CHAPTER 1 GENERAL

The µPD750104, µPD750106, µPD750108, and µPD75P0116 are 75XL series 4-bit single-chip microcomputers. The 75XL series is a successor of the 75X series consisting of many products. These µPD750104, µPD750106, µPD750108, and µPD75P0116 are collectively called the µPD750108 subseries.

The 75XL series takes over the CPUs of the 75X series, realizing a wide range of operating voltages. In addition to having upward compatibility with existing products, the 75XL series is best suited for battery-driven applications.

The µPD750104, µPD750106, µPD750108, and µPD75P0116 have the following features:

- Built-in RC oscillator for main system clock oscillation, enabling the immediate start of processing after the release of standby mode.
- Operable on low voltage: V_{DD} = 1.8 to 5.5 V
- Switchable instruction execution times (useful for power saving)
  4, 8, 16, 64 µs (at 1 MHz)
  2, 4, 8, 32 µs (at 2 MHz)
  122 µs (at 32.768 kHz)
- Enhanced timers: 4 channels
- Easy replacement (The functions and instructions of the µPD75008 and µPD750008 are taken over.)

The 75XL series comes in four models, according to the size and type of program memory (see **Table 1-1**).

Table 1-1. Features of the Products

Model	Program memory (ROM)	Remarks
µPD750104	4096 x 8 bits	Masked ROM
µPD750106	6144 x 8 bits	
µPD750108	8192 x 8 bits	
µPD75P0116	16384 x 8 bits	One-time PROM

The µPD75P0116, having the electrically programmable one-time PROM, is pin-compatible with the µPD750104, µPD750106, and µPD750108. It is suitable for small-scale production or prototype production in system development.

Applications

- Camera
- Meter
- Automobile
- Pager

Remark This manual will explain only the μPD750108 when the μPD750108, μPD750104, μPD750106, and μPD75P0116 are functionally the same. Users of the μPD750104, μPD750106, or μPD75P0116 should read μPD750108 as referring to μPD750104, μPD750106, or μPD75P0116.

1.1 FUNCTION OVERVIEW

<table>
<tr><th colspan="2">Item</th><th colspan="4">Function</th></tr>
<tr><td colspan="2">Instruction execution time</td><td colspan="4">• 4, 8, 16, 64 µs (when the main system clock operates at 1 MHz)
• 2, 4, 8, 32 µs (when the main system clock operates at 2 MHz)
• 122 µs (when the subsystem clock operates at 32.768 kHz)</td></tr>
<tr><td rowspan="5">Internal memory</td><td rowspan="4">ROM</td><td colspan="4">4096 x 8 bits (µPD750104)</td></tr>
<tr><td colspan="4">6144 x 8 bits (µPD750106)</td></tr>
<tr><td colspan="4">8192 x 8 bits (µPD750108)</td></tr>
<tr><td colspan="4">16384 x 8 bits (µPD75P0116)</td></tr>
<tr><td>RAM</td><td colspan="4">512 x 4 bits</td></tr>
<tr><td colspan="2">General register</td><td colspan="4">• When operating in 4 bits: 8 x 4 banks
• When operating in 8 bits: 4 x 4 banks</td></tr>
<tr><td colspan="2" rowspan="3">I/O port</td><td rowspan="3">34</td><td>8</td><td>CMOS input pins</td><td rowspan="2">Can incorporate 25 pull-up resistors that are specified with the software.</td></tr>
<tr><td>18</td><td>CMOS I/O pins
Four pins can directly drive the LED.</td></tr>
<tr><td>8</td><td>N-ch open-drain I/O pins
Eight pins can directly drive the LED.</td><td>Can withstand 13 V.
Can incorporate pull-up resistors that are specified with the mask option.Note</td></tr>
<tr><td colspan="2">Timer</td><td>4</td><td colspan="3">• Timer/event counter: 1 channel
• Timer counter: 1 channel
• Basic interval timer/watchdog timer: 1 channel
• Clock timer: 1 channel</td></tr>
<tr><td colspan="2">Serial interface</td><td colspan="4">• Three-wire serial I/O mode (switchable between the start LSB and the start MSB)
• Two-wire serial I/O mode
• SBI mode</td></tr>
<tr><td colspan="2">Bit sequential buffer</td><td colspan="4">16 bits</td></tr>
<tr><td colspan="2">Clock output</td><td colspan="4">• Φ, 125, 62.5, 15.6 kHz (when the main system clock operates at 1 MHz)
• Φ, 250, 125, 31.3 kHz (when the main system clock operates at 2 MHz)</td></tr>
<tr><td colspan="2">Vectored interrupt</td><td colspan="4">External: 3, Internal: 4</td></tr>
<tr><td colspan="2">Test input</td><td colspan="4">External: 1, Internal: 1</td></tr>
<tr><td colspan="2">System clock oscillator</td><td colspan="4">• RC oscillator for the main system clock
• Crystal oscillator for the subsystem clock</td></tr>
<tr><td colspan="2">Standby function</td><td colspan="4">STOP/HALT mode</td></tr>
<tr><td colspan="2">Operating ambient temperature</td><td colspan="4">T_A = –40˚C to +85˚C</td></tr>
<tr><td colspan="2">Supply voltage</td><td colspan="4">V_{DD} = 1.8 to 5.5 V</td></tr>
<tr><td colspan="2">Package</td><td colspan="4">42-pin plastic shrink DIP (600 mil)
44-pin plastic QFP (10 x 10 mm)</td></tr>
</table>

Note The N-ch open-drain I/O port pins of the µPD75P0116 are not connected to pull-up resistors by mask option, and are always open.

1.2 ORDERING INFORMATION

Part number	Package	On-chip ROM
μPD750104CU-xxx	42-pin plastic shrink DIP (600 mil)	Masked ROM
μPD750104GB-xxx-3BS-MTX	44-pin plastic QFP (10 x 10 mm)	Masked ROM
μPD750106CU-xxx	42-pin plastic shrink DIP (600 mil)	Masked ROM
μPD750106GB-xxx-3BS-MTX	44-pin plastic QFP (10 x 10 mm)	Masked ROM
μPD750108CU-xxx	42-pin plastic shrink DIP (600 mil)	Masked ROM
μPD750108GB-xxx-3BS-MTX	44-pin plastic QFP (10 x 10 mm)	Masked ROM
μPD75P0116CU	42-pin plastic shrink DIP (600 mil)	One-time PROM
μPD75P0116GB-3BS-MTX	44-pin plastic QFP (10 x 10 mm)	One-time PROM

Remark xxx is a ROM code number.

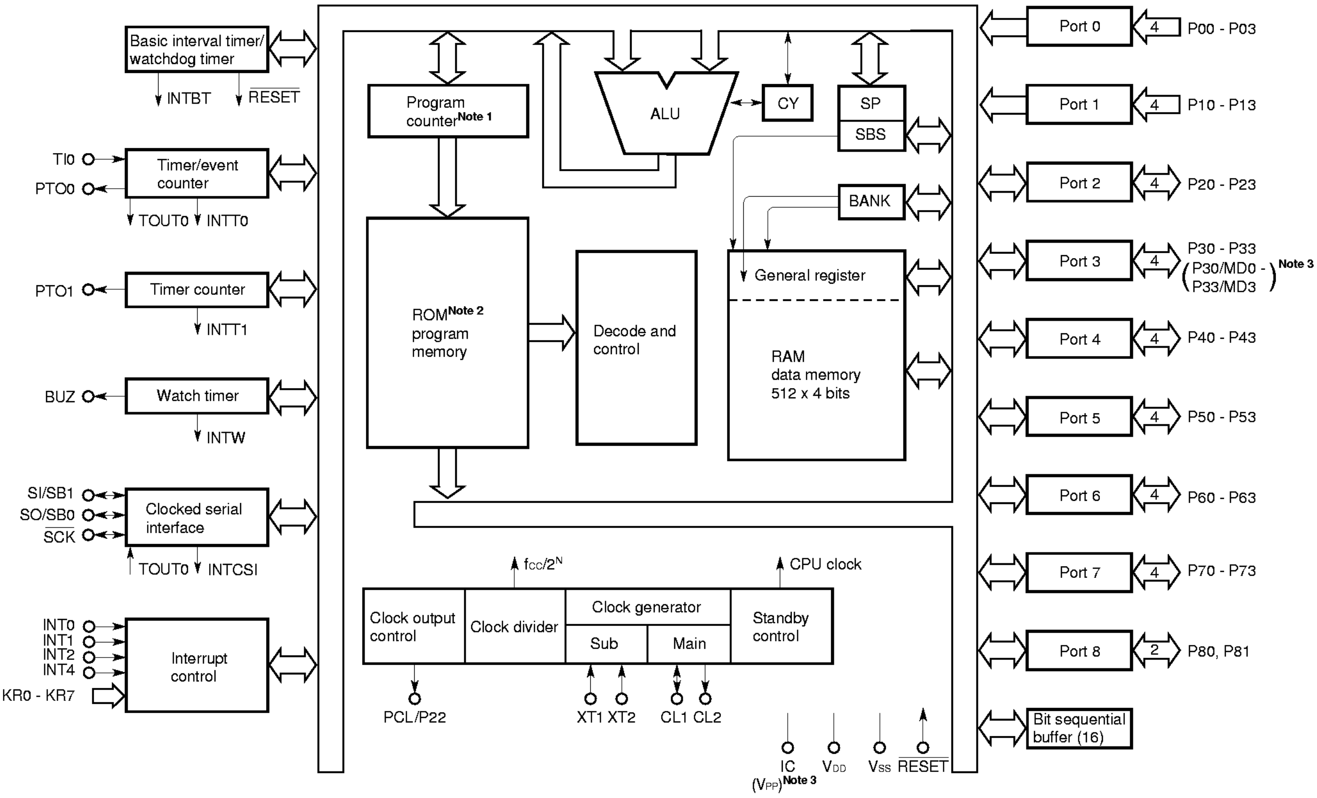
1.3 DIFFERENCES AMONG SUBSERIES PRODUCTS

Item		µPD750104	µPD750106	µPD750108	µPD75P0116
Program counter		12 bits	13 bits		14 bits
Program memory (byte)		Masked ROM 4096	Masked ROM 6144	Masked ROM 8192	One-time PROM 16384
Data memory (x 4 bits)		512			
Mask option	Pull-up resistors at ports 4 and 5	Incorporated (Whether to incorporate pull-up resistors can be specified.)			None (Cannot be incorporated.)
	Wait time applied when STOP mode is released by an interrupt	Available ($2^9/f_{CC}$ or no wait)Note			Not available (Fixed to $2^9/f_{CC}$.)Note
	Selection to use feedback resistors for subsystem clock	Yes (Whether to enable feedback resistors can be specified.)			No (Use of feedback resistors is factory-set)
Pin connection	6-9 (CU)	P33-P30			P33/MD3-P30/MD0
	23-26 (GB)				
	20 (CU)	IC			V_{PP}
	38 (GB)				
Others		Noise immunity and noise radiation vary with the circuit scale and mask layout.			

Note $2^9/f_{CC}$ (256 µs at 2 MHz, 512 µs at 1 MHz)

Caution **The noise immunity and noise radiation of the PROM model differ from those of the mask ROM model. If you replace the PROM model with the ROM model of the course of experimental production to mass production, perform thorough evaluation by using the CS model (not ES model) of the mask ROM model.**

1.4 BLOCK DIAGRAM



Notes 1. The program counter for the µPD750104 consists of 12 bits, 13 bits for the µPD750106 and µPD750108, and 14 bits for the µPD75P0116.

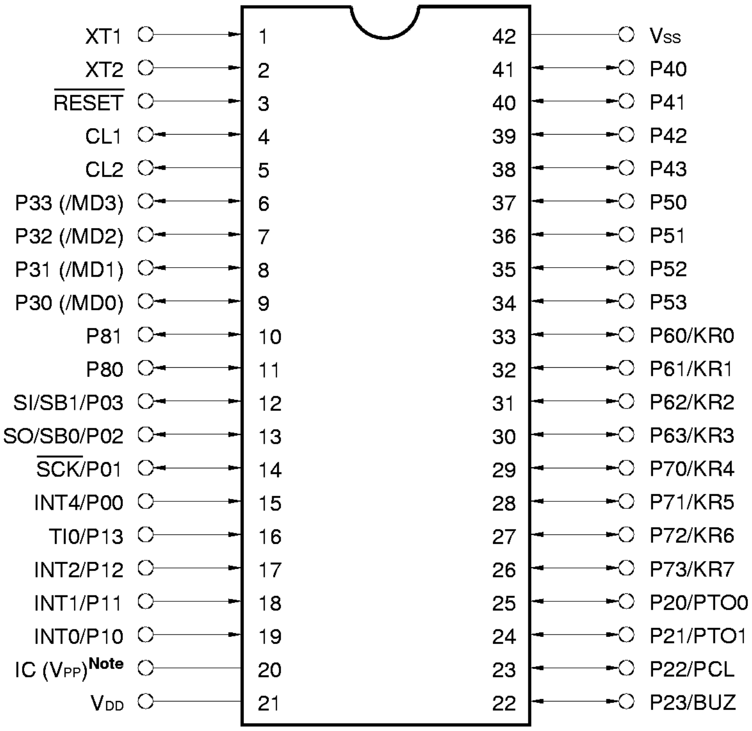
2. The ROM capacity depends on the product.

3. ( ) : µPD75P0116

1.5 PIN CONFIGURATION (TOP VIEW)

(1) 42-pin plastic shrink DIP (600 mil)

µPD750104CU-XXX
µPD750106CU-XXX
µPD750108CU-XXX
µPD75P0116CU

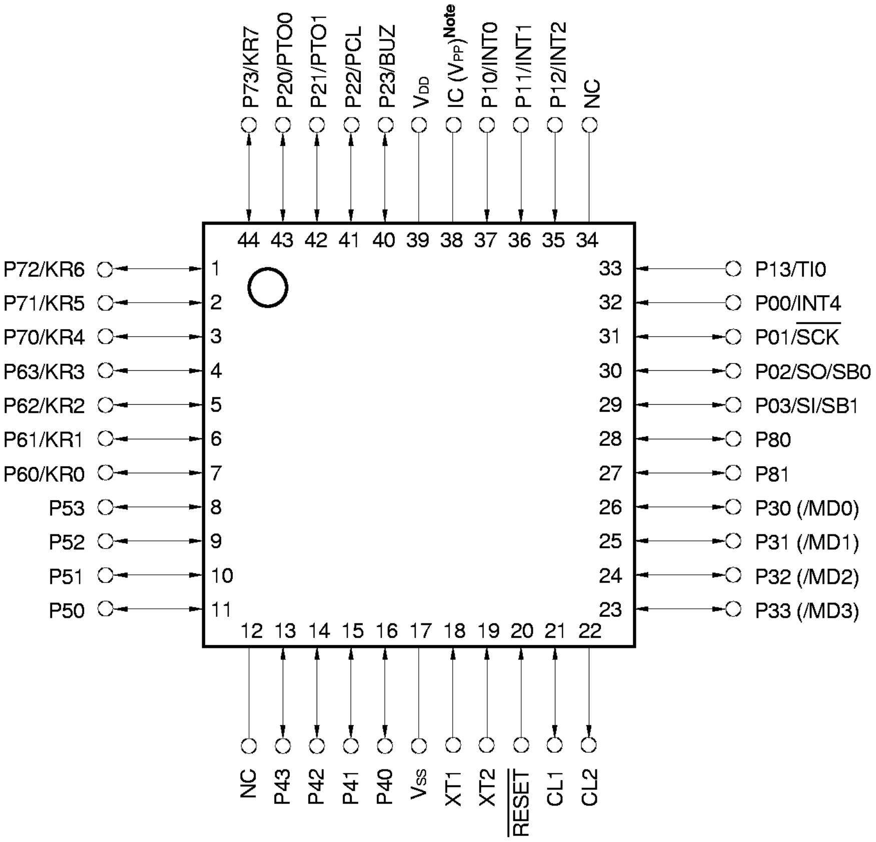


Note Connect IC (V_{PP}) to V_{DD}, keeping the wiring as short as possible.

Remark ( ) : µPD75P0116.

(2) 44-pin plastic QFP (10 x 10 mm)

μPD750104GB-XXX-3BS-MTX
μPD750106GB-XXX-3BS-MTX
μPD750108GB-XXX-3BS-MTX
μPD75P0116GB-3BS-MTX



Note Connect IC (V_{PP}) to V_{DD}, keeping the wiring as short as possible.

Remark ( ) : μPD75P0116.

Pin name

P00-P03	:	Port 0
P10-P13	:	Port 1
P20-P23	:	Port 2
P30-P33	:	Port 3
P40-P43	:	Port 4
P50-P53	:	Port 5
P60-P63	:	Port 6
P70-P73	:	Port 7
P80-P81	:	Port 8
KR0-KR7	:	Key return 0-7
$\overline{\text{SCK}}$	:	Serial clock
SI	:	Serial input
SO	:	Serial output
SB0, 1	:	Serial bus 0, 1

$\overline{\text{RESET}}$	:	Reset input
TI0	:	Timer input 0
PTO0, 1	:	Programmable timer output 0, 1
BUZ	:	Buzzer clock
PCL	:	Programmable clock
INT0, 1, 4	:	External vectored interrupt 0, 1, 4
INT2	:	External test input 2
CL1, 2	:	RC oscillator 1, 2
XT1, 2	:	Subsystem clock oscillation 1, 2
NC	:	No connection
IC	:	Internally connected
V_{DD}	:	Positive power supply
V_{SS}	:	Ground
V_{PP}	:	Programming power supply
MD0-MD3	:	Mode selection 0 - 3

[MEMO]

CHAPTER 2 PIN FUNCTIONS

2.1 PIN FUNCTIONS OF THE µPD750108

Table 2-1. Digital I/O Port Pins (1/2)

Pin	Input/ output	Also used as	Function	8 bit I/O	Upon reset	I/O circuit type[Note 1]
P00	Input	INT4	4-bit input port (PORT0). For P01 to P03, built-in pull-up resistors can be connected by software in units of 3 bits.	x	Input	Ⓑ
P01	I/O	$\overline{\text{SCK}}$				Ⓕ-A
P02	I/O	SO/SB0				Ⓕ-B
P03	I/O	SI/SB1				Ⓜ-C
P10	Input	INT0	4-bit input port (PORT1). Built-in pull-up resistors can be connected by software in units of 4 bits. Only the P10/INT0 pin is provided with noise elimination function.	x	Input	Ⓑ-C
P11		INT1				
P12		INT2				
P13		TI0				
P20	I/O	PTO0	4-bit I/O port (PORT2). Built-in pull-up resistors can be connected by software in units of 4 bits.	x	Input	E-B
P21		PTO1				
P22		PCL				
P23		BUZ				
P30[Note 2]	I/O	(MD0)[Note 3]	Programmable 4-bit I/O port (PORT3). I/O can be specified bit by bit. Built-in pull-up resistors can be connected by software in units of 4 bits.	x	Input	E-B
P31[Note 2]		(MD1)[Note 3]				
P32[Note 2]		(MD2)[Note 3]				
P33[Note 2]		(MD3)[Note 3]				

Notes 1. I/O circuits enclosed in circles have a Schmitt-triggered input.

2. An LED can be driven directly.

3. ( ): µPD75P0116

Table 2-1. Digital I/O Port Pins (2/2)

Pin	Input output	Also used as	Function	8 bit I/O	Upon reset	I/O circuit type[Note 1]
P40- P43[Note 2, 4]	I/O	—	N-ch open-drain 4-bit I/O port (PORT4). Withstand voltage is 13 V in open-drain mode. A pull-up resistor can be provided bit by bit (mask option)[Note 5]. Data input/output pins for writing/ verifying (lower 4 bits) of program memory (PROM).	O	High level (when a pull-up resistor is provided) or high impedance	M-D (M-E)[Note 3]
P50- P53[Note 2, 4]	I/O	—	N-ch open-drain 4-bit I/O port (PORT5). Withstand voltage is 13 V in open-drain mode. A pull-up resistor can be provided bit by bit (mask option)[Note 5]. Data input/output pins for writing/ verifying (higher 4 bits) of program memory (PROM).	O	High level (when a pull-up resistor is provided) or high impedance	M-D (M-E)[Note 3]
P60	I/O	KR0	Programmable 4-bit I/O port (PORT6). I/O can be specified bit by bit. Built-in pull-up resistors can be connected by software in units of 4 bits.	O	Input	Ⓕ-A
P61		KR1				
P62		KR2				
P63		KR3				
P70	I/O	KR4	4-bit I/O port (PORT7). Built-in pull-up resistors can be connected by software in units of 4 bits.		Input	Ⓕ-A
P71		KR5				
P72		KR6				
P73		KR7				
P80	I/O	—	2-bit input port (PORT8). Built-in pull-up resistors can be connected by software in units of 2 bits.	x	Input	E-B
P81		—				

Notes 1. I/O circuits enclosed in circles have a Schmitt-triggered input.

2. An LED can be driven directly.

3. ( ): µPD75P0116

4. When pull-up resistors that can be specified with the mask option are not incorporated (when pins are used as N-ch open-drain input ports), the input leak low current increases when an input instruction or bit operation instruction is executed.

5. These pins of the µPD75P0116 are not provided with pull-up resistors by mask option, and are always open.

Table 2-2. Non-Port Pin Functions

Pin	Input/ output	Also used as	Function		Upon reset	I/O circuit type[Note 1]
TI0	Input	P13	Inputs external event pulse to the timer/event counter		—	Ⓑ-C
PTO0	I/O	P20	Timer/event counter output		Input	E-B
PTO1		P21	Timer counter output			
PCL	I/O	P22	Clock output		Input	E-B
BUZ	I/O	P23	Fixed frequency output (for buzzer or system clock trimming)		Input	E-B
$\overline{\text{SCK}}$	I/O	P01	Serial clock I/O		Input	Ⓕ-A
SO/SB0	I/O	P02	Serial data output or serial data bus I/O		Input	Ⓕ-B
SI/SB1	I/O	P03	Serial data input or serial data bus I/O		Input	Ⓜ-C
INT4	Input	P00	Edge detection vectored interrupt input (Either a rising or falling edge is detected.) The INT0/P10 pin has a noise eliminating function.		—	Ⓑ
INT0	Input	P10	Edge detection vectored interrupt input	Synchronous	—	Ⓑ-C
INT1		P11	(The edge to be detected is selectable.) INT0 and P10 are provided with the noise removing function.	Asynchronous		
INT2	Input	P12	Rising edge detection testable input	Asynchronous	—	Ⓑ-C
KR0-KR3	I/O	P60-P63	Parallel falling edge detection testable input		Input	Ⓕ-A
KR4-KR7	I/O	P70-P73	Parallel falling edge detection testable input		Input	Ⓕ-A
CL1	I/O	—	Connection pin to R and C for main system clock generation.		—	—
CL2	Output					
XT1, XT2	Input	—	Connection pin to a crystal for subsystem clock generation. When external clock is used, it is input to XT1, and its inverted signal is input to XT2.		—	—
$\overline{\text{RESET}}$	Input	—	System reset input		—	Ⓑ
IC[Note 2]	—	—	Internally connected. Connect to V_{DD}, keeping the wiring as short as possible.		—	—
V_{DD}	—	—	Positive power supply		—	—
V_{SS}	—	—	GND potential		—	—
V_{PP}[Note 3]	—	—	Program voltage application for program memory (PROM) write/verify operation. +12.5 V is applied for PROM write/verify operation. Connect to V_{DD}, keeping the wiring as short as possible.		—	—
MD0-MD3[Note 3]	I/O	P30-P33	Mode selection for program memory (PROM) write/verify operation.		Input	E-B
NC	—	—	No connection		—	—

Notes 1. The circuits enclosed in circles have a Schmitt-triggered input.

2. Used as the V_{PP} pin for the µPD75P0116.

3. Provided only in the µPD75P0116.

2.2 PIN FUNCTIONS

2.2.1 P00-P03 (PORT0) : Input Pins Used Also for INT4, $\overline{\text{SCK}}$, SO/SB0 and SI/SB1
P10-P13 (PORT1) : Input Pins Used Also for INT0-INT2, and TI0

These are the input pins of the 4-bit input ports: Ports 0 and 1.

Ports 0 and 1 function as input ports, and also have the functions described below.

(1) Port 0 : Vectored interrupt input (INT4)
Serial interface I/O ($\overline{\text{SCK}}$, SO/SB0, SI/SB1)

(2) Port 1 : Vectored interrupt input (INT0, INT1)
Edge detection test input (INT2)
External event pulse input (TI0) for timer/event counter

Input is always enabled for each pin of ports 0 and 1 regardless of the operation status of the other function of the pin.

Schmitt-triggered inputs are used for the input pin of port 0 and pins of port 1 to prevent malfunction due to noise. In addition, a noise eliminator is provided for P10. (See **(3)** of **Section 6.3**.)

Port 0 can be connected with built-in pull-up resistors in units of 3 bits (P01 to P03) by software. Port 1 can be connected with built-in pull-up resistors in units of 4 bits (P10 to P13) by software. This is done by manipulating pull-up resistor specification register group A (POGA).

A $\overline{\text{RESET}}$ signal input places these pins in the input port mode.

2.2.2 P20-P23 (PORT2) : I/O Pins Used Also for PTO0, PTO1, PCL, and BUZ
P30-P33 (PORT3) : I/O Pins Used Also for MD0-MD3Note
P40-P43 (PORT4),
P50-P53 (PORT5) : N-ch Open-Drain Intermediate Withstand Voltage (13 V) Large-Current Output
P60-P63 (PORT6),
P70-P73 (PORT7) : Tristate I/O

These pins are the I/O pins of the 4-bit I/O ports with output latches: Ports 2 to 7.

Port n (n = 2, 3, 6, and 7) functions as I/O ports, and also have the following functions:

(1) Port 2 : Timer/event counter (PTO0, PTO1)
Timer counter (PTO1)
Clock output (PCL)
Fixed frequency output (BUZ)

(2) Port 3 : Mode selection for program memory (PROM) write/verify operation (MD0-MD3)Note

(3) Ports 6 and 7: Key interrupt input (KR0-KR3, KR4-KR7)

Note Provided only in the µPD75P0116.

Port 3 is a large-current output. Ports 4 and 5 are N-ch open-drain intermediate withstand voltage (13 V) large-current output. These ports can directly drive the LED.

An I/O mode is selected by the port mode register. The I/O mode of port m (m = 2, 4, 5, and 7) can be selected in units of 4 bits, and the I/O mode of ports 3 and 6 can be selected bit by bit.

Port n can be connected with built-in pull-up resistors in units of 4 bits by software. This can be done by manipulating pull-up resistor specification register group A (POGA). For ports 4 and 5, the use of built-in pull-up resistors can be specified bit by bit by mask option.

Ports 4 and 5, and ports 6 and 7 can be paired respectively for 8-bit I/O.

A $\overline{\text{RESET}}$ input clears the output latches in the ports, places port n in the input mode (output high-impedance state), and drives ports 4 and 5 high if pull-up resistors are provided or causes ports 4 and 5 to go into a high-impedance state.

2.2.3 P80, P81 (PORT8)

These pins are the I/O pins of the 2-bit I/O ports with output latches: Port 8.

Port 8 can be connected with built-in pull-up resistors in units of 2 bits by software. This can be done by manipulating pull-up resistor specification register group B (POGB).

2.2.4 TI0: Input Pin Used Also for Port 1

This is an external event pulse input pin for the programmable timer/event counter.

A Schmitt-triggered input is used for the TI0 pin.

2.2.5 PTO0, PTO1: Output Pin Used Also for Port 2

This is the output signal pin of the programmable timer/event counter and programmable timer counter. Square-wave pulses appear on this pin. To output a signal from the programmable timer/event counter and programmable timer counter, the output latch P20 or P21 must be cleared to 0, and the bit for port 2 in the port mode register must be set to 1 (output mode).

The output is cleared to 0 by the timer start instruction.

2.2.6 PCL: Output Pin Used Also for Port 2

This is the programmable clock output pin. It is used to supply the clock pulse to a peripheral LSI circuit such as a slave microcomputer or A/D converter.

A $\overline{\text{RESET}}$ signal clears the clock mode register (CLOM) to 0, disabling clock output, then the pin is placed in the normal mode to function as a normal port.

2.2.7 BUZ: Output Pin Used Also for Port 2

An arbitrary frequency (2.048, 4.096, or 32.768 kHz when WM0 = 1, or 0.488, 0.977, or 7.8125 kHz during 1-MHz operation when WM0 = 0) output on this pin can be used for sounding the buzzer or trimming the system clock frequency. This pin is used also as the P23 pin, and can be used only when bit 7 (WM.7) of the clock mode register (WM) is set to 1.

A $\overline{\text{RESET}}$ signal places this pin in the normal operation mode as a general port (see **Section 5.4.2** for details).

2.2.8 $\overline{\text{SCK}}$, SO/SB0, SI/SB1: Tristate I/O Pins Used Also as Port 0

These are I/O pins for serial interface. They operate according to the setting of the serial operation mode registers (CSIM).

A $\overline{\text{RESET}}$ signal stops serial interface operation and places these pins in the input port mode.

A Schmitt-triggered input is used for each pin.

2.2.9 INT4: Input Pin Used Also as Port 0

INT4 is an external vectored interrupt input pin, which is rising edge active as well as falling edge active. When a signal applied to this pin goes from low to high or from high to low, the interrupt request flag is set.

INT4 is an asynchronous input, and can accept a signal with some high level width or low level width regardless of what the CPU clock is.

The INT4 pin can also be used to release the STOP and HALT modes. A Schmitt-triggered input is used for this pin.

2.2.10 INT0, INT1: Input Pins Used Also for Port 1

These are edge detection vectored interrupt input pins. INT0 has a noise eliminator. The edge to be detected can be selected using the edge detection mode registers (IM0, IM1).

(1) INT0 (bits 0 and 1 of IM0)

(a) Rising edge active
(b) Falling edge active
(c) Both rising and falling edges active
(d) External interrupt signal input disabled

(2) INT1 (bit 0 of IM1)

(a) Rising edge active
(b) Falling edge active

INT0 has a noise eliminator. Two different sampling clocks for noise elimination can be switched. The acceptable width of a signal depends on the CPU clock.

INT1 is an asynchronous input, and can accept a signal with some high level width regardless of what the CPU clock is.

A $\overline{\text{RESET}}$ input clears IM0 and IM1 to 0, selecting rising edge active.

The INT1 pin can also be used to release the STOP and HALT modes, but the INT0 pin cannot.

Schmitt-triggered inputs are used for the INT0 and INT1 pins.

2.2.11 INT2: Input Pin Used Also for Port 1

This is a rising edge active, external test input pin. When INT2 is selected with the edge detection mode register (IM2), or when the signal applied to this pin goes high, the internal test flag (IRQ2) is set.

INT2 is an asynchronous input, and can accept a signal with some high level width regardless of the operating clock of the CPU.

A $\overline{\text{RESET}}$ signal clears IM2 to 0. In this case, the test flag (IRQ2) is set by a rising edge on the INT2 pin.

The INT2 pin can also be used to release the STOP and HALT modes. A Schmitt-triggered input is used for this pin.

2.2.12 KR0-KR3: Input Pins Used Also for Port 6
KR4-KR7: Input Pins Used Also for Port 7

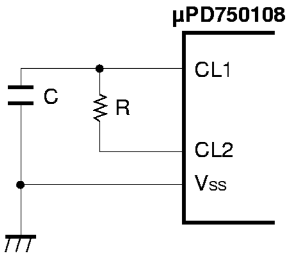
KR0 to KR7 are key interrupt input pins. An interrupt is caused when parallel falling edges are detected on them. The interrupt format can be specified with the edge detection mode register (IM2).

A $\overline{\text{RESET}}$ signal places these pins in the port 6 and 7 input modes.

2.2.13 CL1, CL2

These pins are used for connection to a resistor (R) and capacitor (C) for main system clock generation.

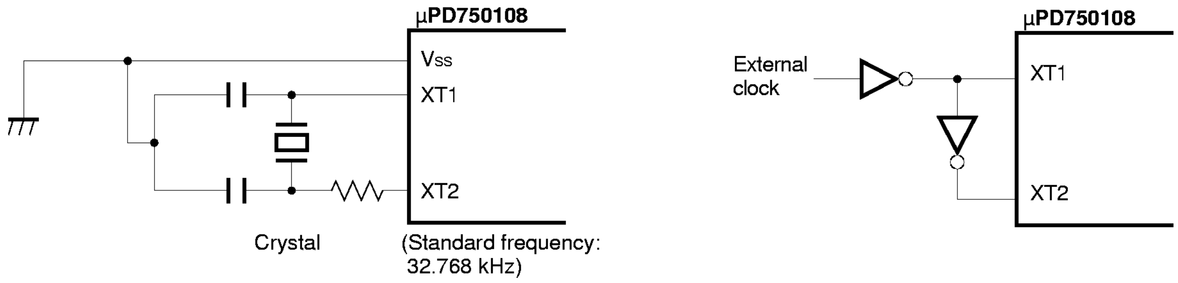
RC oscillation



2.2.14 XT1, XT2

These pins are used for connection to a crystal for subsystem clock oscillation.

An external clock can also be applied.



(a) Crystal oscillation

(b) External clock

2.2.15 $\overline{\text{RESET}}$

This is the pin for active-low reset input.

The $\overline{\text{RESET}}$ input is asynchronous. When a signal with certain low level width is applied to the pin, a $\overline{\text{RESET}}$ signal is generated to cause a system reset, which has priority over any other operations.

The $\overline{\text{RESET}}$ signal is used for normal CPU initialize/start operation, and is also used to release the standby (STOP or HALT) mode.

