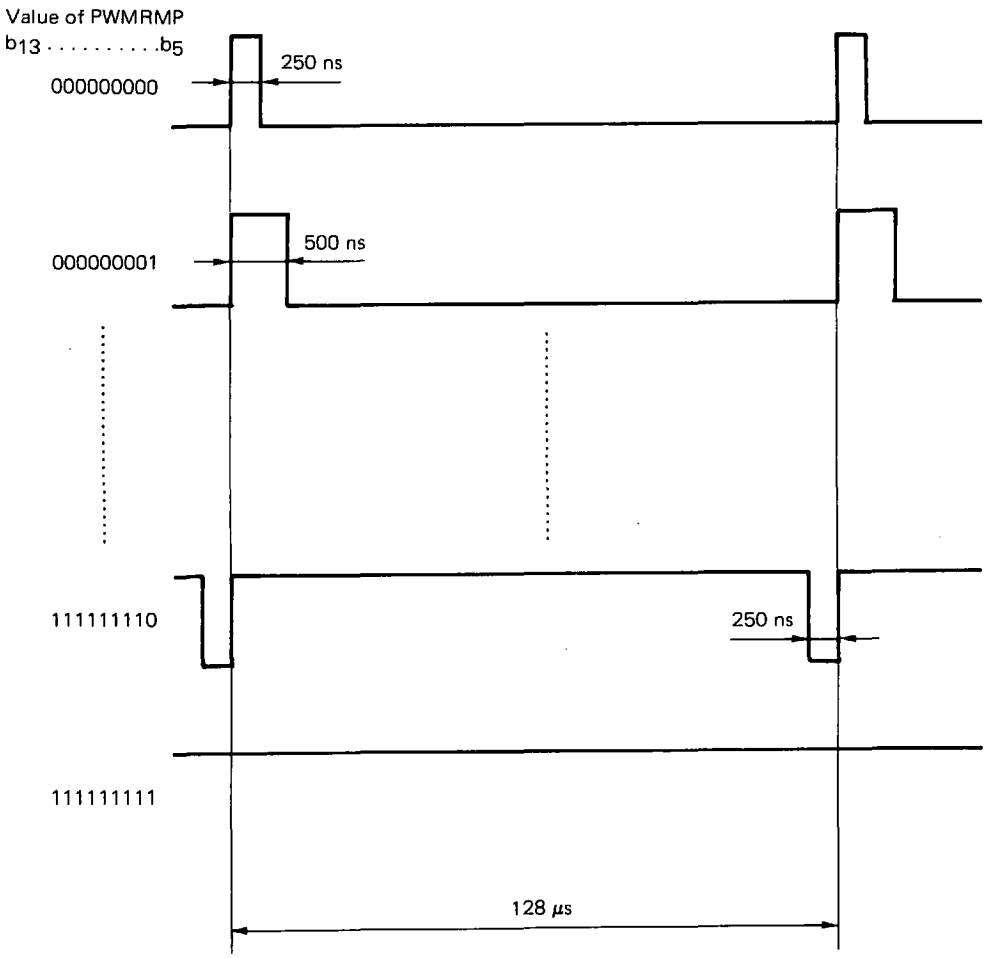
When the value of 14 bits (b_{13} to b_0) of PWMRMP is sequentially incremented from "0", the D/A converter output value will also change proportionally. If the value becomes larger than 3FDFH, that is in the range of 3FEOH to 3FFFH (when b_{13} to b_5 of PWMRMP are "111111111"), a high level is output from the PWM_{RMP} pin.

Fig. 18-2 PWMRMP Configuration and PWMRMP Relations with DBF



The PWM_{RMP} pin output signal has a wave form in which from zero to thirty-two 250 ns throttle pulses are added to 32 cycles of a 9-bit PWM signal using a 4 MHz basic clock. The 9-bit PWM signal is shown in Fig. 18-3.

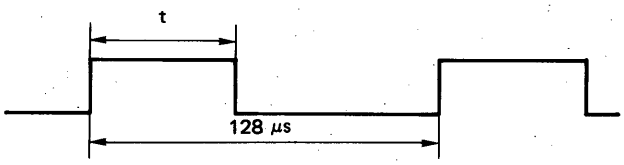
Fig. 18-3 PWM Signal Waveform



Each pulse width is determined by RMP for 32 cycles of the signal in Fig. 18-3. A signal with these pulses set as one cycle is output form the PWM_{RMP} pin. The high level width (t) of each pulse is determined by the PWMRMP value. A pair of pulses with two high-level widths differing from each other by 250 ns are output from the PWM_{RMP} pin. Pairs of two different pulses are determined by the least significant 5 bits (b_4 to b_0) of PWMRMP as shown in Table 18-1.

Table 18-1 PWM_{RMP} Pin Output Signals

b_4 to b_0 of PWMRMP	PWM Signal Cycle 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
00000	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
00001	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
00010	■	■	■	■	■	■	■		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■		■	■	■	■	■	■	■	
00011	■	■	■	■	■	■	■		■	■	■	■	■	■	■		■	■	■	■	■	■	■		■	■	■	■	■	■	■	
00100	■	■	■		■	■	■	■	■	■		■	■	■	■	■	■	■		■	■	■	■	■	■	■		■	■	■	
00101	■	■	■		■	■	■	■	■	■		■	■	■	■		■	■		■	■	■	■	■	■	■		■	■	■	
00110	■	■	■		■	■	■		■	■		■	■	■	■	■	■	■		■	■	■		■	■	■		■	■	■	
00111	■	■	■		■	■	■		■	■		■	■	■	■		■	■		■	■	■		■	■	■		■	■	■	
01000	■		■	■	■		■	■	■		■	■	■		■	■	■		■	■	■		■	■	■		■	■		■	
01001	■		■	■	■		■	■	■		■	■	■		■		■		■	■	■		■	■	■		■	■		■	
01010	■		■	■	■		■		■		■	■	■		■	■	■		■	■	■		■		■		■	■		■	
01011	■		■	■	■		■		■		■	■	■		■		■		■	■	■		■		■		■	■		■	
01100	■		■		■		■	■	■		■		■		■	■	■		■		■		■	■	■		■		■	■	
01101	■		■		■		■	■	■		■		■		■		■		■		■		■	■	■		■		■	■	
01110	■		■		■		■		■		■		■		■	■	■		■		■		■		■		■		■	■	
01111	■		■		■		■		■		■		■		■		■		■		■		■		■		■		■	■	
10000		■		■		■		■		■		■		■		■		■		■		■		■		■		■		■		
10001		■		■		■		■		■		■		■				■		■		■		■		■		■		■		
10010		■		■		■				■		■		■		■		■		■		■				■		■		■		
10011		■		■		■				■		■		■				■		■		■				■		■		■		
10100		■				■		■		■				■		■		■				■		■		■				■		
10101		■				■		■		■				■				■				■		■		■				■		
10110		■				■				■				■		■		■				■				■				■		
10111		■				■				■				■				■				■				■				■		
11000				■				■				■				■				■				■				■				
11001				■				■				■								■				■				■				
11010				■								■				■				■								■				
11011				■								■								■								■				
11100								■								■								■								
11101								■																■								
11101																■																
11111																																



■ : $t = \frac{n}{4}$ (μs)

□ : $t = \frac{n+1}{4}$ (μs)

($0 \leqq n \leqq 2^9 - 2$ where n = value of 9 bits b_5 to b_9 of PWMRMP in decimal notation)

18.2 PWM PIN

The μPD17052 is equipped with four 6-bit PWM output pins and can change the duty of the pulse signal with a frequency of 15.625 kHz in 64 stages. Thus, a D/A converter can be constructed by externally adding a low pass filter to the μPD17052. The PWM pin can also be used as a 1-bit output port.

When a D/A converter is constructed, its output value is set to the latch PWMR for output data setting. The latches for output data setting (PWMR0, PWMR1, PWMR2 and PWMR3) are assigned to the peripheral addresses 05H, 06H, 07H and 08H, respectively, and data can be read and written via DBF. The relations between PWMR addresses and the corresponding pins are shown in Table 18-2.

Table 18-2 PWMR Addresses and Corresponding Pins

Peripheral Device	Peripheral Address	Corresponding Pin
PWM0	05H	PWM_0
PWM1	06H	PWM_1
PWM2	07H	PWM_2
PWM3	08H	PWM_3

Each PWMR consists of 7 bits. The relations between PWMR configuration and DBR are shown in Fig. 18-4. The most significant bit of PWMR is a bit to specify the use as the PWM signal output pin or an output port. The least significant 6 bits are intended to set the PWM signal output value.

The PWMR pin output waveform is shown in Fig. 18-5.

Fig. 18-4 PWMR Configuration and PWMR Relations with DBF

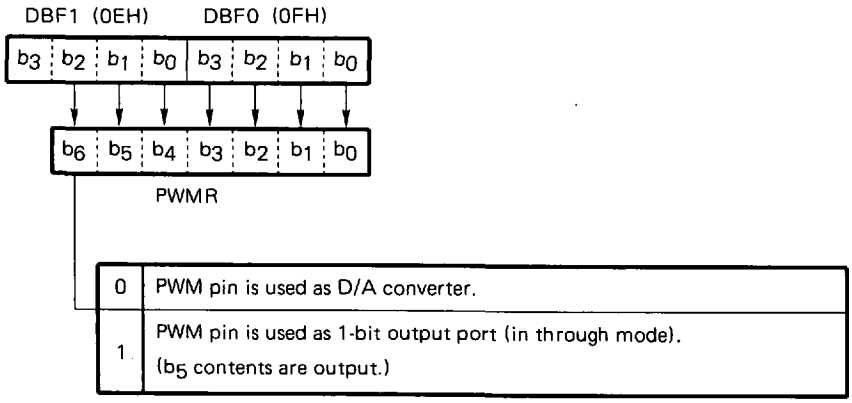
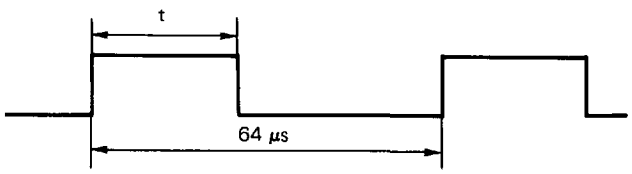


Fig. 18-5 PWM Pin Output Waveform



t = n + 0.75 (μs) (n: Value set to PWMR)

19. A/D CONVERTER

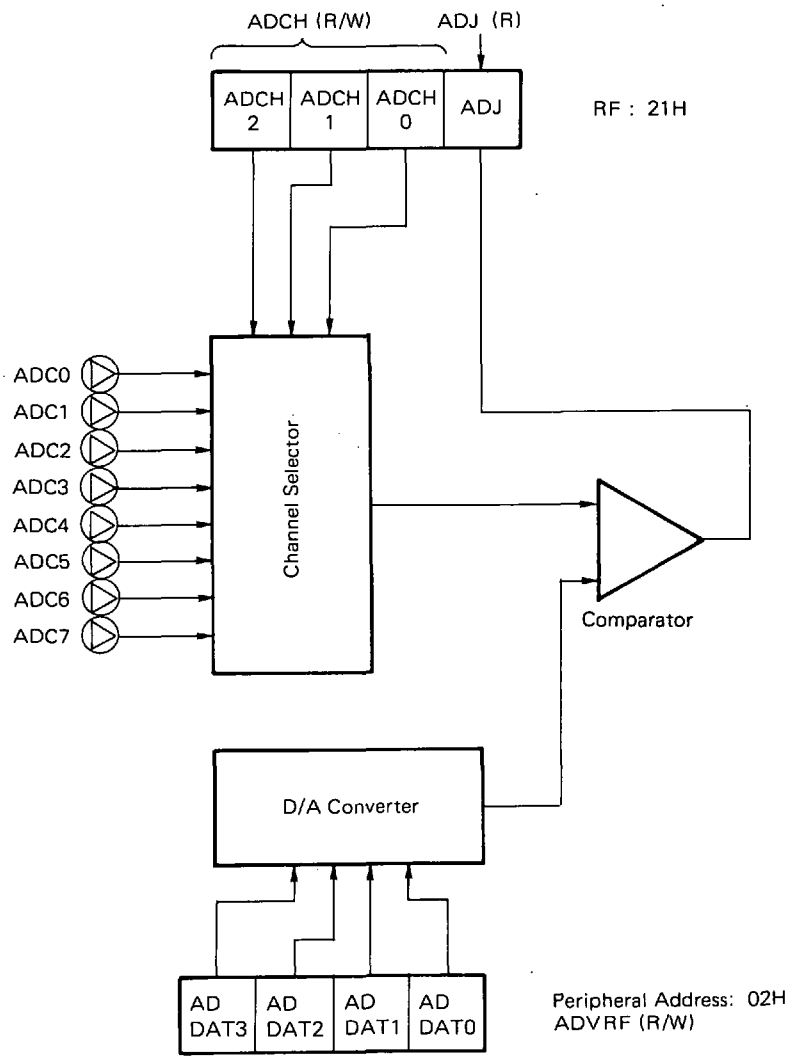
The μPD17052 has an on-chip 4-bit A/D converter based on successive approximation by program.

19.1 OPERATING PRINCIPLE

The A/D converter of the μPD17052 consists of a 4-bit resistance string type D/A converter and a comparator.

Data is set to the D/A converter with the 4-bit register ADCR located at peripheral address 02H. The compare result is judged from the ADCCMP flag in the register file.

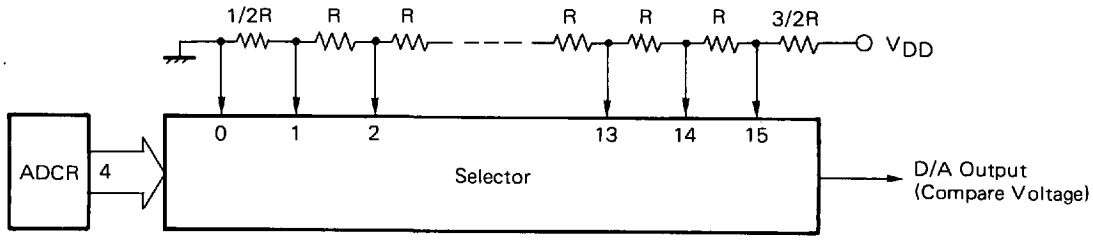
Fig. 19-1 A/D Converter Configuration



19.2 D/A CONVERTER CONFIGURATION

The D/A converter used for the A/D converter of the μPD17052 is a resistance string type D/A converter which selects the voltage at junctions of 16 resistors connected in series between V_{DD} and GND. The D/A converter configuration is shwon in Fig. 19-2.

Fig. 19-2 D/A Converter Configuration



With the D/A converter having this configuration,when 0000B is set to the ADCR, the GND level is output and when 0001B is set, a voltage of 1/32 x V_{DD} is output. Similarly, the value of n (decimal number) is set the V_{REF} compare voltage is given as

$$V_{REF} = V_{DD} \times \frac{2n-1}{32} \text{ (where } 15 \geqq n \geqq 1\text{)}$$

Table 19-1 D/A Converter Compare Voltages

Set Data (ADCR)		Compare Voltage (V_{REF})	
Hexadecimal	Binary	x V_{DD}	V_{DD} = 5 V
0	0000	0	0 [V]
1	0001	1/32	0.15625
2	0010	3/32	0.46875
3	0011	5/32	0.78125
4	0100	7/32	1.09375
5	0101	9/32	1.40625
6	0110	11/32	1.71875
7	0111	13/32	2.03125
8	1000	15/32	2.34375
9	1001	17/32	2.65625
A	1010	19/32	2.96875
B	1011	21/32	3.28125
C	1100	23/32	3.59375
D	1101	25/32	3.90625
E	1110	27/32	4.21875
F	1111	29/32	4.53125

19.3 COMPARE VOLTAGE SET REGISTER (ADCR)

The ADCR is a 4-bit register to set the A/D converter compare voltage. It is located at peripheral address 02H. Data read and write to the ADCR register is carried out by the PUT and GET instructions via a data buffer. Although the ADCR has 4 bits, data transfer with the DBF is executed in 8-bit units. That is, data transfer is executed via 8 bits of DBF1 (0EH) and DBF0 (0FH). Thus, when reading ADCR register contents, executing "GET DBF, ADCR" transfers the ADCR register contents to DBF0 and sets DBF1 to 0000B. Executing "PUT ADCR, DBF" sets DBF0 data to the ADCR register. In this case, DBF1 data can be any data.

The ADCR value is indeterminate upon power-on. Upon clock stop or CE reset, the previous data is held.

19.4 COMPARE JUDGE REGISTER (ADCCMP)

The ADCCMP is a register to store the results of ADC pin input voltage and compare voltage (V_{REF}) comparison by a comparator. It is located at bit 0 of register file address 21H. The ADCCMP is a 1-bit read only register and data cannot be written to this register. When reading the ADCCMP, data is read to the window register by the PEEK instruction. The ADC pin select data is read to the most significant 3 bits of the window register.

The ADCCMP is set as follows when data is read to the window register.

When input voltage $<$ compare voltage, ADCCMP = 0.

When input voltage $\geqq$ compare voltage, ADCCMP = 1.

19.5 ADC PIN SELECT REGISTER (ADCCHn)

The ADCCHn is a register to select the A/D converter input pin. It is located at the most significant 3 bits of the register file address 21H. The relations of ADCCHn and the actually selected pin are shown in Table 19-2.