A Schmitt-triggered input is used for the $\overline{\text{RESET}}$ input pin.

2.2.16 V_{DD}

This is the positive power supply pin.

2.2.17 V_{SS}

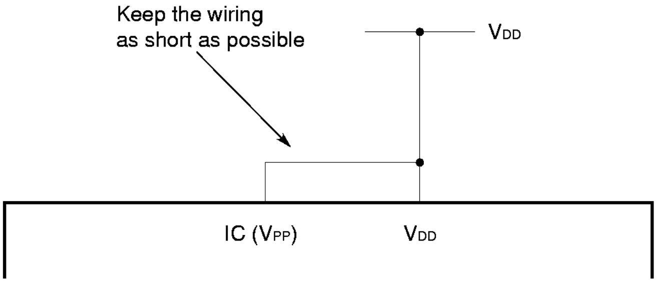
This is the ground pin.

2.2.18 IC (for the µPD750104, µPD750106, and µPD750108 only)

The internally connected (IC) pin is used to set the µPD750108 to test mode for inspection prior to shipping. In normal operation, connect the IC pin to the V_{DD} pin, keeping the writing as short as possible.

When the wiring between the IC pin and the V_{DD} pin is too long, or noise is generated on the IC pin, a potential difference may occur between the IC pin and the V_{DD} pin. This may cause your program to malfunction.

- Connect the IC pin to the V_{DD} pin, keeping the wiring as short as possible.



2.2.19 V_{PP} (for the µPD75P0116 only)

This is a program voltage input pin for program memory (PROM) write/verify operation.

For normal use, connect this pin to V_{DD}, keeping the wiring as short as possible (shown above).

+12.5 V is applied for PROM write/verify operation.

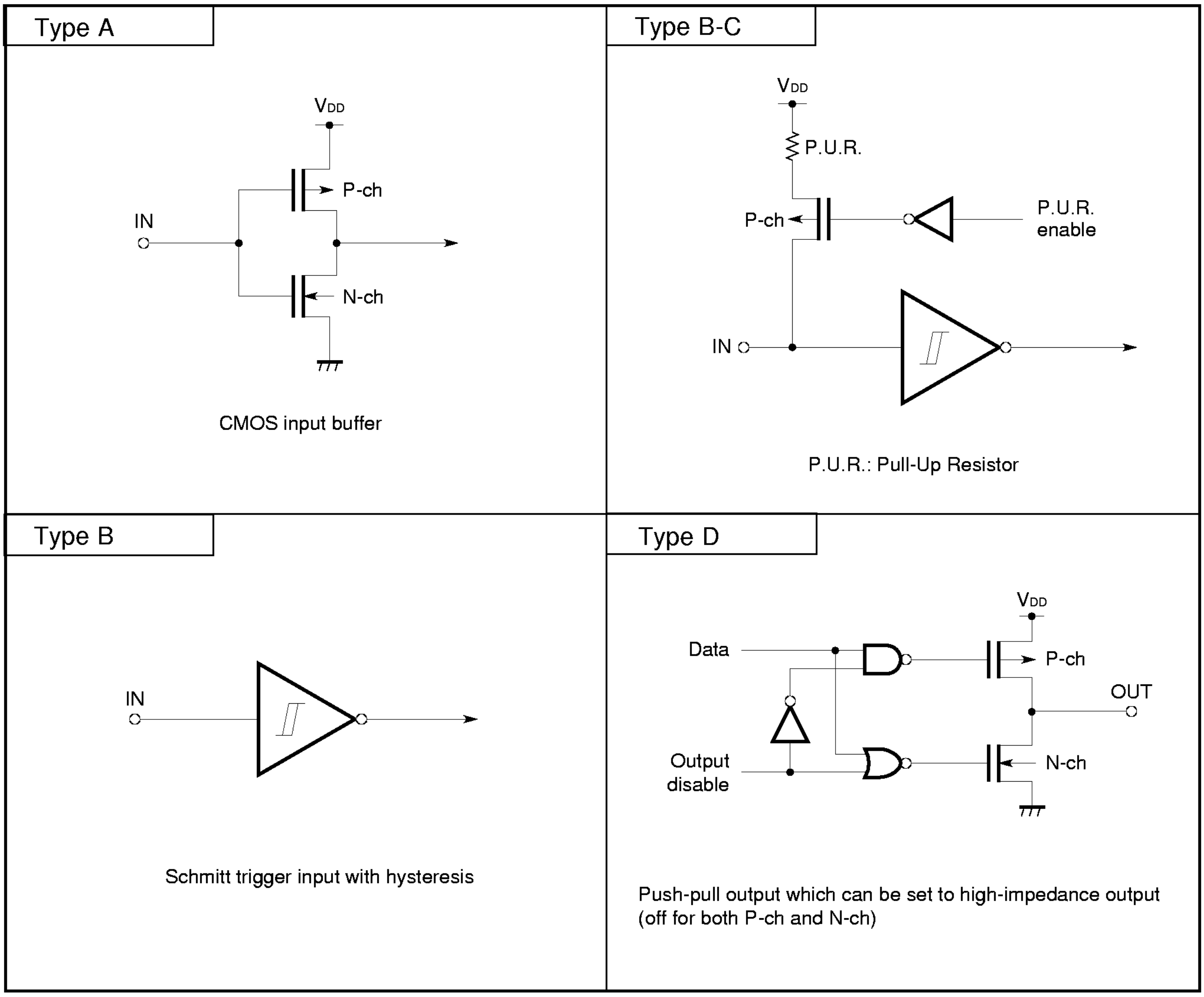
2.2.20 MD0-MD3 (for the µPD75P0116 only): I/O Pins Used Also for Port 3

MD0 to MD3 select a mode for program memory (PROM) write/verify operation.

2.3 PIN INPUT/OUTPUT CIRCUITS

Figure 2-1 shows schematic diagrams of the I/O circuitry of the μPD750108.

Figure 2-1. Pin Input/Output Circuits (1/2)



Type A

CMOS input buffer

Type B-C

P.U.R.: Pull-Up Resistor

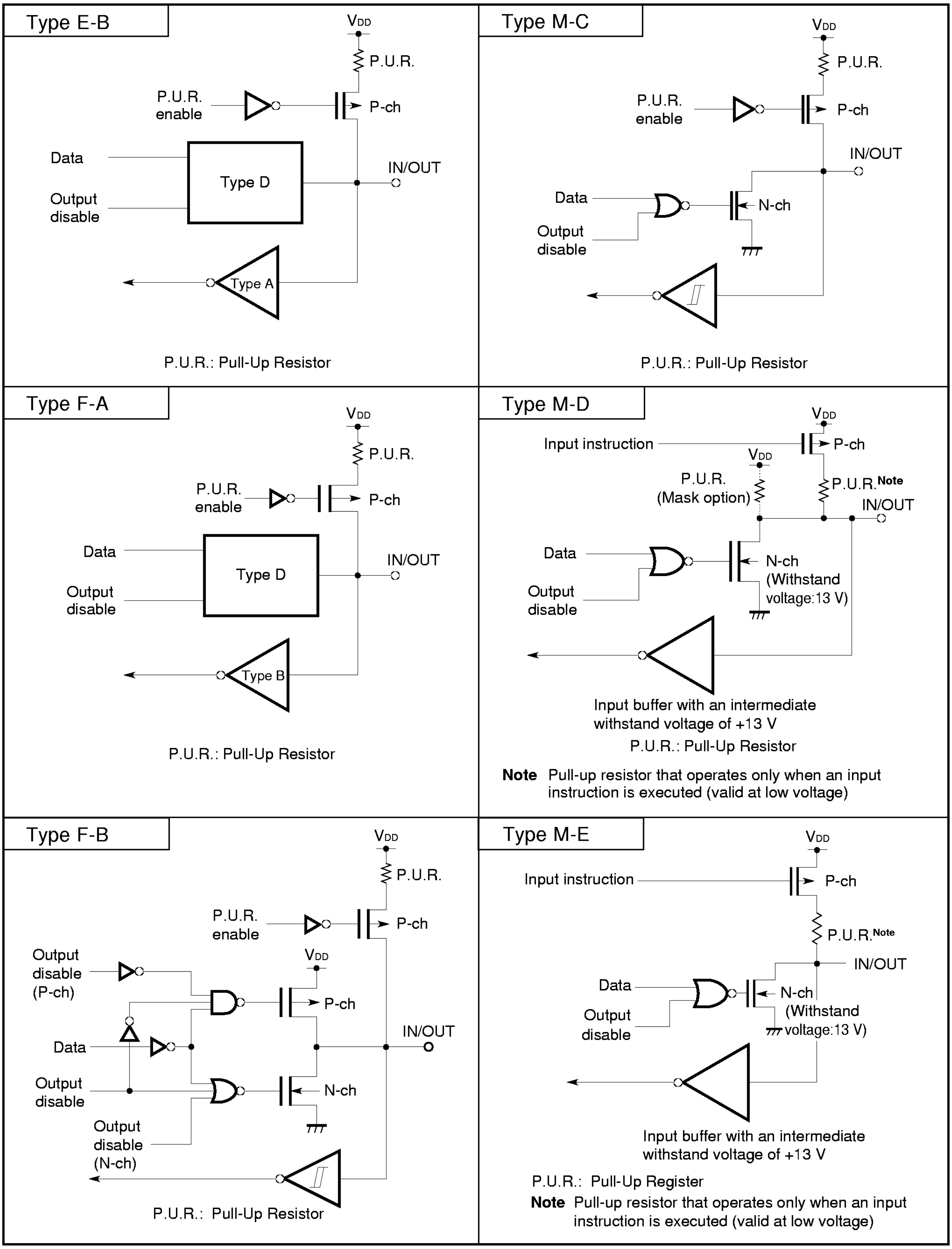
Type B

Schmitt trigger input with hysteresis

Type D

Push-pull output which can be set to high-impedance output (off for both P-ch and N-ch)

Figure 2-1. Pin Input/Output Circuits (2/2)



Type E-B: P.U.R.: Pull-Up Resistor

Type M-C: P.U.R.: Pull-Up Resistor

Type F-A: P.U.R.: Pull-Up Resistor

Type M-D: Input buffer with an intermediate withstand voltage of +13 V

P.U.R.: Pull-Up Resistor

Note Pull-up resistor that operates only when an input instruction is executed (valid at low voltage)

Type F-B: P.U.R.: Pull-Up Resistor

Type M-E: Input buffer with an intermediate withstand voltage of +13 V

P.U.R.: Pull-Up Register

Note Pull-up resistor that operates only when an input instruction is executed (valid at low voltage)

2.4 CONNECTION OF UNUSED PINS

Table 2-3. Connection of Unused Pins

Pin name	Recommended connection
P00/INT4	To be connected to V_{SS}
P01/$\overline{\text{SCK}}$	To be connected to V_{SS} or V_{DD}
P02/SO/SB0	
P03/SI/SB1	
P10/INT0-P12/INT2	To be connected to V_{SS}
P13/TI0	
P20/PTO0	Input state: To be connected to V_{SS} or V_{DD} through a resistor Output state: To be left open
P21/PTO1	
P22/PCL	
P23/BUZ	
P30(/MD0)-P33(/MD3)$^{\textbf{Note 1}}$	
P40-P43	
P50-P53	
P60/KR0-P63/KR3	
P70/KR4-P73/KR7	
P80-P81	
XT1$^{\textbf{Note 2}}$	To be connected to V_{SS} or V_{DD}
XT2$^{\textbf{Note 2}}$	To be left open
IC (V_{PP})$^{\textbf{Note 1}}$	To be connected directly to V_{DD}

Notes 1. ( ): µPD75P0116

2. When the subsystem clock is not to be used, select SOS.0 = 1 (the built-in feedback resistor will not be used).

CHAPTER 3 FEATURES OF THE ARCHITECTURE AND MEMORY MAP

The 75XL series architecture of the µPD750108 has the following features:

- Internal RAM of up to 4K words x 4 bits (12-bit address)
- Peripheral hardware expansibility

To provide these features, the following are used:

(1) Data memory bank structure
(2) General register bank structure
(3) Memory-mapped I/O

This chapter explains these topics.

3.1 DATA MEMORY BANK STRUCTURE AND ADDRESSING MODES

3.1.1 Data Memory Bank Structure

In the µPD750108, addresses 000H to 1FFH in data memory space are assigned to static RAM (512 words x 4 bits), and addresses F80H to FFFH are assigned to peripheral hardware (such as I/O ports and timers). To address a 12-bit location in this data memory space (4K x 4 bits), the µPD750108 uses such a memory bank structure that the low-order eight bits are specified with an instruction directly or indirectly, and the high-order four bits are used to specify a memory bank.

To specify a memory bank (MB), two hardware items are incorporated:

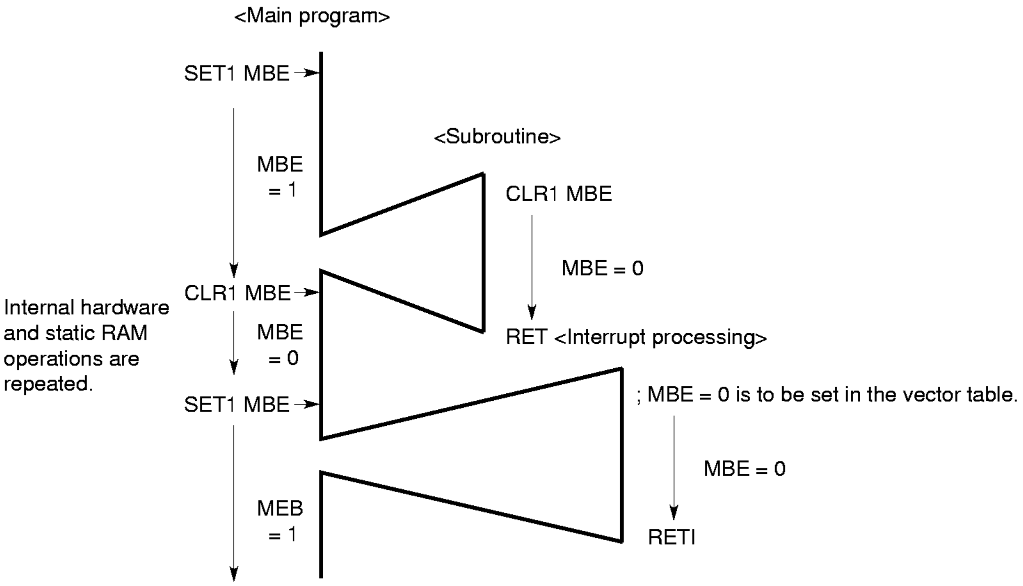
- Memory bank enable flag (MBE)
- Memory bank select register (MBS)

The MBS is a register used to select a memory bank, and the register can be set to 0, 1, or 15. The MBE is a flag used to determine whether the memory bank selected using the MBS is valid. As shown in Figure 3-1, when the MBE is set to 0, a certain memory bank is always selected regardless of the setting of the MBS. When the MBE is set to 1, memory bank selection depends on the setting of the MBS, thus enabling data memory space expansion.

In addressing data memory space, the MBE is usually set to 1 (MBE = 1), and data memory in the memory bank specified in the MBS is operated. However, the MBE = 0 mode or MBE = 1 mode can be selected for each step of processing for more efficient programming.

	Applicable program processing	Effect
MBE = 0 mode	• Interrupt processing	MBS save/restoration becomes unnecessary.
	• Processing that repeats internal hardware and static RAM operations	MBS modification becomes unnecessary.
	• Subroutine processing	MBS save/restoration becomes
MBE = 1 mode	• Usual program processing	

Figure 3-1. Use of MBE = 0 Mode and MBE = 1 Mode



<Main program>
SET1 MBE
MBE = 1
<Subroutine>
CLR1 MBE
MBE = 0
Internal hardware and static RAM operations are repeated.
CLR1 MBE
MBE = 0
RET <Interrupt processing>
SET1 MBE
; MBE = 0 is to be set in the vector table.
MBE = 0
MEB = 1
RETI

The contents of the MBE are automatically saved or restored at the time of subroutine processing, so that the MBE can be freely modified during subroutine processing. In interrupt processing, the MBE is automatically saved or restored, and when interrupt processing is started, the contents of the MBE can be specified for the interrupt processing by setting the interrupt vector table. This speeds up interrupt processing.

The setting of the MBS can be modified for subroutine processing or interrupt processing by saving or restoring the MBS with the PUSH or POP instruction.

The MBE is set using the SET1 or CLR1 instruction. The MBS is set using the SEL instruction.

Examples 1. The MBE is cleared, and a fixed memory bank is used.

```
CLR1    MBE    ; MBE <– 0
```

2. Memory bank 1 is selected.

```
SET1    MBE    ; MBE <– 1
SEL     MB1    ; MBS <– 1
```

3.1.2 Data Memory Addressing Modes

With the architecture of the µPD750108, seven addressing modes summarized in Figures 3-2 and 3-3, and Table 3-1 are available to address data memory space efficiently for each bit length of data to be processed. These addressing modes enable more efficient programming.

(1) 1-bit direct addressing (mem.bit)

In this addressing mode, the operand of an instruction can directly specify any bit in the entire data memory space.

A particular memory bank (MB) is always used in this addressing mode. In the MBE = 0 mode, when an address from 00H to 7FH is specified in the operand, memory bank 0 (MB = 0) is always used. When an address from 80H to FFH is specified, memory bank 15 (MB = 15) is always used. Accordingly, both the data area ranging from 000H to 07FH and the peripheral hardware area ranging from F80H to FFFH can be addressed in the MBE = 0 mode.

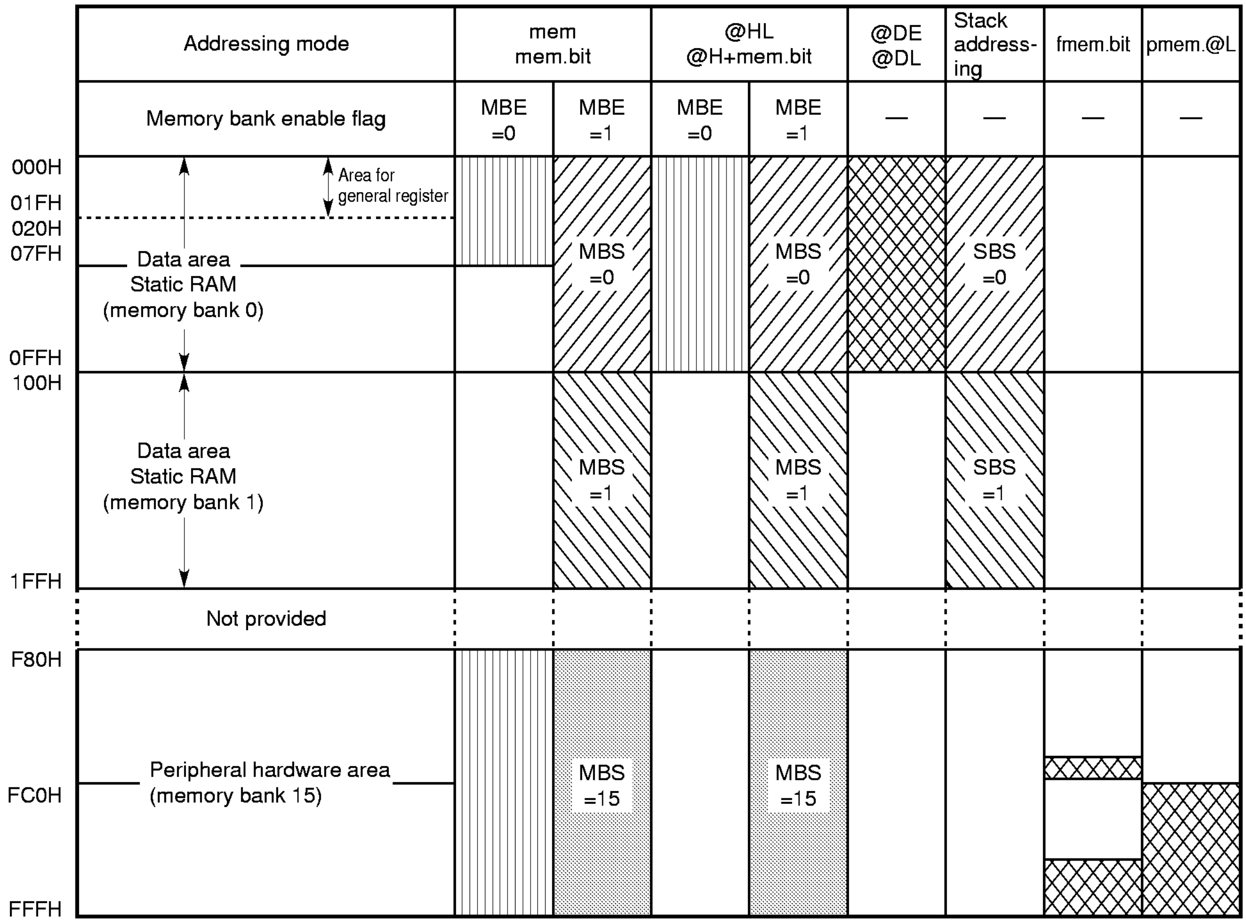
In the MBE = 1 mode, MB = MBS, and specifiable data memory space can be expanded.

This addressing mode can be applied to four instructions: bit set and reset instructions (SET1 and CLR1), and bit test instructions (SKT and SKF).

Example FLAG1 is set, FLAG2 is reset, and whether FLAG3 is zero is tested.

```
FLAG1   EQU     03FH.1  ; Bit 1 at address 3FH
FLAG2   EQU     087H.2  ; Bit 2 at address 87H
FLAG3   EQU     0A7H.0  ; Bit 0 at address A7H

        SET1    MBE     ; MBE   <– 1
        SEL     MB0     ; MBS   <– 0
        SET1    FLAG1   ; FLAG1 <– 1
        CLR1    FLAG2   ; FLAG2 <– 0
        SKF     FLAG3   ; FLAG3 = 0?
```

Figure 3-2. Data Memory Organization and Addressing Range of Each Addressing Mode



Remark – : Don't care

Table 3-1. Addressing Modes

Addressing mode	Representation format	Specified address
1-bit direct addressing	mem.bit	Bit specified by bit at the address specified by MB and mem. • When MBE = 0 and mem = 00H-7FH, MB = 0 mem = 80H-FFH, MB = 15 • When MBE = 1, MB = MBS
4-bit direct addressing	mem	Address specified by MB and mem. • When MBE = 0 and mem = 00H-7FH, MB = 0 mem = 80H-FFH, MB = 15 • When MBE = 1, MB = MBS
8-bit direct addressing		Address specified by MB and mem (mem: even address). • When MBE = 0 and mem = 00H-7FH, MB = 0 mem = 80H-FFH, MB = 15 • When MBE = 1, MB = MBS
4-bit register indirect addressing	@HL @HL+ @HL–	Address specified by MB and HL. In this case, MB = MBE·MBS HL+ automatically increments the L register after addressing. HL– automatically decrements the L register after addressing.
	@DE	Address specified by DE in memory bank 0
	@DL	Address specified by DL in memory bank 0
8-bit register indirect addressing	@HL	Address specified by MB and HL. (Contents of the L register is an even address.) In this case, MB = MBE·MBS
Bit manipulation addressing	fmem.bit	Bit specified by bit at the address specified by fmem. In this case, fmem = { FB0H-FBFH (interrupt-related hardware) FF0H-FFFH (I/O ports)
	pmem.@L	Bit specified by the low-order two bits of the L register at the address specified by the high-order 10 bits of pmem and the high-order two bits of the L register. In this case, pmem = FC0H-FFFH
	@H+mem.bit	Bit specified by bit at the address specified by MB, H, and the low-order four bits of mem. In this case, MB = MBE·MBS
Stack addressing	—	Address specified by the SP in memory bank selected by the SBS

(2) 4-bit direct addressing (mem)

In this addressing mode, the operand of an instruction directly specifies any area in the data memory space in units of four bits.

As with the 1-bit direct addressing mode, in the MBE = 0 mode, a fixed space consisting of the static RAM area ranging from 000H to 07FH and the peripheral hardware area ranging from F80H to FFFH can be addressed. In the MBE = 1 mode, MB = MBS, and specifiable data memory space can be expanded to the entire space.

This addressing mode can be applied to the MOV, XCH, INCS, IN, and OUT instructions.

Caution Less efficient program processing results if data associated with an I/O port is stored in the static RAM area of bank 1 as in Example 1. The modification of the MBS, as contained in Example 2, becomes unnecessary in the programming if data associated with an I/O port is stored at addresses 00H to 7FH of bank 0.

Examples 1. The data contained in BUFF is output on port 5.

```
BUFF    EQU     11AH        ; BUFF located at address 11AH
        SET1    MBE         ; MBE <– 1
        SEL     MB1         ; MBS <– 1
        MOV     A,BUFF      ; A <– (BUFF)
        SEL     MB15        ; MBS <– 15
        OUT     PORT5,A     ; PORT5 <– A
```

2. Data on port 4 is entered, and is saved in DATA1.

```
DATA1   EQU     5FH         ; DATA1 located at address 5FH
        CLR1    MBE         ; MBE <– 0
        IN      A,PORT4     ; A <– PORT4
        MOV     DATA1,A     ; (DATA1) <– A
```

(3) 8-bit direct addressing (mem)

In this addressing mode, the operand of an instruction directly specifies any area in the data memory space in units of eight bits.

The operand can specify an even address. The 4-bit data at the address specified in the operand and the 4-bit data at the address incremented by 1 are processed as a pair on an 8-bit basis with the 8-bit accumulator (XA register pair).

A memory bank is specified in the same way as the 4-bit direct addressing.

This addressing mode can be applied to the MOV, XCH, IN, and OUT instructions.

Example 1. Eight-bit data from port 4 and port 5 is transferred to addresses 20H and 21H.

```
DATA    EQU     020H
        CLR1    MBE         ; MBE <– 0
        IN      XA,PORT4    ; X <– PORT5 , A <– PORT4
        MOV     DATA,XA     ; (21H) <– X, (20H) <– A
```

Example 2. Eight-bit data is latched into the serial interface shift register (SIO), and the transfer data is set at the same time.

```
SEL     MB15        ;  MBS <– 15
XCH     XA,SIO      ;  XA  <—> (SIO)
```

(4) 4-bit register indirect addressing (@rpa)

In this addressing mode, the pointer (general register pair) specified in the operand of an instruction indirectly specifies a data memory space in units of four bits.

There are three types of data pointers. One is the HL register pair, which can specify any area in the data memory space when MB = MBE·MBS is specified. The other two are the DE register pair and DL register pair, with which memory bank 0 is always used regardless of how the MBE and MBS are specified. More efficient programming is possible by selecting a data pointer according to a data memory bank to be used. When the HL register pair is specified, the L register can be incremented or decremented by one in the automatic increment or automatic decrement mode each time an instruction is executed, thus simplifying the program step.

Example The data at 50H to 57H is transferred to 110H to 117H.

```
DATA1   EQU     57H
DATA2   EQU     117H
        SET1    MBE                     ; MBE <– 1
        SEL     MB1                     ; MBS <– 1
        MOV     D,#DATA1 SHR4           ; D <– 5
        MOV     HL,#DATA2 AND 0FFH      ; HL <– 17H
LOOP:   MOV     A,@DL                   ; A <– (DL)
        XCH     A,@HL–                  ; A <—> (HL), L <– L – 1
        BR      LOOP
```

The addressing mode using the HL register pair as the data pointer finds a wide range of operations such as data transfer, operations, comparison, and I/O. The addressing mode using the DE register pair or DL register pair is applied to the MOV and XCH instructions.

This addressing mode, combined with an increment/decrement instruction for a general register or register pair, enables data memory space addresses to be freely updated as shown in Figure 3-3.

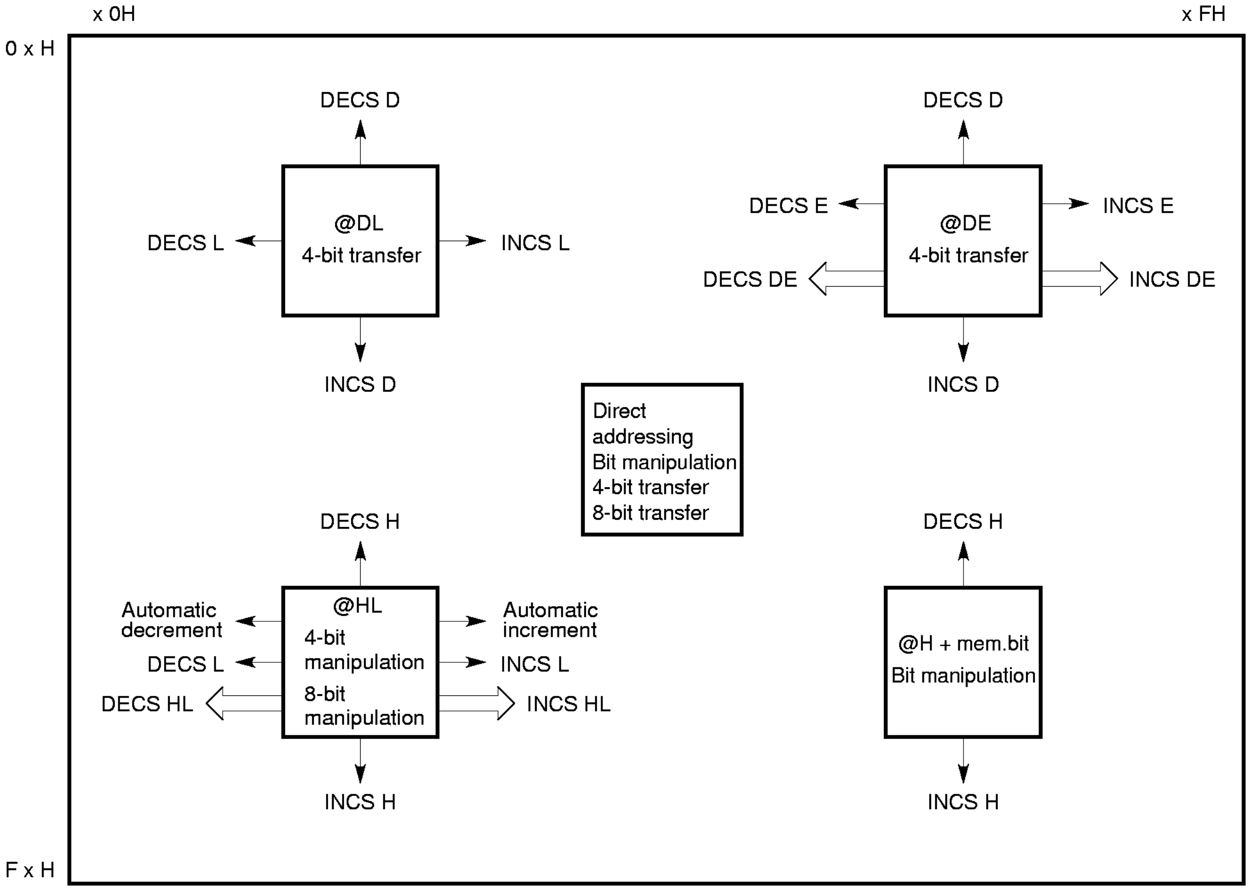
Example 1. The data at 50H to 57H is compared with the data at 110H to 117H.

```
DATA1   EQU     57H
DATA2   EQU     117H
        SET1    MBE
        SEL     MB1
        MOV     D,#DATA1 SHR4
        MOV     HL,#DATA2 AND 0FFH
LOOP:   MOV     A,@DL
        SKE     A,@HL       ; A = (HL)?
        BR      NO          ; NO
        DECS    L           ; YES, L <– L – 1
        BR      LOOP
```

Example 2. The data memory of 00H to FFH is cleared to 0.

```
        CLR1    RBE
        CLR1    MBE
        MOV     XA,#00H
        MOV     HL,#04H
LOOP:   MOV     @HL,A       ; (HL) <– A
        INCS    HL          ; HL <– HL + 1
        BR      LOOP
```

Figure 3-3. Updating Static RAM Addresses



(5) 8-bit register indirect addressing (@HL)

In this addressing mode, the data pointer (HL register pair) indirectly specifies any area in the data memory space in units of eight bits.

The 4-bit data at the address determined with bit 0 of the data pointer (bit 0 of the L register) set to 0 and the 4-bit data at the address incremented by 1 are processed as a pair on an 8-bit basis with the 8-bit accumulator (XA register pair).

A memory bank is specified in the same way as the 4-bit register indirect addressing with the HL register specified. In this case, MB = MBE·MBS.

This addressing mode can be applied to the MOV, XCH, and SKE instructions.

Examples 1. A comparison is made to determine whether the value of the count register (T0) of timer/event counter 0 is equal to the data at addresses 30H and 31H.

```
DATA    EQU     30H
        CLR1    MBE
        MOV     HL,#DATA
        MOV     XA,T0         ; XA <– Count register 0
        SKE     XA,@HL        ; XA = (HL)?
```

2. The data memory of 00H to FFH is cleared to 0.

```
        CLR1    RBE
        CLR1    MBE
        MOV     XA,#00H
        MOV     HL,#04H
LOOP:   MOV     @HL,XA        ; (HL) <– XA
        INCS    HL
        INCS    HL
        BR      LOOP
```

(6) Bit manipulation addressing

This addressing mode is used to perform bit manipulations (such as Boolean operations and bit transfer) for each bit in the data memory space.

The 1-bit direct addressing mode can be applied only to the set, reset, and test instructions. On the other hand, the bit manipulation addressing enables a wide variety of bit manipulations such as Boolean operations using the AND1, OR1, and XOR1 instructions, bit transfers using the MOV1 instruction, and test and reset operations using the SKTCLR instruction.

There are three types of bit manipulation addressing. The user can choose from these options according to the data memory address used.

(a) Specific address bit direct addressing (fmem.bit)

In this addressing mode, peripheral equipment that frequently performs bit manipulations involving, for example, I/O ports and interrupt flags, can be processed at all times regardless of memory bank setting. Accordingly, the data memory addresses that allow this addressing mode to be used are FF0H to FFFH where I/O ports are mapped, and FB0H to FBFH where interrupt-related hardware is mapped. Hardware mapped to these data memory areas can freely perform bit manipulations in the direct addressing mode at any time regardless of MBS and MBE setting.

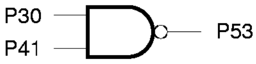
Examples 1. Value input to P02 is inverted, and the result is output on P33.

```
MOV1        CY, PORT0.2
NOT1        CY
MOV1        PORT3.3, CY
```

2. The timer 0 interrupt request flag (IRQT0) is tested. The request flag, if set, is cleared, and P63 is reset.

```
SKTCLR      IRQT0           ; IRQT0 = 1?
BR          NO              ; NO
CLR1        PORT6.3         ; YES
```

3. If both P30 and P41 are set to 1, P53 is reset.



```
MOV1        CY, PORT3.0     ; CY <– P30
AND1        CY, PORT4.1     ; CY ∧ P41
NOT1        CY              ; CY <– C̅Y̅
MOV1        PORT5.3, CY     ; P53 <– CY
```

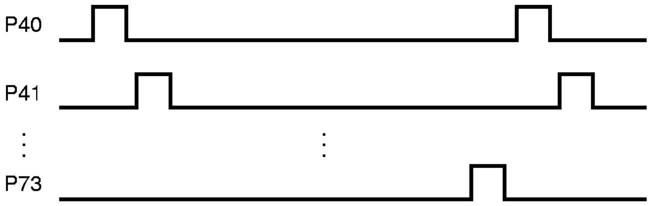
(b) Specific address bit register indirect addressing (pmem.@L)

In this addressing mode, the bits of peripheral hardware I/O ports are indirectly specified using a register to allow continuous manipulations. This addressing mode can be applied to data memory addresses FC0H to FFFH.

In this addressing mode, the high-order 10 bits of a 12-bit data memory address is directly specified in the operand, and the low-order two bits and bit address are indirectly specified using the L register. Thus the use of the L register enables 16 bits (four ports) to be continuously manipulated.

This addressing mode again enables bit manipulation regardless of MBE and MBS setting.

Example Pulses are output on the bits in the order from port 4 to port 7.



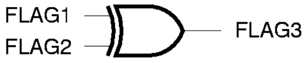
```
        MOV     L,#0
LOOP:   SET1    PORT4.@L    ; Bits (L1-0) of ports 4 to 7 <– 1
        CLR1    PORT4.@L    ; Bits (L1-0) of ports 4 to 7 <– 0
        INCS    L
        NOP
        BR      LOOP
```

(c) Specific 1-bit direct addressing (@H+mem.bit)

This addressing mode enables any bit in the data memory space to be manipulated.

In this addressing mode, the high-order four bits of the data memory address in the memory bank specified by MB = MBE·MBS are indirectly specified using the H register, and the low-order four bits and bit address are directly specified in the operand. This addressing mode enables a wide variety of manipulations for each bit in the entire data memory space.

Example Bit 2 at address 32H (FLAG3) is reset if both bit 3 at address 30H (FLAG1) and bit 0 at address 31H (FLAG2) are set to 0 or 1.



```
FLAG1   EQU    30H.3
FLAG2   EQU    31H.0
FLAG3   EQU    32H.2
        SEL    MB0
        MOV    H,#FLAG1 SHR 6
        MOV1   CY, @H+FLAG1      ; CY <– FLAG1
        XOR1   CY, @H+FLAG2      ; CY <– CY ∀ FLAG2
        MOV1   @H+FLAG3, CY      ; FLAG3 <– CY
```

(7) Stack addressing

This addressing mode is used for save/restoration operation in interrupt processing or subroutine processing.

In this addressing mode, the address indicated by the stack pointer (8 bits) of data memory bank 0 is specified.

This addressing mode can be used for register save/restoration operation using the PUSH or POP instruction as well as save/restoration operation in interrupt and subroutine processing.

Examples **1.** A register is saved and restored in subroutine processing.

```
SUB:  PUSH      XA
      PUSH      HL
      PUSH      BS      ; Save MBS and RBS
             :
      POP       BS
      POP       HL
      POP       XA
      RET
```

2. The contents of the HL register pair are transferred to the DE register pair.

```
      PUSH      HL
      POP       DE      ; DE <– HL
```

3. A branch is made to the address indicated by the [XABC] register.

```
      PUSH      BC
      PUSH      XA
      RET               ; Branch to address XABC
```

3.2 GENERAL REGISTER BANK CONFIGURATION

The μPD750108 contains four register banks, each consisting of eight general registers: X, A, B, C, D, E, H, and L. These registers are mapped to addresses 00H to 1FH in memory bank 0 of the data memory (see **Figure 3-5**). To specify a general register bank, a register bank enable flag (RBE) and a register bank select register (RBS) are contained. The RBS is a register used to select a register bank, and the RBE is a flag used to determine whether a register bank selected using the RBS is to be enabled. The register bank (RB) enabled at instruction execution is determined as

RB = RBE·RBS

Table 3-2. Register Bank to Be Selected with the RBE and RBS

RBE	RBS				Register bank
	3	2	1	0	
0	0	0	x	x	Bank 0 is always selected.
1	0	0	0	0	Bank 0 is selected.
			0	1	Bank 1 is selected.
			1	0	Bank 2 is selected.
			1	1	Bank 3 is selected.

Always 0

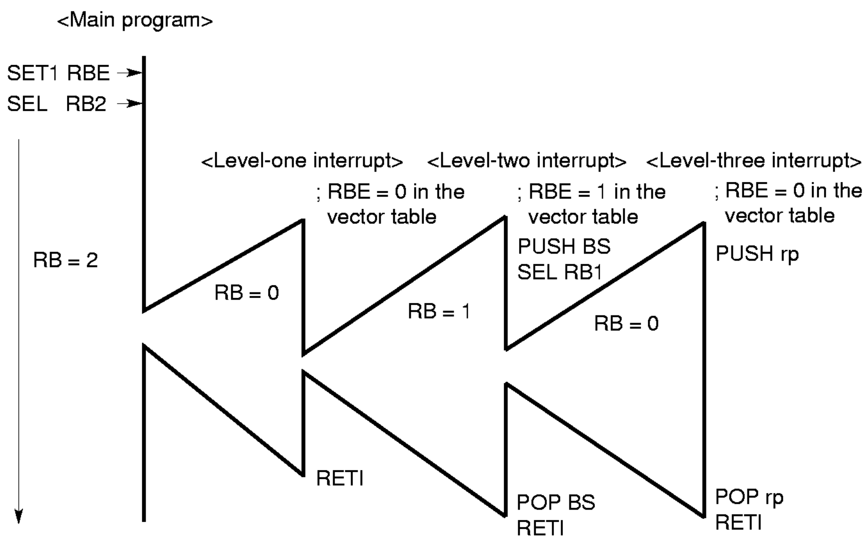
Remark x: Don’t care

The contents of the RBE are automatically saved or restored at the beginning or end of subroutine processing, so that the RBE can be freely modified during subroutine processing. In interrupt processing, the RBE is automatically saved or restored, and when interrupt processing is started, the contents of the RBE can be specified for the interrupt processing by setting the interrupt vector table. Therefore, as indicated in Table 3-3, by selecting a register bank depending on whether the processing is normal or interrupt, the general register need not be saved and restored for the level-one interrupt processing, and only the RBS needs to be saved and restored for the level-two interrupt processing, thus speeding up interrupt processing.

Table 3-3. Recommended Use of Register Banks with Normal Routines and Interrupt Routines

Normal processing	Use register banks 2 and 3 with RBE = 1.
Level-one interrupt processing	Use register bank 0 with RBE = 0.
Level-two interrupt processing	Use register bank 1 with RBE = 1. (In this case, the RBS needs to be saved and restored.)
Multiple (triple or more) interrupt processing	Save and restore the registers with PUSH or POP.

Figure 3-4. Example of Register Bank Selection



The setting of the RBS can be modified for subroutine processing or interrupt processing by saving or restoring the RBS with the PUSH or POP instruction.

The RBE is set using the SET1 or CLR1 instruction. The RBS is set using the SEL instruction.

Example

```
SET1    RBE  ; RBE <– 1
CLR1    RBE  ; RBE <– 0
SEL     RB0  ; RBS <– 0
SEL     RB3  ; RBS <– 3
```

The general register area of the μPD750108 can be used not only on a 4-bit basis, but also on an 8-bit basis with register pairs. This enables users to perform transfers, arithmetic/logical operations, comparisons, and increments and decrements at a speed comparable to that of an 8-bit microcomputer, and thereby enables to program using mainly general registers.

(1) When used as a 4-bit register

When the general register area is used on a 4-bit basis, eight general registers, the X, A, B, C, D, E, H, and L registers, are available in the register bank specified with RB = RBE·RBS as shown in Figure 3-5. The A register functions as a 4-bit accumulator which performs transfers, arithmetic/logical operations, and comparisons. The other general registers perform transfers, comparisons, and increments/decrements with the accumulator.

(2) When used as an 8-bit register

When the general register area is used on an 8-bit basis, the register pairs in the register bank specified by RBE·RBS can be specified as XA, BC, DE, and HL as shown in Figure 3-6, and the register pairs in the register bank that has the inverted value of bit 0 of the register bank (RB) can be specified as XA', BC', DE', and HL', thus providing up to eight 8-bit registers. The XA register pair functions as an 8-bit accumulator which performs transfers, arithmetic/logical operations, comparisons, and increments/ decrements of 8-bit data. The other register pairs perform transfers, arithmetic/logical operations, comparisons, and increments/decrements with the accumulator. The HL register pair functions mainly as a data pointer, and the DE and DL register pairs function as an auxiliary data pointer.

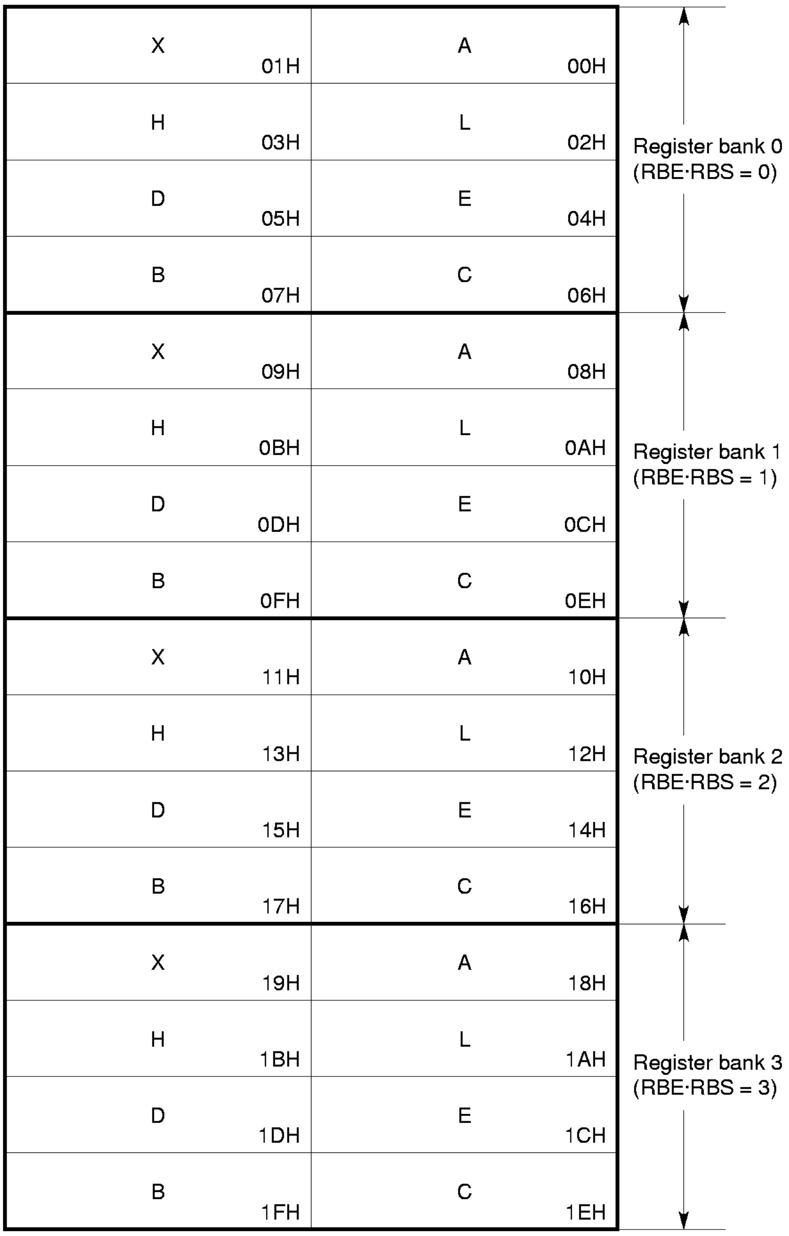
Examples 1.

```
INCS    HL          ; HL <– HL + 1, skip at HL = 00H
ADDS    XA,BC       ; XA <– XA + BC, skip at carry
SUBC    DE',XA      ; DE' <– DE' – XA – CY
MOV     XA,XA'      ; XA <– XA'
MOVT    XA,@PCDE    ; XA <– (PC12-8 + DE) ROM, reference table
SKE     XA, BC      ; Skip if XA = BC
```

2. The value of the count register (T0) for timer/event counter 0 is tested until it becomes greater than the value of the BC' register pair.

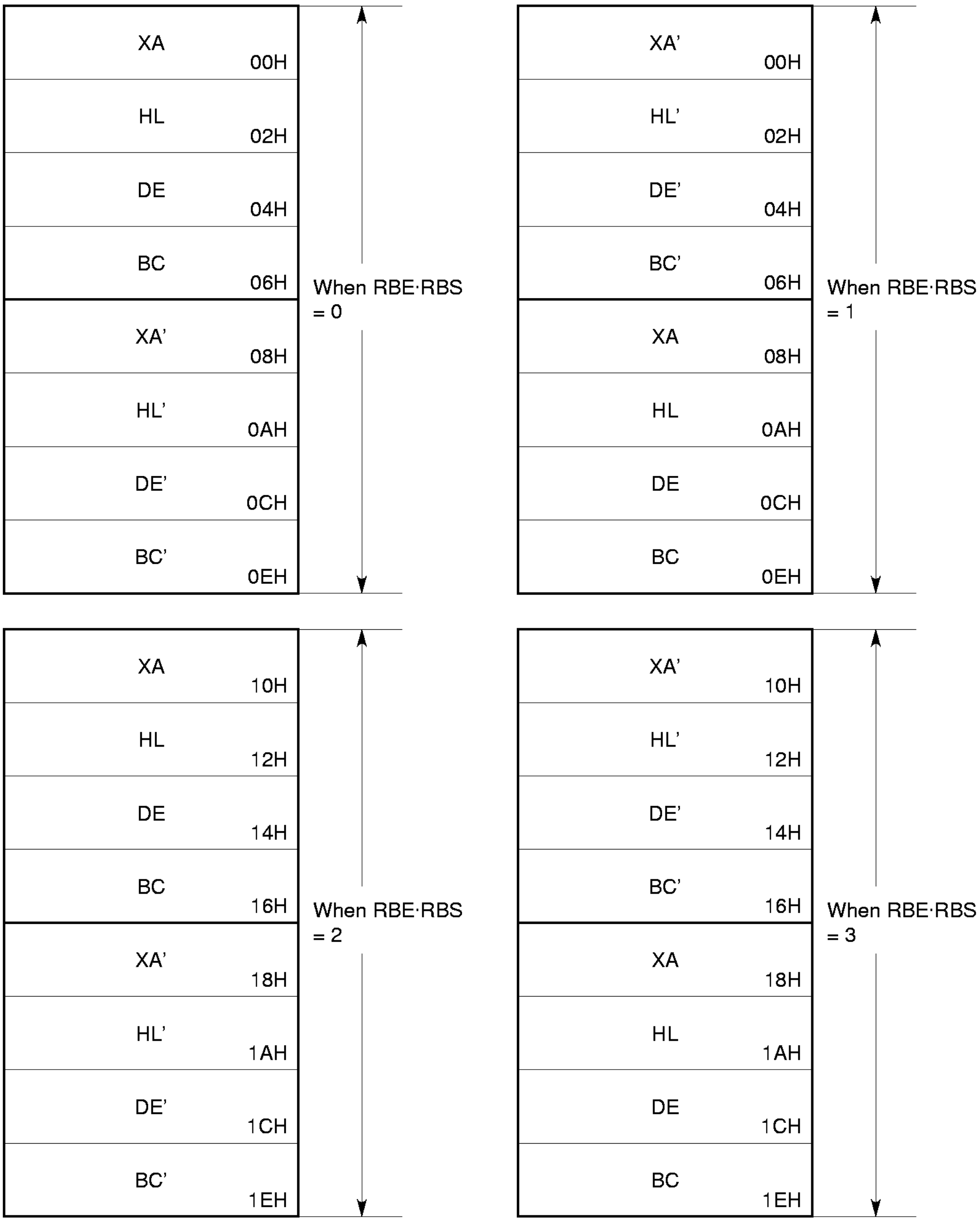
```
        CLR1    MBE     ;
NO:     MOV     XA,T0   ; Read count register
        SUBS    XA,BC'  ; XA ≥ BC'?
        BR      YES     ; YES
        BR      NO      ; NO
```

Figure 3-5. General Register Configuration (4-bit Processing)



X 01H	A 00H	Register bank 0 (RBE·RBS = 0)
H 03H	L 02H	
D 05H	E 04H	
B 07H	C 06H	
X 09H	A 08H	Register bank 1 (RBE·RBS = 1)
H 0BH	L 0AH	
D 0DH	E 0CH	
B 0FH	C 0EH	
X 11H	A 10H	Register bank 2 (RBE·RBS = 2)
H 13H	L 12H	
D 15H	E 14H	
B 17H	C 16H	
X 19H	A 18H	Register bank 3 (RBE·RBS = 3)
H 1BH	L 1AH	
D 1DH	E 1CH	
B 1FH	C 1EH	

Figure 3-6. General Register Configuration (8-bit Processing)



3.3 MEMORY-MAPPED I/O

The µPD750108 employs memory-mapped I/O, which maps peripheral hardware such as timers and I/O ports to addresses F80H to FFFH in data memory space as shown in Figure 3-2. This means that there is no particular instruction to control peripheral hardware, but all peripheral hardware is controlled using memory manipulation instructions. (Some mnemonics for hardware control are available to make programs readable.)

To manipulate peripheral hardware, the addressing modes listed in Table 3-4 can be used.

Table 3-4. Addressing Modes Applicable to Peripheral Hardware Operation

	Applicable addressing mode	Applicable hardware
Bit manipulation	Direct addressing mode specifying mem.bit with MBE = 0 (MBE = 1, MBS = 15)	All hardware allowing bit manipulation
	Direct addressing mode specifying fmem.bit regardless of MBE and MBS setting	IST1, IST0, MBE, RBE, IExxx, IRQxxx, PORTn.x
	Indirect addressing mode specifying pmem.@L regardless of MBE and MBS setting	BSBn.x PORTn.x
4-bit manipulation	Direct addressing mode specifying mem with MBE = 0 or (MBE = 1, MBS = 15)	All hardware allowing 4-bit manipulation
	Register indirect addressing mode specifying @HL with (MBE = 1, MBS = 15)	
8-bit manipulation	Direct addressing mode specifying mem (even address) with MBE = 0 or (MBE = 1, MBS = 15)	All hardware allowing 8-bit manipulation
	Register indirect addressing mode specifying @HL (with the L register containing an even number) with MBE = 1 and MBS = 15	

Figure 3-7 summarizes the I/O map of the µPD750108.

The items in the figure have the following meanings:

- Symbol : Name representing incorporated hardware, which can be coded in the operand field of an instruction
- R/W : Indicates whether the hardware allows read/write operation.
  R/W : Both read and write operations possible
  R : Read only
  W : Write only
- Number of manipulatable bits:
  Indicates the number of bits that can be processed at a time in hardware manipulation
  O : Bit manipulation is possible in units of the indicated number of bits (1, 4, or 8 bits).
  Δ : Particular bits can be manipulated. For these bits, see Remarks.
  – : Bit manipulation is impossible in units of the indicated number of bits (1, 4, or 8 bits).
- Bit manipulation addressing:
  Bit manipulation addressing applicable in hardware bit manipulation

Figure 3-7. µPD750108 I/O Map (1/5)

Address	Hardware name (symbol)				R/W	Number of bits that can be manipulated			Bit manipulation addressing	Remarks
	b3	b2	b1	b0		1 bit	4 bits	8 bits		
F80H	Stack pointer (SP)				R/W	–	–	○	–	Bit 0 is fixed to 0.
F82H	Register bank selection register (RBS)				R	–	○	○	–	Note 1
	Bank selection register (BS)									
F83H	Memory bank selection register (MBS)					–	○			
F84H	Stack bank selection register (SBS)				R/W	–	○	–	mem.bit	
F85H	Basic interval timer mode register (BTM)				W	△	○	–	mem.bit	Only bit 3 can be manipulated.
F86H	Basic interval timer (BT)				R	–	–	○	–	
F8BH	WDTM[Note 2]				W	○	–	–	mem.bit	

F98H	Clock mode register (WM)				R/W	△ (R)	–	○	mem.bit	Only bit 3 can be tested.
						–	–		–	

Notes 1. Can be manipulated separately as the RBS and MBS in 4-bit units.
Can also be manipulated as the BS in 8-bit units.
Use SEL MBn and SEL RBn instructions to write data to MBS and RBS respectively.

2. WDTM: Watchdog timer enable flag (W); cannot be cleared by an instruction.

Figure 3-7. μPD750108 I/O Map (2/5)

Address	Hardware name (symbol)				R/W	Number of bits that can be manipulated			Bit manipulation addressing	Remarks
	b3	b2	b1	b0		1 bit	4 bits	8 bits		
FA0H	Timer/event counter mode register (TM0)				R/W	△ (W)	–	○ (R/W)	mem.bit	Bit write manipulation is enabled only for bit 3.
						–	–		–	
FA2H	TOE0[Note 1]				W	○	–	–	mem.bit	
FA4H	Timer/event counter count register (T0)				R	–	–	○	–	
FA6H	Timer/event counter modulo register (TMOD0)				R/W	–	–	○	–	
FA8H	Timer counter mode register (TM1)				R/W	△ (W)	–	○ (R/W)	mem.bit	Bit write manipulation is enabled only for bit 3.
						–	–		–	
FAAH	TOE1[Note 2]				W	○	–	–	mem.bit	
FACH	Timer counter count register (T1)				R	–	–	○	–	
FAEH	Timer counter modulo register (TMOD1)				R/W	–	–	○	–	

Notes 1. TOE0: Timer/event counter output enable flag (W)

2. TOE1: Timer counter output enable flag (W)

Figure 3-7. µPD750108 I/O Map (3/5)

Address	Hardware name (symbol)				R/W	Number of bits that can be manipulated			Bit manipulation addressing	Remarks
	b3	b2	b1	b0		1 bit	4 bits	8 bits		
FB0H	IST1	IST0	MBE	RBE	R/W	○ (R/W)	○ (R/W)	○ (R)	fmem.bit	Manipulation in 8-bit units is enabled only for reading.
	Program status word (PSW)									
	CY	SK2	SK1	SK0		–	–			
FB2H	Interrupt priority select register (IPS)				R/W	–	○	–		**Note 1**
FB3H	Processor clock control register (PCC)				R/W	–	○	–		**Note 2**
FB4H	INT0 edge detection mode register (IM0)				R/W	–	○	–	–	
FB5H	INT1 edge detection mode register (IM1)				R/W	–	○			Bits 3, 2, and 1 are fixed to 0.
FB6H	INT2 edge detection mode register (IM2)				R/W	–	○			Bits 3 and 2 are fixed to 0.
FB7H	System clock control register (SCC)				R/W	△ (R/W)	○ (R)	–	–	Bits 2 and 1 are fixed to 0.
FB8H	IE4	IRQ4	IEBT	IRQBT	R/W	○	○	–	fmem.bit	
FBAH			IEW	IRQW	R/W	○	○			
FBCH	IET1	IRQT1	IET0	IRQT0	R/W	○	○	–		
FBDH			IECSI	IRQCSI	R/W	○	○			
FBEH	IE1	IRQ1	IE0	IRQ0	R/W	○	○	–		
FBFH			IE2	IRQ2	R/W	○	○			

FC0H	Bit sequential buffer 0 (BSB0)	R/W	○	○	○	mem.bit pmem.@L	
FC1H	Bit sequential buffer 1 (BSB1)	R/W	○	○			
FC2H	Bit sequential buffer 2 (BSB2)	R/W	○	○	○		
FC3H	Bit sequential buffer 3 (BSB3)	R/W	○	○			
FCFH	Sub-oscillator control register (SOS)	R/W	–	○	–	–	

Notes 1. Only bit 3 can be manipulated by an EI/DI instruction.

2. Bits 3 and 2 can be manipulated bit by bit by a STOP/HALT instruction.

Remarks 1. IExxx : Interrupt enable flag

2. IRQxxx : Interrupt request flag

Figure 3-7. µPD750108 I/O Map (4/5)

Address	Hardware name (symbol)				R/W	Number of bits that can be manipulated			Bit manipulation addressing	Remarks
	b3	b2	b1	b0		1 bit	4 bits	8 bits		
FD0H	Clock output mode register (CLOM)				R/W	–	○	–	–	
FDCH	Pull-up resistor specification register group A (POGA)				R/W	–	–	○	–	
FDEH	Pull-up resistor specification register group B (POGB)				R/W	–	–	○	–	
FE0H	Serial operation mode register (CSIM)				R/W	–	–	○	–	Note
	CSIE	COI	WUP			△ (R) (W)	–		mem.bit	
FE2H	CMDD	RELD	CMDT	RELT	R/W	○	–	–	mem.bit	Whether this location is read- or write-accessible depends on the bit.
	SBI control register (SBIC)									
	BSYE	ACKD	ACKE	ACKT						
FE4H	Serial I/O shift register (SIO)				R/W	–	–	○	–	
FE6H	Slave address register (SVA)				R/W	–	–	○	–	
FE8H	PM33	PM32	PM31	PM30	R/W	–	–	○	–	
	Port mode register group A (PMGA)									
	PM63	PM62	PM61	PM60						
FECH	–	PM2	–	–	R/W	–	–	○	–	
	Port mode register group B (PMGB)									
	PM7	–	PM5	PM4						
FEEH	–	–	–	PM8	R/W	–	–	○	–	
	Port mode register group C (PMGC)									
	–	–	–	–						

Note Whether a bit can be read or written depends on the bit.

Figure 3-7. µPD750108 I/O Map (5/5)

Address	Hardware name (symbol)				R/W	Number of bits that can be manipulated			Bit manipulation addressing	Remarks
	b3	b2	b1	b0		1 bit	4 bits	8 bits		
FF0H	Port 0 (PORT0)		SCKP		R/W	○ (R) (R/W)	○ (R)	–	fmem.bit pmem.@L	Note 1
FF1H	Port 1 (PORT1)				R	○	○			
FF2H	Port 2 (PORT2)				R/W	○	○	–		
FF3H	Port 3 (PORT3)				R/W	○	○			
FF4H	Port 4 (PORT4)				R/W	○	○	○		
FF5H	Port 5 (PORT5)				R/W	○	○			
FF6H[Note 2]	KR3 Port 6 (PORT6)	KR2	KR1	KR0	R/W	○	○	○		
FF7H[Note 2]	KR7 Port 7 (PORT7)	KR6	KR5	KR4	R/W	○	○			
FF8H	Port 8 (PORT8)				R/W	○	○	–		

Notes 1. Bit 1 can be read or written only in serial operation enable mode. It can be read when four-bit manipulation is performed.

2. KR0 to KR7 can be read (R) bit by bit. When inputting 4 bits at a time, specify PORT6 or PORT7.

CHAPTER 4 INTERNAL CPU FUNCTIONS

4.1 Mk I MODE/Mk II MODE SWITCH FUNCTIONS

4.1.1 Differences between Mk I Mode and Mk II Mode

The CPU of the µPD750108 subseries has two modes (Mk I mode and Mk II mode) and which mode is used is selectable. Bit 3 of the stack bank selection register (SBS) determines the mode.

- Mk I mode: This mode has the upward compatibility with the µPD75008 subseries.
  It can be used in the 75XL CPUs having a ROM of up to 16KB.
- Mk II mode: This mode is not compatible with the µPD75008 subseries.
  It can be used in all 75XL CPUs, including those having a ROM of 16KB or more.

Table 4-1 shows the differences between Mk I mode and Mk II mode.

Table 4-1. Differences between Mk I Mode and Mk II Mode

	Mk I mode	Mk II mode
Number of stack bytes in a subroutine instruction	2 bytes	3 bytes
BRA !addr1 instruction CALLA !addr1 instruction	Not supported	Supported
MOVT XA, @BCXA instruction MOVT XA, @BCDE instruction BR BCXA instruction BR BCDE instruction	Supported	Supported
CALL !addr instruction	3 machine cycles	4 machine cycles
CALLF !faddr instruction	2 machine cycles	3 machine cycles

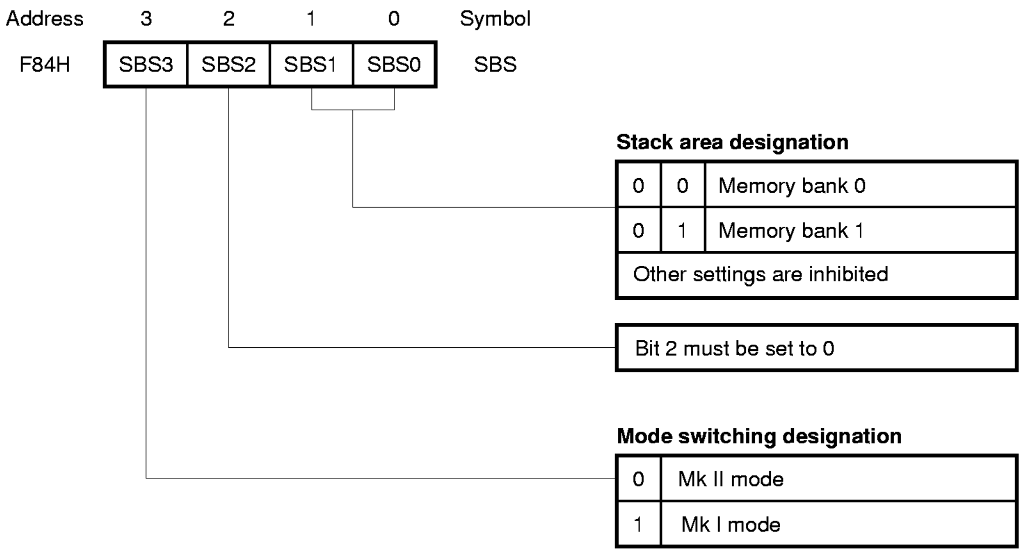
Caution Mk II mode is for maintaining a software compatibility with products in the 75X series or 75XL series whose program memory is more than 24K bytes.
Therefore, Mk I mode is recommended for applications with a focus on the ROM efficiency or speed.

4.1.2 Setting of the Stack Bank Selection Register (SBS)

The Mk I mode and Mk II mode are switched by stack bank selection register. Figure 4-1 shows the register configuration.

The stack bank selection register is set with a 4-bit memory operation instruction. To use the CPU in Mk I mode, initialize the register to 10xxB[Note] at the beginning of the program. To use the CPU in Mk II mode, initialize it to 00xxB[Note].

Figure 4-1. Stack Bank Selection Register Format



Note Specify the desired value in xx.

Caution The CPU operates in Mk I mode after the $\overline{\text{RESET}}$ signal is issued, because bit 3 of SBS is set to 1. Set bit 3 of SBS to 0 (Mk II mode) to use the CPU in Mk II mode.

4.2 PROGRAM COUNTER (PC): 12 BITS (µPD750104) 13 BITS (µPD750106 AND µPD750108) 14 BITS (µPD75P0116)

The program counter is a binary counter which retains the address data of the program memory. The program counter consists of 12 bits in the µPD750104 (see **Figure 4-2(a)**), 13 bits in the µPD750106 and µPD750108 (see **Figure 4-2(b)**), and 14 bits in the µPD75P0116 (see **Figure 4-2(c)**).

Figure 4-2. Program Counter Organization

(a) µPD750104

PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0

(b) µPD750106 and µPD750108

PC12	PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0

(c) µPD75P0116

PC13	PC12	PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0

Usually, each time an instruction is executed, the program counter is automatically incremented according to the number of bytes in the instruction.

When a branch instruction (BR, BRA, BRCB) is executed, immediate data indicating the branch destination and the contents of a register pair are set in all or some bits of the program counter.

When a subroutine call instruction (CALL, CALLA, CALLF) is executed, or a vectored interrupt occurs, the current contents of the program counter (already incremented return address for fetching the next instruction) are saved in the stack memory (data memory indicated by the stack pointer), then the jump destination address is loaded.

When a return instruction (RET, RETS, RETI) is executed, the contents of the stack memory are set in the program counter.

When the $\overline{\text{RESET}}$ signal is issued, the program counter is initialized to the contents of the program memory at addresses 000H and 001H. The program can be started from any address according to the contents.