Table 19-2 ADC Pin Selection

(MSB) (LSB)

#3	#2	#1	#0	(RF: 21H)
			ADCCMP	

ADCCH2	ADCCH1	ADCCH0	Selected Pin (Pin No.)
0	0	0	ADC (63)
0	0	1	$P0D_0/ADC_1$ (62)
0	1	0	$P0D_1/ADC_2$ (61)
0	1	1	$P0D_2/ADC_3$ (60)
1	0	0	$P0D_3/ADC_4$ (59)
1	0	1	$P1C_0/ADC_5$ (58)
1	1	0	$P1C_1/ADC_6$ (57)
1	1	1	$P1C_2/ADC_7$ (56)

When using one of the $P1C_0/ADC_5$ to $P1C_2/ADC_7$ pins as an A/D converter, specify P1C for the input port.

$P0D_0/ADC_1$ to $P0D_3/ADC_4$ are pins with pull-down resistor. When they are selected as A/D converters, the pull-down resistor of the selected pins are releases.

When P1C or P0D reads the pin selected for the A/D converter as a port, "0" is read.

19.6 A/D CONVERSION PROGRAM EXAMPLE

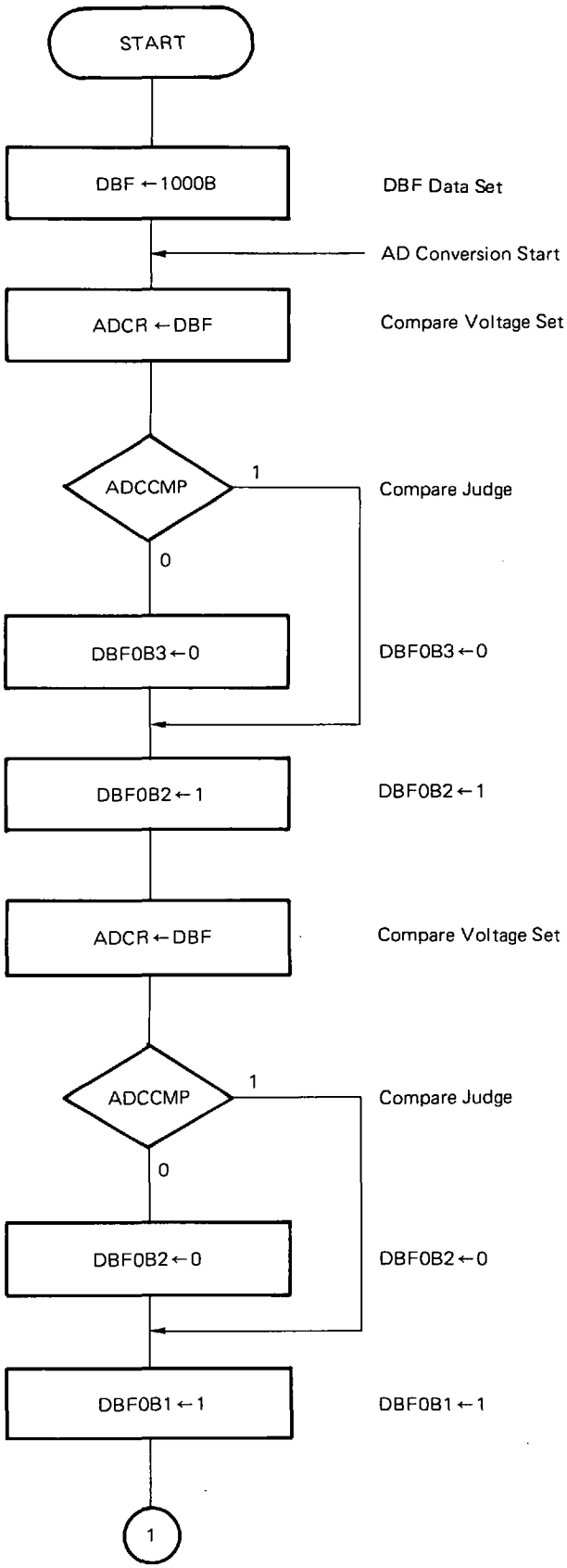
An example of A/D conversion program based on successive approximation is described below.
The conversion result is stored in DBF0.

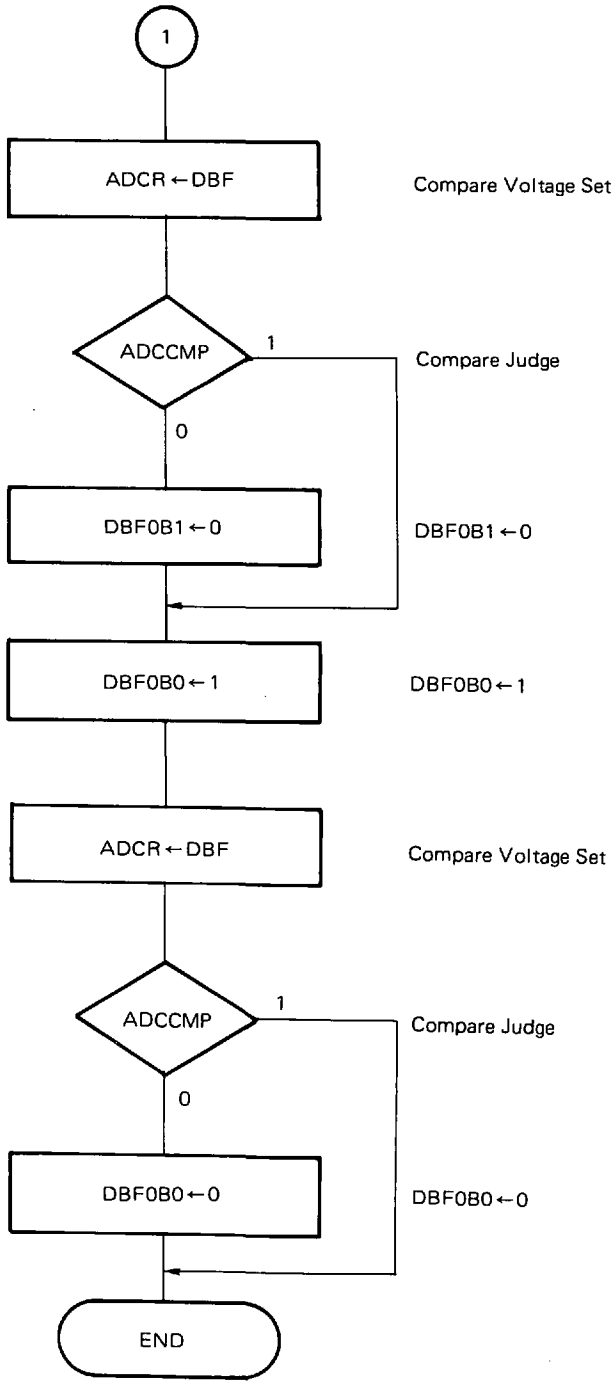
Program example

```
    DBF0B3      FLG 0.0FH.3
    DBF0B2      FLG 0.0FH.2
    DBF0B1      FLG 0.0FH.1
    DBF0B0      FLG 0.0FH.0
  START:
    BANK0
    INITFLG     DBF0B3, NOT DBF0B2, NOT DBF0B1, NOT DBF0B0
                                ; DBF data set
    PUT         ADCR, DBF       ; Compare voltage set
    SKT1        ADCCMP          ; Compare judge
    CLR1        DBF0B3          ; DBF0B3 ← 0
    SET1        DBF0B2          ; DBF0B2 ← 1
    PUT         ADCR, DBF       ; Compare voltage set
    SKT1        ADCCMP          ; Compare judge
    CLR1        DBF0B2          ; DBF0B2 ← 0
    SET1        DBF0B1          ; DBF0B1 ← 1
    PUT         ADCR, DBF       ; Compare voltage set
    SKT1        ADCCMP          ; Compare judge
    CLR1        DBF0B1          ; DBF0B1 ← 0
    SET1        DBF0B0          ; DBF0B0 ← 1
    PUT         ADCR, DBF       ; Compare voltage set
    SKT1        ADCCMP          ; Compare judge
    CLR1        DBF0B0          ; DBF0B0 ← 0
  END:
```

No. of conversion steps: : 17
Conversion time : 34 μs (with DMA mode OFF)
204 μs (with DMA mode ON)

Flowchart


START
DBF ← 1000B
DBF Data Set
AD Conversion Start
ADCR ← DBF
Compare Voltage Set
ADCCMP
1
Compare Judge
0
DBF0B3 ← 0
DBF0B3 ← 0
DBF0B2 ← 1
DBF0B2 ← 1
ADCR ← DBF
Compare Voltage Set
ADCCMP
1
Compare Judge
0
DBF0B2 ← 0
DBF0B2 ← 0
DBF0B1 ← 1
DBF0B1 ← 1
1


1
ADCR ← DBF
Compare Voltage Set
ADCCMP
1
Compare Judge
0
DBF0B1 ← 0
DBF0B1 ← 0
DBF0B0 ← 1
DBF0B0 ← 1
ADCR ← DBF
Compare Voltage Set
ADCCMP
1
Compare Judge
0
DBF0B0 ← 0
DBF0B0 ← 0
END

20. IDC (IMAGE DISPLAY CONTROLLER)

The IDC is a function to display channel numbers, sound volume and timer time on the TV screen. Display patterns are user programmable and display patterns are defined in the CROM area.

The actually displayed patterns are stored in the VRAM. The VRAM is assigned to BANK2 and BANK3 of the data memory.

20.1 SPECIFICATIONS OUTLINE AND RESTRICTIONS

(1) No. of characters to be displayed on one screen: 97 max.

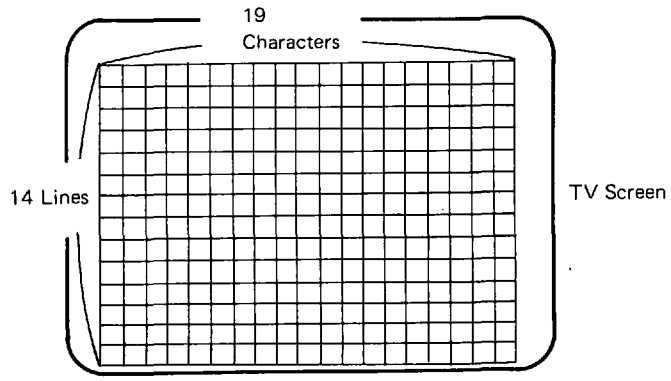
When a control code is used once on one line, the maximum number of display characters varies depending on the number of times control data is used.

Character Size	Max. No. of Display Characters on One Line	No. of Times Control Data is Used (on One Line)
x 1 (min.)	19	Up to 3
x 2	9	Up to 6
x 3	5	Up to 5
x 4	4	Up to 4

"Up to 3 times for control data usage" means that color specification can be made 3 times for one line.

(2) Display position variable range: 19 characters horizontally and 14 lines vertically

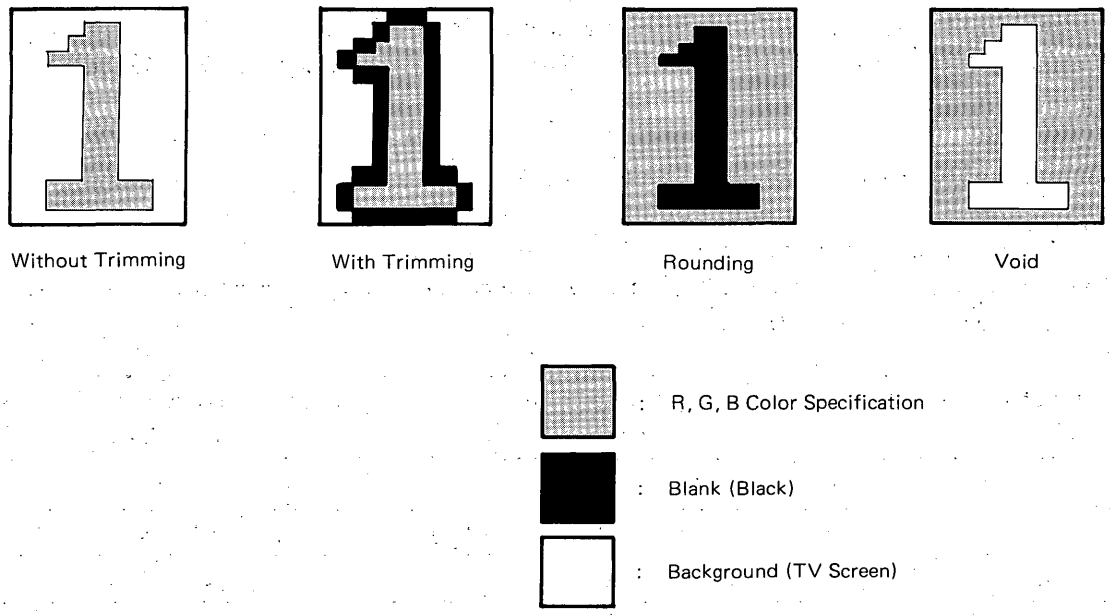
The following range is available for display on the TV screen.



(3) Color specifiable in character units (8 colors including white and black)

R, G and B independently (in compliance with control codes*1)

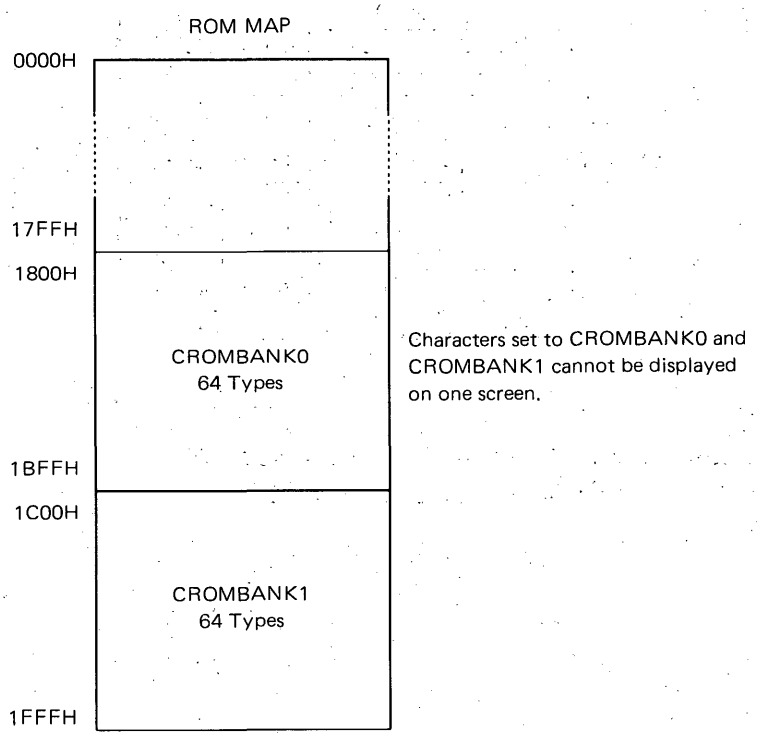
(4) Rounding, trimming and void setting enable in character units



(5) Types of font: 128 (user programmable)

64 types of font can be displayed simultaneously on one screen.

Character pattern data is located in the program memory (CROM) and 128 types of character pattern data are set in 64 types x 2 addresses (only 64 types can be displayed simultaneously on one screen [in the same CROM-BANK]).



(6) 4 vertical/horizontal sizes specifiable

Vertical size setting on each line and horizontal size setting in character units (using control codes*1)

(7) 10 x 15-dot character configuration

No gap between two characters*2

(8) Character pattern data to be located in program memory space

If there is few character pattern data, the CROM area can also be used as a program area.

(9) Character data to be located in the data memory space

Data transfer and data read/write can be executed as is the case with normal data memory data.

*1: Up to 3 control data can be specified on one line.

*2: To provide no gap between characters, kanji and graphics can be displayed by combining two or more characters.

20.2 DMA

DMA (Direct Memory Access) is a mode in which memory contents are directly transferred to the peripheral devices without the use of a CPU.

The DMA mode is used to operate the IDC for the μPD17052.

Although the μPD17052 instruction cycle is 2 μs, the instruction cycle in the DMA mode apparently becomes 12 μs. This is not because the actual instruction cycle becomes 12 μs but because 5 instructions (10 μs) are used for IDC data transfer and the remaining 2 μs or one instruction is used for normal instruction execution. In the DMA mode a normal instruction is executed every 5 instruction cycles.

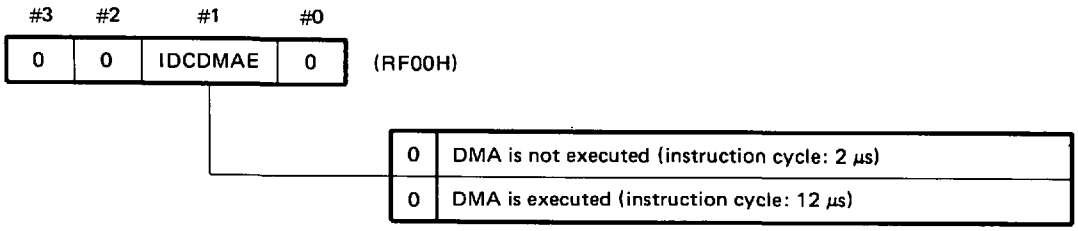
Thus, when the IDC is used, the instruction processing time apparently becomes 12 μs. If a 12 μs instruction cycle mixes with a 2 μs instruction for the sake of the program and the mixture poses a problem, set only the DMA mode when processing the routine and stop the IDC operation. In this case, 5 instructions for IDC data transfer are only used for clock operation and the μPD17052 itself carries out no operation.