µPD750104 :

PC_{11}-PC_8 <– $(000H)_{3\text{-}0}$, PC_7-PC_0 <– $(001H)_{7\text{-}0}$

µPD750106 and µPD750108 :

PC_{12}-PC_8 <– $(000H)_{4\text{-}0}$, PC_7-PC_0 <– $(001H)_{7\text{-}0}$

µPD75P0116 :

PC_{13}-PC_8 <– $(000H)_{5\text{-}0}$, PC_7-PC_0 <– $(001H)_{7\text{-}0}$

4.3 PROGRAM MEMORY (ROM): 4096 WORDS x 8 BITS (µPD750104: MASKED ROM) 6144 WORDS x 8 BITS (µPD750106: MASKED ROM) 8192 WORDS x 8 BITS (µPD750108: MASKED ROM) 16384 WORDS x 8 BITS (µPD75P0116: ONE-TIME PROM)

The program memory is used for storing programs, an interrupt vector table, GETI instruction reference table, table data, and so forth. The µPD750104, µPD750106, and µPD750108 are provided with a mask-programmable ROM as the program memory, and the µPD75P0116 is provided with a one-time PROM.

Figures 4-3 to 4-6 show the program memory maps.

Program memory is addressed by the program counter. Table data can be referenced using the table reference instruction (MOVT).

Figures 4-3 to 4-6 also show the allowable branch address ranges for the branch instructions and subroutine call instructions. The relative branch instruction (BR $addr) allows a branch to addresses (contents of the PC less 15 to one, or plus two to 16) regardless of block.

The program memory is located at following addresses.

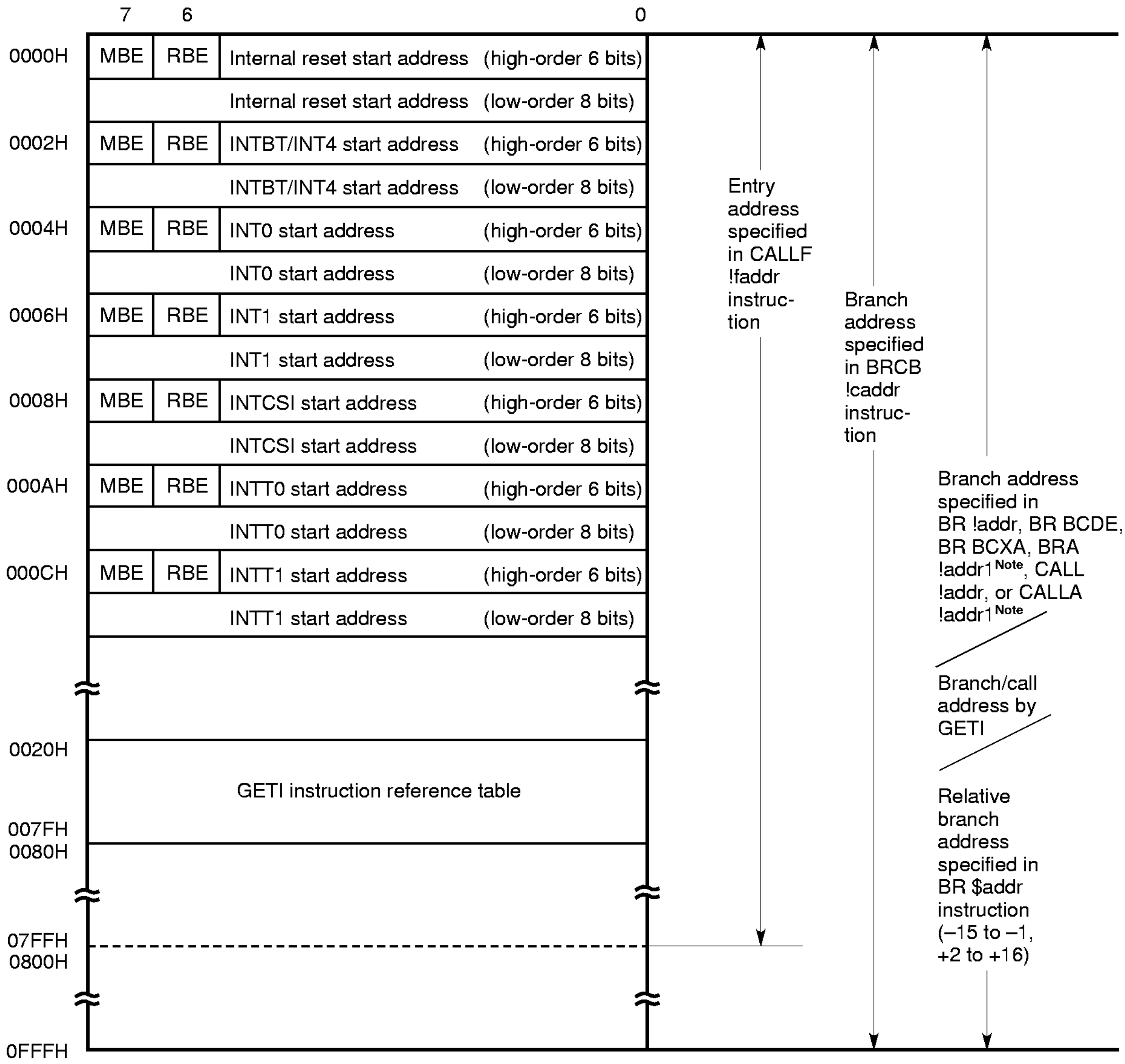
- 0000H to 0FFFH: µPD750104
- 0000H to 17FFH: µPD750106
- 0000H to 1FFFH: µPD750108
- 0000H to 3FFFH: µPD75P0116

The following addresses are assigned to special functions. All areas excluding 0000H and 0001H can be used as normal program memory.

- 0000H to 0001H
  Vector address table for holding the RBE and MBE values and program start address when a $\overline{\text{RESET}}$ signal is issued (allowing a reset start at an arbitrary address)
- 0002H to 000DH
  Vector address table for holding the RBE and MBE values and program start address for each vectored interrupt (allowing interrupt processing to be started at an arbitrary address)
- 0020H to 007FH
  Table area referenced by the GETI instruction[Note]

Note The GETI instruction can represent an arbitrary two-byte or three-byte instruction or two one-byte instructions in one byte and is used to reduce the number of program bytes. (See **Section 11.1.1.**)

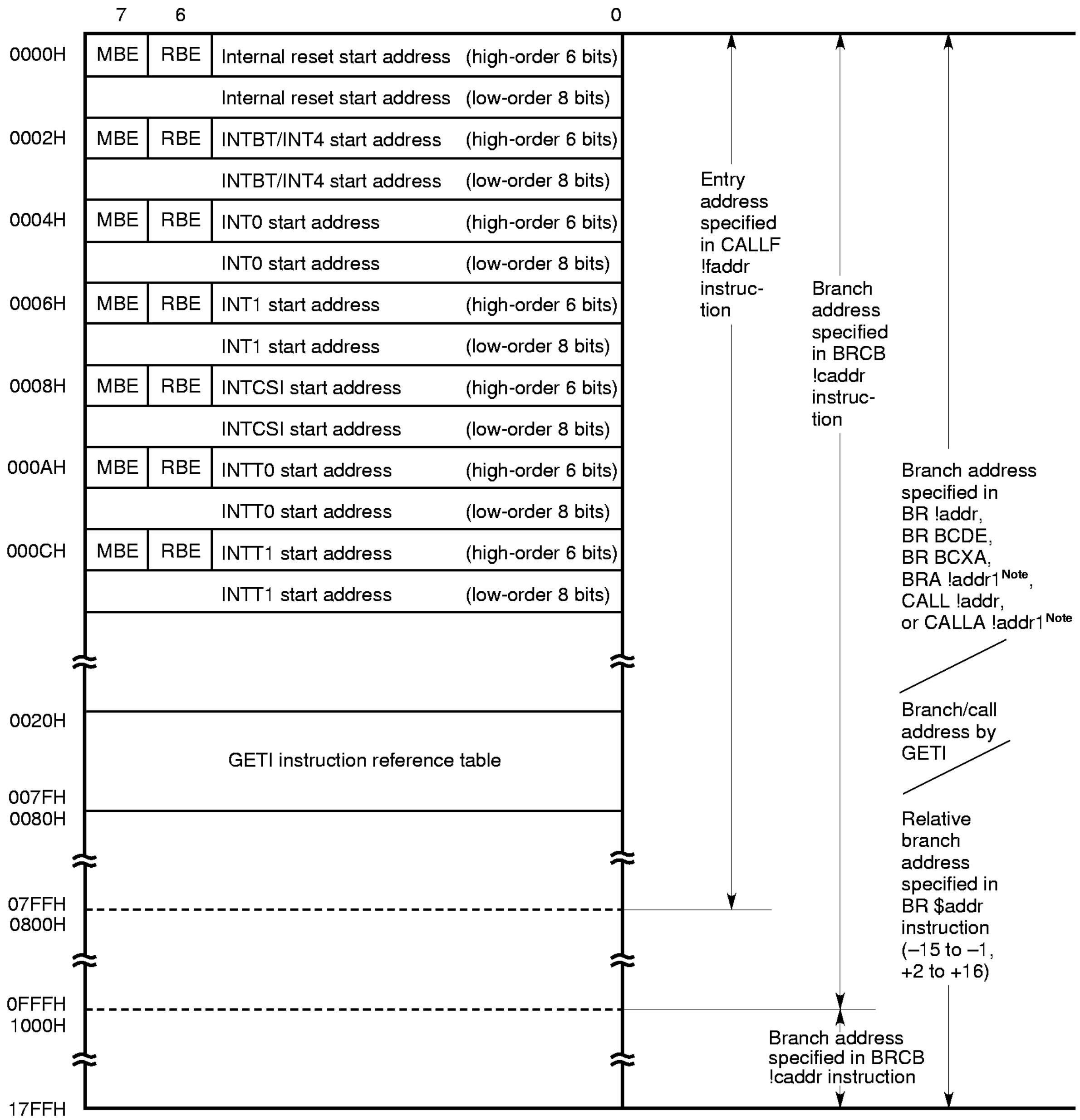
Figure 4-3. Program Memory Map (in μPD750104)


7
6
0
0000H
MBE
RBE
Internal reset start address (high-order 6 bits)
Internal reset start address (low-order 8 bits)
0002H
MBE
RBE
INTBT/INT4 start address (high-order 6 bits)
INTBT/INT4 start address (low-order 8 bits)
0004H
MBE
RBE
INT0 start address (high-order 6 bits)
INT0 start address (low-order 8 bits)
0006H
MBE
RBE
INT1 start address (high-order 6 bits)
INT1 start address (low-order 8 bits)
0008H
MBE
RBE
INTCSI start address (high-order 6 bits)
INTCSI start address (low-order 8 bits)
000AH
MBE
RBE
INTT0 start address (high-order 6 bits)
INTT0 start address (low-order 8 bits)
000CH
MBE
RBE
INTT1 start address (high-order 6 bits)
INTT1 start address (low-order 8 bits)
0020H
GETI instruction reference table
007FH
0080H
07FFH
0800H
0FFFH
Entry address specified in CALLF !faddr instruction
Branch address specified in BRCB !caddr instruction
Branch address specified in BR !addr, BR BCDE, BR BCXA, BRA !addr1 Note, CALL !addr, or CALLA !addr1 Note
Branch/call address by GETI
Relative branch address specified in BR $addr instruction (–15 to –1, +2 to +16)

Note Can be used only in the MkII mode.

Remark In addition to the above, the BR PCDE and BR PCXA instructions can cause a branch to an address with only the 8 low-order bits of the PC changed.

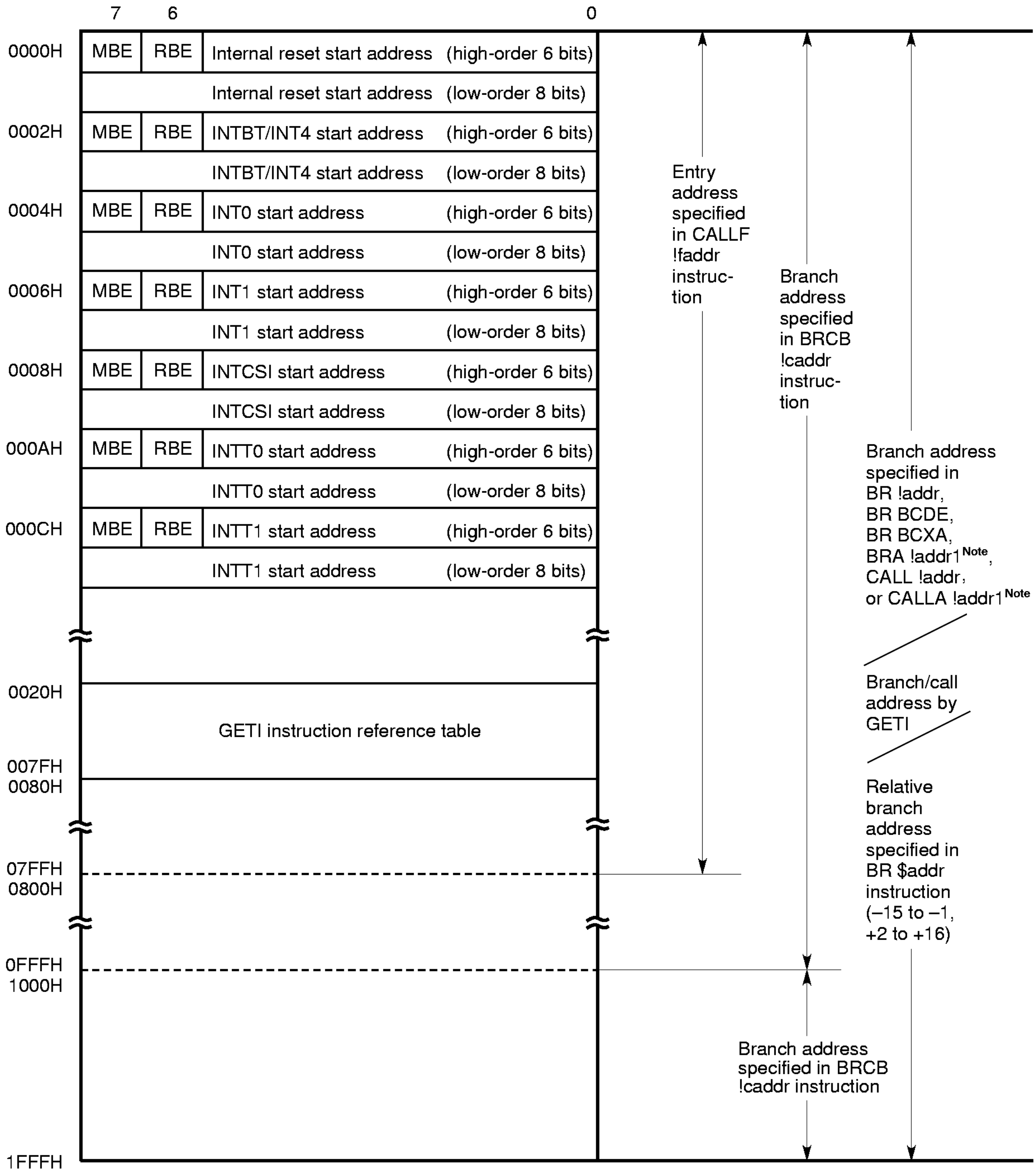
Figure 4-4. Program Memory Map (in µPD750106)


7
6
0
0000H
MBE
RBE
Internal reset start address (high-order 6 bits)
Internal reset start address (low-order 8 bits)
0002H
MBE
RBE
INTBT/INT4 start address (high-order 6 bits)
INTBT/INT4 start address (low-order 8 bits)
0004H
MBE
RBE
INT0 start address (high-order 6 bits)
INT0 start address (low-order 8 bits)
0006H
MBE
RBE
INT1 start address (high-order 6 bits)
INT1 start address (low-order 8 bits)
0008H
MBE
RBE
INTCSI start address (high-order 6 bits)
INTCSI start address (low-order 8 bits)
000AH
MBE
RBE
INTT0 start address (high-order 6 bits)
INTT0 start address (low-order 8 bits)
000CH
MBE
RBE
INTT1 start address (high-order 6 bits)
INTT1 start address (low-order 8 bits)
0020H
GETI instruction reference table
007FH
0080H
07FFH
0800H
0FFFH
1000H
17FFH
Entry address specified in CALLF !faddr instruction
Branch address specified in BRCB !caddr instruction
Branch address specified in BR !addr, BR BCDE, BR BCXA, BRA !addr1 Note, CALL !addr, or CALLA !addr1 Note
Branch/call address by GETI
Relative branch address specified in BR $addr instruction (–15 to –1, +2 to +16)
Branch address specified in BRCB !caddr instruction

Note Can be used only in the MkII mode.

Remark In addition to the above, the BR PCDE and BR PCXA instructions can cause a branch to an address with only the 8 low-order bits of the PC changed.

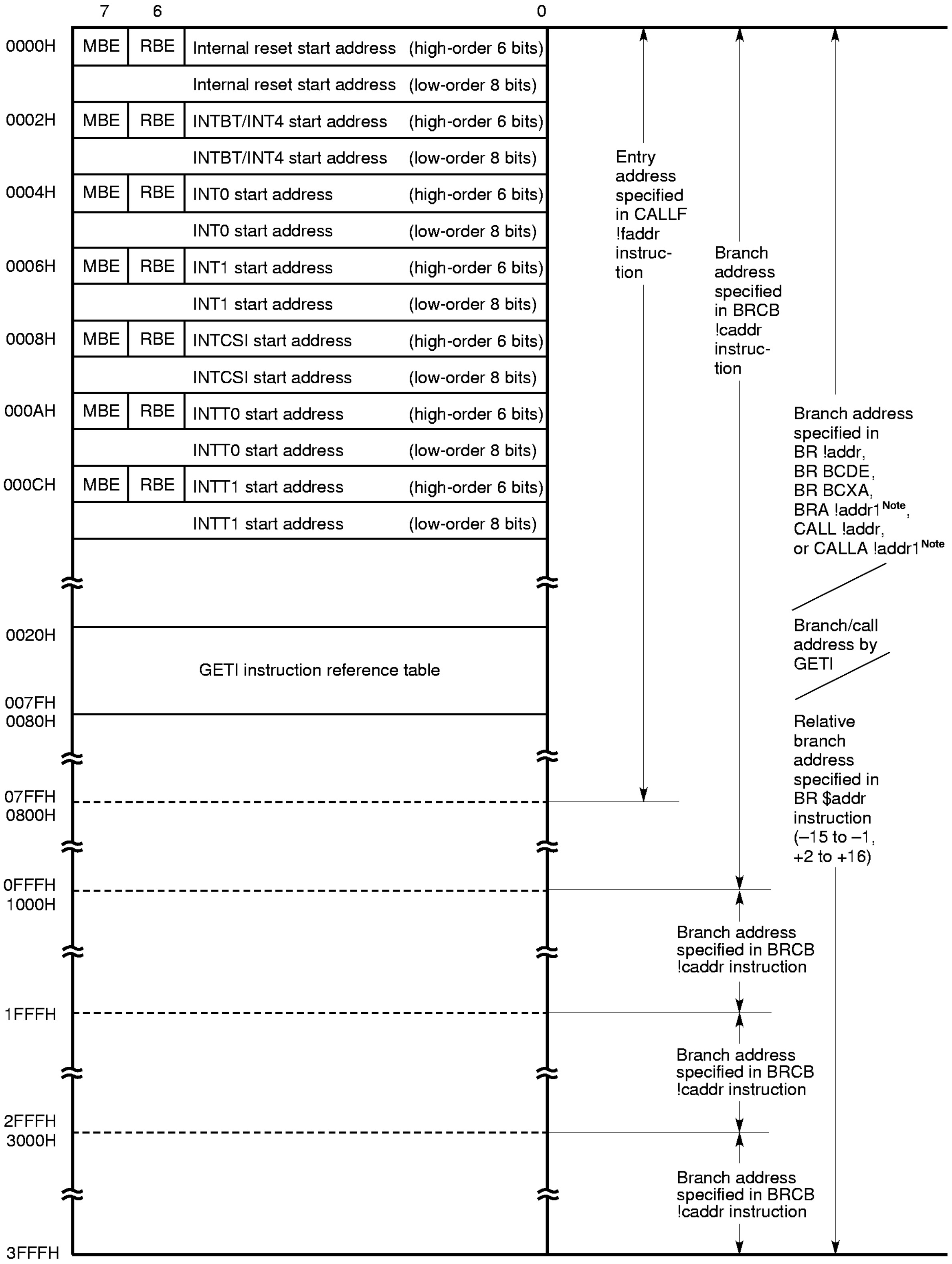
Figure 4-5. Program Memory Map (in μPD750108)


7
6
0
0000H
MBE
RBE
Internal reset start address (high-order 6 bits)
Internal reset start address (low-order 8 bits)
0002H
MBE
RBE
INTBT/INT4 start address (high-order 6 bits)
INTBT/INT4 start address (low-order 8 bits)
0004H
MBE
RBE
INT0 start address (high-order 6 bits)
INT0 start address (low-order 8 bits)
0006H
MBE
RBE
INT1 start address (high-order 6 bits)
INT1 start address (low-order 8 bits)
0008H
MBE
RBE
INTCSI start address (high-order 6 bits)
INTCSI start address (low-order 8 bits)
000AH
MBE
RBE
INTT0 start address (high-order 6 bits)
INTT0 start address (low-order 8 bits)
000CH
MBE
RBE
INTT1 start address (high-order 6 bits)
INTT1 start address (low-order 8 bits)
0020H
GETI instruction reference table
007FH
0080H
07FFH
0800H
0FFFH
1000H
1FFFH
Entry address specified in CALLF !faddr instruction
Branch address specified in BRCB !caddr instruction
Branch address specified in BR !addr, BR BCDE, BR BCXA, BRA !addr1Note, CALL !addr, or CALLA !addr1Note
Branch/call address by GETI
Relative branch address specified in BR $addr instruction (–15 to –1, +2 to +16)
Branch address specified in BRCB !caddr instruction

Note Can be used only in the MkII mode.

Remark In addition to the above, the BR PCDE and BR PCXA instructions can cause a branch to an address with only the 8 low-order bits of the PC changed.

Figure 4-6. Program Memory Map (in μPD75P0116)


7
6
0
0000H
MBE
RBE
Internal reset start address (high-order 6 bits)
Internal reset start address (low-order 8 bits)
0002H
MBE
RBE
INTBT/INT4 start address (high-order 6 bits)
INTBT/INT4 start address (low-order 8 bits)
0004H
MBE
RBE
INT0 start address (high-order 6 bits)
INT0 start address (low-order 8 bits)
0006H
MBE
RBE
INT1 start address (high-order 6 bits)
INT1 start address (low-order 8 bits)
0008H
MBE
RBE
INTCSI start address (high-order 6 bits)
INTCSI start address (low-order 8 bits)
000AH
MBE
RBE
INTT0 start address (high-order 6 bits)
INTT0 start address (low-order 8 bits)
000CH
MBE
RBE
INTT1 start address (high-order 6 bits)
INTT1 start address (low-order 8 bits)
0020H
GETI instruction reference table
007FH
0080H
07FFH
0800H
0FFFH
1000H
1FFFH
2FFFH
3000H
3FFFH
Entry address specified in CALLF !faddr instruction
Branch address specified in BRCB !caddr instruction
Branch address specified in BR !addr, BR BCDE, BR BCXA, BRA !addr1Note, CALL !addr, or CALLA !addr1Note
Branch/call address by GETI
Relative branch address specified in BR $addr instruction (–15 to –1, +2 to +16)
Branch address specified in BRCB !caddr instruction
Branch address specified in BRCB !caddr instruction
Branch address specified in BRCB !caddr instruction

Note Can be used only in the MkII mode.

Remark In addition to the above, the BR PCDE and BR PCXA instructions can cause a branch to an address with only the 8 low-order bits of the PC changed.

4.4 DATA MEMORY (RAM): 512 WORDS x 4 BITS

The data memory consists of a data area and peripheral hardware area as shown in Figure 4-7.
The data memory consists of the following memory banks with each bank made of 256 words x 4 bits.

- Memory banks 0 and 1 (data area)
- Memory bank 15 (peripheral hardware area)

4.4.1 Data Memory Configuration

(1) Data area

The data area consists of a static RAM, and is used for storing program data and as stack memory for subroutine and interrupt execution. Battery backup enables the memory to hold data for a long time even if the CPU is stopped in the standby mode. The data area can be manipulated with memory manipulation instructions.

The static RAM is mapped to memory banks 0 and 1, with each made up of 256 words x 4 bits. Bank 0 is used as a data area, but can also be used as a general register area (000H to 01FH) and stack area[Note] (000H to 1FFH).

Whole locations in memory banks 0, 1, 2, and 3 (000H to 3FFH) can be used as a stack area.

The static RAM has a configuration of four bits per address. However, the memory can be manipulated in 8 bit units using an 8-bit memory manipulation instruction, and in bit units using a bit manipulation instruction. Note that an even address must be specified in an 8-bit manipulation instruction.

Note Memory bank 0 or 1 can be selected as the stack area.

- **General register area**

  The general register area can be manipulated with either general register manipulation instructions or memory manipulation instructions. Up to eight 4-bit registers are available. Of the 8 general registers, registers not used by the program can be used as a data area or stack area. (See **Section 4.5.**)

- **Stack memory area**

  The stack memory area is set by the instruction. This area can be used as a save area for subroutine or interrupt execution. (See **Section 4.7.**)

(2) Peripheral hardware area

The peripheral hardware area is mapped at addresses F80H to FFFH of memory bank 15.

Memory manipulation instructions are used to manipulate the peripheral hardware area as well as the static RAM area. Note that, however, the number of bits to be manipulated at a time varies according to the individual addresses. Addresses to which no peripheral hardware is assigned cannot be accessed since such address locations contain no data memory. (See **Figure 3-7**.)

4.4.2 Specification of a Data Memory Bank

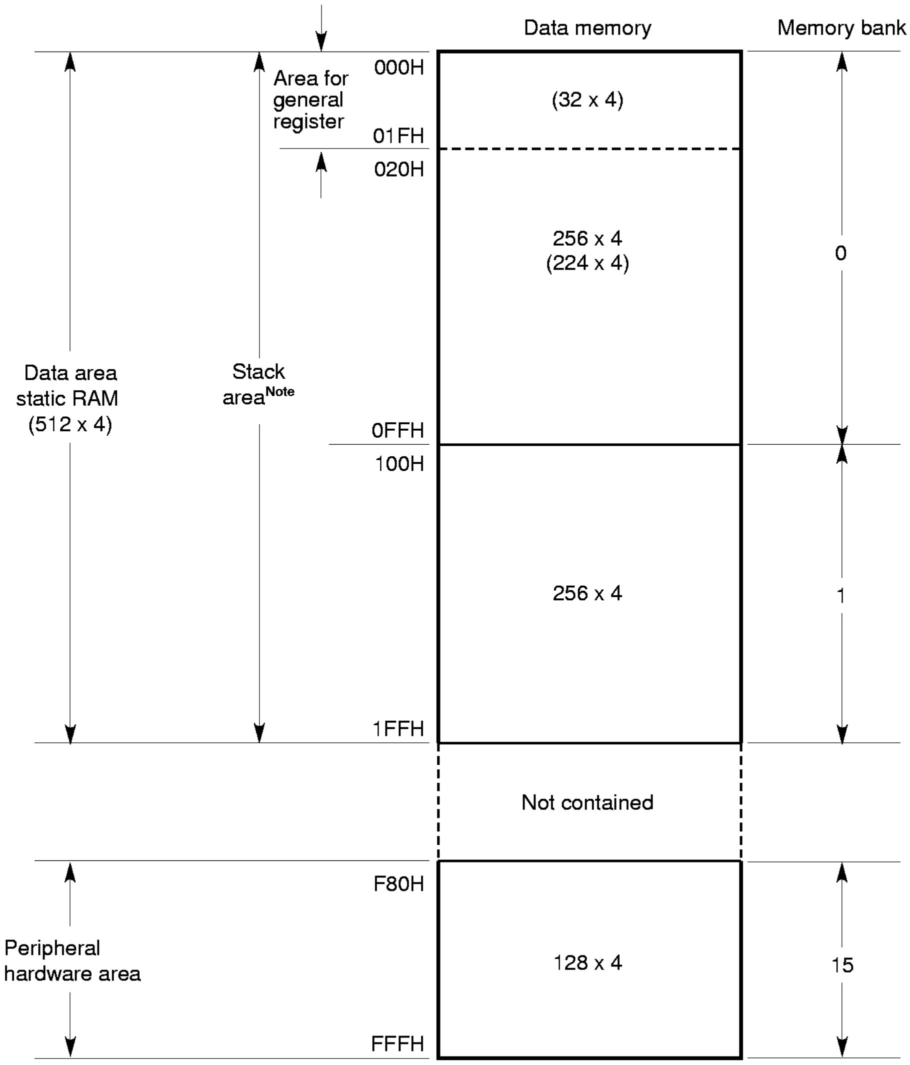
If the memory bank enable flag (MBE) enables bank specification (MBE = 1), a memory bank is specified with the 4-bit memory bank select register (MBS = 0, 1, 15). If the MBE disables bank specification (MBE = 0), memory bank 0 or 15 is automatically selected according to the addressing mode. Locations in a bank is addressed by 8-bit immediate data or a register pair.

For details on the selection of a memory bank and addressing, see **Section 3.1**.

For how to use the particular data memory areas, see the following sections and chapter.

- General register area : **Section 4.5**
- Stack memory area : **Section 4.7**
- Peripheral hardware area: **Chapter 5**

Figure 4-7. Data Memory Map



Note Memory bank 0 or 1 can be selected as the stack area.

Data memory is undefined when it is reset. For this reason, it is to be initialized to zero (RAM clear) usually at the start of a program. Remember to perform this initialization. Otherwise, unexpected bugs may occur.

Example The following program clears data at addresses 000H to 1FFH in RAM.

```
         SET1   MBE
         SEL    MB0
         MOV    XA,#00H
         MOV    HL,#04H
RAMC0:   MOV    @HL,A          ; Clear 04H to FFH[Note]
         INCS   L              ; L <– L + 1
         BR     RAMC0
         INCS   H              ; H <– H + 1
         BR     RAMC0
         SEL    MB1
RAMC1:   MOV    @HL,A          ; Clear 100H to 1FFH
         INCS   L              ; L <– L + 1
         BR     RAMC1
         INCS   H              ; H <– H + 1
```

Note Data memory locations at 000H to 003H are allocated to general registers XA and HL, so these are not cleared.

4.5 GENERAL REGISTER: 8 x 4 BITS x 4 BANKS

The general registers are mapped to particular addresses in data memory. Four banks of registers are provided, with each bank consisting of eight 4-bit registers (B, C, D, E, H, L, X, and A).

The register bank (RB) to be enabled at the time of instruction execution is determined by:

RB = RBE·RBS: (RBS = 0 to 3)

Each general register allows 4-bit manipulation. In addition, BC, DE, HL, or XA serves as a register pair for 8-bit manipulation. DL also makes a register pair as well as DE and HL. These three register pairs can be used as data pointers.

In 8-bit manipulation, the register pairs in the register banks (0 <—> 1, 2 <—> 3) that have the inverted value of bit 0 of the register bank (RB) address can be specified as BC', DE', HL', and XA' in addition to the register pairs BC, DE, HL, and XA. (See **Section 3.2**.)

A general register area can be addressed and accessed as normal RAM, regardless of whether it is used as a register.

Figure 4-8. General Register Format

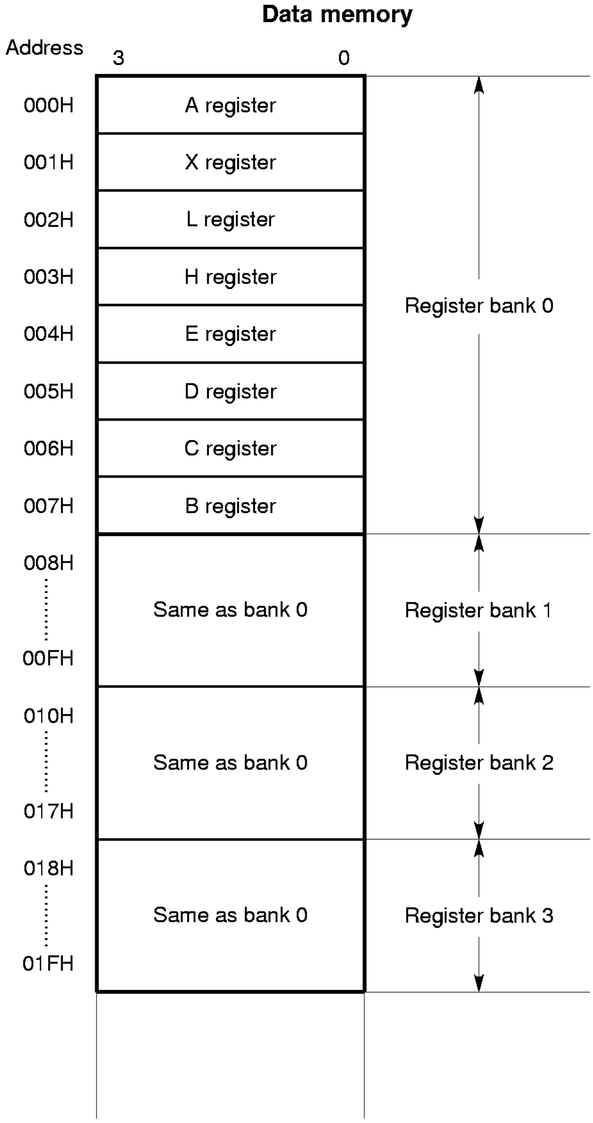
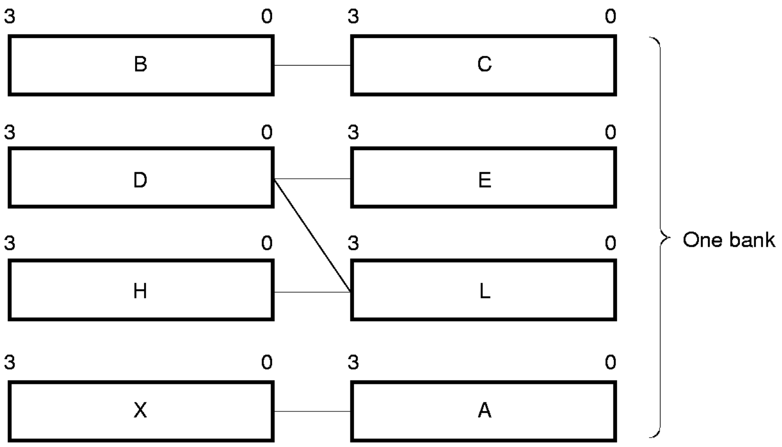


Figure 4-9. Register Pair Format

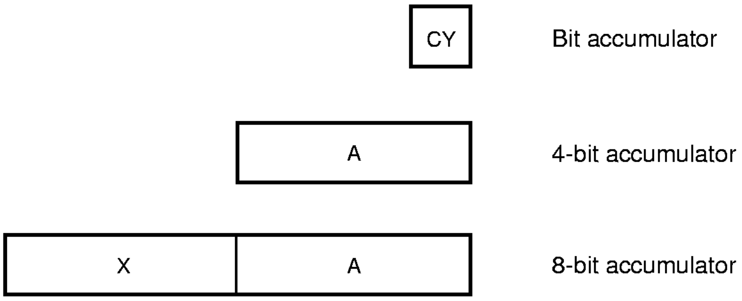


4.6 ACCUMULATOR

In the µPD750108, the A register and XA register pair function as accumulators. The A register is mainly used for 4-bit data processing instructions, and the XA register pair is mainly used for 8-bit data processing instructions.

For a bit manipulation instruction, the carry flag (CY) functions as a bit accumulator.

Figure 4-10. Accumulator



4.7 STACK POINTER (SP) AND STACK BANK SELECT REGISTER (SBS)

The μPD750108 uses static RAM as stack memory (LIFO scheme), and the 8-bit register holding the start address of the stack area is the stack pointer (SP).

The stack area is located at addresses 000H to 1FFH in memory banks 0 and 1. One memory bank is selected according to the value of the 2-bit SBS. (See **Table 4-2**.)

Table 4-2. Stack Area to Be Selected by the SBS

SBS		Stack area
SBS1	SBS0	
0	0	Memory bank 0
0	1	Memory bank 1
Other than above		Not to be set

The SP is decremented before a write (save) operation to stack memory, and is incremented after a read (restoration) operation from stack memory.

Figures 4-12 to 4-15 show data saved to and restored from stack memory in these stack operations.

To place the stack area at a given location, the SP can be initialized with an 8-bit memory manipulation instruction, and the SBS can be initialized with a 4-bit memory manipulation instruction. Both can be read from as well.

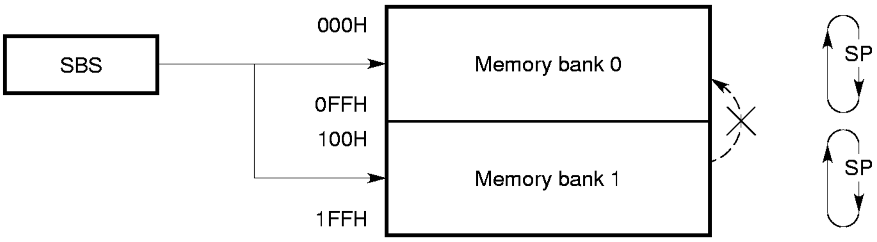
When the SP is initialized to 00H, a stack operation starts at the high-order address (nFFH) of memory bank (n) specified with the SBS.

A stack area must be within the memory bank specified with the SBS. If a stack operation exceeds address n00H, the operation returns to address nFFH in the same bank. Linear stacking beyond memory bank boundaries is enabled only by resetting the SBS.

A $\overline{\text{RESET}}$ signal causes the contents of the SP to be undefined, and causes the contents of the SBS to be 1000B. Remember to initialize the SP and SBS to a desired value at the start of a program.

Figure 4-11. Format of Stack Pointer and Stack Bank Select Register

Address									Symbol
F80H	SP7	SP6	SP5	SP4	SP3	SP2	SP1	0	SP
F84H					SBS3[Note]	0	SBS1	SBS0	SBS



Note The Mk I mode and Mk II mode can be switched by bit 3 of SBS. The stack bank selection function can be used in both Mk I mode and Mk II mode. (See **Section 4.1** for details.)

Example SP initialization

Specify memory bank 1 as a stack area to start stack operation at address 1FFH.

```
SEL     MB15        ; or CLR1 MBE
MOV     A,#1
MOV     SBS,A       ; Specify memory bank 1 as a stack area
MOV     XA,#00H
MOV     SP,XA       ; SP <– 00H
```

Figure 4-12. Data Saved to the Stack Memory (Mk I Mode)

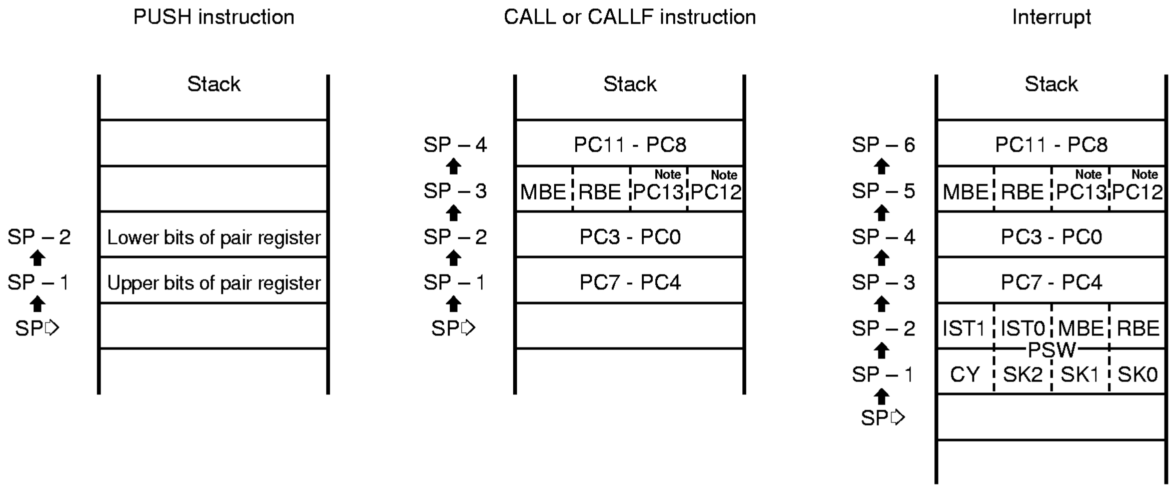
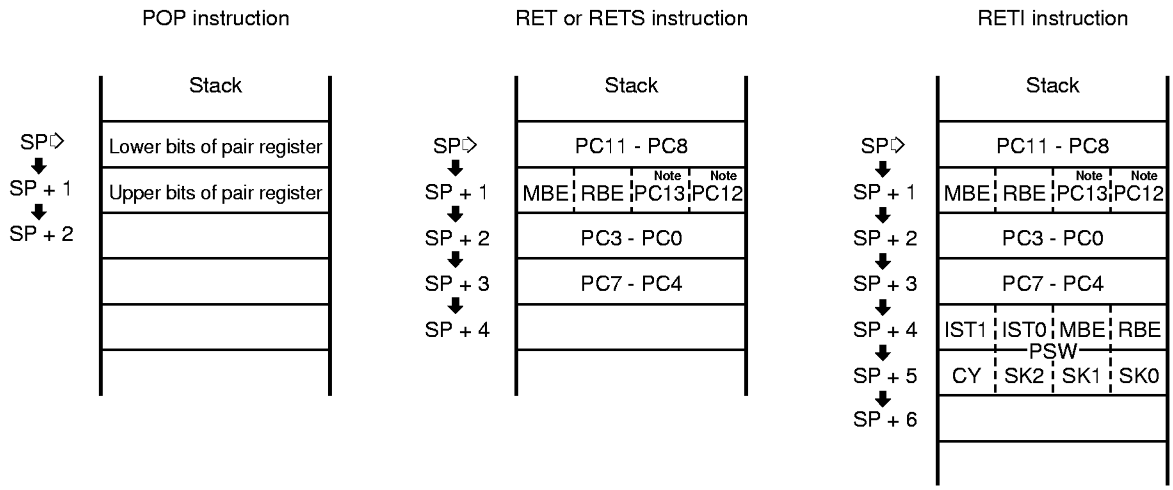


Figure 4-13. Data Restored from the Stack Memory (Mk I Mode)



Note PC12 and PC13 are 0 in the µPD750104. PC13 is 0 in the µPD750106 and µPD750108.

Figure 4-14. Data Saved to the Stack Memory (Mk II Mode)

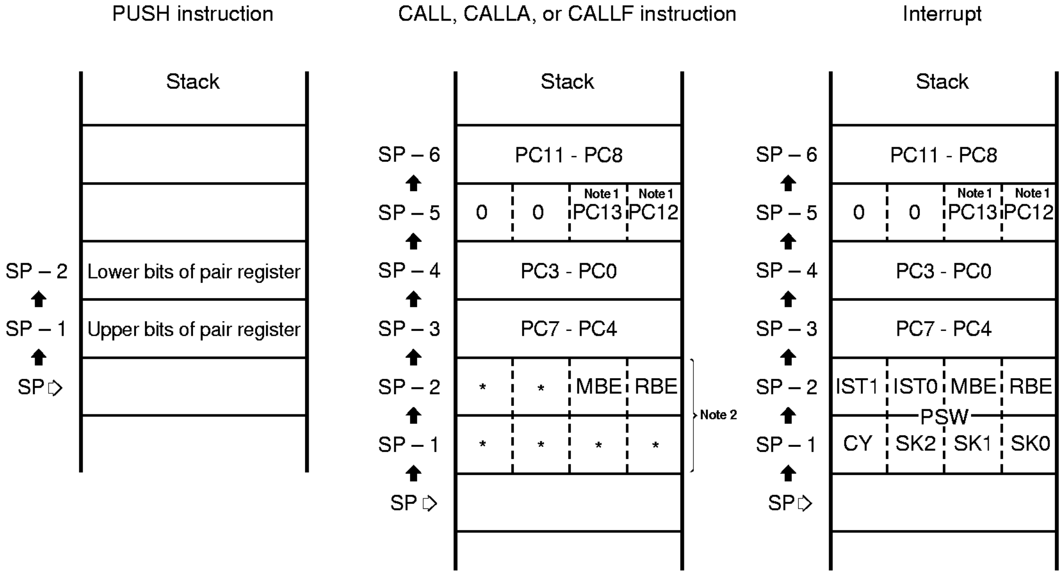
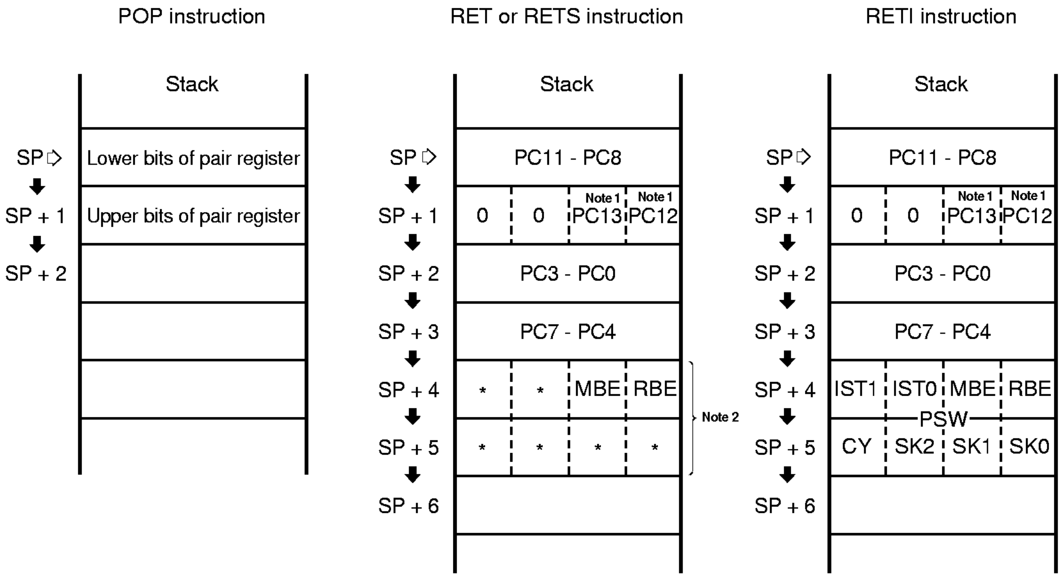


Figure 4-15. Data Restored from the Stack Memory (Mk II Mode)



Notes 1. PC12 and PC13 are 0 in the µPD750104. PC13 is 0 in the µPD750106 and µPD750108.

2. PSW bits other than MBE and RBE are not saved or restored.

Remark * indicates an undefined bit.

4.8 PROGRAM STATUS WORD (PSW): 8 BITS

The program status word (PSW) consists of various flags closely associated with processor operations.

The PSW is mapped to addresses FB0H and FB1H in data memory space. Four bits at address FB0H can be manipulated with a memory manipulation instruction.

Figure 4-16. Program Status Word Format

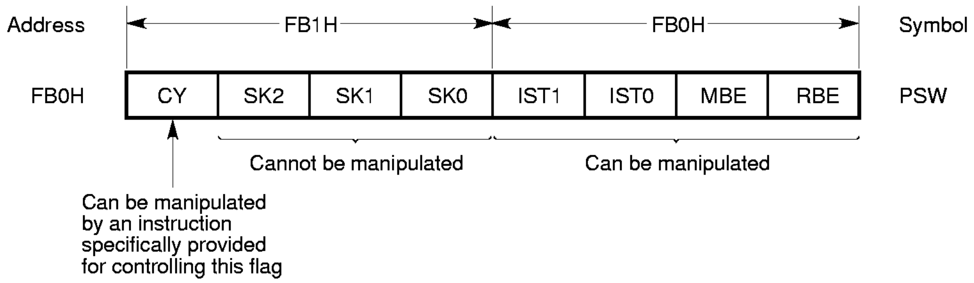


Table 4-3. PSW Flags Saved/Restored in Stack Operation

		Saved/restored flag
Save	When a CALL, CALLA, or CALLF instruction is executed	MBE and RBE are saved.
	When a hardware interrupt occurs	All PSW bits are saved.
Restore	When a RET or RETS instruction is executed	MBE and RBE are restored.
	When a RETI is executed	All PSW bits are restored.

(1) Carry flag (CY)

The carry flag is a 1-bit flag used to store information about an overflow or underflow that occurs when an arithmetic operation with a carry (ADDC, SUBC) is executed.

The carry flag functions as a bit accumulator, and therefore can be used to store the result of a Boolean algebra operation performed on the CY and a bit at a specified data memory bit address.

The carry flag is manipulated using special instructions, independently of the other PSW bits.

A $\overline{\text{RESET}}$ signal causes the carry flag to be undefined.

Table 4-4. Carry Flag Manipulation Instructions

	Instruction (mnemonic)	Carry flag operation/processing
Instruction dedicated to carry flag manipulation	SET1 CY CLR1 CY NOT1 CY SKT CY	Sets CY to 1. Clears CY to 0. Inverts the state of CY. Skips if CY is 1.
Bit transfer instruction	MOV1 mem*.bit, CY MOV1 CY, mem*.bit	Transfers the state of CY to a specified bit. Transfers the state of a specified bit to CY.
Bit Boolean instruction	AND1 CY, mem*.bit OR1 CY, mem*.bit XOR1 CY, mem*.bit	ANDs, ORs, or XORs CY with a specified bit, then sets the result in CY.
Interrupt handling	Interrupt execution	Saves CY and all other PSW bits to stack memory in parallel.
	RETI	Restores CY together with the other PSW bits from stack memory in parallel.

Remark mem*.bit represents the following bit addressing:

- fmem.bit
- pmem.@L
- @H+mem.bit

Example Bit 3 at address 3FH is ANDed with P33, then the result is set in P50.

```
MOV     H,#3H          ; Set the high-order 4 bits of the address in H register
MOV1    CY,@H+0FH.3    ; CY <– bit 3 at 3FH
AND1    CY,PORT3.3     ; CY <– CY∧P33
MOV1    PORT5.0,CY     ; P50 <– CY
```

(2) Skip flags (SK2, SK1, SK0)

The skip flags are used to store skip status, and are automatically set or reset when the CPU executes an instruction.

The user cannot directly manipulate these flags by specifying an operand.

(3) Interrupt status flag (IST1, IST0)

The interrupt status flag is a 2-bit flag used to store the status of processing being performed.

See **Table 6-3** for details.

Table 4-5. Information Indicated by the Interrupt Status Flag

IST1	IST0	Status of processing	Processing and interrupt control being performed
0	0	Status 0	Normal program processing is being performed. Any interrupts are acceptable.
0	1	Status 1	A lower- or higher-priority interrupt is being serviced. Higher-priority interrupts are acceptable.
1	0	Status 2	A higher-priority interrupt is being serviced. No interrupts are acceptable.
1	1	—	Not to be set

The interrupt priority control circuit (**Figure 6-1**) checks this flag to control multiple interrupts.

The contents of the IST1 and IST0 are saved as part of the PSW to stack memory if an interrupt is accepted, then are automatically set to a one-step higher status. The RETI instruction restores the contents present before an interrupt occurs.

The interrupt status flag can be manipulated using a memory manipulation instruction, and the status of processing being performed can be changed by program control.

Caution The user must always disable interrupts with the DI instruction before manipulating this flag, and must enable interrupts with the EI instruction after manipulating this flag.

(4) Memory bank enable flag (MBE)

The memory bank enable flag is a 1-bit flag used to specify the address information generation mode for the high-order four bits of a 12-bit data memory address.

The MBE can be set or reset any time with a bit manipulation instruction, regardless of memory bank setting.

When the MBE is set to 1, the data memory address space is expanded, allowing all data memory space to be addressed.

When the MBE is reset to 0, the data memory address space is fixed, regardless of MBS setting. (See **Figure 3-2**.)

A $\overline{\text{RESET}}$ signal automatically initializes the MBE by setting the MBE to the content of bit 7 at program memory address 0.

In vectored interrupt processing, the MBE is automatically set to the content of bit 7 in the vector address table for servicing the interrupt.

Usually, the MBE is set to 0 in interrupt processing, and static RAM in memory bank 0 is used.

(5) Register bank enable flag (RBE)

The register bank enable flag is a 1-bit flag used to determine whether to expand the general register bank configuration.

The RBE can be set or reset any time with a bit manipulation instruction, regardless of memory bank setting.

When the RBE is set to 1, a set of general registers can be selected from register banks 0 to 3, depending on the setting of the register bank select register (RBS).

When the RBE is reset to 0, register bank 0 is always selected as general registers, regardless of the setting of the RBS.

A $\overline{\text{RESET}}$ signal automatically initializes the RBE by setting the RBE to the state of bit 6 at program memory address 0.

When a vectored interrupt occurs, the RBE is automatically set to the state of bit 6 in the vector address table for servicing the interrupt. Usually, the RBE is set to 0 in interrupt processing. Register bank 0 is used for 4-bit processing, and register banks 0 and 1 are used for 8-bit processing.

4.9 BANK SELECT REGISTER (BS)

The bank select register (BS) consists of a register bank select register (RBS) and memory bank select register (MBS), which specify a register bank and memory bank to be used, respectively.

The RBS and MBS are set using the SEL RBn instruction and SEL MBn instruction, respectively.

The contents of the BS can be saved to or restored from a stack memory eight bits at a time by using the PUSH BS/POP BS instruction.

Figure 4-17. Bank Select Register Format

Address	F83H				F82H				Symbol
F82H	MBS3	MBS2	MBS1	MBS0	0	0	RBS1	RBS0	BS

(1) Memory bank select register (MBS)

The memory bank select register is a 4-bit register used to store the high-order four bits of a 12-bit data memory address. The contents of this register specify a memory bank to be accessed. The μPD750108 allows memory banks 0, 1, and 15 only to be specified.

The MBS is set with the SEL MBn instruction (n = 0, 1, 15).

Figure 3-2 shows the range of addressing using MBE and MBS settings.

A $\overline{\text{RESET}}$ signal initializes the MBS to 0.

(2) Register bank select register (RBS)

The register bank select register specifies a register bank to be used as general registers; a register bank can be selected from register banks 0 to 3.

The RBS is set by the SEL RBn instruction (n = 0 to 3).

A $\overline{\text{RESET}}$ signal initializes the RBS to 0.

Table 4-6. Register Bank to Be Selected with the RBE and RBS

RBE	RBS				Register bank
	3	2	1	0	
0	0	0	x	x	Bank 0 is always selected.
1	0	0	0	0	Bank 0 is selected.
			0	1	Bank 1 is selected.
			1	0	Bank 2 is selected.
			1	1	Bank 3 is selected.

Always 0

x: Don't care

CHAPTER 5 PERIPHERAL HARDWARE FUNCTIONS

5.1 DIGITAL I/O PORTS

The µPD750108 employs the memory mapped I/O method. Thus, all input/output ports are mapped on the data memory space.

Figure 5-1. Data Memory Addresses of Digital Ports

Address	3	2	1	0	
FF0H	P03	P02	P01	P00	PORT 0
FF1H	P13	P12	P11	P10	PORT 1
FF2H	P23	P22	P21	P20	PORT 2
FF3H	P33	P32	P31	P30	PORT 3
FF4H	P43	P42	P41	P40	PORT 4
FF5H	P53	P52	P51	P50	PORT 5
FF6H	P63	P62	P61	P60	PORT 6
FF7H	P73	P72	P71	P70	PORT 7
FF8H	–	–	P81	P80	PORT 8

Remark Some I/O parts can be used as static RAM.

Input/output port manipulation instructions are as listed in Table 5-2. Ports 4 to 7 can be manipulated not only in 4-bit units, but also in 8-bit or 1-bit units so that these ports can be controlled in various ways.

Example 1. To test the condition of P13 and output different values to ports 4 and 5 according to the test result:

```
SKT   PORT1. 3   ; Skips if bit 3 of port 1 is 1
MOV   XA, #18H   ; XA <– 18H  ⎫
MOV   XA, #14H   ; XA <– 14H  ⎭  String-effect instructions
SEL   MB15       ; Or CLR1 MBE
OUT   PORT4, XA  ; Port 5, 4 <– XA
```

2.

```
SET1  PORT4. @L; Sets the bit(s) specified by the L register, in ports 4 to 7, to 1.
```

5.1.1 Types, Features, and Configurations of Digital I/O Ports

Table 5-1 lists the types of digital I/O ports.

Figures 5-2 to 5-6 show the configurations of the ports.

Table 5-1. Types and Features of Digital Ports

Port name (symbol)	Function	Operation and feature	Remarks
PORT0	4-bit I/O	Allows read and test at any time regardless of the operation modes of another functions assigned to these pins.	Also used as INT4, SCK, SO/SB0, and SI/SB1.
PORT1			Also used as INT0 to INT2 and TI0.
PORT3[Note 1]	4-bit I/O	Allows input or output mode setting bit by bit.	Also used as MD0 to MD3[Note 2].
PORT6			Also used as KR0 to KR3.
PORT2		Ports 6 and 7 can be paired, allowing data I/O in units of 8 bits. Allows input or output mode setting in units of 4 bits.	Also used as PTO0, PTO1, PCL, and BUZ.
PORT7			Also used as KR4 to KR7.
PORT4[Note 1] PORT5[Note 1]	4-bit I/O (N-ch open-drain; can withstand 13 V)	Allows input or output mode setting in units of 4 bits. Ports 4 and 5 can be paired, allowing data I/O in units of 8 bits.	Whether to use pull-up resistors can be specified bit by bit with a mask option[Note 3].
PORT8	2-bit I/O	Allows input or output mode setting in units of 2 bits.	—

Notes 1. Can directly drive the LED.

2. Only for the µPD75P0116.

3. The µPD75P0116 does not have a mask option and cannot be connected with a pull-up resistor.

P10 is also used as an external vectored interrupt input pin. This input is provided with a noise eliminator. (See **Section 6.3** for details.)

When the $\overline{\text{RESET}}$ signal is generated, output latches of ports 2 to 8 are cleared to 0 and the output buffer is turned off so that these ports are in the input mode.

Figure 5-2. Configurations of Ports 0 and 1

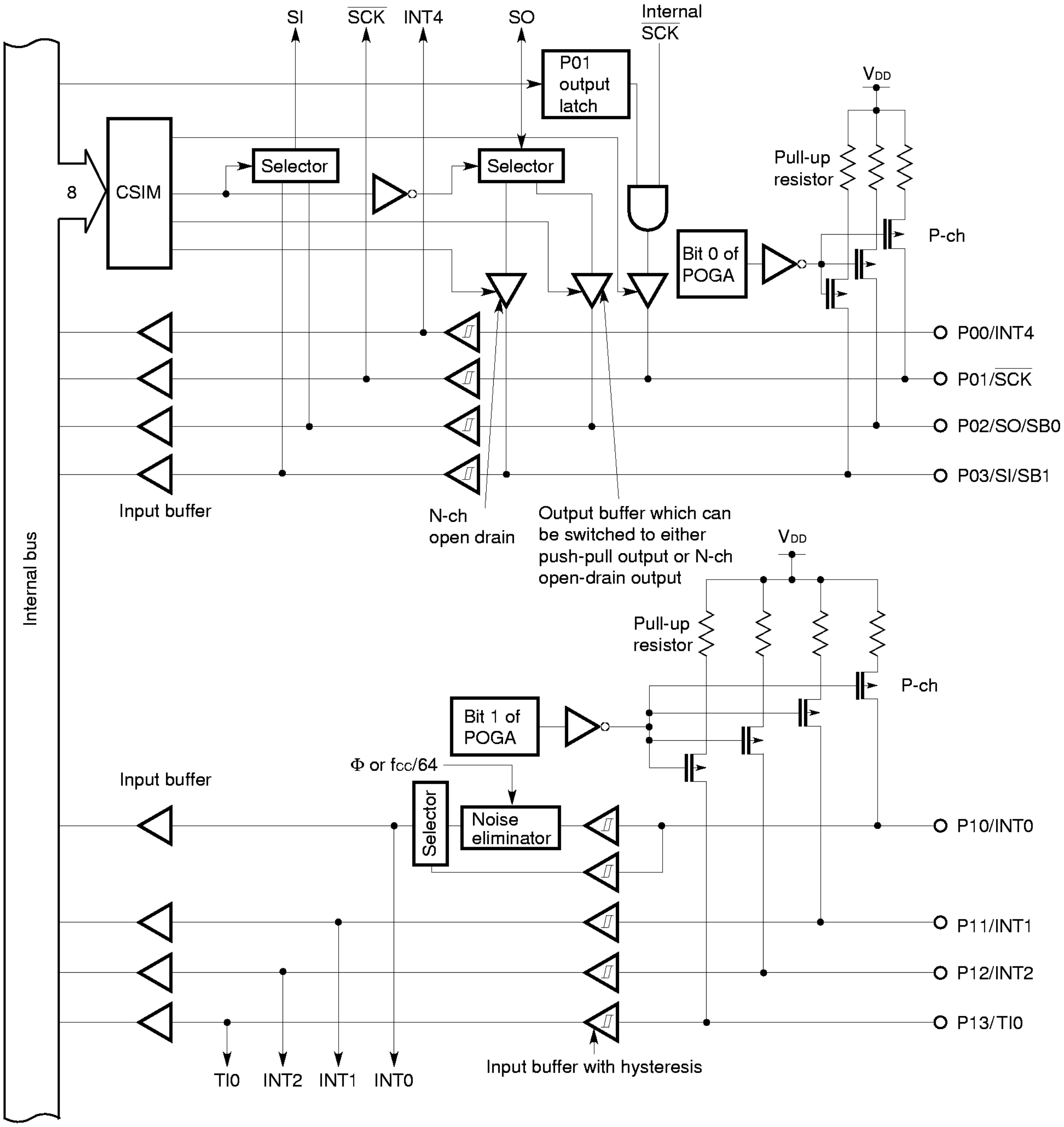
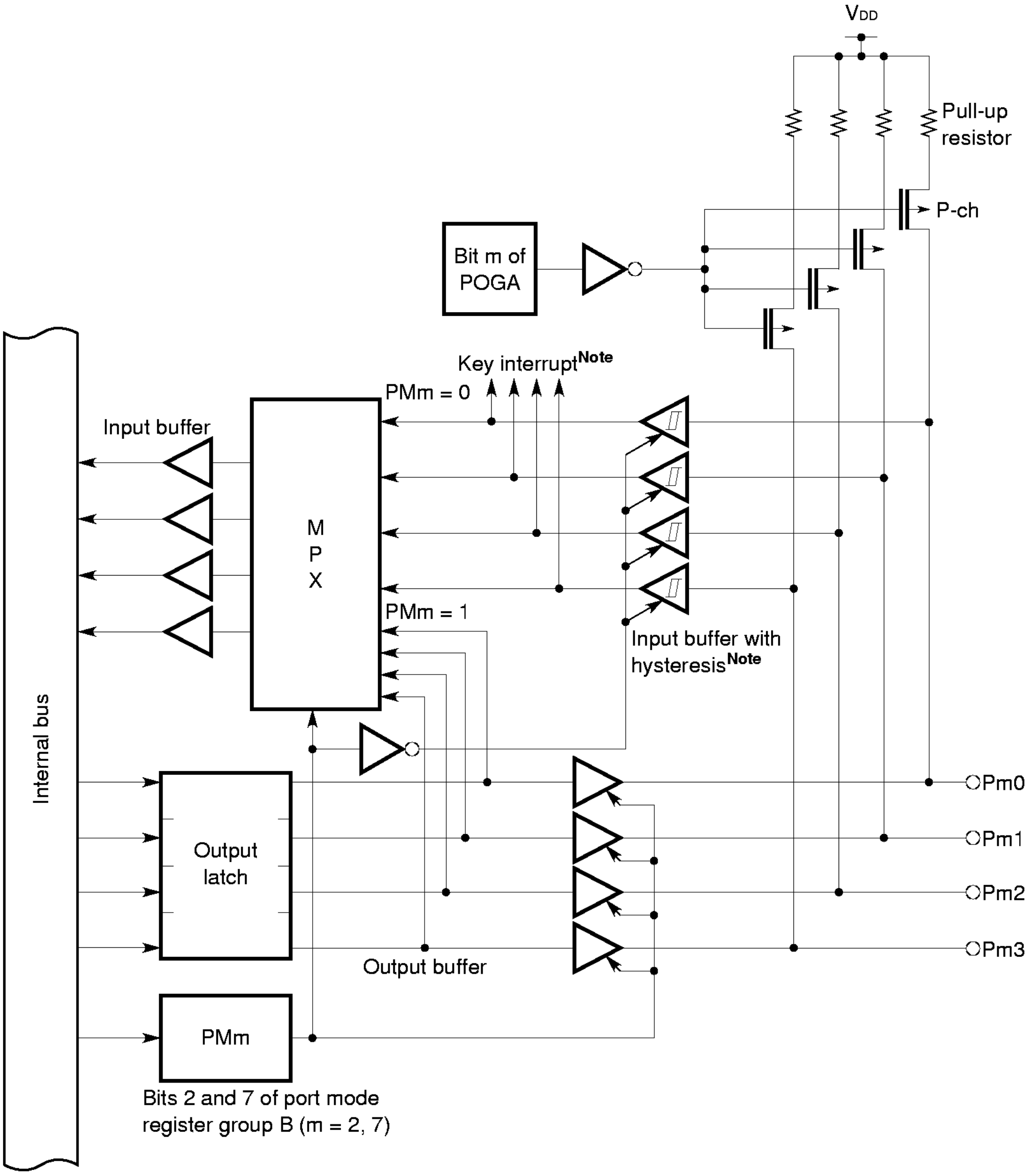
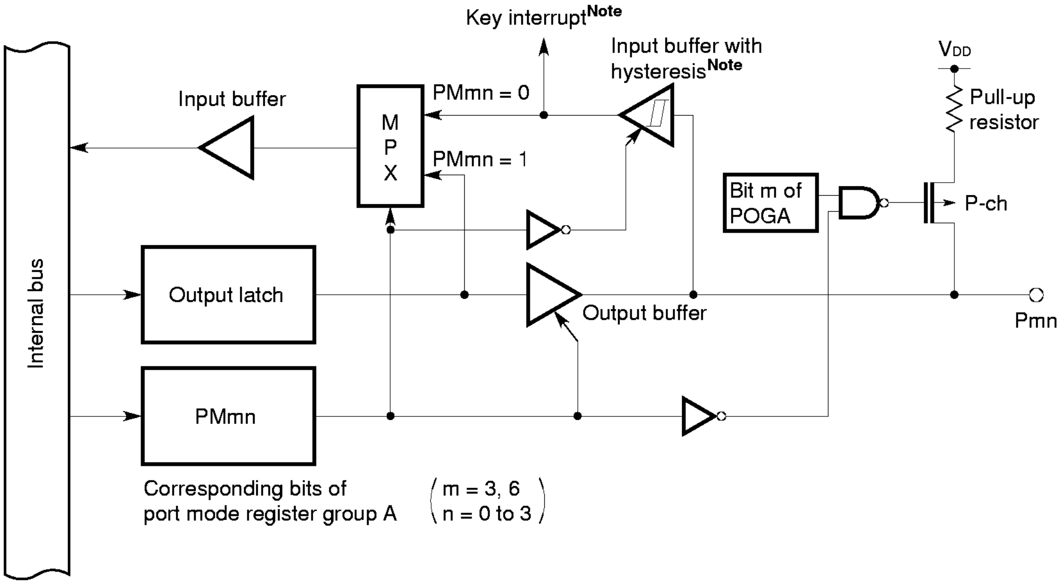