In the DMA mode the ROM addresses of 5 of 6 instructions are pointed not by the program counter but by the CROM address pointer. The ROM addresses are pointed by the VRAM address pointer.

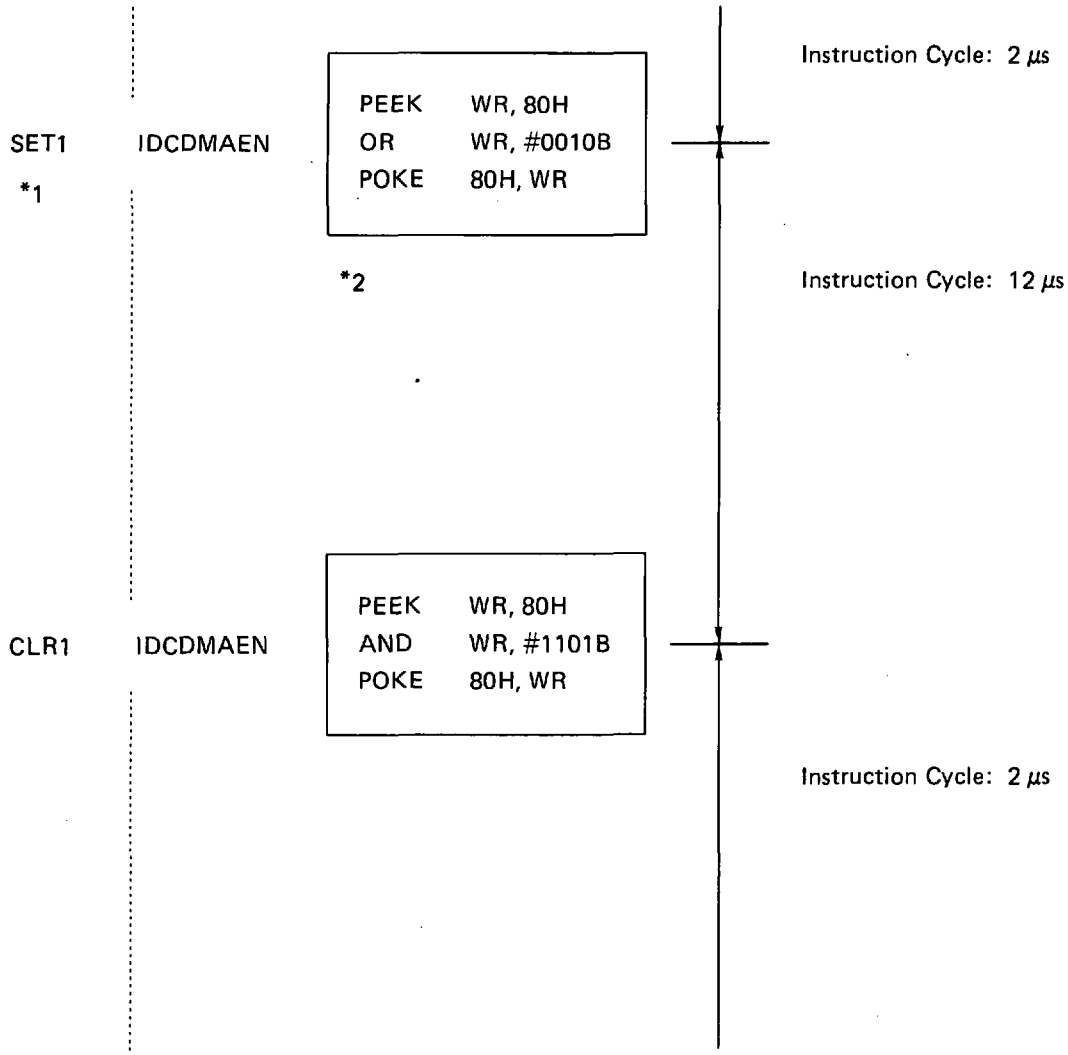
For DMA, manipulate the IDCDMAEN flag.

The IDCDMAEN flag is a read and write enable 1-bit flag located in the register file. When this flag is set, the DMA request is acknowledged with priority over any interrupt and the mode shifts to the DAM mode. When this flag is reset, the DAM request is not acknowledged. If this flag is reset while in the DMA mode, the DMA is released from the instruction cycle following the reset instruction.

Table 20-1 IDCDMAEN Flag



Program example

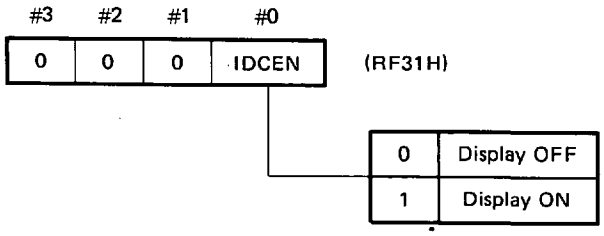


Remarks: "SET1" and "CLR1" are not μPD17052 instructions. They are intrinsic macro instructions avilable with the 17K series assembler and are used to set/reset a 1-bit flag. They are described in the source program as in *1. When assembled, they are developed as in *2.

20.3 IDC ENABLE FLAG

To start IDC operation (display ON), manipulate the IDCEN flag. The IDCEN (IDC enable) flag is assigned to the least significant bit (#0) at register file address 31H.

Table 20-2 IDCEN Flag



(1) Display ON precautions

(a) "1" must be set to IDCEN (display ON) while the vertical synchronous signal (Vsync) is at the high level (vertical retrace line: $\overline{\text{Vsync}}$ = low level) after the IDCDMAEN flag (RF, 00H, #1) is turned on.

(b) VRAM data must be set when IDCEN is "0" (display OFF).

Program example

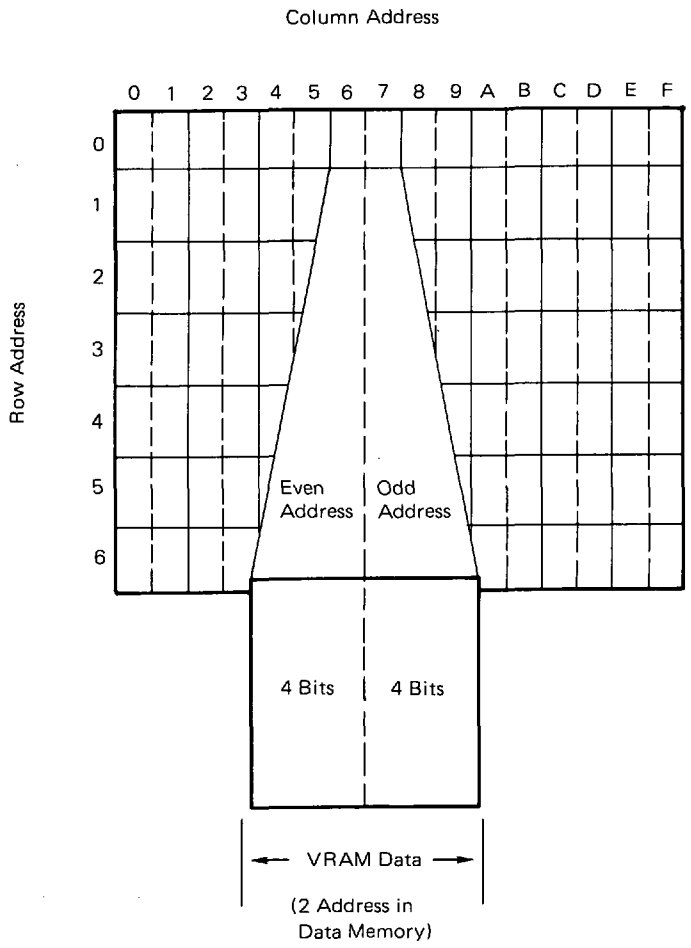
```
SET1    IDCDMAEN      ; DMA mode is set.
        :
        :
CLR1    IDCEN         ; When setting VRAM data from display-ON status, reset IDCEN (display OFF).
        :
        :
        Data set to VRAM  ; VRAM data is set.
        :
        :
LOOP
  SKF1  INTVSYN       ; Vsync = low level is checked and IDCEN is set.
  BR    LOOP
  SET1  IDCEN         ; Display ON
        :
```

20.4 VRAM

VRAM is a memory in which data is stored to select image patterns displayed on the TV screen by IDC. In the μPD17052, VRAM data is positioned in BANK2 and BANK3 of the data memory. One data consists of two adjacent addresses (8 bits), an even address and an odd address, in the data memory.

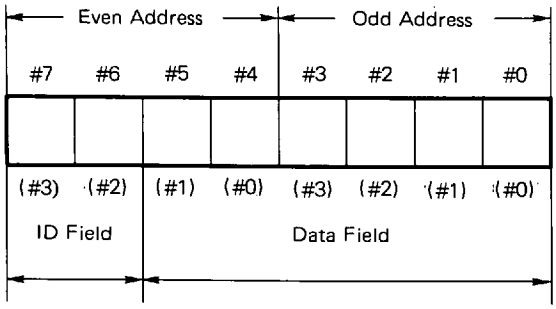
A 112-nibble data memory is installed in each of BANK2 and BANK3 (224 nibbles in total: 224 x 4 bits). That is, 112 data can be set as VRAM data.

Fig. 20-1 VRAM Configuration



VRAM data consists of 8 bits. Among 8 bits, the most significant 2 bits are called the ID field indicating the VRAM data type. The least significant 6 bits are called the data field which is display and control data to be actually displayed.

Fig. 20-2 VRAM Data Format



20.4.1 ID Field

The ID field indicates the data field data type.

The following three types of data are set in the data field.

(1) Character pattern select data
(2) Carriage return data
(3) Control data select data

Table 20-3 ID Field

ID field		Data Field Data Type
#7	#6	
0	0	Character pattern select data
0	1	Carriage return to BANK2 (return address) (From BANK2 to BANK2 and from BANK3 to BANK2)
1	0	Control data
1	1	Carriage return to BANK3 (return address) (From BANK3 to BANK3)

20.4.2 Character Pattern Select Data

A character pattern is image data to be displayed on the TV screen. It is located in the CROM area (1800H to 1FFFH) of the program memory. Character pattern select data makes up some CROM addresses (#9 to #4). That is, 6 bits of data field data indicate CROM addresses #9 to #4.

However, the CROM is divided into BANK0 and BANK1 and, depending on the value of CROMBNK (30H, #0) in the register file, the indicated CROM address differs if the 6-bit VRAM data is the same. That is, if the data field data is 0 (000000B), the CROM address will be 180xH (110000000xxxxB) or 1C0xH (111000000xxxxB). 180xH or 1C0xH is determined by specifying CROM BANK. Table 20-4 lists CROM addresses which the VRAM character pattern select data actually indicates.

Table 20-4 Relations between VRAM Data (Character Pattern Select Data) and CROM Addresses

VRAM Data (8 Bits)	CROM Address		VRAM Data (8 Bits)	CROM Address	
	BANK0	BANK1		BANK0	BANK1
00H	1800H-180EH	1C00H-1C0EH	20H	1A00H-1A0EH	1E00H-1E0EH
01H	1810H-181EH	1C10H-1C1EH	21H	1A10H-1A1EH	1E10H-1E1EH
02H	1820H-182EH	1C20H-1C2EH	22H	1A20H-1A2EH	1E20H-1E2EH
03H	1830H-183EH	1C30H-1C3EH	23H	1A30H-1A3EH	1E30H-1E3EH
04H	1840H-184EH	1C40H-1C4EH	24H	1A40H-1A4EH	1E40H-1E4EH
05H	1850H-185EH	1C50H-1C5EH	25H	1A50H-1A5EH	1E50H-1E5EH
06H	1860H-186EH	1C60H-1C6EH	26H	1A60H-1A6EH	1E60H-1E6EH
07H	1870H-187EH	1C70H-1C7EH	27H	1A70H-1A7EH	1E70H-1E7EH
08H	1880H-188EH	1C80H-1C8EH	28H	1A80H-1A8EH	1E80H-1E8EH
09H	1890H-189EH	1C90H-1C9EH	29H	1A90H-1A9EH	1E90H-1E9EH
0AH	18A0H-18AEH	1CA0H-1CAEH	2AH	1AA0H-1AAEH	1EA0H-1EAEH
0BH	18B0H-18BEH	1CB0H-1CBEH	2BH	1AB0H-1ABEH	1EB0H-1EBEH
0CH	18C0H-18CEH	1CC0H-1CCEH	2CH	1AC0H-1ACEH	1EC0H-1ECEH
0DH	18D0H-18DEH	1CD0H-1CDEH	2DH	1AD0H-1ADEH	1ED0H-1EDEH
0EH	18E0H-18EEH	1CE0H-1CEEH	2EH	1AE0H-1AEEH	1EE0H-1EEEH
0FH	18F0H-18FEH	1CF0H-1CFEH	2FH	1AF0H-1AFEH	1EF0H-1EFEH
10H	1900H-190EH	1D00H-1D0EH	30H	1B00H-1B0EH	1F00H-1F0EH
11H	1910H-191EH	1D10H-1D1EH	31H	1B10H-1B1EH	1F10H-1F1EH
12H	1920H-192EH	1D20H-1D2EH	32H	1B20H-1B2EH	1F20H-1F2EH
13H	1930H-193EH	1D30H-1D3EH	33H	1B30H-1B3EH	1F30H-1F3EH
14H	1940H-194EH	1D40H-1D4EH	34H	1B40H-1B4EH	1F40H-1F4EH
15H	1950H-195EH	1D50H-1D5EH	35H	1B50H-1B5EH	1F50H-1F5EH
16H	1960H-196EH	1D60H-1D6EH	36H	1B60H-1B6EH	1F60H-1F6EH
17H	1970H-197EH	1D70H-1D7EH	37H	1B70H-1B7EH	1F70H-1F7EH
18H	1980H-198EH	1D80H-1D8EH	38H	1B80H-1B8EH	1F80H-1F8EH
19H	1990H-199EH	1D90H-1D9EH	39H	1B90H-1B9EH	1F90H-1F9EH
1AH	19A0H-19AEH	1DA0H-1DAEH	3AH	1BA0H-1BAEH	1FA0H-1FAEH
1BH	19B0H-19BEH	1DB0H-1DBEH	3BH	1BB0H-1BBEH	1FB0H-1FBEH
1CH	19C0H-19CEH	1DC0H-1DCEH	3CH	1BC0H-1BCEH	1FC0H-1FCEH
1DH	19D0H-19DEH	1DD0H-1DDEH	3DH	1BD0H-1BDEH	1FD0H-1FDEH
1EH	19E0H-19EEH	1DE0H-1DEEH	3EH	1BE0H-1BEEH	1FE0H-1FEEH
1FH	19F0H-19FEH	1DF0H-1DFEH	3FH	1BF0H-1BFEH	1FF0H-1FFEH

Program example

VRAM Data

	0	1	2	3	4	5	6	7	8	9	A	B
0	8	0	0	0	0	1	4	0				
1												

CROM Data

```
1800H
  :          "C"
180FH                    ;  Control Data 1
1810H
  :          "H"
181FH                    ;  Control Data 2
  :
  :
1C00H
  :          "V"
1C0FH                    ;  Control Data 1'
1C10H
  :          "O"
1C1FH                    ;  Control Data 2'
  :
```

If CROM data and VRAM data have been specified as shown above, the screen display contents differ depending on the CROM bank.

The CROM bank is specified by CROMBNK (30H, #0) in the register file.

In the above example, the following results.

(1) CROMBNK = 0

"CH" is displayed on the screen. In this case, control data has "control data 1" contents.

(2) CROMBNK = 1

"VO" is displayed on the screen. In this case, control data has "control data 1" contents.

20.4.3 Carriage Return Data

Carriage return data is a data which specifies the address of VRAM data specifying the start character on the line displayed on the screen.

Carriage return data means the termination of one display line.

If carriage return data continues twice, they mean the termination of one screen.

There are two types of carriage return data, data for carriage return to BANK2 and data for carriage return to BANK3. Whether the data field carriage return is for BANK2 or BANK3 is determined by the ID field data. When the ID field is 01B or 11B, carriage return is for BANK2 or BANK3, respectively.

Carriage return data consists of 6 bits and the most significant 3 bits point to the VRAM row address and the least significant 3 bits point to the most significant 3 bits of the VRAM column address. The least significant bit of the VRAM column address is fixed to "0". Thus, if carriage return data is 010011B, the VRAM row address is 010B (2H) and the column address is 0110B (6H) or the return data to 26H.