Figure 5-3. Configurations of Ports 2 and 7



Note For port 7 only

Figure 5-4. Configurations of Ports 3n and 6n (n = 0 to 3)



Note For port 6n only

Figure 5-5. Configurations of Ports 4 and 5

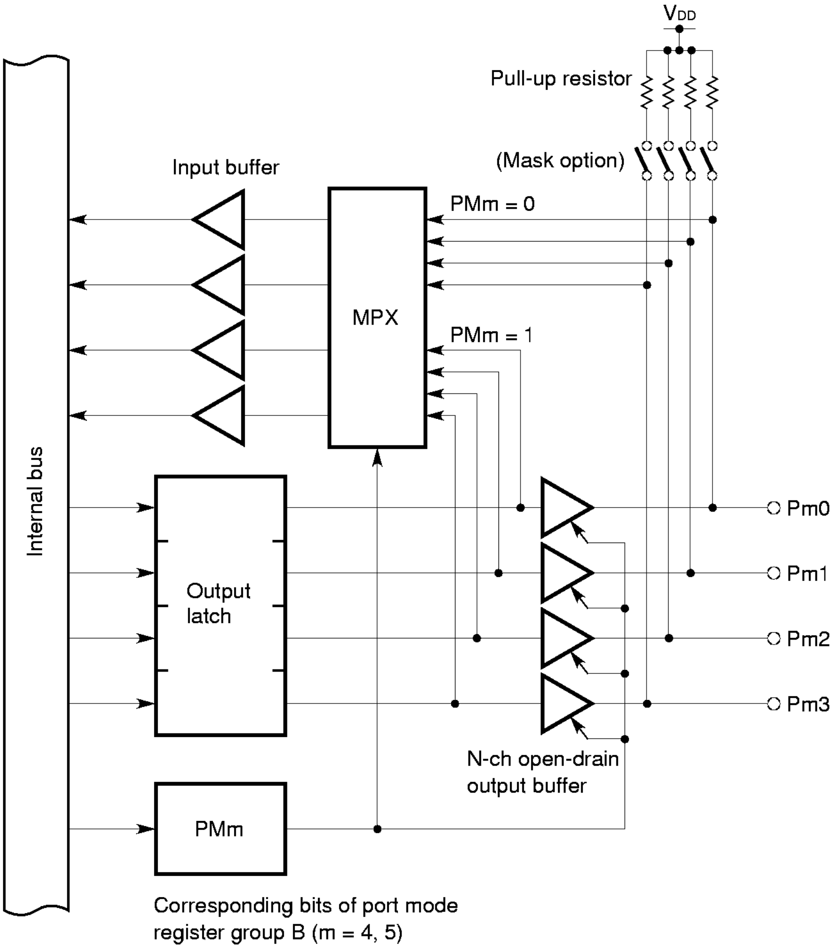
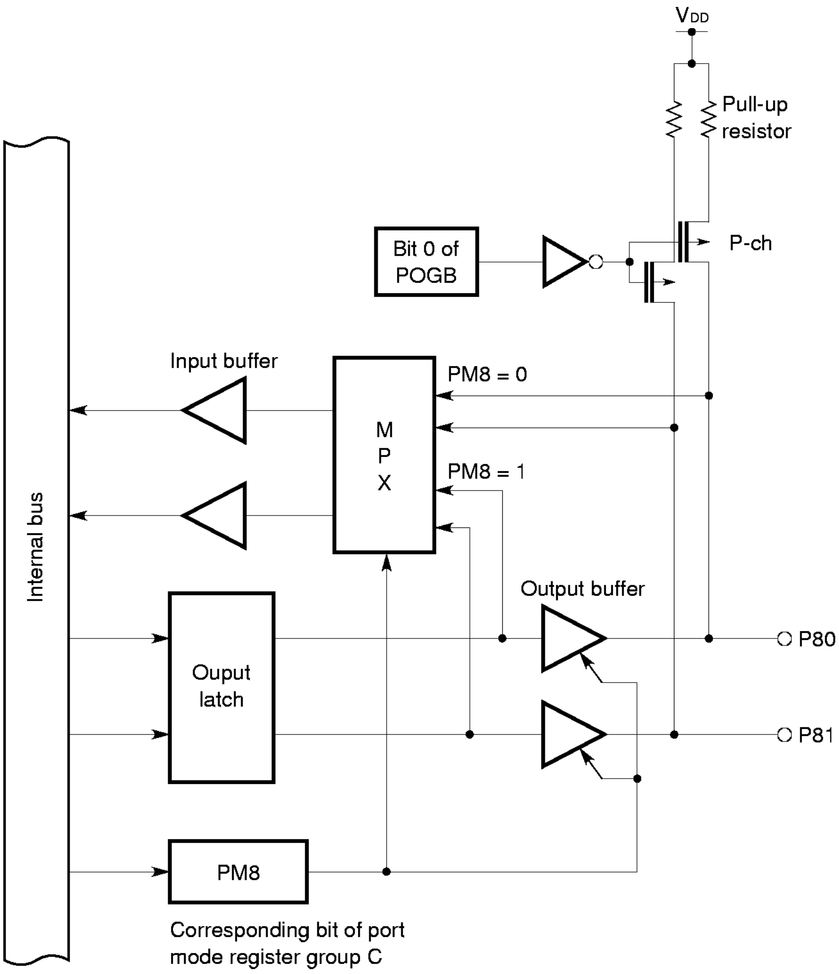


Figure 5-6. Configuration of Port 8



5.1.2 I/O Mode Setting

The I/O mode of each I/O port is set by the port mode register as shown in Figure 5-7. The I/O modes of ports 3 and 6 can be set bit by bit by port mode register group A (PMGA). The I/O modes of ports 2, 4, 5, and 7 can be set in units of four bits by port mode register group B (PMGB). The I/O mode of port 8 can be set in units of two bits by port mode register group C (PMGC).

Each port functions as an input port when the corresponding bit of the port mode register is set to 0, and functions as an output port when the same corresponding bit is set to 1.

When the output mode is selected by the port mode register, the contents of the output latch appear on the output pins, and so the contents of the output latch must be changed to a desired value before the output mode is set.

An 8-bit memory manipulation instruction is used to set port mode register group A, B, or C.

A $\overline{\text{RESET}}$ signal clears all bits of each port mode register to 0. This means that the output buffers are set off, and all ports are placed in the input mode.

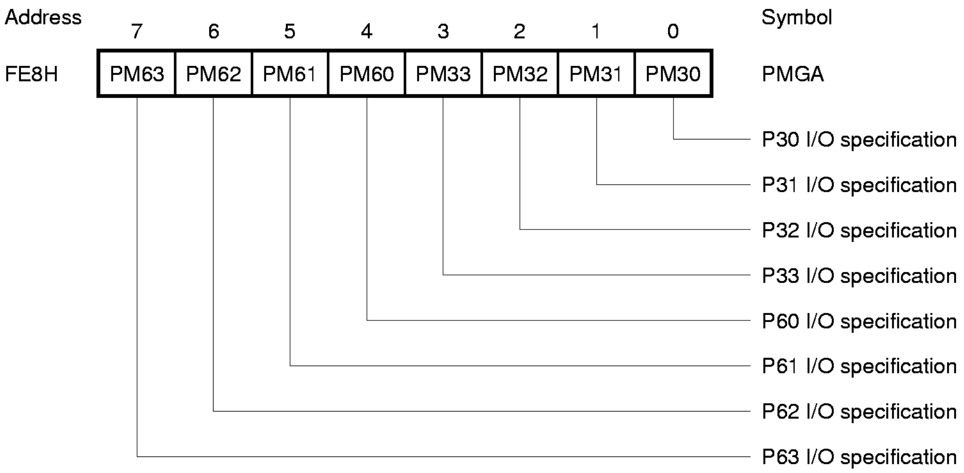
Example P30, P31, P62, and P63 are used as input pins, and P32, P33, P60, and P61 are used as output pins.

```
CLR1    MBE        ; or SEL MB15
MOV     XA,#3CH
MOV     PMGA,XA
```

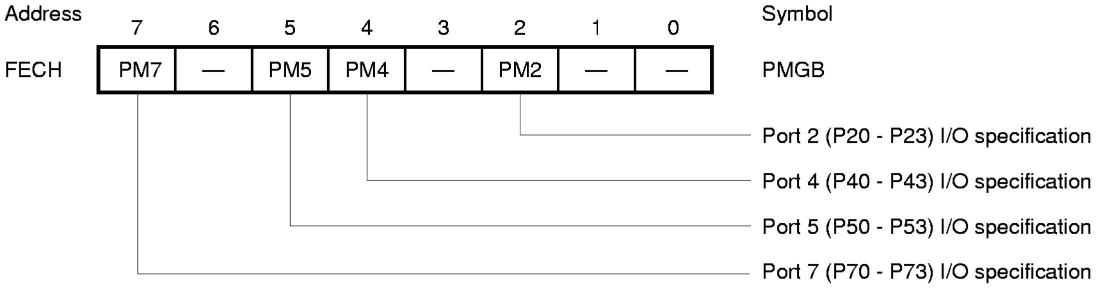
Figure 5-7. Formats of Port Mode Registers

	Contents of specification
0	Input mode (Output buffer off)
1	Output mode (Output buffer on)

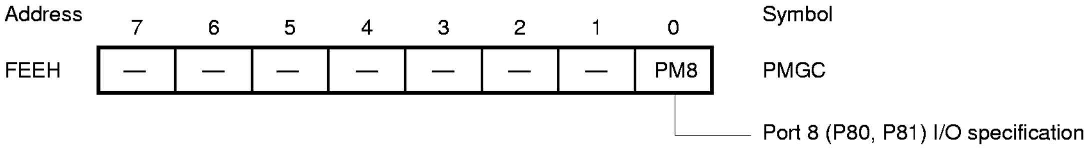
Port mode register group A



Port mode register group B


Address
7
6
5
4
3
2
1
0
Symbol
FECH
PM7
—
PM5
PM4
—
PM2
—
—
PMGB
Port 2 (P20 - P23) I/O specification
Port 4 (P40 - P43) I/O specification
Port 5 (P50 - P53) I/O specification
Port 7 (P70 - P73) I/O specification

Port mode register group C



5.1.3 Digital I/O Port Manipulation Instructions

All I/O ports contained in the µPD750108 are mapped to data memory space, so that all data memory manipulation instructions can be used. Table 5-3 lists the instructions that are particularly useful for I/O pin manipulation and their application ranges.

(1) Bit manipulation instructions

For digital I/O ports PORT0 to PORT8, specific address bit direct addressing (fmem.bit) and specific address bit register indirect addressing (pmem.@L) can be used. This means that bit manipulation can be freely performed for these ports regardless of MBE and MBS settings.

Example P50 is ORed with P41, then the result is output to P61.

```
        SET1    CY              ; CY <– 1
        AND1    CY,PORT5.0      ; CY <– CY∧P50
        OR1     CY,PORT4.1      ; CY <– CY∨P41
        SKT     CY
        BR      CLRP
        SET1    PORT6.1         ; P61 <– 1
          :
CLRP :  CLR1    PORT6.1         ; P61 <– 0
```

(2) 4-bit manipulation instructions

All 4-bit memory manipulation instructions including the IN, OUT, MOV, XCH, ADDS, and INCS instructions can be used. However, before these instructions can be executed, memory bank 15 must be selected.

Examples **1.** The contents of the accumulator are output to port 3.

```
SEL     MB15            ; or CLR1 MBE
OUT     PORT3,A
```

2. The value of the accumulator is added to the data output on port 5, then the result is output.

```
SET1    MBE
SEL     MB15
MOV     HL,#PORT5
ADDS    A,@HL           ; A <– A+PORT5
NOP
MOV     @HL,A           ; PORT5 <– A
```

3. Whether the data on port 4 is greater than the value of the accumulator is tested.

```
SET1    MBE
SEL     MB15
MOV     HL,#PORT4
SUBS    A,@HL           ; A < PORT4
BR      NO              ; NO
                        ; YES
```

(3) 8-bit manipulation instructions

The MOV, XCH, and SKE instructions as well as the IN and OUT instructions can be used for ports 4 and 5 that allow 8-bit manipulation. As with 4-bit manipulation, memory bank 15 must be selected in advance.

Example The data contained in the BC register pair is output on the output port specified by 8-bit data applied to ports 4 and 5.

```
SET1   MBE
SEL    MB15
IN     XA,PORT4     ; XA <– ports 5,4
MOV    HL,XA        ; HL <– XA
MOV    XA,BC        ; XA <– BC
MOV    @HL,XA       ; Port (L) <– XA
```

Table 5-2. I/O Pin Manipulation Instructions

<table>
<tr><th>PORT
Instruction</th><th>PORT 0</th><th>PORT 1</th><th>PORT 2</th><th>PORT 3</th><th>PORT 4</th><th>PORT 5</th><th>PORT 6</th><th>PORT 7</th><th>PORT 8</th></tr>
<tr><td>IN A, PORTn Note 1</td><td colspan="9">○</td></tr>
<tr><td>IN XA, PORTn Note 1</td><td colspan="2">—</td><td colspan="2">—</td><td colspan="2">○</td><td colspan="2">○</td><td>—</td></tr>
<tr><td>OUT PORTn, A Note 1</td><td colspan="2">—</td><td colspan="7">○</td></tr>
<tr><td>OUT PORTn, XA Note 1</td><td colspan="2">—</td><td colspan="2">—</td><td colspan="2">○</td><td colspan="2">○</td><td>—</td></tr>
<tr><td>SET1 PORTn.bit</td><td colspan="2">—</td><td colspan="7">○</td></tr>
<tr><td>SET1 PORTn.@L Note 2</td><td colspan="2">—</td><td colspan="7">○</td></tr>
<tr><td>CLR1 PORTn.bit</td><td colspan="2">—</td><td colspan="7">○</td></tr>
<tr><td>CLR1 PORTn.@L Note 2</td><td colspan="2">—</td><td colspan="7">○</td></tr>
<tr><td>SKT PORTn.bit</td><td colspan="9">○</td></tr>
<tr><td>SKT PORTn.@L Note 2</td><td colspan="9">○</td></tr>
<tr><td>SKF PORTn.bit</td><td colspan="9">○</td></tr>
<tr><td>SKF PORTn.@L Note 2</td><td colspan="9">○</td></tr>
<tr><td>MOV1 CY, PORTn.bit</td><td colspan="9">○</td></tr>
<tr><td>MOV1 CY, PORTn.@L Note 2</td><td colspan="9">○</td></tr>
<tr><td>MOV1 PORTn.bit,CY</td><td>—</td><td>—</td><td colspan="7">○</td></tr>
<tr><td>MOV1 PORTn.@L,CY Note 2</td><td>—</td><td>—</td><td colspan="7">○</td></tr>
<tr><td>AND1 CY, PORTn.bit</td><td colspan="9">○</td></tr>
<tr><td>AND1 CY, PORTn.@L Note 2</td><td colspan="9">○</td></tr>
<tr><td>OR1 CY, PORTn.bit</td><td colspan="9">○</td></tr>
<tr><td>OR1 CY, PORTn.@L Note 2</td><td colspan="9">○</td></tr>
<tr><td>XOR1 CY, PORTn.bit</td><td colspan="9">○</td></tr>
<tr><td>XOR1 CY, PORTn.@L Note 2</td><td colspan="9">○</td></tr>
</table>

Notes 1. MBE = 0 or (MBE = 1, MBS = 15) must be set before execution.

2. The low-order two bits of an address and bit address are indirectly specified using the L register.

5.1.4 Digital I/O Port Operation

When a data memory manipulation instruction is executed for a digital I/O port, the operation of the port and pins depends on the I/O mode setting (Table 5-3). This is because data taken in on the internal bus is the data input from the pins in the input mode, or the output latch data in the output mode, as obvious from the configurations of I/O ports.

(1) Operation when the input mode is set

Data from each pin is manipulated when a test instruction such as the SKT instruction ,a bit input instruction such as MOV1,or an instruction for taking in port data on the internal bus in units of four or eight bits (such as an IN, OUT, arithmetic/logical or comparison instruction) is executed.

When an instruction (the OUT or MOV instruction) is executed to transfer the contents of the accumulator to a port in units of four or eight bits, the data of the accumulator is latched in the output latch, with the output buffers kept off.

When the XCH instruction is executed, the data on each pin is loaded into the accumulator, and the data in the accumulator is latched in the output latch, with the output buffers kept off.

When the INCS instruction is executed, the 4-bit data existing on the pins plus 1 is latched in the output latch, with the output buffers kept off.

When an instruction such as the SET1, CLR1, or SKTCLR instruction is executed to rewrite a data memory bit, the output latch data of the specified bit can be rewritten according to the instruction, but the states of the other output latch bits are undefined.

(2) Operation when the output mode is set

When a test instruction or instruction for taking in port data on the internal bus in units of four or eight bits is executed, output latch data is manipulated.

When an instruction is executed to transfer the contents of the accumulator in units of four or eight bits, the output latch data is rewritten, and is output on the pins.

When the XCH instruction is executed, the output latch data is transferred to the accumulator. The contents of the accumulator are latched in the output latches, and are output on the pins.

When the INCS instruction is executed, the contents of the output latch incremented by 1 are latched in the output latch, and are output on the pins.

When a bit output instruction is executed, the specified bit of the output latch is rewritten, and is output on the pin.

Table 5-3. Operations by I/O Port Manipulation Instructions

Instruction	Port and pin operation	
	Input mode	Output mode
SKT <1> SKF <1>	Pin data is tested.	Output latch data is tested.
MOV1 CY, <1>	Pin data is transferred to CY.	Output latch data is transferred to CY.
AND1 CY, <1> OR1 CY, <1> XOR1 CY, <1>	An operation is performed on pin data and CY.	An operation is performed on output latch data and CY.
IN A,PORTn IN XA,PORTn MOV A,@HL MOV XA,@HL	Pin data is transferred to the accumulator.	Output latch data is transferred to the accumulator.
ADDS A,@HL ADDC A,@HL SUBS A,@HL SUBC A,@HL AND A,@HL OR A,@HL XOR A,@HL	An operation is performed on pin data and the accumulator.	An operation is performed on output latch data and the accumulator.
SKE A,@HL SKE XA,@HL	Pin data is compared with the accumulator.	Output latch data is compared with the accumulator.
OUT PORTn,A OUT PORTn,XA MOV @HL,A MOV @HL,XA	Accumulator data is transferred to the output latch (with the output buffers kept off).	Accumulator data is transferred to the output latch and is output on the pins.
XCH A,PORTn XCH XA,PORTn XCH A,@HL XCH XA,@HL	Pin data is transferred to the accumulator, and accumulator data is transferred to the output latch (with the output buffers kept off).	Data is exchanged between the output latch and accumulator.
INCS PORTn INCS @HL	Pin data incremented by 1 is latched in the output latch.	Output latch data is incremented by 1.
SET1 <1> CLR1 <1> MOV1 <1> ,CY SKTCLR <1>	The output latch data of a specified bit is rewritten, but the output latch data of the other bits is undefined.	The output pin state is modified according to the instruction.

<1> : Represents an addressing mode PORTn.bit or PORTn.@L.

5.1.5 Specification of Built-in Pull-Up Resistors

A pull-up resistor can be contained at each port pin of the µPD750108 (except for P00). Whether to use the pull-up resistor can be specified by software (for some pins) or a mask option (for the other pins).

Table 5-4 shows how a built-in pull-up resistor is specified for each port pin. The built-in pull-up resistor is connected by software in the format shown in Figure 5-8.

Table 5-4. Specification of Built-in Pull-Up Resistors

Port (pin name)	Pull-up resistor incorporation specification method	Bit of POGA	Bit of POGB
Port 0 (P01-P03)Note	Connection specification by software in 3-bit units	Bit 0	—
Port 1 (P10-P13)	Connection specification by software in 4-bit units	Bit 1	—
Port 2 (P20-P23)		Bit 2	—
Port 3 (P30-P33)		Bit 3	—
Port 6 (P60-P63)		Bit 6	—
Port 7 (P70-P73)		Bit 7	—
Port 4 (P40-P43)	Incorporation specification by mask option in 1-bit units	—	—
Port 5 (P50-P53)			
Port 8 (P80, P81)	Connection specification by software in 2-bit units	—	Bit 0

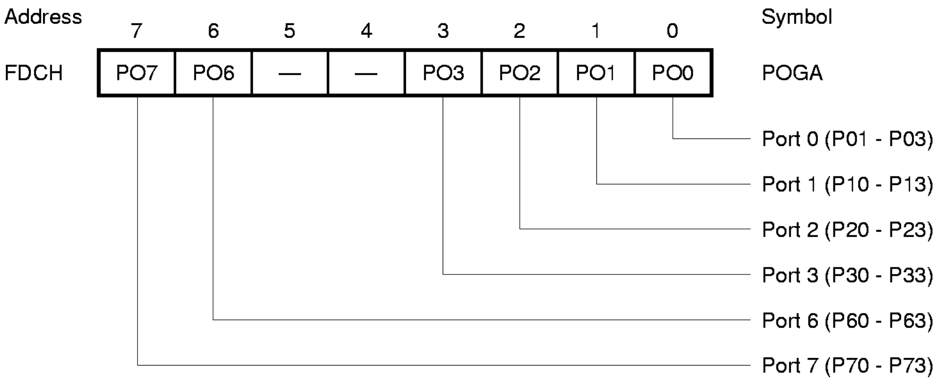
Note The P00 pin cannot specify connection of a built-in pull-up resistor.

Remark The port pins of the µPD75P0116 are not connected to a pull-up resistor by mask option, and are always open.

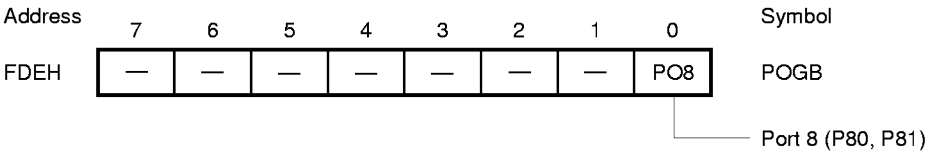
Figure 5-8. Pull-Up Resistor Specification Register Format

	Specification contents
0	Built-in pull-up resistor not connected
1	Built-in pull-up resistor connected

Pull-up resistor specification register group A



Pull-up resistor specification register group B



5.1.6 I/O Timing of Digital I/O Ports

Figure 5-9 shows the timing of data output to an output latch and the timing of taking in pin data or output latch data on the internal bus.

Figure 5-10 shows an ON timing chart when a built-in pull-up resistor is connected to a port pin by software.

Figure 5-9. I/O Timing Chart of Digital I/O Ports (1/2)

(a) When data is input by a 1-machine cycle instruction

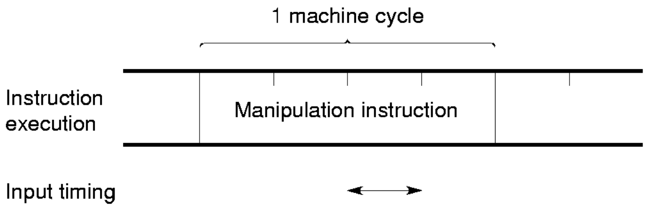
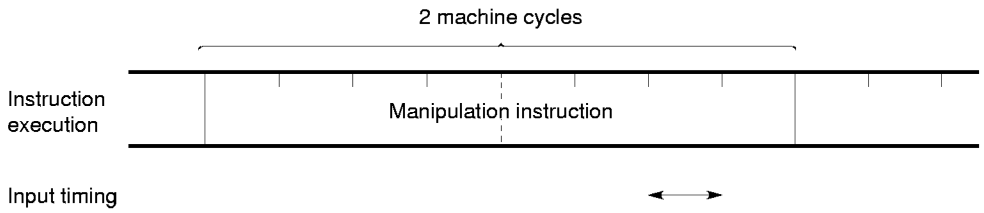
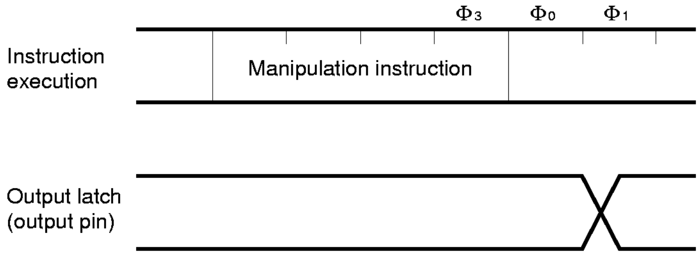


Figure 5-9. I/O Timing Chart of Digital I/O Ports (2/2)

(b) When data is input by a 2-machine cycle instruction



(c) When data is latched by a 1-machine cycle instruction



(d) When data is latched by a 2-machine cycle instruction

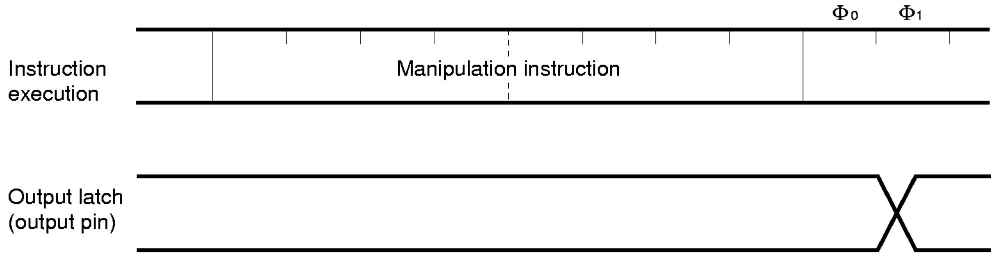
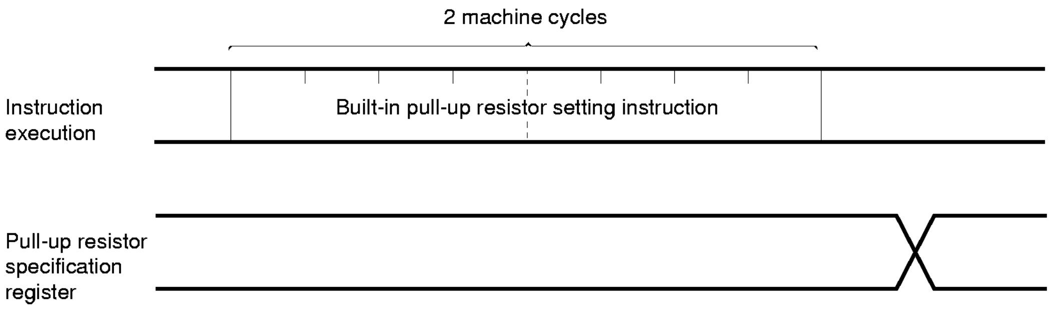


Figure 5-10. ON Timing Chart of Built-in Pull-Up Resistor Connected by Software


2 machine cycles
Instruction execution
Built-in pull-up resistor setting instruction
Pull-up resistor specification register

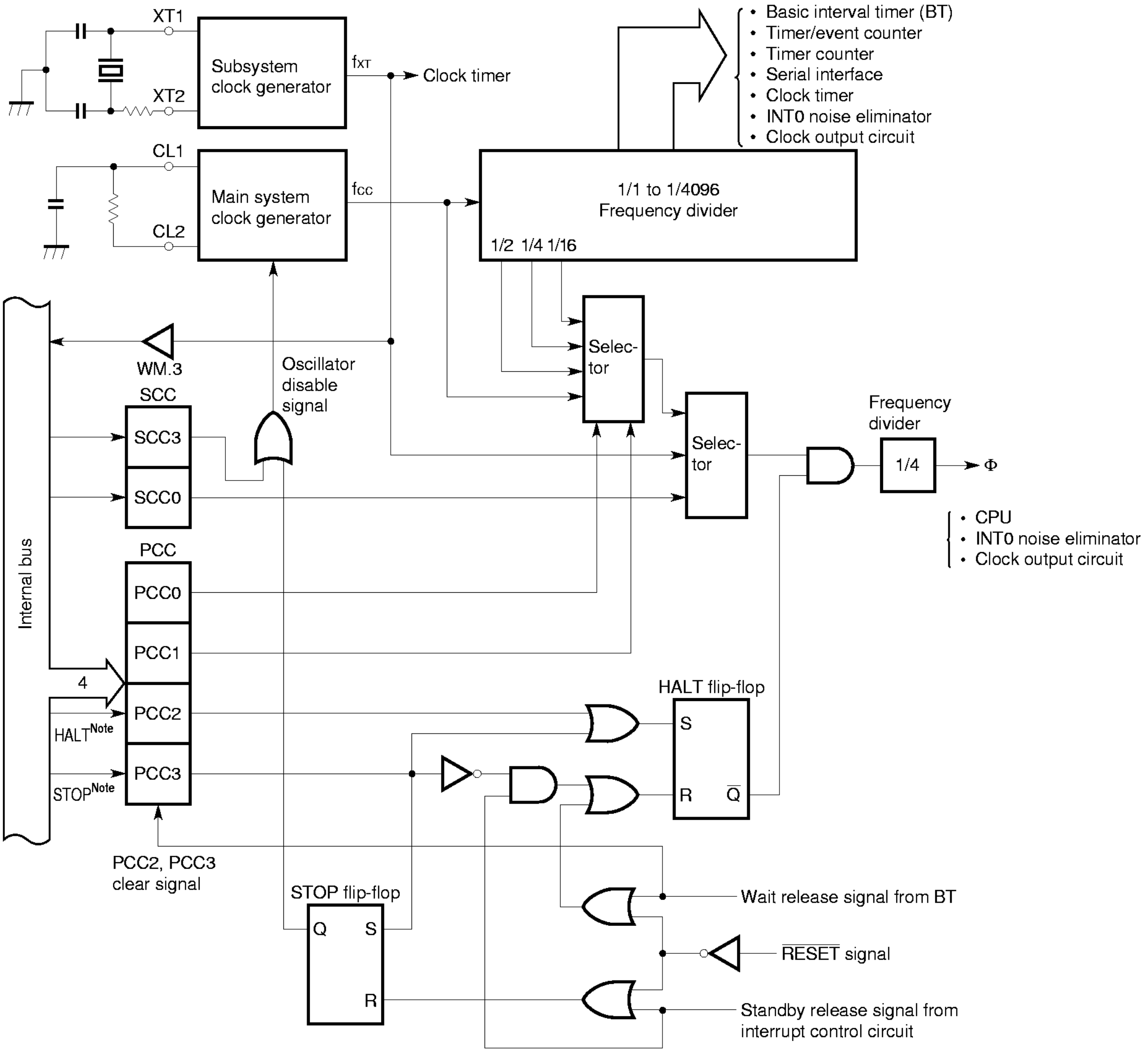
5.2 CLOCK GENERATOR

The clock generator supplies various clock signals to the CPU and peripheral hardware to control the CPU operation mode.

5.2.1 Clock Generator Configuration

Figure 5-11 shows the configuration of the clock generator.

Figure 5-11. Block Diagram of the Clock Generator



Note Instruction execution

Remarks 1. f_{CC}: Main system clock frequency

2. f_{XT}: Subsystem clock frequency

3. Φ = CPU clock

4. PCC: Processor clock control register

5. SCC: System clock control register

6. One clock cycle (t_{CY}) of the CPU clock (Φ) is equal to one machine cycle of an instruction.

5.2.2 Functions and Operations of the Clock Generator

The clock generator generates the following clocks, and controls the CPU operation modes such as the standby mode.

- Main system clock f_{CC}
- Subsystem clock f_{XT}
- CPU clock Φ
- Clock to peripheral hardware

The operation of the clock generator is determined by the processor clock control register (PCC) and system clock control register (SCC). The function and operation of the clock generator are described in (a) to (g) below.

(a) A $\overline{\text{RESET}}$ signal selects the lowest-speed mode (32 μs at 2 MHz)**Note 1** for the main system clock (PCC = 0, SCC = 0).

(b) When the main system clock is selected, the PCC can be set to select one of four CPU clocks (2, 4, 8, and 32 μs at 2 MHz)**Note 2**.

(c) When the main system clock is selected, the two standby modes, STOP mode and HALT mode, are available.

(d) The SCC can be set to select the subsystem clock for very low-speed, low-current operation (122 μs at 32.768 kHz). The value in the PCC does not affect the CPU clock.

(e) When the subsystem clock is selected, main system clock generation can be stopped with the SCC. In addition, the HALT mode can be used, but the STOP mode cannot be used. (Subsystem clock generation cannot be stopped.)

(f) The clock to be supplied to peripheral hardware is produced by frequency-dividing the main system clock signal. The subsystem clock can directly be supplied only to the clock timer. This enables the clock function and the buzzer output function to continue operating even in the standby state.

(g) When the subsystem clock is selected, the clock timer can continue to operate normally. The serial interface, timer/event counter, and timer counter can continue to operate when the external clock is selected. However, other hardware cannot be used when the main system clock is stopped because they operate with the main system clock.

Notes 1. At f_{CC} = 1 MHz: 64 μs

2. At f_{CC} = 1 MHz: 4, 8, 16, and 64 μs

(1) Processor clock control register (PCC)

The PCC is a 4-bit register for selecting a CPU clock Φ with the low-order two bits and for controlling the CPU operation mode with the high-order two bits (see **Figure 5-12**).