Fig. 20-3 Carriage Return Data Format

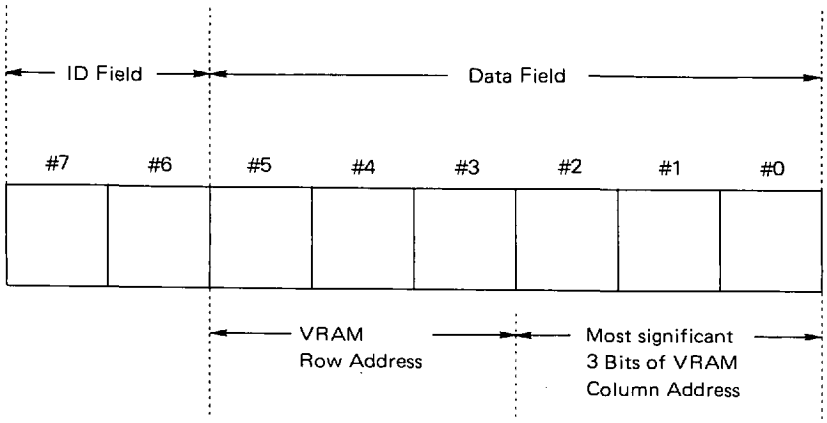


Fig. 20-4 List of Carriage Return Data (8 Bits Including ID Field)

BANK2

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	40		41		42		43		44		45		46		47	
1	48		49		4A		4B		4C		4D		4E		4F	
2	50		51		52		53		54		55		56		57	
3	58		59		5A		5B		5C		5D		5E		5F	
4	60		61		62		63		64		65		66		67	
5	68		69		6A		6B		6C		6D		6E		6F	
6	70		71		72		73		74		75		76		77	

BANK3

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	C0		C1		C2		C3		C4		C5		C6		C7	
1	C8		C9		CA		CB		CC		CD		CE		CF	
2	D0		D1		D2		D3		D4		D5		D6		D7	
3	D8		D9		DA		DB		DC		DD		DE		DF	
4	E0		E1		E2		E3		E4		E5		E6		E7	
5	E8		E9		EA		EB		EC		ED		EE		EF	
6	F0		F1		F2		F3		F4		F5		F6		F7	

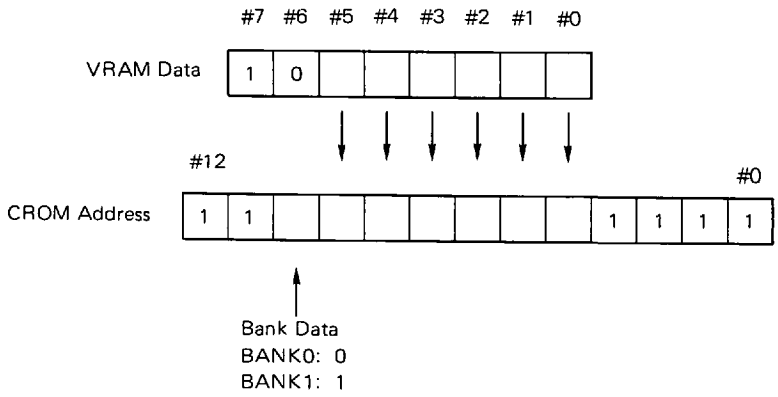
20.4.4 Control Data Select Data

Control data is intended to specify the character sizes, display positions and colors for character patterns on the screen. It is stored in the CROM (address xxxFH).

Control data select data is used to specify in the VRAM the control data in the CROM.

6 bits of the data field correspond to #9 to #4 of CROM address. As is the case with character pattern select data, CROM bank specification is necessary with CROMBNK (30H, #0) in the register file.

Fig. 20-5 Relations between Control Data and CROM Address



VRAM Data (8 Bits)	CROM Address	
	BANK0	BANK1
80H	180FH	1C0FH
81H	181FH	1C1FH
82H	182FH	1C2FH
83H	183FH	1C3FH
84H	184FH	1C4FH
85H	185FH	1C5FH
86H	186FH	1C6FH
87H	187FH	1C7FH
88H	188FH	1C8FH
89H	189FH	1C9FH
8AH	18AFH	1CAFH
8BH	18BFH	1CBFH
8CH	18CFH	1CCFH
8DH	18DFH	1CDFH
8EH	18EFH	1CEFH
8FH	18FFH	1CFFH
90H	190FH	1D0FH
91H	191FH	1D1FH
92H	192FH	1D2FH
93H	193FH	1D3FH
94H	194FH	1D4FH
95H	195FH	1D5FH
96H	196FH	1D6FH
97H	197FH	1D7FH
98H	198FH	1D8FH
99H	199FH	1D9FH
9AH	19AFH	1DAFH
9BH	19BFH	1DBFH
9CH	19CFH	1DCFH
9DH	19DFH	1DDFH
9EH	19EFH	1DEFH
9FH	19FFH	1DFFH

VRAM Data (8 Bits)	CROM Address	
	BANK0	BANK1
A0H	1A0FH	1E0FH
A1H	1A1FH	1E1FH
A2H	1A2FH	1E2FH
A3H	1A3FH	1E3FH
A4H	1A4FH	1E4FH
A5H	1A5FH	1E5FH
A6H	1A6FH	1E6FH
A7H	1A7FH	1E7FH
A8H	1A8FH	1E8FH
A9H	1A9FH	1E9FH
AAH	1AAFH	1EAFH
ABH	1ABFH	1EBFH
ACH	1ACFH	1ECFH
ADH	1ADFH	1EDFH
AEH	1AEFH	1EEFH
AFH	1AFFH	1EFFH
B0H	1B0FH	1F0FH
B1H	1B1FH	1F1FH
B2H	1B2FH	1F2FH
B3H	1B3FH	1F3FH
B4H	1B4FH	1F4FH
B5H	1B5FH	1F5FH
B6H	1B6FH	1F6FH
B7H	1B7FH	1F7FH
B8H	1B8FH	1F8FH
B9H	1B9FH	1F9FH
BAH	1BAFH	1FAFH
BBH	1BBFH	1FBFH
BCH	1BCFH	1FCFH
BDH	1BDFH	1FDFH
BEH	1BEFH	1FEFH
BFH	1BFFH	1FFFH

20.4.5 VRAM Data Setting Precautions

(1) Before setting VRAM data, be sure to set the IDCEN flag to "0".

(2) The VRAM data must start with 00H of BANK2.

(3) Do not set VRAM data to 7xH of BANK2 and BANK3.

(4) Be sure to set control data at the start of one screen. To prevent program errors, be sure to set control data at the start of each line. If control data is not set, the data set just before becomes valid.

(5) Data setting
   (a) Set character pattern select data sequentially at VRAM addresses starting with the smallest from the top left of the screen.
   (b) Control data can be used up to three times on one line.
   (c) Character pattern data after control data select data is modified by control data.
   Horizontal start position data and vertical start position data are only used for the characters just after control data select data. Characters following those characters are output in succession.
   (d) Be sure to set carriage return data at the end of one line.
   (e) Be sure to set two carriage return data at the end of data on one screen.

20.5 CROM (CHARACTER ROM)

IDC pattern data and control data are stored in the CROM. The CROM also serves as a program memory. It has a capacity of 2K steps (2048 x 16 bits). The area which is not used as CROM can be used as a normal program area.

The CROM area in the ROM ranges from address 1800H to address 1FFFH. It is divided into CROM BANK0 and CROM BANK1. Only when the CROM area is used as CROM, the concept of bank is necessary. When it is used as a program area, that concept is not necessary. CROM BANK0 is a 1K step of 1800H to 1BFFH and CROM BANK1 is a 1K step of 1C00H to 1FFFH.

CROM bank selection is done with the CROMBNK flag (30H, #0) of the register file.

Table 20-6 CROM Bank

CROMBNK Flag	CROM Bank	CROM Address
0	BANK0	1800H to 1BFFH
1	BANK1	1C00H to 1FFFH

Remarks: Change CROM banks when the IDCEN flag is "0".

Read and write are enabled for address 30H of the register file but "0" is always set to flags other than the CROMBANK flag (#0).

CROM data has a data length of 16 bits because it is located in the program memory area.

There are two types of CROM data.

(1) Character pattern data

(2) Control data

20.5.1 Character Pattern Data

This is character and graphic pattern data. One character consists of 10 horizontal dots and 15 vertical dots and the character pattern data consists of 16 bits x 15 steps. 10 horizontal data corresponds to CROM1 step. In the CROM, 15 steps of address xx0H to xxEH become one character pattern data.

With or without trimming, character pattern data has different configurations.

Fig. 20-6 shows character pattern data formats.

Trimming is selected using the most significant bit. Set the most significant bit to "0" for data without trimming and "1" for data with trimming.

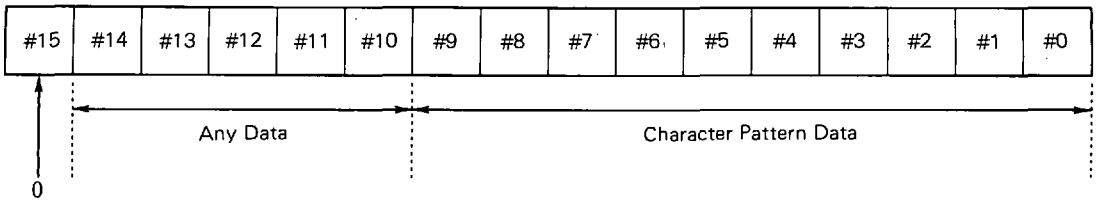
For data without trimming, set a character pattern dot image for the most significant 10 bits. #9 and #0 correspond to the left and right of the display screen, respectively. Set the bit corresponding to the ON bit to "1" and the bit corresponding to the OFF bit to "0".

For data with trimming, character pattern data is 5-bit data as shown in Fig. 20-6. In this case, 2 dots of the actual display pattern correspond to 1 bit of character pattern data. 10 bits of trimming data (to be set in 1-dot unit) are added to create character pattern with trimming.

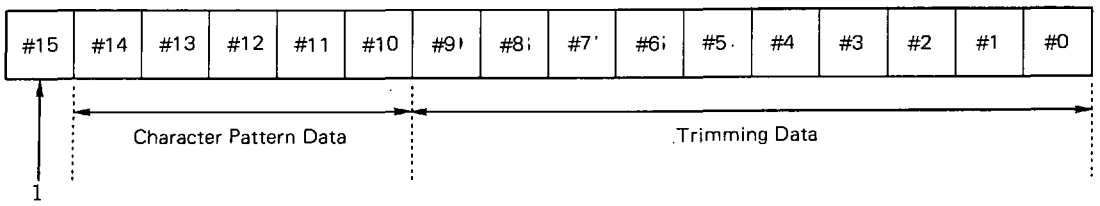
Character pattern can be easily defined by the DCP pseudo-instruction using a 17K series assembler. Using the DCP pseudo-instruction automatically creates data with or without trimming shown in Fig. 20-6.

Fig. 20-6 Character Pattern Data Formats

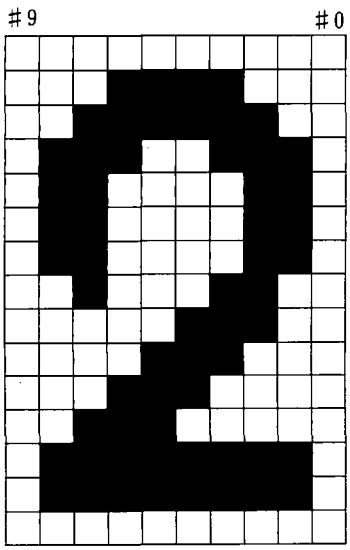
(a) Data without trimming



(b) Data with trimming



When displaying "2", set a character pattern as shown in Fig. 20-7. "0" and "1" in the pattern data correspond to □ and ■, respectively. Specify the character size, location and color with control data. Fig. 20-8 shows a character pattern example with trimming.

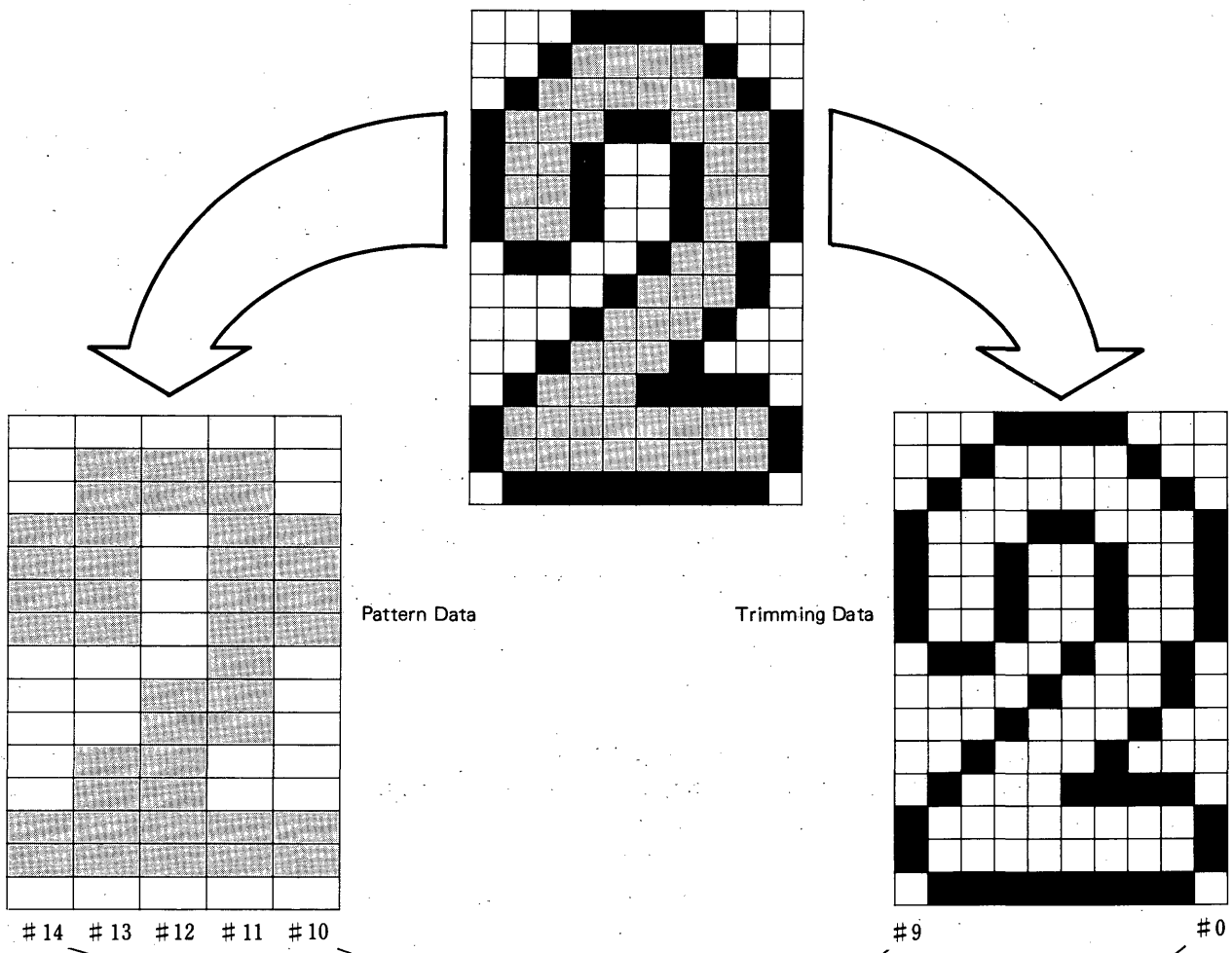


	#15						#9								#0
×××0H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
×××1H	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
×××2H	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0
×××3H	0	0	0	0	0	0	0	1	1	1	0	0	1	1	1
×××4H	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1
×××5H	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1
×××6H	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1
×××7H	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0
×××8H	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
×××9H	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
×××AH	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
×××BH	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0
×××CH	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0
×××DH	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0
×××EH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ROM Address (in CROM Area)

Any Data — Pattern Data

"0" (without Trimming)



	#15							#9							#0
× × × 0 H	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0
× × × 1 H	1	0	1	1	1	0	0	0	1	0	0	0	0	1	0	0
× × × 2 H	1	0	1	1	1	0	0	1	0	0	0	0	0	0	1	0
× × × 3 H	1	1	1	0	1	1	1	0	0	0	1	1	0	0	0	1
× × × 4 H	1	1	1	0	1	1	1	0	0	1	0	0	1	0	0	1
× × × 5 H	1	1	1	0	1	1	1	0	0	1	0	0	1	0	0	1
× × × 6 H	1	1	1	0	1	1	1	0	0	1	0	0	1	0	0	1
× × × 7 H	1	0	0	0	1	0	0	1	1	0	0	1	0	0	1	0
× × × 8 H	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	0
× × × 9 H	1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	0
× × × A H	1	0	1	1	0	0	0	0	1	0	0	0	1	0	0	0
× × × B H	1	0	1	1	0	0	0	1	0	0	0	1	1	1	1	0
× × × C H	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1
× × × D H	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1
× × × E H	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0

ROM Address (in CROM Area)

Pattern Data

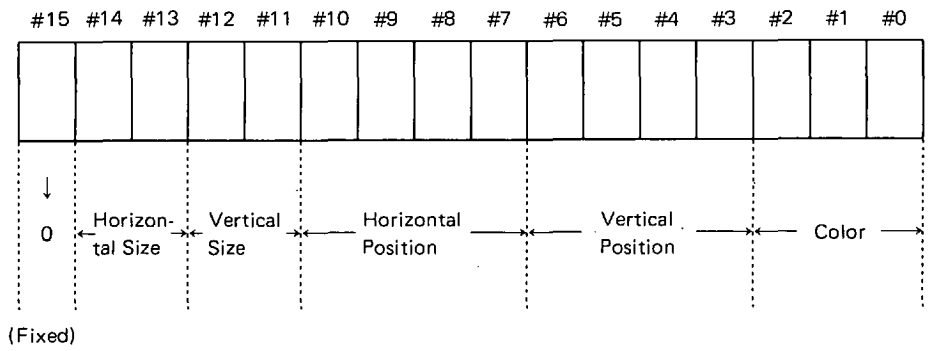
Trimming Data

"1" (with Trimming)

20.5.2 Control Data

Control data is intended to specify character pattern display positions, sizes and colors. It is stored at xxxFH in the CROM area. One data consists of 16 bits. Be sure to set the most significant bit to "0". Fig. 20-9 shows the control data format.