When bit 3 or bit 2 is set to 1, the standby mode is set. When the standby mode is released by the standby release signal, these bits are automatically cleared to return to the normal operation mode. (See **Chapter 7** for details.)

A 4-bit memory manipulation instruction is used to set the low-order two bits of the PCC. (The high-order two bits are set to 0.)

Bit 3 and bit 2 are set to 1 using the STOP instruction and HALT instruction, respectively.

The STOP instruction and HALT instruction can always be executed regardless of MBE setting.

The CPU clock can be selected only while the processor is operated by the main system clock. When the processor is operated by the subsystem clock, the low-order 2 bits of the PCC are invalidated, and $f_{XT}/4$ is automatically set. The STOP instruction can be executed only when the processor is operated by the main system clock.

Examples 1. The machine cycle is entered in highest-speed mode (2 µs at f_{CC} = 2 MHz).

```
SEL  MB15
MOV  A,#0011B
MOV  PCC,A
```

2. The machine cycle is set to 8 µs (at f_{CC} = 1 MHz).

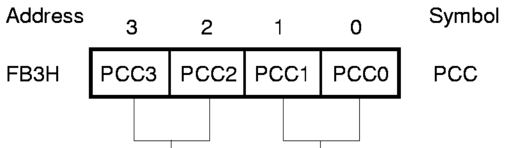
```
SEL  MB15
MOV  A,#0010B
MOV  PCC,A
```

3. The STOP mode is set. (A STOP instruction or HALT instruction must always be followed by an NOP instruction.)

```
STOP
NOP
```

A $\overline{\text{RESET}}$ signal clears the PCC to 0.

Figure 5-12. Format of the Processor Clock Control Register



CPU clock selection bit
(Operation with f_{CC} = 2 MHz)

		SCC3, SCC0 = 00 ( ) is actual frequency at f_{CC} = 2 MHz		SCC3, SCC0 = 01 or 11 ( ) is actual frequency at f_{XT} = 32.768 kHz	
		CPU clock frequency	1 machine cycle	CPU clock frequency	1 machine cycle
0	0	$\Phi = f_{CC}/64$ (31.3 kHz)	32 μs	$\Phi = f_{XT}/4$ (8.192 kHz)	122 μs
0	1	$\Phi = f_{CC}/16$ (125 kHz)	8 μs		
1	0	$\Phi = f_{CC}/8$ (250 kHz)	4 μs		
1	1	$\Phi = f_{CC}/4$ (500 kHz)	2 μs		

(Operation with f_{CC} = 1 MHz)

		SCC3, SCC0 = 00 ( ) is actual frequency at f_{CC} =1 MHz		SCC3, SCC0 = 01 or 11 ( ) is actual frequency at f_{XT} = 32.768 kHz	
		CPU clock frequency	1 machine cycle	CPU clock frequency	1 machine cycle
0	0	$\Phi = f_{CC}/64$ (15.6 kHz)	64 μs	$\Phi = f_{XT}/4$ (8.192 kHz)	122 μs
0	1	$\Phi = f_{CC}/16$ (62.5 kHz)	16 μs		
1	0	$\Phi = f_{CC}/8$ (125 kHz)	8 μs		
1	1	$\Phi = f_{CC}/4$ (250 kHz)	4 μs		

Remarks 1. f_{CC} : Output frequency from the main system clock oscillator

2. f_{XT} : Output frequency from the subsystem clock oscillator

CPU operation mode control bits

0	0	Normal operation mode
0	1	HALT mode
1	0	STOP mode
1	1	Not to be set

(2) System clock control register (SCC)

The SCC is a 4-bit register for selecting CPU clock Φ with the least significant bit and for controlling the termination of main system clock generation with the most significant bit (see **Figure 5-13**).

Bits 0 and 3 of the SCC are located at the same data memory address, but both bits cannot be changed at the same time. Accordingly, bits 0 and 3 of the SCC are set using bit manipulation instructions. Bits 0 and 3 of the SCC can be manipulated regardless of MBE setting.

Main system clock generation can be terminated by setting bit 3 of the SCC only when the subsystem clock is used for operation. The STOP instruction must be used to terminate main system clock generation.

A $\overline{\text{RESET}}$ signal clears the SCC to 0.

Figure 5-13. Format of the System Clock Control Register

Address	3	2	1	0	Symbol
FB7H	SCC3	—	—	SCC0	SCC

SCC3	SCC0	CPU clock frequency	Main system clock operation
0	0	Main system clock	Can oscillate
0	1	Subsystem clock	
1	0	Not to be set	
1	1	Subsystem clock	Oscillation stopped

Cautions 1. A time period of up to 1/f_{XT} is needed to change the system clock. This means that to terminate main system clock generation, bit 3 of the SCC must be set to 1 when the machine cycles indicated in Table 5-4 or more have elapsed after the clock is switched from the main system clock to the subsystem clock.

2. When the main system clock is used for operation, setting bit 3 of the SCC to stop clock generation does not enter the normal STOP mode.

3. When the PCC is set to 0001B (Φ = f_{CC}/16), do not set SCC.0 to 1. Before switching the main system clock to the subsystem clock, be sure to manipulate the PCC so other than 0001B is set. When the system operates on the subsystem clock, the PCC must also be other than 0001B.

(3) System clock oscillator

The main system clock oscillator operates with a resistor (R) and capacitor (C) connected to the CL1 and CL2 pins, as shown in Figure 5-14.

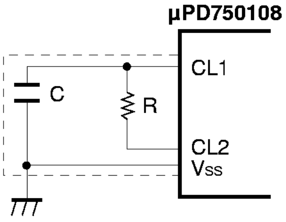
The output frequency (f_{CC}) of the main system clock oscillator is determined from the resistance (R) and capacitance (C), as follows:

$$f_{CC} = \frac{1}{2\pi CR}$$

Caution f_{CC} may be subject to a frequency deviation caused by a variation in the supply voltage or temperature.

Figure 5-14. External Circuit for the Main System Clock Oscillator

RC oscillation



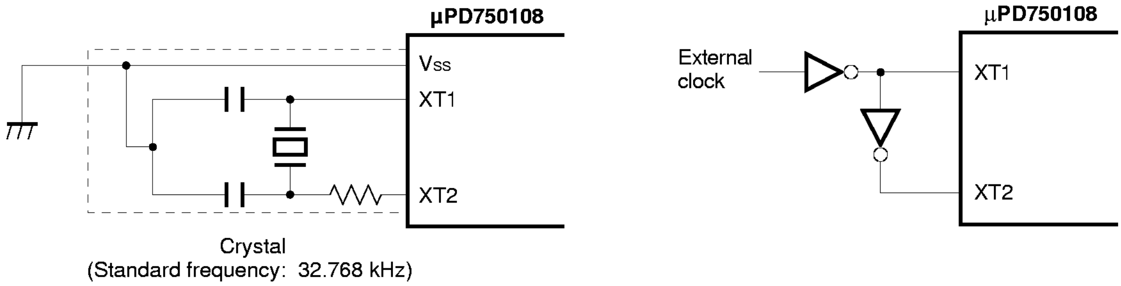
The subsystem clock oscillator operates with a crystal resonator (32.768 kHz standard) connected to the XT1 and XT2 pins.

An external clock can also be input. Input the clock signal to the XT1 pin and its inverted signal to the XT2 pin.

The state of the XT1 pin is tested by bit 3 of the clock mode register (WM).

Figure 5-15. External Circuit for the Subsystem Clock Oscillator

(a) Crystal oscillation **(b) External clock**



Caution When the main system clock or subsystem clock oscillator is used, conform to the following guidelines when wiring enclosed in broken lines of Figures 5-14 and 5-15 to eliminate the influence of the stray capacitance around the wiring.

- **The wiring must be as short as possible.**
- **Other signal lines must not run in these areas.**
  Any line carrying a high pulsating current must be kept away as far as possible.
- **The grounding point of the capacitor of the oscillator must have the same potential as that of V_{SS}. It must not be grounded to a grounding pattern carrying a high current.**
- **No signal must be taken directly from the resonator.**

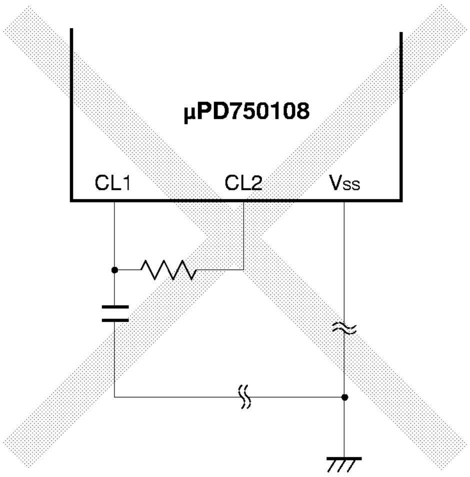
The subsystem clock oscillator has low amplification to minimize current consumption. For this reason, more malfunctions can occur due to noise than the main system clock oscillator. So pay special attention to wiring when using the subsystem clock.

Figure 5-16 gives examples of oscillator connections which should be avoided.

Figure 5-16. Examples of Oscillator Connections Which Should Be Avoided (1/4)

(a) The wiring is too long.

- **Main system clock**



- **Subsystem clock**

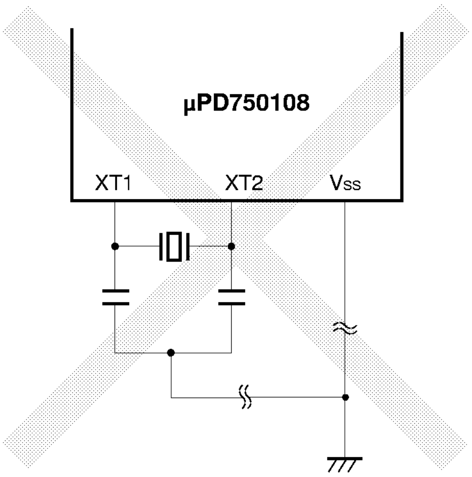
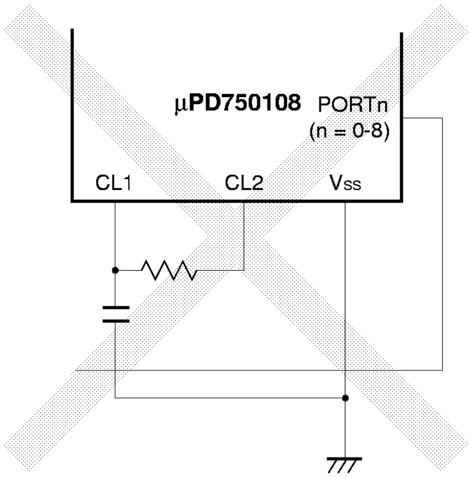


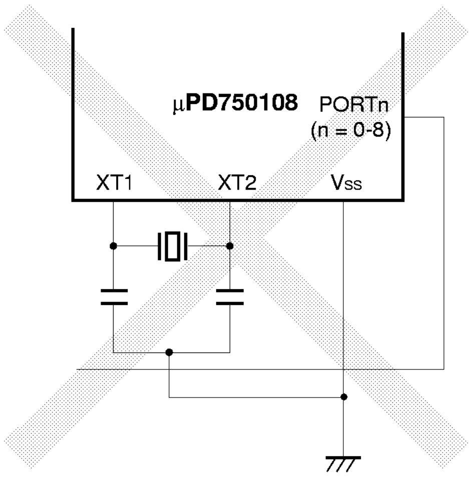
Figure 5-16. Examples of Oscillator Connections Which Should Be Avoided (2/4)

(b) The signal lines cross.

- Main system clock

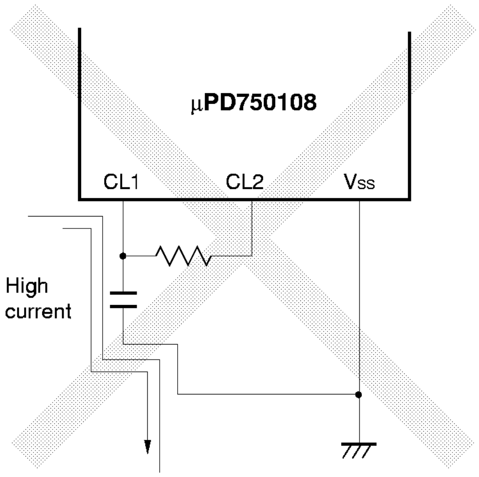


- Subsystem clock



(c) A high pulsating current is too close to the signal line.

- Main system clock



- Subsystem clock

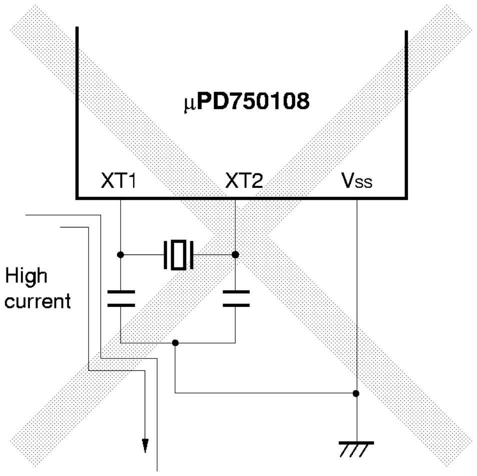
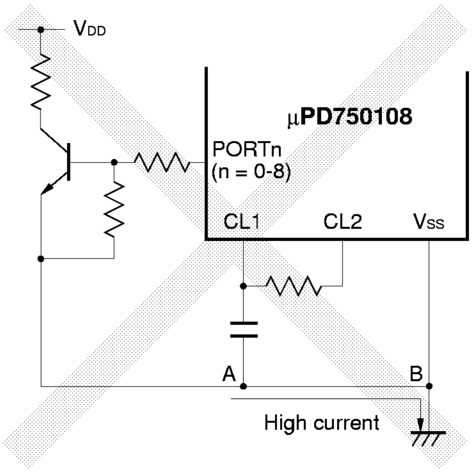


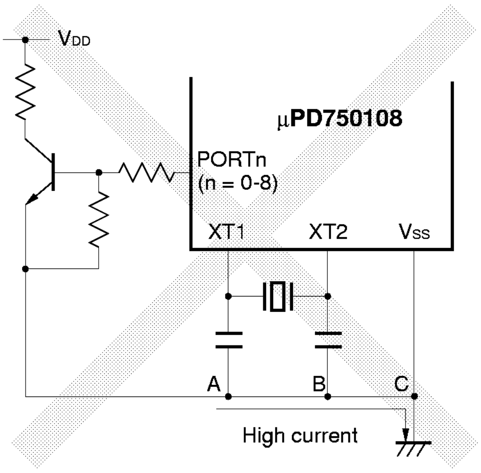
Figure 5-16. Examples of Oscillator Connections Which Should Be Avoided (3/4)

(d) The current flows through the ground line of the oscillator. (The potential at points A, B, and C fluctuates.)

- Main system clock

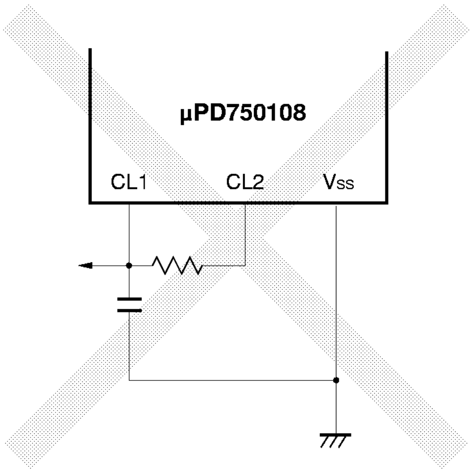


- Subsystem clock



(e) A signal is taken directly from the resonator.

- Main system clock



- Subsystem clock

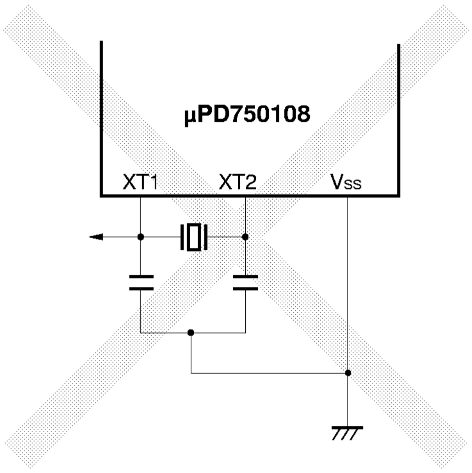
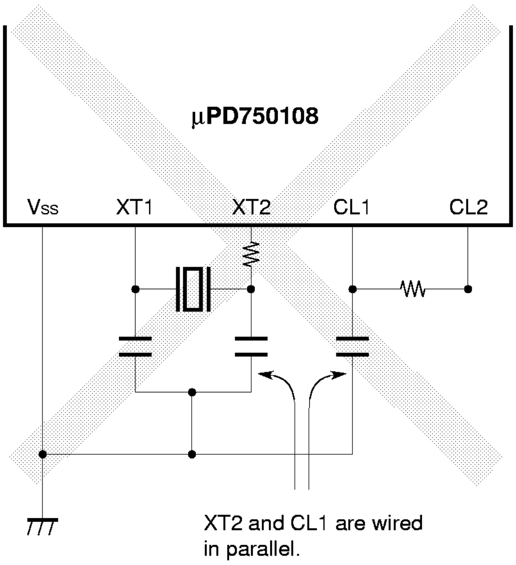


Figure 5-16. Examples of Oscillator Connections Which Should Be Avoided (4/4)

(f) The signal lines of the main system clock and subsystem clock are parallel and adjacent to each other.



(4) Frequency divider

The frequency divider divides the output (f_{CC}) of the main system clock oscillator to generate various clocks.

(5) Control functions of subsystem clock oscillator

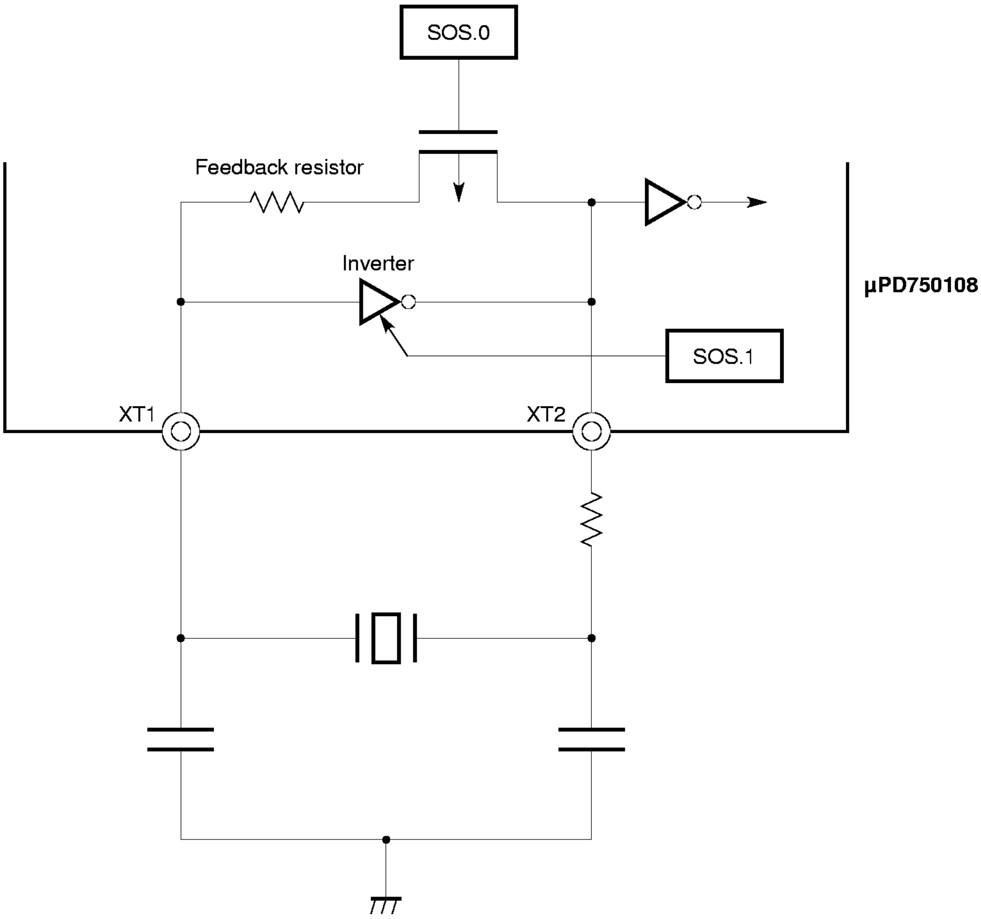
The subsystem clock oscillator of the μPD750108 subseries has two control functions to decrease the supply current.

- The function to select with the software whether to use the built-in feedback resistor**Note**
- The function to suppress the supply current by reducing the drive current of the built-in inverter when the operating supply voltage is high ($V_{DD} \geq 2.7$ V)

Note When the subsystem clock is not to be used, select SOS.0 = 1 (the built-in feedback resistor will not be used), connect the XT1 pin to V_{SS} or V_{DD}, and leave the XT2 pin open. This reduces the supply current which flows upon the execution of the STOP instruction.

Each function can be used by switching bits 0 and 1 in the sub-oscillator control register (SOS). (See **Figure 5-17**.)

Figure 5-17. Subsystem Clock Oscillator



(6) Sub-oscillator control register (SOS)

The SOS register specifies whether to use the built-in feedback resistor and controls the drive current of the built-in inverter. (See **Figure 5-18**.)

Inputting a $\overline{\text{RESET}}$ signal clears all bits of the SOS register. The functions of each flag in the SOS register are described below.

(a) SOS.0 (feedback resistor cut flag)

To use the feedback resistor of the subsystem clock, the mask option setup and switching SOS.0 by software are required. Set SOS.0 to 0 to turn on the feedback circuit.

When the resonator is not used, set SOS.0 to 1. The feedback circuit is turned off, reducing the current drain.

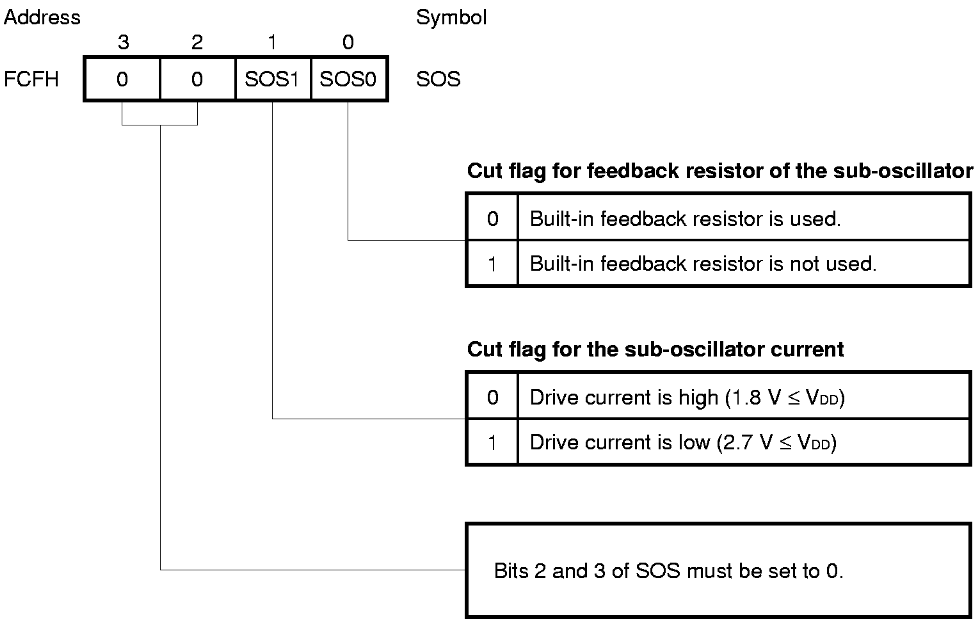
To use the resonator, be sure to select "Enable the feedback resistor" upon setting the mask option. Then, set SOS.0 to 0 (feedback circuit is turned on).

(b) SOS.1 (drive capability switch flag)

The built-in inverter in the subsystem clock oscillator of the µPD750108 subseries has a large drive current because it can be used at low supply voltage (V_{DD} = 1.8 V), so that the supply current becomes too high to use at high supply voltage (V_{DD} ≥ 2.7 V). To reduce the supply current, set SOS.1 to 1 so as to reduce the drive current of the inverter.

However, if SOS.1 is set to 1 when V_{DD} is less than 2.7 V, the oscillation may stop for insufficient drive current. Set this flag to 0 when V_{DD} is less than 2.7 V.

Figure 5-18. Sub-Oscillator Control Register (SOS) Format



Cut flag for feedback resistor of the sub-oscillator

0	Built-in feedback resistor is used.
1	Built-in feedback resistor is not used.

Cut flag for the sub-oscillator current

0	Drive current is high (1.8 V ≤ V_{DD})
1	Drive current is low (2.7 V ≤ V_{DD})

Bits 2 and 3 of SOS must be set to 0.

Remark If the subsystem clock is not required , the XT1 and XT2 pins and SOS register must be treated as follows:

XT1 : Connected to V_{SS} or V_{DD}.
XT2 : Open
SOS: 0001B

5.2.3 System Clock and CPU Clock Setting

(1) Time required to change the system clock and CPU clock

The system clock and CPU clock can be changed by using the least significant bit of the SCC and the low-order two bits of the PCC. This switching is not performed immediately after the contents of the registers are rewritten, but the system operates with the previous clock for some machine cycles. Accordingly, after this time period, the STOP instruction must be executed to terminate main system clock generation.

Table 5-5. Maximum Time Required to Change the System Clock and CPU Clock

Setting before switching			Setting after switching														
SCC0	PCC1	PCC0	SCC0	PCC1	PCC0	SCC0	PCC1	PCC0	SCC0	PCC1	PCC0	SCC0	PCC1	PCC0	SCC0	PCC1	PCC0
			0	0	0	0	0	1	0	1	0	0	1	1	1	x	x
0	0	0				1 machine cycle			1 machine cycle			1 machine cycle			$f_{CC}/64f_{XT}$ machine cycles (1 machine cycle)		
	0	1	4 machine cycles						4 machine cycles			4 machine cycles			$f_{CC}/16f_{XT}$ machine cycles (4 machine cycles)		
	1	0	8 machine cycles			8 machine cycles						8 machine cycles			$f_{CC}/8f_{XT}$ machine cycles (8 machine cycles)		
	1	1	16 machine cycles			16 machine cycles			16 machine cycles						$f_{CC}/4f_{XT}$ machine cycles (15 machine cycles)		
1	x	x	1 machine cycle			Not to be set			1 machine cycle			1 machine cycle					

Remarks 1. Time indicated in parentheses is required when f_{CC} = 2 MHz and f_{XT} = 32.768 kHz.

2. X: Don't care

3. CPU clock Φ is supplied to the CPU of the μPD750108. The reciprocal of this frequency is a minimum instruction time (defined as one machine cycle in this manual).

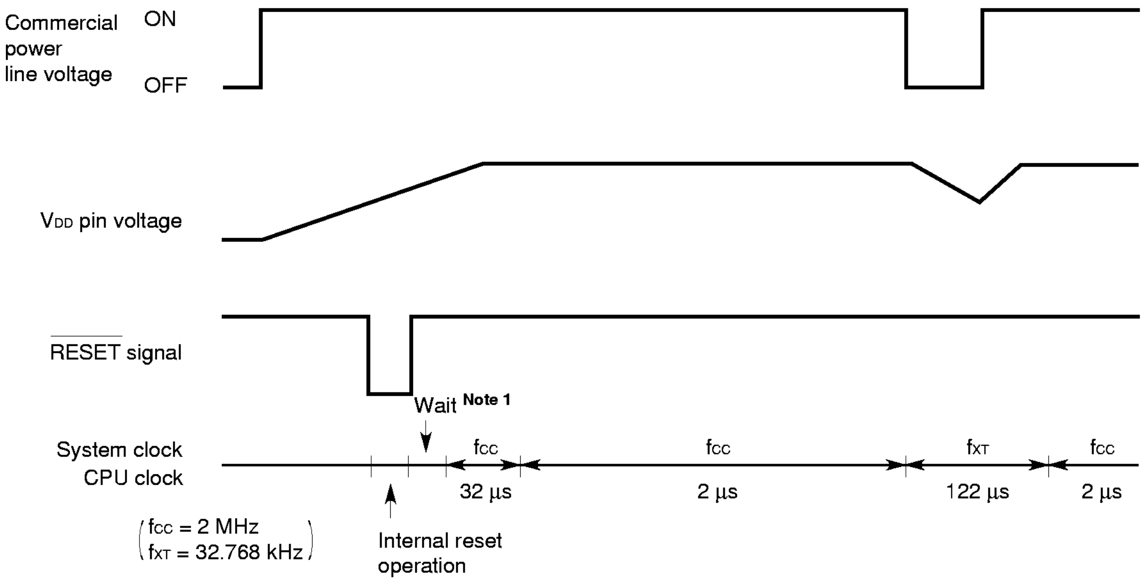
Cautions 1. When the PCC is set to 0001B (Φ = f_{CC}/16), do not set SCC.0 to 1. Before switching the main system clock to the subsystem clock, be sure to manipulate the PCC so other than 0001B is set. When the system operates on the subsystem clock, the PCC must also be other than 0001B.

2. The fluctuation of the ambient temperature around an oscillator and the performance of a load capacity change f_{CC} and f_{XT}. In particular, when f_{CC} is higher than the nominal value or f_{XT} is lower than the nominal value, the machine cycles calculated by f_{CC}/ 64f_{XT}, f_{CC}/16f_{XT}, f_{CC}/ 8f_{XT}, and f_{CC}/4f_{XT} in Table 5-5 are longer than the machine cycle calculated by the nominal values of f_{CC} and f_{XT}. Therefore, the wait time required to change the system clock and CPU clock should be longer than the machine cycle calculated by the nominal values of f_{CC} and f_{XT}.

(2) Procedure for changing the system clock and CPU clock

The procedure for changing the system clock and CPU clock is explained using Figure 5-19.

Figure 5-19. Changing the System Clock and CPU Clock



<1> A $\overline{\text{RESET}}$ signal starts CPU operation at the lowest speed of the main system clock (32 µs at 2 MHz, 64 µs at 1 MHz) after a wait time[Note 1] for stable oscillation.

<2> The PCC is rewritten for highest-speed operation after a time elapse which is sufficient for the voltage on the V_{DD} pin to be high enough for highest-speed operation.

<3> The removal of commercial current is detected using, for example, an interrupt input[Note 2], then bit 0 of the SCC is set to 1 to operate with the subsystem clock. (In this case, subsystem clock generation must have been started.) After a time (15 machine cycles) required to switch to the subsystem clock elapses, bit 3 of the SCC is set to 1 to terminate main system clock generation.

<4> After detecting the input of commercial current by using an interrupt, bit 3 of the SCC is cleared to start main system clock generation. After a time required for stable generation, bit 0 of the SCC is cleared to 0 to operate at the highest speed.

Notes 1. The wait time is fixed to $56/f_{CC}$ (28 µs at 2 MHz, 56 µs at 1 MHz)

2. INT4 is useful.

5.2.4 Clock Output Circuit

(1) Configuration of the clock output circuit

Figure 5-20 shows the configuration of the clock output circuit.

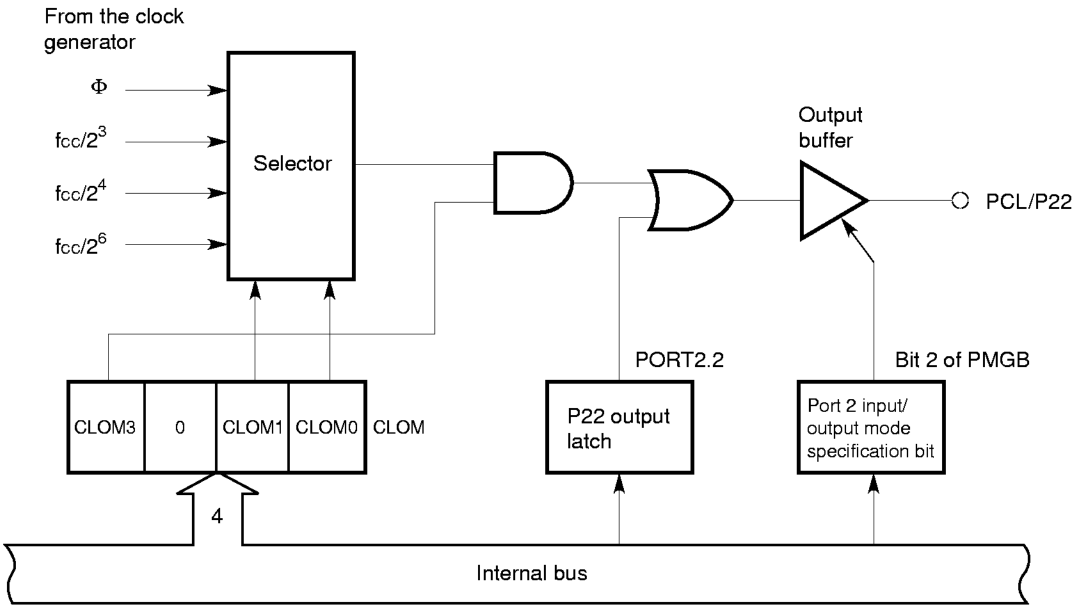
(2) Functions of the clock output circuit

The clock output circuit outputs a clock pulse signal on the P22/PCL pin to output remote control signals or to supply clock pulses to a peripheral LSI device.

The procedure for outputting a clock pulse signal is as follows:

(a) Select a clock output frequency, and disable clock output.
(b) Write a 0 in the P22 output latch.
(c) Set the output mode for port 2.
(d) Enable clock output.