Fig. 20-9 Control Data Format



Although control data is attached to each character pattern data, it has nothing to do with character pattern data at the addresses before and after the control code. Any control data can be specified by the data to be set to the VRAM.

(1) Horizontal size data (2 bits, #14 and #13, of control data)

Horizontal size data determines the size of character image in the horizontal direction. 4 types of sizes can be specified for each character (up to 3 times on one line). Table 20-6 shows the setting contents.

Table 20-6 Horizontal Size Setting

Horizontal Size Data		Size	Horizontal Width of One Character	Max. No. of Display Characters on One Line
#14	#13			
0	0	x 1	2.5 μs	16
0	1	x 2	5.0 μs	8
1	0	x 3	7.5 μs	5
1	1	x 4	10.0 μs	4

(2) Vertical size (2 bits #12 and #11 of control data)

The vertical size data is intended to determine the size of a character image in the vertical direction. Four sizes can be set for each line as shown in Table 20-7.

Vertical size data specified at the start of a line is valid on the line. Other vertical size control data on the same line is ignored.

Table 20-7 Vertical Size Setting

Vertical Size Data		Size	Vertical Width of One Character (Interlace)	Max. No. of Display Characters in Vertical Direction
#12	#11			
0	0	x 1	15H	12
0	1	x 2	30H	6
1	0	x 3	45H	4
1	1	x 4	60H	3

(3) Horizontal position data (4 bits #10 to #7 of control data)

Horizontal position data is intended to determine one of the 16 digits in the horizontal direction to start display in Fig. 20-10. Although there are actual display positions for 19 characters in the horizontal direction, the horizontal display start position can only be specified within 16 characters from the left of the screen.

The start of the line is specified with an absolute digit (0 to 15 digits in Fig. 20-10). Data has 4 bits having #10 as MSB and #7 as LSB. Horizontal position data is 0H to FH with 0 digit corresponding to "0" and 15th digit corresponding to "FH".

When having a gap between characters on one line, specify the number of characters for the gap using horizontal position data. In other words, specify in hexadecimal notation the number of characters to be displayed after the currently displayed character on the same line.

For example, the horizontal position data of "A" and "C" in Fig. 20-10 are 8H and 1H, respectively. When the horizontal position data of "C" control data is "O", "C" is displayed on the 9th digit. If no control data is used after "A", "C" is displayed on the 9th digit.

Remarks: The number of characters is an example when horizontal size data is "00". If horizontal size is changed, the number of characters in the new size is counted. For example, if horizontal size data is doubled, one line will consists of 8 digits.

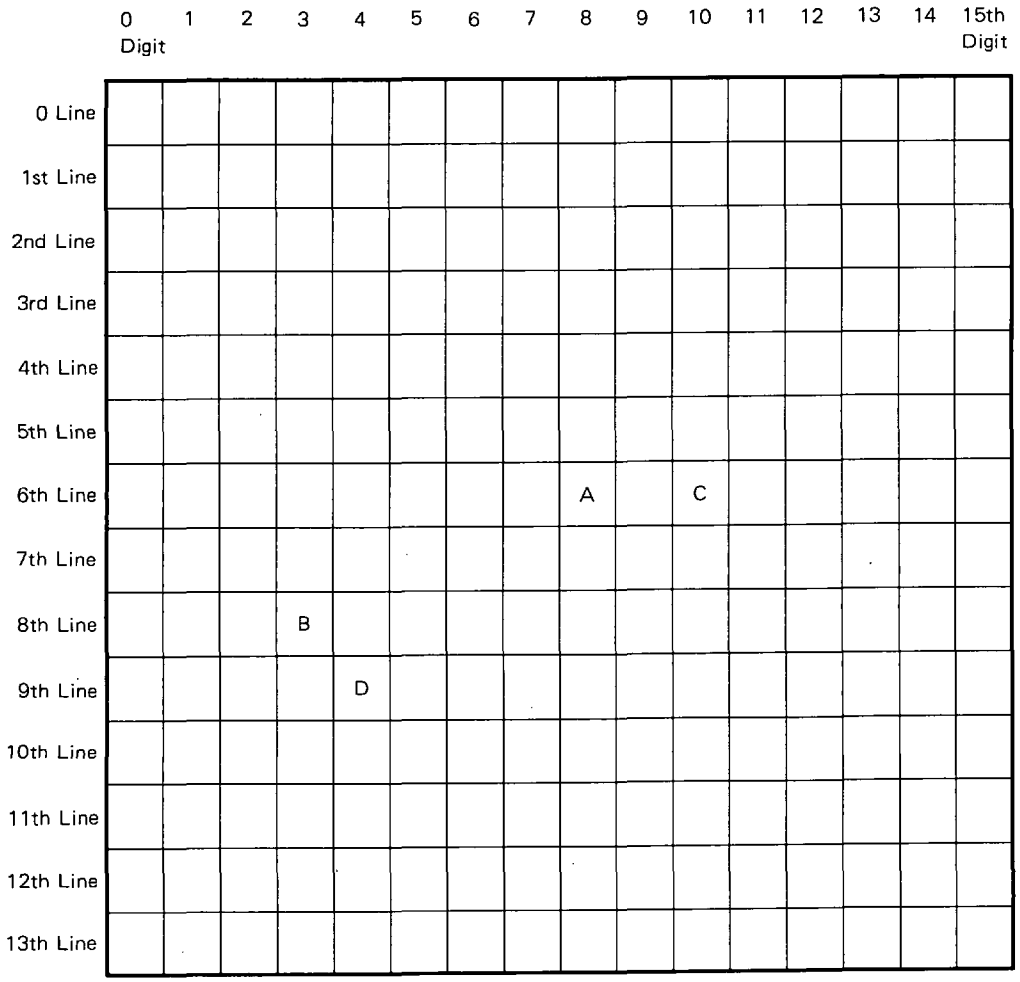
(4) Vertical position data (bit #6 to #3 of control data)

Vertical position data is intended to determine one of the 12 lines in the vertical direction to start display in Fig. 20-10. Data has 4 bits having #6 as MSB and #3 as LSB and is specified with OH to DH (do not set EH and FH data). The 0 and 13th line of the set data correspond to "0" and DH", respectively.

The character at the start of the screen is specified with an absolute line (0 to 11th line in Fig. 20-10). When having a gap between lines, specify the number of lines for the gap using vertical position data. In other words, specify in hexadecimal notation the number of lines to be displayed after the currently displayed character. For example, the vertical position data of "A", "B" and "D" in Fig. 20-10 are 6H, 1H and 0H, respectively.

Remarks: One line is an example of x1 when vertical size data is "00". If vertical size data changes, the number of lines is counted accordingly. For example, if the vertical size data is doubled, one screen will consist of 6 lines.

Fig. 10-10 Display Positions



(5) Color data (3 bits #2 to #0 of control data)

Color data is intended to specify character colors. Data is output from the specified output pin (R, G and B pins). The relations between color data and pins are shown in Fig. 20-8.

The relations between color data settings and actually output colors are shown in Table 20-9.

Table 20-8 Color Data

#2	#1	#0
R	G	B

Table 20-9 Character Colors

Color Data			Character Color
R	G	B	
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

20.5.3 Method of Definition Using Assembler

With the 17K series assembler, display patterns can easily be defined by the DCP pseudo-instruction. The description method for the DCP pseudo-instruction is shown below.

(1) Description format

Symbol field	Mnemonic field	Operand field	Comment field
[Label:]	DCP	Expression, 'Display pattern'	[; Comment]

(2) Explanation

(a) The expression has a value of "0" or "1" specifying whether or not trimming should be done for the display pattern described by the 2nd operand.

"0": No trimming
"1": Trimming

When the evaluation value of the expression is not "0" or "1", an error occurs.

(b) Only three types of characters are used for display pattern. They are "0", "#" and " " (blank) and a total of 10 characters are described.
If characters except these three types are described or more than or less than 10 characters are described, errors will result. The three types of characters correspond to one dot of display pattern for the following relationships.

"0": ON
"#": Trimming
" ": Blank

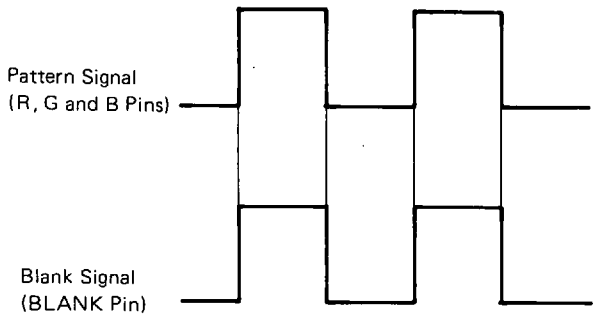
When the evaluation value of the 1st operand expression is "0" , "#" cannot be used for display pattern.

20.6 BLANK, R, G AND B PINS

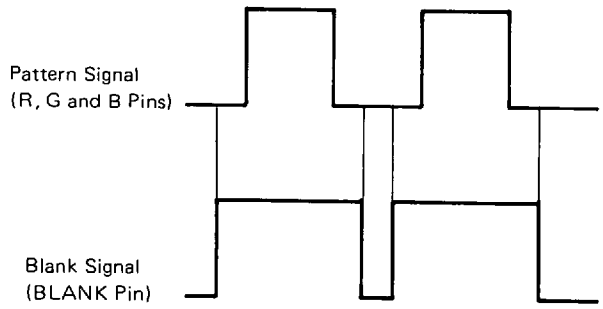
These pins are CMOS push-pull type output pins. An active-high signal is output from these pins. A signal which erases broadcast images is output from the BLANK pin. Character pattern data is output form the R, G and B pins. Without trimming, the BLANK signal and the character pattern signal (R, G and B OR signals) are the same signals. With trimming, the BLANK signal having a waveform surrounding the character pattern signal is output from the BLANK pin.

Fig. 20-11 IDC Output Waveforms

(a) Without trimming



(b) With trimming



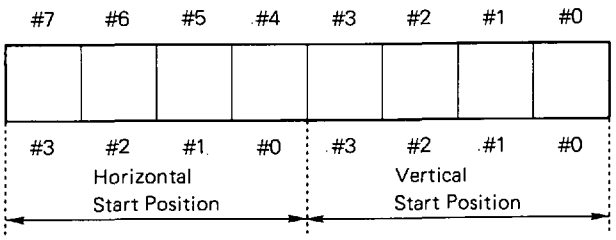
20.7 DISPLAY START POSITION SETTING

IDC display start position (top left on the screen) can be set in a total of 16 locations in horizontal and vertical directions by setting data to the IDC start position setting register. This means that the display position of the whole screen can be moved. The IDC start position setting register consists of a vertical start position setting register and a horizontal start position setting register, each having 4 bits (8 bits in total).

The IDC start position setting register is assigned peripheral address 01H and used to set or check data by the "GET" and "PUT" instructions.

Set the IDC start position setting register when the IDCEN flag is "0".

Fig. 20-12 IDC Start Position Setting Register Configuration



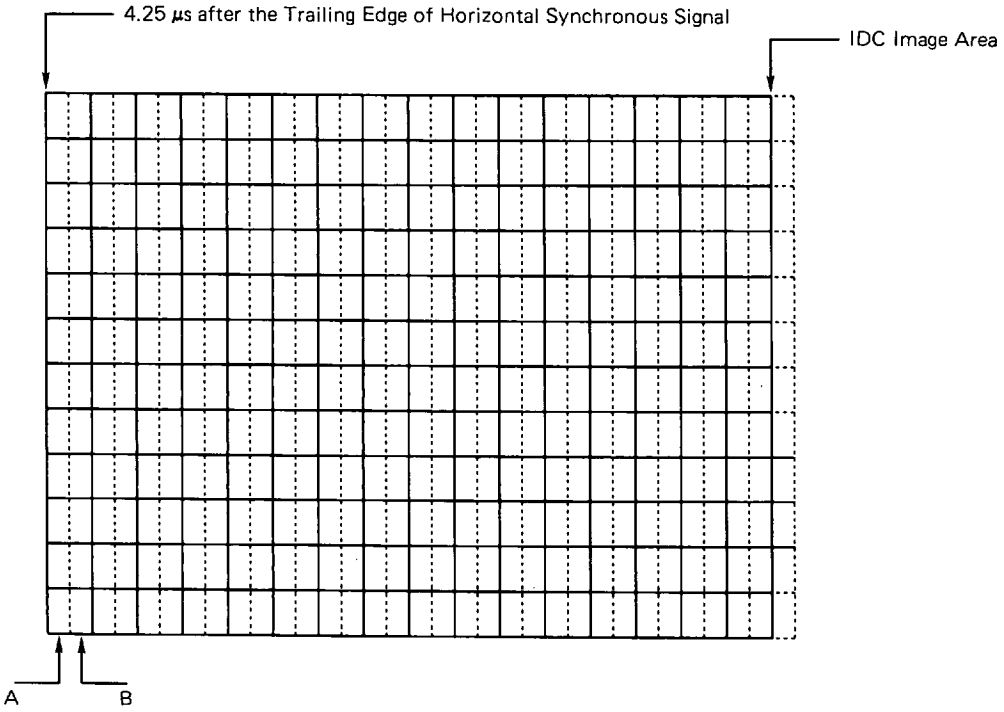
20.7.1 Horizontal Start Position Setting Register

When "0H" data is set to the horizontal start position setting register, the horizontal start position is set 4.25 μs after the trailing edge of the horizontal synchronous signal. As this data increments by "1", the horizontal start position shifts to the right by 250 ns: It is given in the following expression.

Horizontal start position = 4.25 μs + 250 ns x (horizontal start position setting data)

In Fig. 20-13, the position is A when the horizontal start position setting data is "0H". When the horizontal start position setting data is set to "1", the position will move to the right by 250 ns (for 1 dot of the minimum size character) to position B (the screen with data set to "0" is indicated in sold line and that with data set to "1" is indicated in dotted line).

Fig. 20-13 Horizontal Move



20.7.2 Vertical Start Position Setting Register

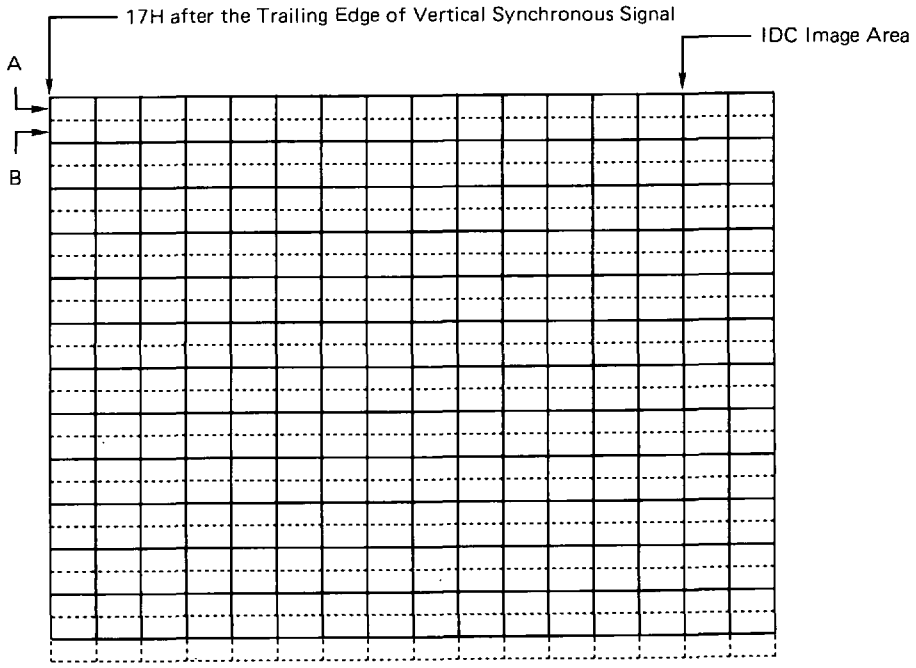
When "0H" data is set to the vertical start position setting register, the vertical start position is set 17H (interlace) after the trailing edge of the vertical synchronous signal. As this data increments by "1", the vertical start position shifts downward by 1H.

It is given in the following expression.

Vertical start position = 17H + 1H x (vertical start position setting data)

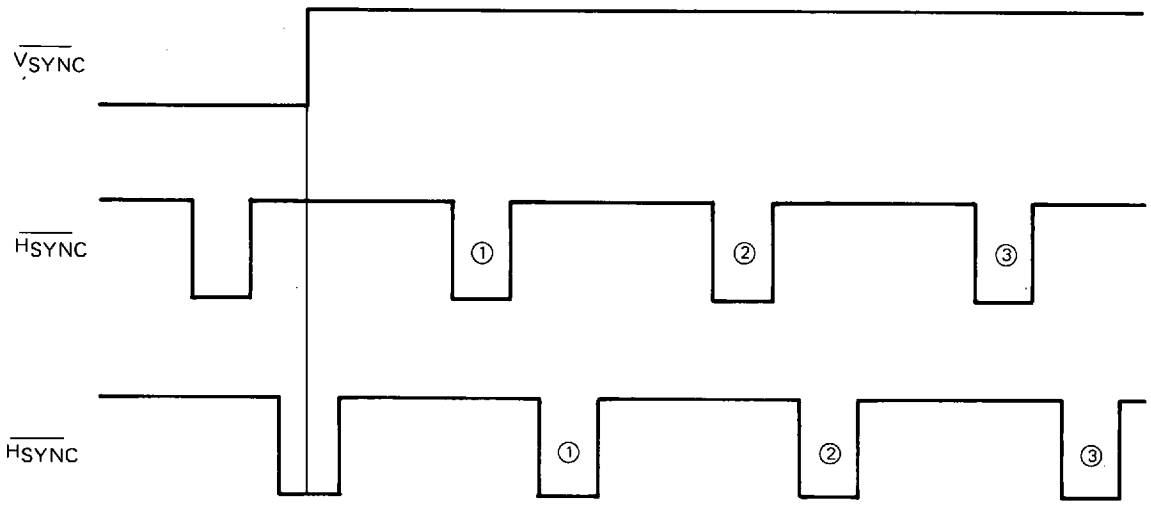
In Fig. 20-14, the position is A when the vertical start position setting data is "0H". When the vertical start position setting data is set to "1", the position will move downward by 1H to position B (the screen with data set to "0" is indicated in sold line and that with data set to "1" is indicated in dotted line).

Fig. 20-14 Vertical Move



The vertical start position of display character is determined by the vertical start position register. The vertical start position (number of horizontal scanning lines) is determined according to the statuses of $\overline{V_{SYNC}}$ and $\overline{H_{SYNC}}$ signals to be input to the μPD17052 as shown in Fig. 20-15. That is, the first $\overline{H_{SYNC}}$ signal after the rising edge of the $\overline{V_{SYNC}}$ signal is counted as 1H.

Fig. 20-15 Vertical Start Position Counting

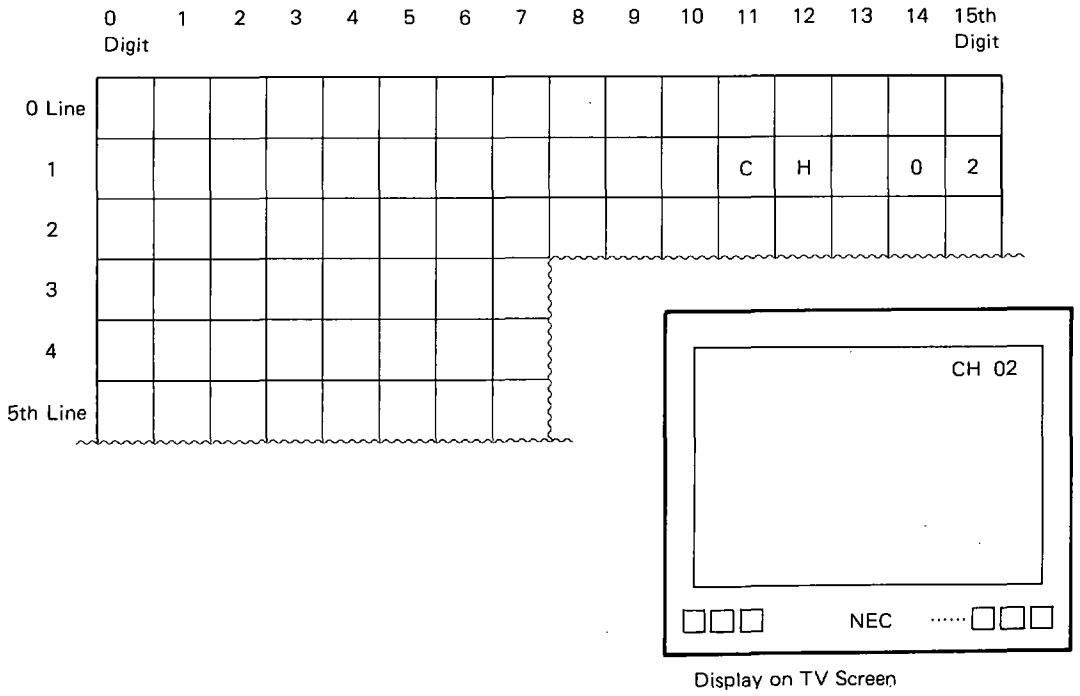


The circled number is that of scanning lines.

20.8 PROGRAM EXAMPLE

This section shows a program example for the following screen display.

Display contents



Display on TV Screen

VRAM RAM names are defined as follows (temporarily).

```
; **RAM SET**
VRAM0   MEM   2.00H
VRAM1   MEM   2.01H
VRAM2   MEM   2.02H
VRAM3   MEM   2.03H
VRAM4   MEM   2.04H
VRAM5   MEM   2.05H
VRAM6   MEM   2.06H
VRAM7   MEM   2.07H
VRAM8   MEM   2.08H
VRAM9   MEM   2.09H
VRAMA   MEM   2.0AH
VRAMB   MEM   2.0BH
VRAMC   MEM   2.0CH
VRAMD   MEM   2.0DH
VRAME   MEM   2.0EH
VRAMF   MEM   2.0FH
        :
```

VRAM Map (BANK2)

	0	1	2	3	4	5	6	7
0	VRAM0	VRAM1	VRAM2	VRAM3	VRAM4	VRAM5	VRAM6	VRAM7
1	------	------	------	------	------	------	------	------
2	------	------	------	------	------	------	------	------

	8	9	A	B	C	D	E	F
0	VRAM8	VRAM9	VRAMA	VRAMB	VRAMC	VRAMD	VRAME	VRAMF
1	------	------	------	------	------	------	------	------
2	------	------	------	------	------	------	------	------

A program is created as follows.

```
        Program Start
                                        ; Initialize such as RAM clear is executed.
        ┌──────────┐
        │Initialize│
        └──────────┘
        SET1    IDCDMAEN        ; DMA mode is set.
        CLR1    IDCEN           ; Display is turned off.
;
;**Channel Display Routine**
;
        CLR1    CROMBNK         ; CROM bank is set to 0.
;
        MOV     VRAM0, #1000B   ; Control code 1 is set.
        MOV     VRAM1, #0000B
;
        MOV     VRAM2, #0       ; Display character data "C" is set.
        MOV     VRAM3, #0CH
;
        MOV     VRAM4, #0       ; Display character data "H" is set.
        MOV     VRAM5, #0DH
;
        MOV     VRAM6, #1000B   ; Control code 2 is set.
        MOV     VRAM7, #0001B
;
        MOV     VRAM8, #0       ; Display character data "0" is set.
        MOV     VRAM9, #0
;
        MOV     VRAMA, #0       ; Display character data "2" is set.
        MOV     VRAMB, #2
;
        MOV     VRAMC, #0100B   ; CR (Carriage Return)
        MOV     VRAMD, #0000B
;
        MOV     VRAME, #0100B   ; CR (Carriage Return)
        MOV     VRAMF, #0000B
        ;①
LOOP :
        SKF1    INTVSYN         ; Vsync=low level is checked and display is
        BR      LOOP              turned on.
        SET1    IDCEN           ; Display ON
        ⋮
```

VRAM (BANK2) has the following contents at point 1 .

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	8	0	0	C	0	D	8	1	0	0	0	2	4	0	4	0
1																

The CROM contents in this example are as follows.

```
                        CROM DATA
                        ;**************************
                        ;***  Image Display Controller data set  ***
ROM ADDRESS             ;**************************
    1800                        ORG    1800H
                                ;********
                                ;***  0  ***
                                ;********
    1800   0000         DCP 0, '                '     ; "0"
    1801   007C         DCP 0, '     OOOOO      '
    1802   00FE         DCP 0, '   OOOOOOO      '
    1803   01C7         DCP 0, ' OOO     OOO    '
    1804   0183         DCP 0, ' OO        OO   '
    1805   0183         DCP 0, ' OO        OO   '
    1806   0183         DCP 0, ' OO        OO   '
    1807   0183         DCP 0, ' OO        OO   '
    1808   0183         DCP 0, ' OO        OO   '
    1809   0183         DCP 0, ' OO        OO   '
    180A   0183         DCP 0, ' OO        OO   '
    180B   0183         DCP 0, ' OO        OO   '
    180C   01C7         DCP 0, ' OOO     OOO    '
    180D   00FE         DCP 0, '   OOOOOOO      '
    180E   007C         DCP 0, '     OOOOO      '
                                ;**  CD1  **      ;**Control Data 1**
    180F   058A         DW     0000010110001010B ;Horizontal size : x1,
                                                   vertical size : x1.
                                                  ;Horizontal position : 11 digits,
                                                   vertical position : 1 line
                                                  ;Color : Green(G), trimming : None
```

```
                        ; ********
                        ; ***  1  ***
                        ; ********
1810  0000        DCP 0, '              '    ; "1"
1811  0006        DCP 0, '        OO    '
1812  000E        DCP 0, '        OO    '
1813  001E        DCP 0, '      OOOO    '
1814  0076        DCP 0, '      OOOO    '
1815  00C6        DCP 0, '        OO    '
1816  0186        DCP 0, '        OO    '
1817  0006        DCP 0, '        OO    '
1818  0006        DCP 0, '        OO    '
1819  0006        DCP 0, '        OO    '
181A  0006        DCP 0, '        OO    '
181B  0006        DCP 0, '        OO    '
181C  0006        DCP 0, '        OO    '
181D  0006        DCP 0, '      OOOOOO  '
181E  0006        DCP 0, '      OOOOOO  '
                        ; **  CD2  **    ; **Control Data 2**
181F  0082        DW    0000000010000010B ; Horizontal size : x1, vertical size : x1
                                          ; Horizontal position : 1 digit,
                                            vertical position : 0 line
                                          ; Color : Green(G), trimming : None
                        ; ********
                        ; ***  2  ***
                        ; ********
1820  0000        DCP 0, '              '    ; "2"
1821  007C        DCP 0, '     OOOOO    '
1822  00FE        DCP 0, '    OOOOOOO   '
1823  01C7        DCP 0, '  OOO     OOO '
1824  0183        DCP 0, '  OO       OO '
1825  0003        DCP 0, '           OO '
1826  0007        DCP 0, '          OOO '
1827  000E        DCP 0, '         OOO  '
1828  0038        DCP 0, '        OOO   '
1829  00E0        DCP 0, '       OOO    '
182A  01C0        DCP 0, '     OOO      '
182B  0180        DCP 0, '    OOO       '
182C  0180        DCP 0, '  OOO         '
182D  01FF        DCP 0, '  OOOOOOOOO   '
182E  01FF        DCP 0, '  OOOOOOOOO   '
                        ; **  CD2  **
182F  0000        DW    0000000000000000B ; NO USE
```

```
                                ;********
                                ;*** 3 ***
                                ;********
1830  0000      DCP 0, '              '    ; "3"
1831  007C      DCP 0, '    OOOOO     '
1832  00FE      DCP 0, '  OOOOOOO     '
1833  01C7      DCP 0, ' OOO    OOO   '
1834  0183      DCP 0, ' OO       OO  '
1835  0003      DCP 0, '          OO  '
                  :
                  :
                  :
                                ;********
                                ;*** C ***
                                ;********
18C0  0000      DCP 0, '              '    ; "C"
18C1  007F      DCP 0, '    OOOOO     '
18C2  00FF      DCP 0, '  OOOOOOO     '
18C3  01C0      DCP 0, ' OOO    OOO   '
18C4  0180      DCP 0, ' OO       OO  '
18C5  0180      DCP 0, ' OO           '
18C6  0180      DCP 0, ' OO           '
18C7  0180      DCP 0, ' OO           '
18C8  0180      DCP 0, ' OO           '
18C9  0180      DCP 0, ' OO           '
18CA  0180      DCP 0, ' OO           '
18CB  0180      DCP 0, ' OO       OO  '
18CC  01C0      DCP 0, ' OOO    OOO   '
18CD  00FF      DCP 0, '  OOOOOOO     '
18CE  007F      DCP 0, '    OOOOO     '
                                ;
18CF  0000      DW    0000000000000000B    ; NO USE
```

```
                        ; ********
                        ; *** H ***
                        ; ********
18D0  0000      DCP 0, '              '     ; "H"
18D1  0183      DCP 0, '  OO      OO '
18D2  0183      DCP 0, '  OO      OO '
18D3  0183      DCP 0, '  OO      OO '
18D4  0183      DCP 0, '  OO      OO '
18D5  0183      DCP 0, '  OO      OO '
18D6  0183      DCP 0, '  OO      OO '
18D7  01FF      DCP 0, '  OOOOOOOOO '
18D8  01FF      DCP 0, '  OOOOOOOOO '
18D9  0183      DCP 0, '  OO      OO '
18DA  0183      DCP 0, '  OO      OO '
18DB  0183      DCP 0, '  OO      OO '
18DC  0183      DCP 0, '  OO      OO '
18DD  0183      DCP 0, '  OO      OO '
18DE  0183      DCP 0, '  OO      OO '
                        ;
18DF  0000      DW    0000000000000000B     ; NO USE
```

21. HORIZONTAL SYNCHRONOUS SIGNAL COUNTER

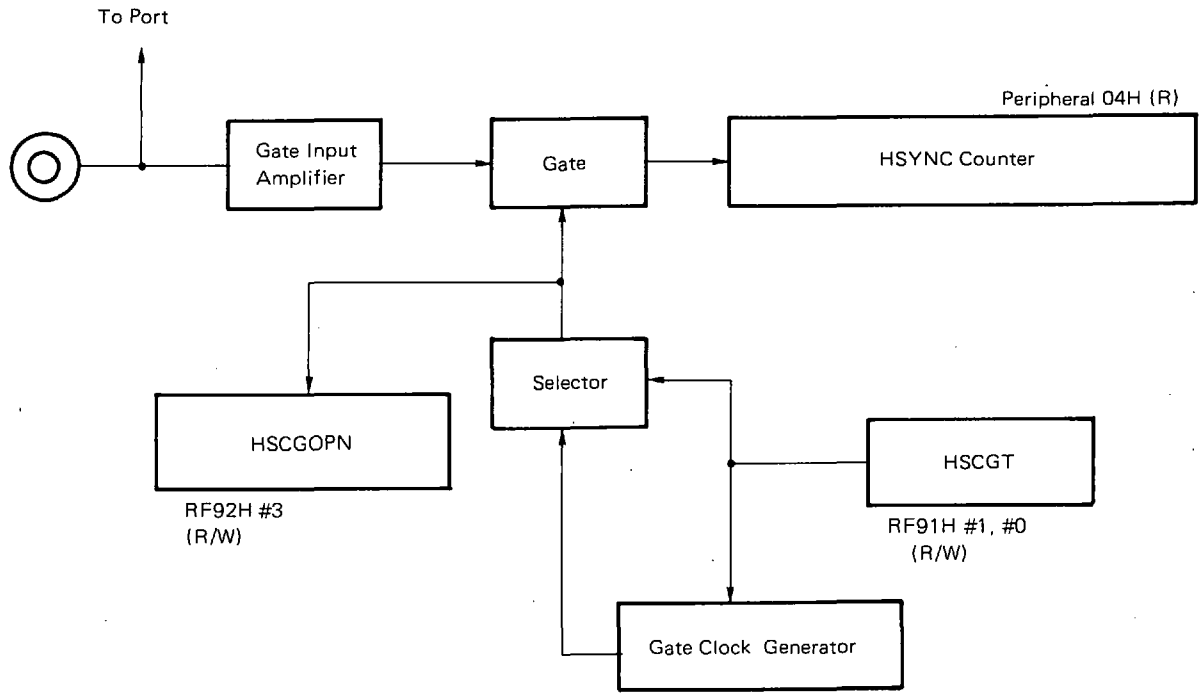
21.1 HORIZONTAL SYNCHRONOUS SIGNAL COUNTER CONFIGURATION

The horizontal synchronous signal counter is used to measure the frequency of TV or other horizontal synchronous signal. This counter can be used to check if there is any broadcasting station in the received frequency range utilizing the fact that in TV broadcast reception the specified horizontal synchronous signal is output.

The horizontal synchronous signal counter consists of a 6-bit HSYNC counter (HSC), a gate clock generator, a gate control register (HSCGT), a gate input amplifier and a test gate open register (HSCGOPN).

A signal input from the $P0B_3$/HSCNT pin (pin No. 35) is amplified by a self-biased input amplifier, passes through the gate controlled by the gate control register to open for the specified time, and is counted by the 6-bit HSYNC counter. When the gate closes, the HSYNC counter stops counting and sets the test gate open register to "1". Because the HSYNC counter is a read only register, the number of pulses counted while the gate is open can be checked by reading the HSYNC counter. Thus, the frequency can be obtained by dividing the value read from the HSYNC counter by the time the gate is open (1.69 ms). The $P0B_3$/HSCNT pin also serves as an I/O port and is assigned to $P0B_3$ port. When using this pin as a horizontal synchronous signal counter, set $P0B_3$ as an input port. When using is as a port, set the HSCGT to 0000B. If $P0B_3$ is read when the pin is used as a horizontal synchronous signal counter, the read value is always "0".

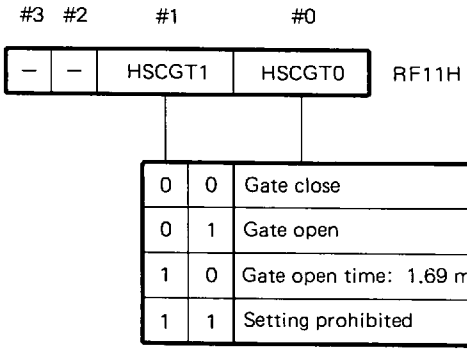
Fig. 21-1 Horizontal Synchronous Signal Counter Block Diagram



21.2 GATE CONTROL REGISTER (HSCGT)

The gate control register is a 2-bit register to control the gate. It consists of HSCGT1 flag and HSCGT0 flag and is located at register file address 11H. Data can be written/read to this register by the "PEEK" and "POKE" instructions via the window register of the system register.

The following modes are available by setting the gate control register.



0	0	Gate close
0	1	Gate open
1	0	Gate open time: 1.69 ms*
1	1	Setting prohibited

* Only in this mode the gate clock generator operates.

21.2.1 Gate Close Mode

In this mode the gate remains closed and the HSC and the gate clock generator do not operate (HSYNC counter value remains unchanged). Because the horizontal synchronous signal counter input bias is also turned off, be sure to set this mode when using as a port.

This mode is set upon power-on reset or clock stop.

21.2.2 Gate Open Mode

When this mode is set, the gate opens and the HSYNC counter is reset and input signal counting starts.

If the HSYNC counter overflows, "0" is set.

In this mode the input pin is biased.

21.2.3 Gate 1.69 ms Mode

When this mode is set the HSYNC counter is reset and counting starts by a maximum delay of 62.5 μs. The gate time is 1.69 ms. In this mode the input pin is biased.

21.3 HSYNC COUNTER (HSC)

This counter is assigned at peripheral address 04H. It is a 6-bit read only binary counter which reads data by the GET instruction via a data buffer.

Because this counter has 6 bits, it starts counting again with 00H if an overflow occurs.

Upon power-on reset or clock stop, the HSYNC counter value is reset at 00H.

(1) Gate open bit (HSCGOPN)

HSCGOPN is assigned to MSB (b_3) of register file 12H. When the $\overline{H_{sync}}$ input gate is opened, HSCGOPN remains at the high level. However, if the gate 1.69 ms mode is selected, a high level is output when data is set if no gate clock arrives.

21.4 HORIZONTAL SYNCHRONOUS SIGNAL COUNTER USAGE EXAMPLE

The following is a program example when a horizontal synchronous signal counter is used.

In the case of 1.69 ms gate open

```
        :
        :
        CLR1    P0BBIO3         ; Sets P0B3 to the input mode.
        :
        :
        :
        :
        :
        :
        :
```

```
PEEK    WR, 0B6H
AND     WR, #0111B
POKE    0B6H, WR
```

```
LOOP:
        PEEK    WR, #92H        ; Checks that the gate closes once.
        SKF     WR, #1000B
        BR      LOOP
        ;
        MOV     WR, #0010B      ; Sets the 1.69 ms gate open mode.
        POKE    91H, WR
                :
                :
                :
LOOP2:
        PEEK    WR, #92H        ; Checks that the gate closes.
        SKF     WR, #1000B
        BR      LOOP2
        GET     DBF, HSC        ; Fetches the HSYNC counter contents.
        :
        :
        :
```

22. μPD17052 INSTRUCTIONS

22.1 GENERAL DESCRIPTION ON INSTRUCTION SET

		0		1	
0 0 0 0	0	ADD	r, m	ADD	m, #i
0 0 0 1	1	SUB	r, m	SUB	m, #i
0 0 1 0	2	ADDC	r, m	ADDC	m, #i
0 0 1 1	3	SUBC	r, m	SUBC	m, #i
0 1 0 0	4	AND	r, m	AND	m, #i
0 1 0 1	5	XOR	r, m	XOR	m, #i
0 1 1 0	6	OR	r, m	OR	m, #i
0 1 1 1	7	INC INC MOVT BR CALL RET RETSK EI DI RETI PUSH POP GET PUT PEEK POKE RORC STOP HALT NOP	AR IX DBF, @AR @AR @AR AR AR DBF, p p, DBF WR, rf rf, WR r 0 h		
1 0 0 0	8	LD	r, m	ST	m, r
1 0 0 1	9	SKE	m, #i	SKGE	m, #i
1 0 1 0	A	MOV	@r, m	MOV	m, @r
1 0 1 1	B	SKNE	m, #i	SKLT	m, #i
1 1 0 0	C	BR	addr (Page 0)	CALL	addr (Page 0)
1 1 0 1	D	BR	addr (Page 1)	MOV	m, #i
1 1 1 0	E	BR	addr (Page 2)	SKT	m, #n
1 1 1 1	F	BR	addr (Page 3)	SKF	m, #n

22.2 LEGEND

M	:	Data memory indicated by [(BANK), m]
m	:	Data memory address indicated by [m_H, m_L]
m_H	:	Data memory row address (3 bits)
m_L	:	Data memory column address (4 bits)
R	:	General register indicated by [(RP), r]
r	:	General register column address (4 bits)
RP	:	General register pointer
RF	:	Register file indicated by rf
rf	:	Register file address indicated by [rf_H, rf_L]
rf_H	:	Register file address (most significant 3 bits)
rf_L	:	Register file address (least significant 3 bits)
AR	:	Address register
IX	:	Index register
IXE	:	Index enable flag
DBF	:	Data buffer
WR	:	Window register
MT	:	Data memory row address pointer
MPE	:	Memory pointer enable flag
PE	:	Peripheral register
p	:	Peripheral address
p_H	:	Peripheral address (most significant 3 bits)
p_L	:	Peripheral address (least significant 4 bits)
PC	:	Program counter
SP	:	Stack pointer
STACK	:	Stack value indicated by stack pointer
$STACK_{PC}$	:	Program counter value indicated by stack pointer
BANK	:	Bank register
$(ROM)_{PC}$	:	Program memory data indicated by (PC)
INTEF	:	Interrupt enable flag
SGR	:	Program memory segment register
i	:	Immediate data (4 bits)
n	:	Bit position (4 bits)
addr	:	Program memory address (11 bits)
c	:	Carry
b	:	Borrow
h	:	Halt release condition
[ ]	:	Data memory or register address
( )	:	Data memory or register value

22.3 INSTRUCTION SET LIST

Instruction Group	Mnemonic	Operand	Operation	Machine Code			
				Operation Code			
Add instructions	ADD	r, m	(R)←(R) + (M)	00000	m_H	m_L	r
		m, #i	(M)←(M) + i	10000	m_H	m_L	i
	ADDC	r, m	(R)←(R) + (M) + c	00010	m_H	m_L	r
		m, #i	(M)←(M) + i + c	10010	m_H	m_L	i
	INC	AR	(AR)←(AR) + 1	00111	000	1001	0000
		IX	(IX)←(IX) + 1	00111	000	1000	0000
Subtract instructions	SUB	r, m	(R)←(R) − (M)	00001	m_H	m_L	r
		m, #i	(M)←(M) − i	10001	m_H	m_L	i
	SUBC	r, m	(R)←(R) − (M) − b	00011	m_H	m_L	r
		m, #i	(M)←(M) − i − b	10011	m_H	m_L	i
Compare instructions	SKE	m, #i	(M) − i, skip if zero	01001	m_H	m_L	i
	SKGE	m, #i	(M) − i, skip if not borrow	11001	m_H	m_L	i
	SKLT	m, #i	(M) − i, skip if borrow	11011	m_H	m_L	i
	SKNE	m, #i	(M) − i, skip if not zero	01011	m_H	m_L	i
Logical operation instructions	AND	m, #i	(M)←(M) AND i	10100	m_H	m_L	i
		r, m	(R)←(R) AND (M)	00100	m_H	m_L	r
	OR	m, #i	(M)←(M) OR i	10110	m_H	m_L	i
		r, m	(R)←(R) OR (M)	00110	m_H	m_L	r
	XOR	m, #i	(M)←(M) XOR i	10101	m_H	m_L	i
		r, m	(R)←(R) XOR (M)	00101	m_H	m_L	r
Transfer instructions	LD	r, m	(R)←(M)	01000	m_H	m_L	r
	ST	m, r	(M)←(R)	11000	m_H	m_L	r
	MOV	@r, m	if MPE=1 : [(MP), (R)]←(M) if MPE=0 : [(m_H), (R)]←(M)	01010	m_H	m_L	r
		m, @r	if MPE=1 : (M)←[(MP), (R)] if MPE=0 : (M)←[(m_H), (R)]	11010	m_H	m_L	r
		m, #i	(M)← i	11101	m_H	m_L	i
	MOVT*	DBF, @AR	($STACK_{PC}$)←(PC), (PC)←(AR), (DBF)←$(ROM)_{PC}$, (PC)←($STACK_{PC}$)	00111	000	0001	0000
	PUSH	AR	(SP)←(SP) − 1, ($STACK_{PC}$)←(AR)	00111	000	1101	0000
	POP	AR	(AR)←($STACK_{PC}$), (SP)←(SP) + 1	00111	000	1100	0000
	PEEK	WR, rf	(WR)←(RF)	00111	rf_H	0011	rf_L
	POKE	rf, WR	(RF)←(WR)	00111	rf_H	0010	rf_L
	GET	DBF, p	(DBF)←(PE)	00111	p_H	1011	p_L
	PUT	p, DBF	(PE)←(DBF)	00111	p_H	1010	p_L

*: 2 machine cycles (for 2 instructions) is necessary for MOVT instruction execution. The stack is temporarily used for execution.

Instruction Group	Mnemonic	Operand	Operation	Machine Code			
				Operation Code			
Judge instructions	SKT	m, #n	if $(M)_n$ = all "1", then skip	11110	m_H	m_L	n
Judge instructions	SKF	m, #n	if $(M)_n$ = all "0", then skip	11111	m_H	m_L	n
Branch instructions	BR	addr	(PC) ← addr, (PC) $\#_{12}$, $\#_{11}$ ← 00	01100	addr (least significant 11 bits)		
			(PC) ← addr, (PC) $\#_{12}$, $\#_{11}$ ← 01	01101			
			(PC) ← addr, (PC) $\#_{12}$, $\#_{11}$ ← 10	01110			
			(PC) ← addr, (PC) $\#_{12}$, $\#_{11}$ ← 11	01111			
		@AR	(PC) ← (AR)	00111	000	0100	0000
Shift	RORC	r	C ← (R) $\#_3$ → (R) $\#_2$ → (R) $\#_1$ → (R) $\#_0$ → C	00111	000	0111	r
Subroutine instructions	CALL	addr	(SP) ← (SP) − 1, $(STACK_{PC})$ ← ((PC) + 1) (PC) $\#_{11}$ ← 0, (PC) ← addr	11100	addr (11bits)		
		@AR	(SP) ← (SP) − 1, $(STACK_{PC})$ ← ((PC) + 1) (PC) ← (AR)	00111	000	0101	0000
	RET		(PC) ← $(STACK_{PC})$, (SP) ← (SP) + 1	00111	000	1110	0000
	RETSK		(PC) ← $(STACK_{PC})$, (SP) ← (SP) + 1, and skip	00111	001	1110	0000
	RETI		(PC), (BANK), (IXE) ← (STACK), (SP) ← (SP) + 1	00111	100	1110	0000
Interrupt	EI		INTEF ← 1	00111	000	1111	0000
Interrupt	DI		INTEF ← 0	00111	001	1111	0000
Others	STOP	0	stop clock if CE = low	00111	010	1111	0000
Others	HALT	h	halt	00111	011	1111	h
Others	NOP		No operation	00111	100	1111	0000

22.4 INTRINSIC MACRO INSTRUCTIONS

The following macro instructions are available as intrinsic macro instructions for the 17K series assembler (AS 17K). For details, refer to the assembler user's manual.

Legend

flag : One of flags flag1 to flagn
flag1-flagn: Flag name indicated by reserved word
n : Number
< > : Omissible

	Mnemonic	Operand	n	Operation
Assemble macro instructions	SKTn	flag1, ... flagn	$1 \leqq n \leqq 4$	If (flag1)–(flagn) = all '1", then skip
	SKFn	flag1, ... flagn	$1 \leqq n \leqq 4$	If (flag1)–(flagn) = all "0", then skip
	SETn	flag1, ... flagn	$1 \leqq n \leqq 4$	(flag1)–(flagn) ← 1
	CLRn	flag1, ... flagn	$1 \leqq n \leqq 4$	(flag1)–(flagn) ← 0
	NOTn	flag1, ... flagn	$1 \leqq n \leqq 4$	if (flag) = "0", then (flag) ← 1, if (flag) = "1", then (flag) ← 0
	INITFLG	<NOT> flag1, ... <NOT> flagn	$1 \leqq n \leqq 4$	if description = NOT flag, (flag) ← 0 if description = flag, (flag) ← 1
	BANKn		$0 \leqq n \leqq 3$	(BANK) ← n

23. RESERVED SYMBOLS OF ASSEMBLER

When an assembler is used, the μPD17052 reserved symbols are as follows.

23.1 SYSTEM REGISTER (SYSREG)

Reserved Word	Type	Address	Read/ Write	Function Outline
AR3	MEM	0.74H	R	Address register bits b15 to b12
AR2	MEM	0.75H	R	Address register bits b11 to b8
AR1	MEM	0.76H	R/W	Address register bits b9 to b4
AR0	MEM	0.77H	R/W	Address register bits b3 to b0
WR	MEM	0.78H	R/W	Window register
BANK	MEM	0.79H	R/W	Bank register
IXH	MEM	0.7AH	R/W	Index register high
MPH	MEM	0.7AH	R/W	Data memory low address pointer high
MPE	FLG	0.7AH.3	R/W	Memory pointer enable flag
IXM	MEM	0.7BH	R/W	Index register middle
MPL	MEM	0.7BH	R/W	Data memory low address pointer low
IXL	MEM	0.7CH	R/W	Index register low
RPH	MEM	0.7DH	R/W	General register pointer high
RPL	MEM	0.7EH	R/W	General register pointer low
PSW	MEM	0.7FH	R/W	Program status word
BCD	FLG	0.7EH.0	R/W	BCD flag
CMP	FLG	0.7FH.3	R/W	Compare flag
CY	FLG	0.7FH.2	R/W	Carry flag
Z	FLG	0.7FH.1	R/W	Zero flag
IXE	FLG	0.7FH.0	R/W	Index enable flag

23.2 DATA BUFFER (DBF)

Reserved Word	Type	Address	Read/ Write	Function Outline
DBF3	MEM	0.0CH	R/W	Data buffer bits b15 to b12
DBF2	MEM	0.0DH	R/W	Data buffer bits b11 to b8
DBF1	MEM	0.0EH	R/W	Data buffer bits b7 to b4
DBF0	MEM	0.0FH	R/W	Data buffer bits b3 to b0

23.3 GENERAL-PURPOSE PORT REGISTER

Reserved Word	Type	Address	Read/ Write	Function Outline
P0A3	FLG	0.70H.3	R/W	Port 0A bit b3
P0A2	FLG	0.70H.2	R/W	Port 0A bit b2
P0A1	FLG	0.70H.1	R/W	Port 0A bit b1
P0A0	FLG	0.70H.0	R/W	Port 0A bit b0
P0B3	FLG	0.71H.3	R/W	Port 0B bit b3
P0B2	FLG	0.71H.2	R/W	Port 0B bit b2
P0B1	FLG	0.71H.1	R/W	Port 0B bit b1
P0B0	FLG	0.71H.0	R/W	Port 0B bit b0
P0C3	FLG	0.72H.3	R/W	Port 0C bit b3
P0C2	FLG	0.72H.2	R/W	Port 0C bit b2
P0C1	FLG	0.72H.1	R/W	Port 0C bit b1
P0C0	FLG	0.72H.0	R/W	Port 0C bit b0
P0D3	FLG	0.73H.3	R	Port 0D bit b3
P0D2	FLG	0.73H.2	R	Port 0D bit b2
P0D1	FLG	0.73H.1	R	Port 0D bit b1
P0D0	FLG	0.73H.0	R	Port 0D bit b0
P1A3	FLG	1.70H.3	R/W	Port 1A bit b3
P1A2	FLG	1.70H.2	R/W	Port 1A bit b2
P1A1	FLG	1.70H.1	R/W	Port 1A bit b1
P1A0	FLG	1.70H.0	R/W	Port 1A bit b0
P1B3	FLG	1.71H.3	R/W	Port 1B bit b3
P1B2	FLG	1.71H.2	R/W	Port 1B bit b2
P1B1	FLG	1.71H.1	R/W	Port 1B bit b1
P1B0	FLG	1.71H.0	R/W	Port 1B bit b0
P1C3	FLG	1.72H.3	R/W	Port 1C bit b3
P1C2	FLG	1.72H.2	R/W	Port 1C bit b2
P1C1	FLG	1.72H.1	R/W	Port 1C bit b1
P1C0	FLG	1.72H.0	R/W	Port 1C bit b0
P1D3	FLG	1.73H.3	R/W	Port 1D bit b3
P1D2	FLG	1.73H.2	R/W	Port 1D bit b2
P1D1	FLG	1.73H.1	R/W	Port 1D bit b1
P1D0	FLG	1.73H.0	R/W	Port 1D bit b0
P2A3	FLG	2.70H.3	R/W	Port 2A bit b3
P2A2	FLG	2.70H.2	R/W	Port 2A bit b2
P2A1	FLG	2.70H.1	R/W	Port 2A bit b1

Reserved Word	Type	Address	Read/ Write	Function Outline
P2A0	FLG	2.70H.0	R/W	Port 2A bit b0
P2B3	FLG	2.71H.3	R/W	Port 2B bit b3
P2B2	FLG	2.71H.3	R/W	Port 2B bit b2
P2B1	FLG	2.71H.2	R/W	Port 2B bit b1
P2B0	FLG	2.71H.1	R/W	Port 2B bit b0
P2C3	FLG	2.72H.3	R/W	Port 2C bit b3
P2C2	FLG	2.72H.2	R/W	Port 2C bit b2
P2C1	FLG	2.72H.1	R/W	Port 2C bit b1
P2C0	FLG	2.72H.0	R/W	Port 2C bit b0

23.4 REGISTER FILE (CONTROL REGISTER)

Reserved Word	Type	Address	Read/ Write	Function Outline
IDCDMAEN	FLG	0.80H.1	R/W	DMA enable flag
SP	MEM	0.81H	R/W	Stack pointer
CE	FLG	0.87H.0	R	CE pin status flag
SIOCH	FLG	0.88H.3	R/W	SIO channel select flag
SB	FLG	0.88H.2	R/W	SIO mode slect flag
SIOMS	FLG	0.88H.1	R/W	SIO clock mode select flag
SIOTX	FLG	0.88H.0	R/W	SIO TX/RX select flag
ZCROSS	FLG	0.89H.3	R/W	Timer interrupt mode select flag
TMMD2	FLG	0.89H.2	R/W	Timer carry FF mode select flag
TMMD1	FLG	0.89H.1	R/W	Timer carry FF mode select flag
TMMD0	FLG	0.89H.0	R/W	Timer carry FF mode select flag
INTVSYN	FLG	0.8FH.2	R	Vsync pin status flag
INT	FLG	0.8FH.0	R	RMC pin status flag
HSCGT3	FLG	0.91H.3	R	Hsync counter mode select flag (dummy: 0)
HSCGT2	FLG	0.91H.2	R	Hsync counter mode select flag (dummy: 0)
HSCGT1	FLG	0.91H.1	R/W	Hsync counter mode select flag
HSCGT0	FLG	0.91H.0	R/W	Hsync counter mode select flag
HSCGOPN	FLG	0.92H.3	R/W	Hsync counter gate open flag
RMCSTAT3	FLG	0.95H.3	R	RMC pin status flag (dummy: 0)
RMCSTAT2	FLG	0.95H.2	R/W	RMC pin status flag
RMCSTAT1	FLG	0.95H.1	R/W	RMC pin status flag
RMCSTAT0	FLG	0.95H.0	R/W	RMC pin status flag
TMCY	FLG	0.97H.0	R	Timer carry FF status flag
SBACK	FLG	0.98H.3	R/W	Serial bus acknowledge flag
SIONWT	FLG	0.98H.2	R/W	SIO no wait flag
SIOWRQ1	FLG	0.98H.1	R/W	SIO wait request flag
SIOWRQ0	FLG	0.98H.0	R/W	SIO wait request flag
IEGVSYN	FLG	0.9FH.2	R/W	Vsync interrupt edge select flag
IEG	FLG	0.9FH.0	R/W	RMC interrupt edge select flag
ADCCH2	FLG	0.0A1H.3	R/W	A/D converter channel select flag
ADCCH1	FLG	0.0A1H.2	R/W	A/D converter channel select flag
ADCCH0	FLG	0.0A1H.1	R/W	A/D converter channel select flag
ADCCMP	FLG	0.0A1H.0	R	A/D converter judge flag
P0CGI0	FLG	0.0A7H.0	R/W	Port 0C I/O select flag
SIOSF8	FLG	0.0A8H.3	R/W	SIO shift 8 clock flag

Reserved Word	Type	Address	Read/ Write	Function Outline
SIOSF9	FLG	0.0A8H.2	R/W	SIO shift 9 clock flag
SBSTT	FLG	0.0A8H.1	R/W	Serial bus start test flag
SBBSY	FLG	0.0A8H.0	R/W	Serial bus busy flag
IPSIO	FLG	0.0AFH.3	R/W	SIO interrupt permission flag
IPVSYN	FLG	0.0AFH.2	R/W	Vsync interrupt permission flag
IPTM	FLG	0.0AFH.1	R/W	Timer interrupt permission flag
IP	FLG	0.0AFH.0	R/W	RMC interrupt permission flag
CROMBNK	FLG	0.0B0H.0	R/W	CROM bank select flag
IDCEN	FLG	0.0B1H.0	R/W	IDC enable flag
P2ABI03	FLG	0.0B4H.3	R/W	P2A3I/O select flag
P2ABI02	FLG	0.0B4H.2	R/W	P2A2I/O select flag
P2ABI01	FLG	0.0B4H.1	R/W	P2A1I/O select flag
P2ABI00	FLG	0.0B4H.0	R/W	P2A0I/O select flag
P1BBI03	FLG	0.0B5H.3	R/W	P1B3I/O select flag
P1BBI02	FLG	0.0B5H.2	R/W	P1B2I/O select flag
P1BBI01	FLG	0.0B5H.1	R/W	P1B1I/O select flag
P1BBI00	FLG	0.0B5H.0	R/W	P1B0I/O select flag
P0BBI03	FLG	0.0B6H.3	R/W	P0B3I/O select flag
P0BBI02	FLG	0.0B6H.2	R/W	P0B2I/O select flag
P0BBI01	FLG	0.0B6H.1	R/W	P0B1I/O select flag
P0BBI00	FLG	0.0B6H.0	R/W	P0B0I/O select flag
P0ABI03	FLG	0.0B7H.3	R/W	P0A3I/O select flag
P0ABI02	FLG	0.0B7H.2	R/W	P0A2I/O select flag
P0ABI01	FLG	0.0B7H.1	R/W	P0A1I/O select flag
P0ABI00	FLG	0.0B7H.0	R/W	P0A0I/O select flag
SIOIMD3	FLG	0.0B8H.3	R	SIO interrupt mode select flag (dummy: 0)
SIOIMD2	FLG	0.0B8H.2	R	SIO interrupt mode select flag (dummy: 0)
SIOIMD1	FLG	0.0B8H.1	R/W	SIO interrupt mode select flag
SIOIMD0	FLG	0.0B8H.0	R/W	SIO interrupt mode select flag
SIOCK3	FLG	0.0B9H.3	R	SIO shift clock select flag (dummy: 0)
SIOCK2	FLG	0.0B9H.2	R	SIO shift clock select flag (dummy: 0)
SIOCK1	FLG	0.0B9H.1	R/W	SIO shift clock select flag
SIOCK0	FLG	0.0B9H.0	R/W	SIO shift clock select flag
IRQSIO	FLG	0.0BFH.3	R/W	SIO interrupt request flag
IRQVSYN	FLG	0.0BFH.2	R/W	Vsync interrupt request flag
IRQTM	FLG	0.0BFH.1	R/W	Timer interrupt request flag
IRQ	FLG	0.0BFH.0	R/W	RMC interrupt request flag

Remarks: Dummy is "0".

23.5 PERIPHERAL HARDWARE ADDRESS

Reserved Word	Type	Address	Read/ Write	Function Outline
DBF	DAT	0FH	R/W	GET/PUT instruction data buffer address
IX	DAT	01H	R/W	INC instruction index register address
IDCORG	DAT	01H	R/W	IDC start position setting register
ADCR	DAT	02H	R/W	A/D converter V_{REF} data register
SIOSFR	DAT	03H	R/W	SIO presettable shift register
HSC	DAT	04H	R/W	Hsync counter data register
PWMR0	DAT	05H	R/W	PWM data register 0
PWMR1	DAT	06H	R/W	PWM data register 1
PWMR2	DAT	07H	R/W	PWM data register 2
PWMR3	DAT	08H	R/W	PWM data register 3
AR	DAT	40H	R/W	GET/PUT/PUSH/CALL/BR/MOVT/INC instruction address register address
PWMRMP	DAT	41H	R/W	PWMRMP data register

24. ELECTRICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS

Power Supply Voltage	V_{DD}		−0.3 to +6.0	V
Input Voltage	V_I		−0.3 to V_{DD}	V
Output Voltage	V_O	Except P1A, P2B, P2C and PWM	−0.3 to V_{DD}	V
Output Current High	I_{OH}	1 pin	−12	mA
		All pins	−20	mA
Output Current Low	I_{OL1}	1 pin (except P1A)	12	mA
		All pins (except P1A)	20	mA
Output Current Low	I_{OL2}	1 pin (P1A only)	17	mA
		All pins (P1A only)	60	mA
Output Withstand Voltage	V_{BDS}	P1A, P2B, P2C, PWM	13	V
Operating Temperature	T_{opt}		−20 to +70	°C
Storage Temperature	T_{stg}		−55 to +125	°C

RECOMMENDED OPERATING CONDITION

CHARACTERISTICS	SYMBOL	MIN.	TYP.	MAX.	UNIT	CONDITIONS
Power Supply Voltage	V_{DD1}	4.5	5.0	5.5	V	All functional operations
Power Supply Voltage	V_{DD2}	4.0	5.0	5.5	V	Only IDC stop
Data Retention Voltage	V_{DR}	2.2		5.5	V	Clock oscillation stop
Output Withstand Voltage	V_{BDS}			12.5	V	P1A, P2B, P2C, PWM
Power Supply Voltage Rise Time	t_{rise}			500	ms	V_{DD}: 0 → 4.0 V

DC CHARACTERISTICS (T_a = –20 to +70 °C, V_{DD} = 4.0 to 5.5 V)

CHARACTERISTICS	SYMBOL	MIN.	TYP.	MAX.	UNIT	CONDITIONS
Supply Current	I_{DD1}		7	15	mA	CPU and IDC operations V_{DD}=5.5 V
Supply Current	I_{DD2}		3.5	15	mA	CPU operation, IDC stop V_{DD}=5.5 V
Input Voltage High	V_{IH1}	0.7 V_{DD}		V_{DD}	V	P0A, P0B, P0D, P1B, P1C, P2A
Input Voltage High	V_{IH2}	0.8 V_{DD}		V_{DD}	V	CE, RMC, $\overline{V_{SYNC}}$, $\overline{H_{SYNC}}$
Input Voltage Low	V_{IL1}	0		0.3 V_{DD}	V	P0A, P0B, P0D, P1B, P1C, P2A
Input Voltage Low	V_{IL2}	0		0.2 V_{DD}	V	CE, RMC, $\overline{V_{SYNC}}$, $\overline{H_{SYNC}}$
Output Current High	I_{OH}	–1	–2		mA	$P0A_2$, $P0A_3$, P0B, P0C, P1B, P1C, P1D, RED, GREEN, BLUE, BLANK V_{OH}=V_{DD}–1 V
Output Current Low	I_{OL1}	2	3		mA	P0A, P0B, P0C, P1B, P1C, P1D, RED, GREEN, BLUE, BLANK V_{OL}=1 V
Output Current Low	I_{OL2}	15	20		mA	P1A V_{OL}=1 V
Output Current Low	I_{OL3}	1	2		mA	P2B, P2C, PWM V_{OL}=1 V
Input Current High	I_{IH}		50		μA	When P0D is pulled down V_{IH}=V_{DD}
Data Retention Current	I_{DR}			10	μA	Clock oscillation stop T_a=25 °C, V_{DD}=5.5 V
Output Leakage	I_L			1	μA	$P0A_0$, $P0A_1$, P1A, P2B, P2C, PWM V_{OH}=5 V

AC CHARACTERISTICS (T_a = –20 to +70 °C, V_{DD} = 4.0 to 5.5 V)

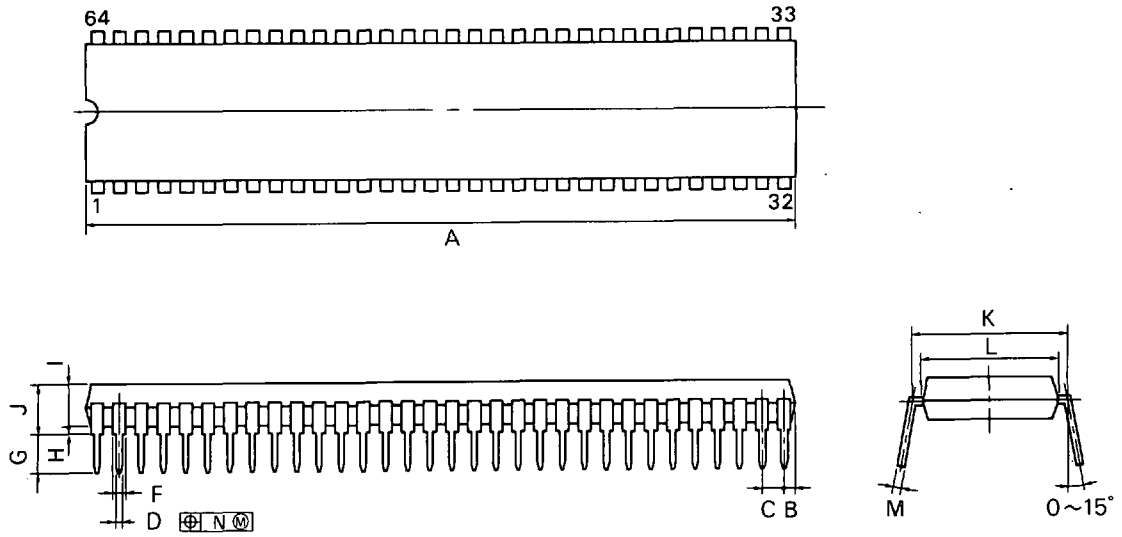
CHARACTERISTICS	SYMBOL	MIN.	TYP.	MAX.	UNIT	CONDITIONS
Input Frequency	f_{TMIN}	50		60	Hz	$P1B_3$/TMIN
Input Frequency	f_{HS}	10		20	kHz	$P0B_3$/HSCNT
IDC Jitter	IDC_G		3	4	ns	V_{DD}=4.5 to 5.5 V

A/D CONVERTER CHARACTERISTICS (T_a = –20 to +70 °C, V_{DD} = 4.0 to 5.5 V)

CHARACTERISTICS	SYMBOL	MIN.	TYP.	MAX.	UNIT	CONDITIONS
A/D Conversion Total Error		±1/2		±1	LSB	
A/D Input Impedance		1			MΩ	

25. PACKAGE DIMENSION

64PIN PLASTIC SHRINK DIP (750 mil)



P64C-70-750A,C

NOTES

1) Each lead centerline is located within 0.17 mm (0.007 inch) of its true position (T.P.) at maximum material condition.

2) Item "K" to center of leads when formed parallel.

ITEM	MILLIMETERS	INCHES
A	58.68 MAX.	2.311 MAX.
B	1.78 MAX.	0.070 MAX.
C	1.778 (T.P.)	0.070 (T.P.)
D	$0.50^{\pm 0.10}$	$0.020^{+0.004}_{-0.005}$
F	0.9 MIN.	0.035 MIN.
G	$3.2^{\pm 0.3}$	$0.126^{\pm 0.012}$
H	0.51 MIN.	0.020 MIN.
I	4.31 MAX.	0.170 MAX.
J	5.08 MAX.	0.200 MAX.
K	19.05 (T.P.)	0.750 (T.P.)
L	17.0	0.669
M	$0.25^{+0.10}_{-0.05}$	$0.010^{+0.004}_{-0.003}$
N	0.17	0.007

26. RECOMMENDED SOLDERING CONDITIONS

The following conditions (see table below) must be met when soldering this product. Please consult with our sales offices in case other soldering process is used, or in case soldering is done under different conditions.

TYPES OF SURFACE MOUNT DEVICE

For more details, refer to our document "SMT MANUAL" (IEI-1207).

μPD17052CW-XXX

Soldering process	Soldering conditions	Symbol
Wave soldering	Solder temperature : 260 °C or below, Flow time : 10 seconds or below, Number of flow process : 1, Exposure limit* : None	WS60-00
Partial heating method	Terminal temperature : 300 °C or below, Flow time : 10 seconds or below, Exposure limit* : None	

*: Exposure limit before soldering after dry-pack package is opened.
Storage conditions : 25 °C and relative humidity at 65% or less.

Note: Do not apply more than a single process at once, except for "Partial heating method".

APPENDIX. DEVELOPMENT TOOLS

The following support tools are available for μPD17052 program development.

Hardware		
Name	Description	Ordering Code
Incircuit emulator (IE-17K)	IE-7K is an incircuit emulator for evaluation for common use with the 17K series. The IE-17K is used with the SE-17052 system evaluation board for μPD17052 program development. The IE-17K is operated based on the RAM. Simply connecting a console to the IE-17K enables to add or correct the program instantaneously on the console. Operating the SIMPLEHOST™ support software provides more advanced program development environment.	IE-17K
SE board (SE-17052)	SE-17052 is a μPD17052 system evaluation board which is used independently or jointly with the IE-17K.	SE-17052
Probe (EP-17052CW)	EP-17052CW is a probe to connect the SE-17052 to the target system.	EP-17052CW

Software					
Name		Description	Host Machine	OS	Ordering Code
Assembler	Main assembler unit (AS17K)	AS17K is a main assembler unit for common use with the 17K series. This AS17K is used with a device file AS17052.	PC-9801 series IBM PC/AT™	MS-DOS™ Ver. 2.11 Ver 3.1 PC DOS™ Ver. 3.1	MS-DOS version μS5A1AS17K (8-inch 2D) μS5A10AS17K (5-inch 2HD) PS DOS version μS7B11AS17K (5-inch 2D)
	Device file (AS17052)	AS17052 is used with the AS17K to assemble the μPD17052 program.			MS-DOS version μS5A1AS17052 (8-inch 2D) μS5A10AS17052 (5-inch 2HD) PC DOS version μS7B11AS17052 (5-inch 2D)
Support software (SIMPLEHOST)		SIMPLEHOST is software to provide a man-machine interface in MS-WINDOWS™ for program development using the IE-17K and a personal computer.		MS-WINDOWS	----

MS-DOS™ and MS-WINDOWS™ are trademarks of Microsoft Corporation.
IBM PC/AT™ and PC DOS™ are trademarks of IBM Corporation.
SIMPLEHOST™ is trademark of NEC Corporation.

(MEMO)