Figure 5-20. Configuration of the Clock Output Circuit



Remark The clock output circuit is designed so that pulses with short widths do not appear in enabling or disabling clock output.

(3) Clock output mode register (CLOM)

The CLOM is a 4-bit register to control clock output.

The CLOM is set by a 4-bit memory manipulation instruction. No read operation is allowed on this register.

Example CPU clock Φ is output on the PCL/P22 pin.

```
SEL     MB15          ; or CLR1 MBE
MOV     A,#1000B
MOV     CLOM,A
```

A $\overline{\text{RESET}}$ signal clears the CLOM to 0, disabling clock output.

Figure 5-21. Format of the Clock Output Mode Register

Address	3	2	1	0	Symbol
FD0H	CLOM3	0	CLOM1	CLOM0	CLOM

Clock output frequency selection bit
(fcc = 2 MHz)

0	0	Φ output[Note] (500, 250, 125, 31.3 kHz)
0	1	fcc/2^3 output (250 kHz)
1	0	fcc/2^4 output (125 kHz)
1	1	fcc/2^6 output (31.3 kHz)

(fcc = 1 MHz)

0	0	Φ output[Note] (250, 125, 62.5 15.6 kHz)
0	1	fcc/2^3 output (125 kHz)
1	0	fcc/2^4 output (62.5 kHz)
1	1	fcc/2^6 output (15.6 kHz)

Note Φ is the CPU clock selected by PCC.

Clock output enable/disable bit

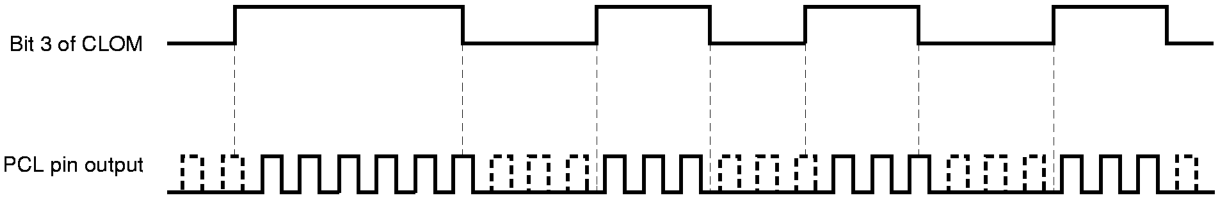
0	Output disable
1	Output enable

Caution Be sure to write a 0 in bit 2 of the CLOM.

(4) Application to remote control output

The clock output function of the µPD750108 is applicable to remote control output. The frequency of the carrier for remote control output is selected by the clock frequency select bit of the clock output mode register. Pulse output is enabled or disabled by controlling the clock output enable/disable bit by software. The clock output circuit is designed so that pulses with short widths do not appear in enabling or disabling clock output.

Figure 5-22. Application to Remote Control Output



5.3 BASIC INTERVAL TIMER/WATCHDOG TIMER

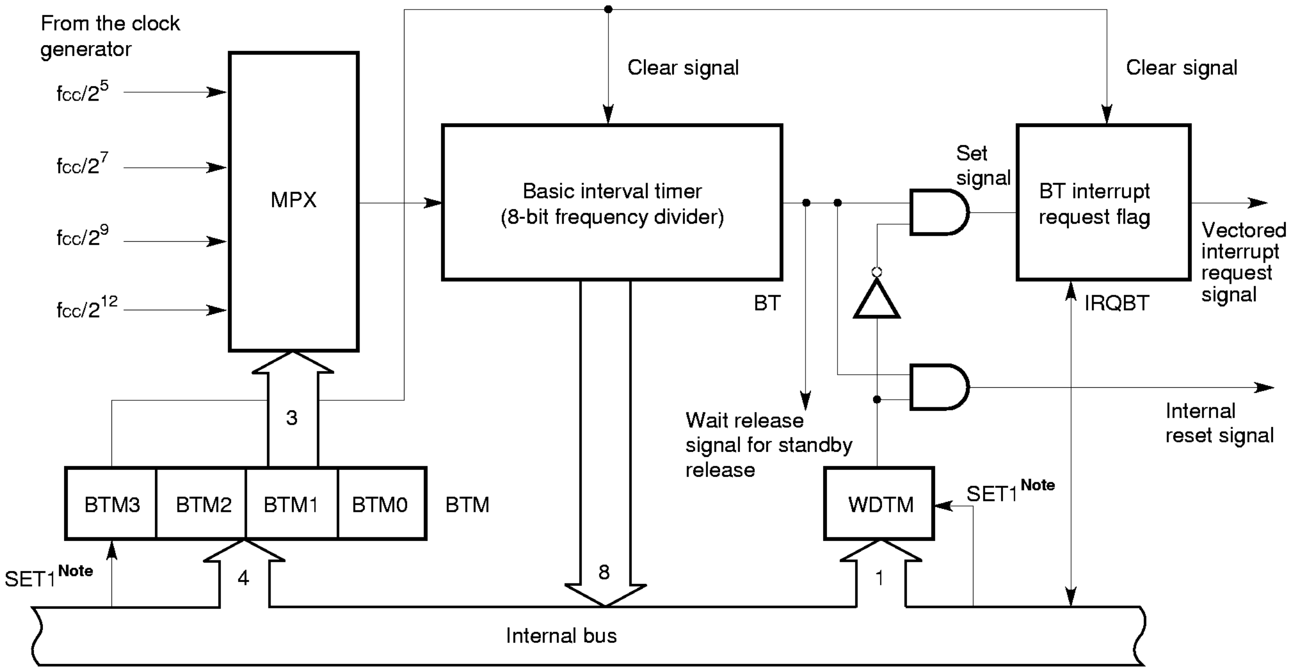
The μPD750108 contains an 8-bit basic interval timer/watchdog timer, which has the following functions:

(a) Interval timer operation which generates a reference timer interrupt
(b) Operation as a watchdog timer for detecting program crashes and resetting the CPU
(c) Selection of a wait time for releasing the standby mode, and counting
(d) Reading the count value

5.3.1 Configuration of the Basic Interval Timer/Watchdog Timer

Figure 5-23 shows the configuration of the basic interval timer/watchdog timer.

Figure 5-23. Block Diagram of the Basic Interval Timer/Watchdog Timer



Note Instruction execution

5.3.2 Basic Interval Timer Mode Register (BTM)

The BTM is a 4-bit register for controlling operation of the basic interval timer (BT).
A 4-bit memory manipulation instruction is used to set the BTM.
Bit 3 can be independently manipulated using a bit manipulation instruction.

Example The interrupt generation interval is set to 4.10 ms (at 2 MHz).

```
SEL   MB15      ; or CLR1 MBE
MOV   A,#1111B
MOV   BTM,A     ; BTM <– 1111B
```

When bit 3 is set to 1, the BT is cleared, and the basic interval timer/watchdog timer interrupt request flag (IRQBT) is also cleared (to start the basic interval timer/watchdog timer).

A $\overline{\text{RESET}}$ signal clears the interval timer to 0, and the longest interrupt request signal generation interval time is set.

Figure 5-24. Format of the Basic Interval Timer Mode Register

Address	3	2	1	0	Symbol
F85H	BTM3	BTM2	BTM1	BTM0	BTM

(fcc = 2 MHz)

			Input clock specification	Interrupt interval time
0	0	0	fcc/2^{12}(488 Hz)	2^{20}/fcc(524 ms)
0	1	1	fcc/2^{9}(3.91 kHz)	2^{17}/fcc(65.5 ms)
1	0	1	fcc/2^{7}(15.6 kHz)	2^{15}/fcc(16.4 ms)
1	1	1	fcc/2^{5}(62.5 kHz)	2^{13}/fcc(4.10 ms)
Other than above			Not to be set	—

(fcc = 1 MHz)

			Input clock specification	Interrupt interval time
0	0	0	fcc/2^{12}(244 Hz)	2^{20}/fcc(1.05 s)
0	1	1	fcc/2^{9}(1.95 kHz)	2^{17}/fcc(131 ms)
1	0	1	fcc/2^{7}(7.81 kHz)	2^{15}/fcc(32.8 ms)
1	1	1	fcc/2^{5}(31.3 kHz)	2^{13}/fcc(8.19 ms)
Other than above			Not to be set	—

Basic interval timer/watchdog timer start control bit

When 1 is written to this bit, the basic interval timer/watchdog timer operation starts (the counter and the interrupt request flag are cleared).
When the operation starts, this bit is automatically reset to 0.

5.3.3 Watchdog Timer Enable Flag (WDTM)

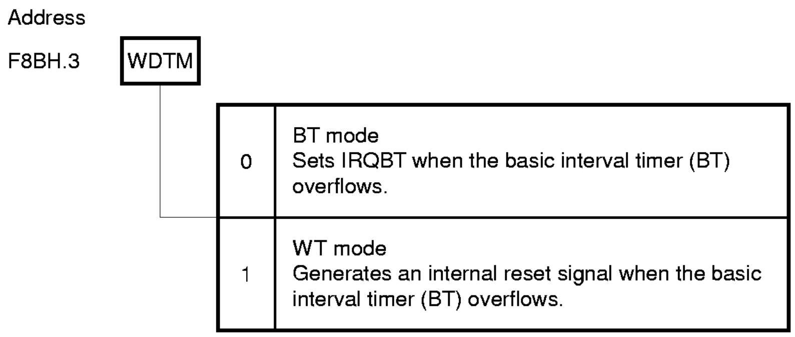
WDTM, when set, is a flag for enabling the generation of the reset signal when the basic interval timer overflows. WDTM is set by a bit manipulation instruction. It cannot be cleared by an instruction.

Example Set the watchdog timer function.

```
SEL    MB15      ; or CLR1 MBE
SET1   WDTM
        :
        :
SET1   BTM.3     ; Set bit 3 of BTM to 1
```

The generation of a $\overline{\text{RESET}}$ signal clears WDTM to 0.

Figure 5-25. Format of the Watchdog Timer Enable Flag (WDTM)



Address

F8BH.3 WDTM

0	BT mode Sets IRQBT when the basic interval timer (BT) overflows.
1	WT mode Generates an internal reset signal when the basic interval timer (BT) overflows.

5.3.4 Operation of the Basic Interval Timer

When WDTM is set to 0, the basic interval timer (BT) functions as an interval timer. An interrupt request flag (IRQBT) is set when the timer overflows. BT is constantly incremented by the clock supplied from the clock generator. So it is impossible to stop the timer from incrementing.

One of four interrupt generation intervals can be selected by setting BTM. (See **Figure 5-24**.)

BT and IRQBT can be cleared by setting bit 3 of BTM to 1 (instruction for starting as an interval timer).

The count status of BT can be read by an 8-bit manipulation instruction. No data can be loaded to the timer.

Perform the timer operation as follows (**<1>** and **<2>** can be performed with the same instruction):

<1> Set the interval in BTM.

<2> Set 1 in bit 3 of BTM.

Example Generate an interrupt at intervals of 4.10 ms (at 2 MHz).

```
SET1   MBE
SEL    MB15
MOV    A,#1111B
MOV    BTM,A     ; Set the interval and start processing
EI               ; Enable interrupt
EI     IEBT      ; Enable BT interrupt
```

5.3.5 Operation of the Watchdog Timer

When WDTM is set to 1, the basic interval timer/watchdog timer functions as a watchdog timer. An internal reset signal is generated when the basic interval timer (BT) overflows. No reset signal, however, is generated during the oscillation settling time following the STOP instruction has been released (WDTM cannot be cleared without using reset). BT is constantly incremented by the clock supplied from the clock generator. It cannot be stopped from counting.

In the watchdog timer mode, program crashes are detected using the intervals at which BT overflows. The interval can be selected from among four values depending on bits 2 to 0 of BTM (see **Figure 5-24**). Select an interval for detecting crashes according to the user system. A large program should be divided into modules each of which can be executed within the set interval. Include an instruction which clears BT at the end of each module. If execution does not reach the instruction which clears BT within the set interval (in which case a program error leading to a program crash may have occurred), BT overflows and an internal reset signal is generated to forcibly terminate the program. The occurrence of internal reset possibly means that a program crash has occurred. A crash can thus be detected.

Set the watchdog timer as follows (**<1>** and **<2>** can be performed with the same instruction):

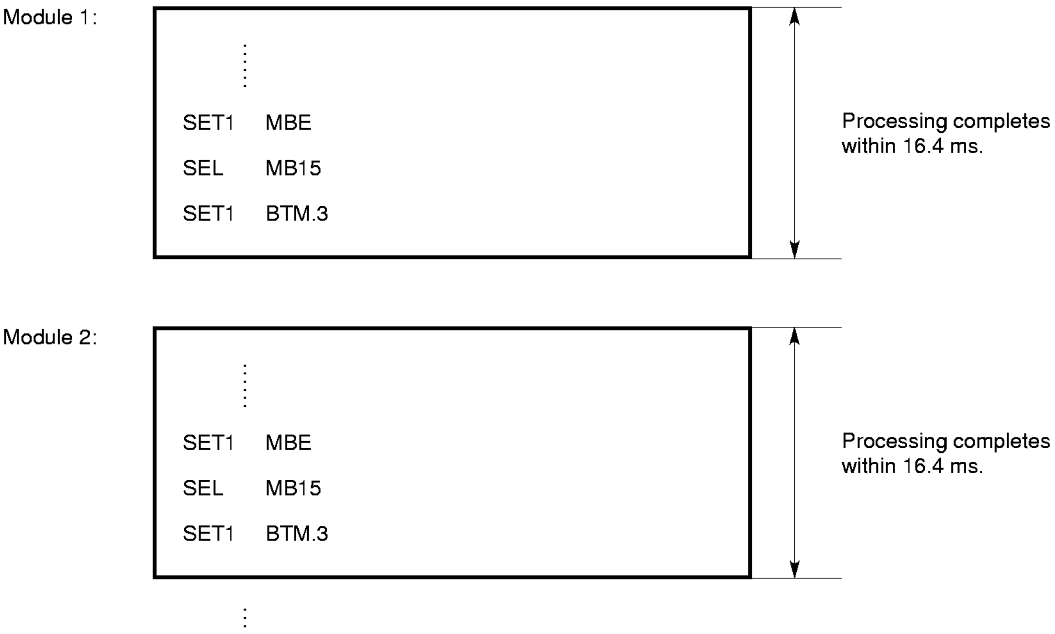
<1> Set the interval in BTM.
<2> Set 1 in bit 3 of BTM. Initial settings
<3> Set 1 in WDTM.
<4> After **<1>** to **<3>** are set, set 1 in bit 3 of BTM within each interval.

Example Use the basic interval/watchdog timer as a watchdog timer with 16.4-ms interval (at 2 MHz) A program is divided into several modules each of which can be executed within the interval set in BTM (16.4 ms). BT is cleared at the end of each module. If a program crash occurs, BT overflows and an internal reset signal is generated because BT is not cleared within the set interval.

Initial setting:

```
SET1   MBE
SEL    MB15
MOV    A, #1101B
MOV    BTM, A          ; Specifies a time interval and
                       ; starts processing.
SET1   WDTM            ; Enables the watchdog timer.
         :
```

(From now on, 1 is set in bit 3 of BTM at intervals of 16.4 ms.)



Module 1:

```
      :
SET1  MBE
SEL   MB15
SET1  BTM.3
```

Processing completes within 16.4 ms.

Module 2:

```
      :
SET1  MBE
SEL   MB15
SET1  BTM.3
      :
```

Processing completes within 16.4 ms.

5.3.6 Other Functions

The basic interval timer/watchdog has the following functions regardless of whether it operates as a basic interval timer or watchdog timer:

- **Reading the count**

  The count status of the basic interval timer (BT) can be read by using an 8-bit manipulation instruction. No data can be loaded to the timer.

Caution When reading the count value of BT, execute a read instruction twice so that unstable data which has been counted will not be read. If the two read values are reasonable, use the second one as the result. If the two read values are far apart, retry from the beginning.

Example 1. Read the count value of BT.

```
       SET1   MBE
       SEL    MB15
       MOV    HL, #BT    ; Set the BT address in HL
LOOP:  MOV    XA, @HL    ; First read
       MOV    BC, XA
       MOV    XA, @HL    ; Second read
       SKE    XA, BC
       BR     LOOP
```

Example 2. Set the high level width of pulses applied to the INT4 interrupt pin (both edges detected). (The pulse width is assumed not to exceed the value (16.4 ms or longer at 2 MHz) set in the BTM.)

<INT4 interrupt routine (MBE = 0)>

```
LOOP:   MOV     XA, BT      ; First read
        MOV     BC, XA      ; Store data
        MOV     XA, BT      ; Second read
        SKE     A, C
        BR      LOOP
        MOV     A, X
        SKE     A, B
        BR      LOOP
        SKT     PORT0.0     ; P00 = 1?
        BR      AA          ; NO
        MOV     XA, BC      ; Store data in data memory
        MOV     BUFF, XA
        CLR1    FLAG        ; Clear data presence flag
        RETI
AA:     MOV     HL, #BUFF
        MOV     A, C
        SUBC    A, @HL
        INCS    L
        MOV     C, A
        MOV     A, B
        SUBC    A, @HL
        MOV     B, A
        MOV     XA, BC
        MOV     BUFF, XA    ; Store data
        SET1    FLAG        ; Set data presence flag
        RETI
```

5.4 CLOCK TIMER

The μPD750108 contains one clock timer, which has the following functions.

(a) The clock timer sets the test flag (IRQW) every 0.5 seconds (when WM = 1).
The IRQW can release the standby mode.

(b) The subsystem clock can be used to produce 0.5-second intervals.

(c) The fast-forward mode produces an interval 128 times faster, which is useful for program debugging and testing.

(d) An arbitrary frequency**Note** can be output to the P23/BUZ pin, so that it can be used for sounding the buzzer and for system clock frequency trimming.

(e) The frequency divider can be cleared to start the clock from zero second.

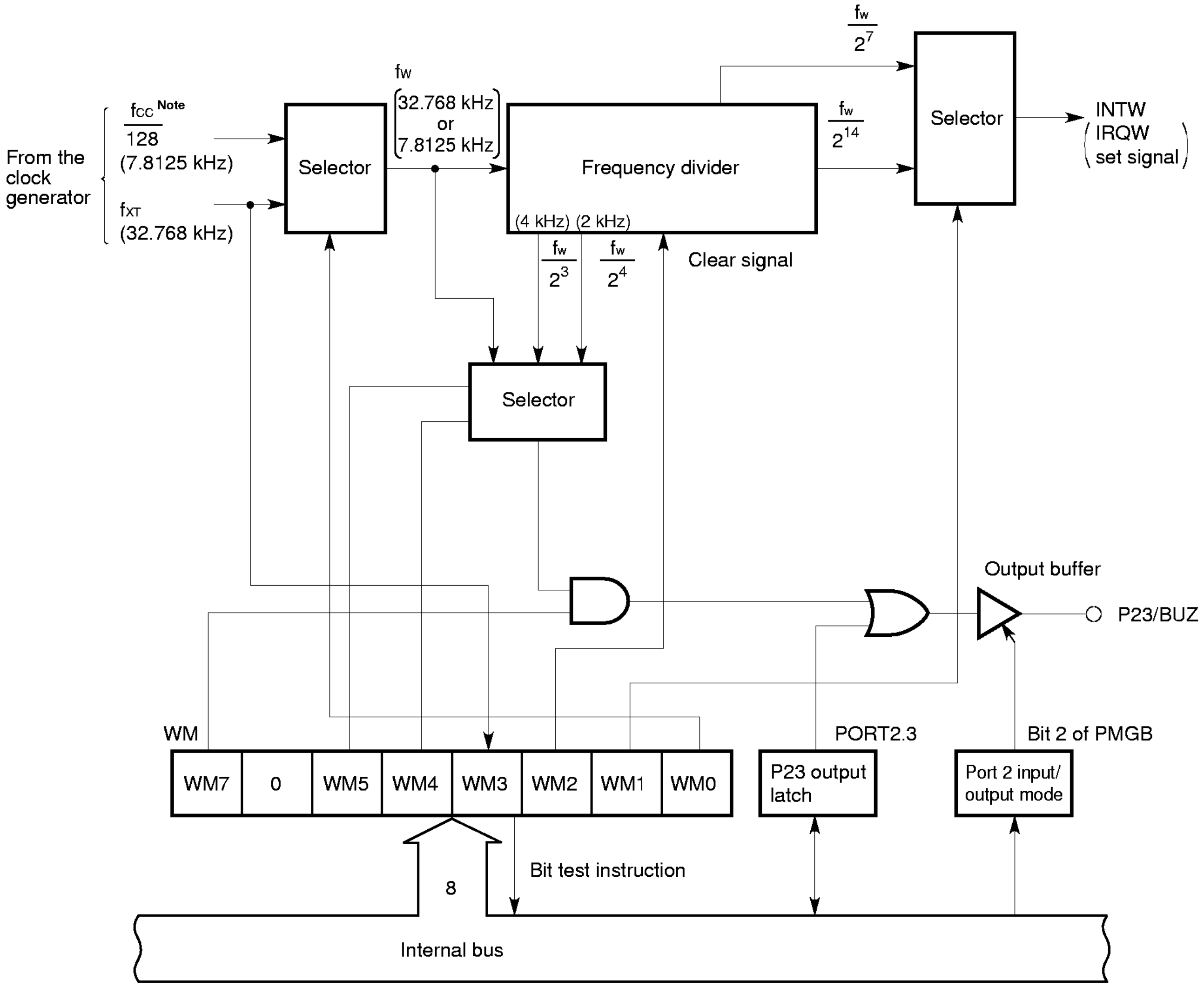
Note 2.048, 4.096, or 32.768 kHz when WM0 = 1.
0.488, 0.977, or 7.8125 kHz during 1-MHz operation when WM0 = 0.

Caution Set WM0 = 1 when using the clock function.

5.4.1 Configuration of the Clock Timer

Figure 5-26 shows the configuration of the clock timer.

Figure 5-26. Block Diagram of the Clock Timer



Note When a frequency-divided main system clock is used, 32.768 kHz cannot be selected as the source clock.

Remark The values in parentheses are for f_{CC} = 1 MHz and f_{XT} = 32.768 kHz.

5.4.2 Clock Mode Register

The clock mode register (WM) is an 8-bit register which controls the clock timer. Figure 5-27 shows the format of the clock mode register.

All bits except bit 3 of the clock mode register are controlled by an 8-bit manipulation instruction. Bit 3 is for testing the XT1 pin input level. The input level of the XT1 pin can be tested by bit test operation. No data can be written to this register.

When the $\overline{\text{RESET}}$ signal is generated, all bits except bit 3 of this register are cleared to 0.

Example Time is set using the subsystem clock (32.768 kHz), and buzzer output is enabled:

```
CLR1   MBE
MOV    XA, #85H
MOV    WM, XA      ; Sets WM
```

Figure 5-27. Clock Mode Register Format

Address	7	6	5	4	3	2	1	0	Symbol
F98H	WM7	0	WM5	WM4	WM3	WM2	WM1	WM0	WM

BUZ output enable/disable bit

WM7	0	Disables BUZ output
	1	Enables BUZ output

BUZ output frequency selection bit

WM5	WM4	BUZ output frequency
0	0	$\frac{f_W}{2^4}$ (2.048 kHz: when WM0 = 1, 0.488 kHz: during 1-MHz operation when WM0 = 0)
0	1	$\frac{f_W}{2^3}$ (4.096 kHz: when WM0 = 1, 0.977 kHz: during 1-MHz operation when WM0 = 0)
1	0	Not to be set
1	1	f_W (32.768 kHz: when WM0 = 1, 7.8125 kHz: during 1-MHz operation when WM0 = 0)

XT1 pin input level (bit test only)

WM3	0	Input to the XT1 pin is low level
	1	Input to the XT1 pin is high level

Clock operation enable/disable bit

WM2	0	Disables clock operation (clears the frequency dividing circuit)
	1	Enables clock operation

Operation mode selection bit

WM1	0	Normal clock mode (sets IRQW at $\frac{f_W}{2^{14}}$ Note 1)
	1	Advanced clock mode (sets IRQW at $\frac{f_W}{2^7}$ Note 2)

Count clock (fw) selection bit

WM0	0	Selects divided system clock output: $\frac{f_{CC}}{128}$
	1	Selects subsystem clock: f_{XT}

Notes 1. $f_W/2^{14}$ is 0.5 s when WM0 = 1 (2.1 s during 1-MHz operation when WM0 = 0).

2. $f_W/2^7$ is 3.91 ms when WM0 = 1 (16.4 ms during 1-MHz operation when WM0 = 0).

Remark ( ) for f_W = 32.768 kHz

5.5 TIMER/EVENT COUNTER

The µPD750108 has one timer/event counter channel (channel 0) and one timer counter channel (channel 1). Figures 5-28 and 5-29 show the configuration of these channels.

In this section, the timer/event counter and timer counters are referred to as "timer/event counters." When you read this section for description of channel 1, take "timer/event counter" as "timer counter."

The timer/event counter has the following functions.

(a) Programmable interval timer operation
(b) Square wave output of any frequency to the PTOn pin.
(c) Event counter operation (Channel 0 only)
(d) Divides the frequency of signal input via the TI0 pin to 1-Nth of the original signal and outputs the divided frequency to the PTO0 pin (frequency divider operation) (Channel 0 only).
(e) Supplies the serial shift clock to the serial interface circuit (Channel 0 only).
(f) Calls the counting status.

5.5.1 Configuration of Timer/Event Counter

Figures 5-28 and 5-29 shows the configuration of the timer/event counter.

Figure 5-28. Block Diagram of the Timer/Event Counter (Channel 0)

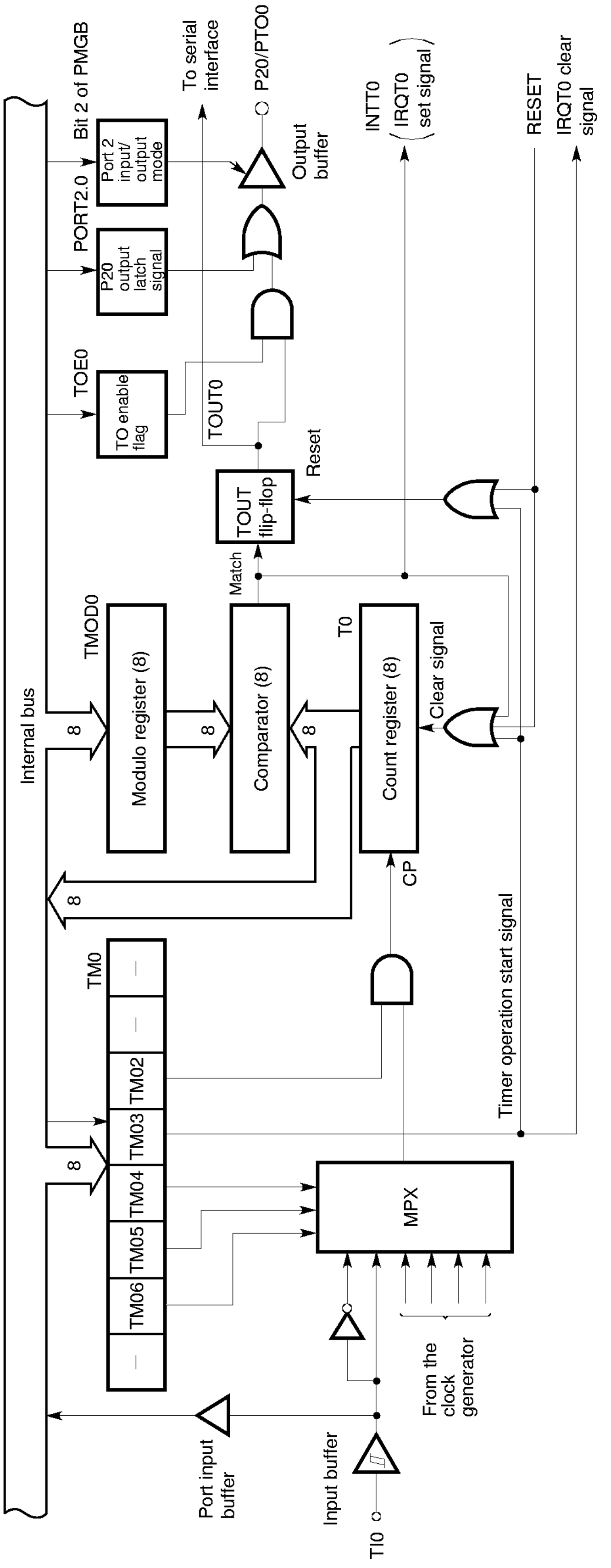
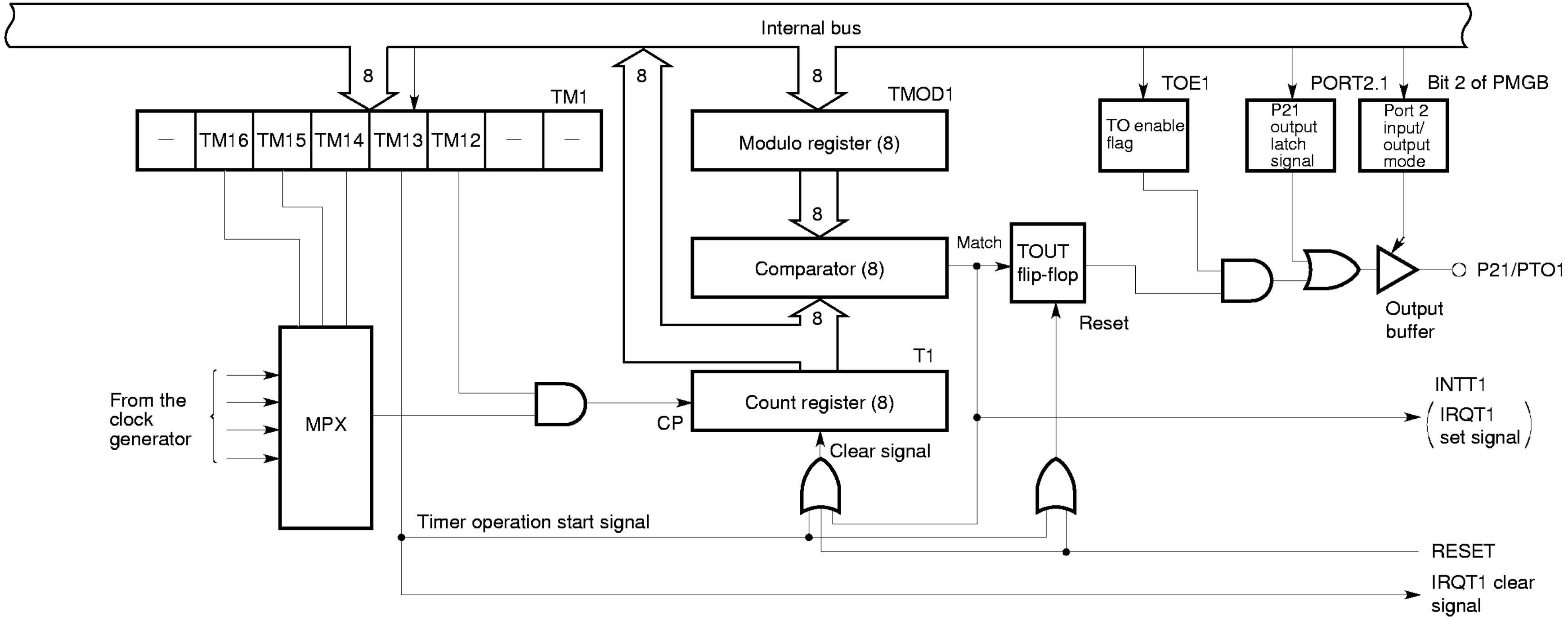


Figure 5-29. Block Diagram of the Timer Counter (Channel 1)



(1) Timer/event counter mode register (TM0, TM1)

The mode register (TMn) is an 8-bit register which controls the timer/event counter.

Its format is shown in Figures 5-30 and 5-31.

The timer/event counter mode register is set by an 8-bit memory manipulation instruction.

Bit 3 is a timer start bit and can be operated bit-wise. It is automatically reset to 0 when the timer operation starts.

All the bits of the timer/event counter mode register are cleared to 0 by a $\overline{\text{RESET}}$ signal generation.

Examples **1.** Start the timer in the interval timer mode of CP = 1.95 kHz (during 2 MHz operation).

```
SEL     MB15            ; or CLR1 MBE
MOV     XA, #01001100B
MOV     TMn, XA         ; TMn <– 4CH
```

2. Restart the timer according to the setting of the timer/event counter mode register.

```
SEL     MB15            ; or CLR1 MBE
SET1    TMn.3           ; TMn.bit3 <– 1
```

Figure 5-30. Timer/Event Counter Mode Register (Channel 0) Format

Address	7	6	5	4	3	2	1	0	Symbol
FA0H	—	TM06	TM05	TM04	TM03	TM02	—	—	TM0

Count pulse (CP) selection bit

When f_{CC} = 2 MHz

TM06	TM05	TM04	Count pulse (CP)
0	0	0	TI0 rising edge
0	0	1	TI0 falling edge
1	0	0	$f_{CC}/2^{10}$ (1.95 kHz)
1	0	1	$f_{CC}/2^{8}$ (7.81 kHz)
1	1	0	$f_{CC}/2^{6}$ (31.3 kHz)
1	1	1	$f_{CC}/2^{4}$ (125 kHz)
Other than above			Not to be set

When f_{CC} = 1 MHz

TM06	TM05	TM04	Count pulse (CP)
0	0	0	TI0 rising edge
0	0	1	TI0 falling edge
1	0	0	$f_{CC}/2^{10}$ (977 Hz)
1	0	1	$f_{CC}/2^{8}$ (3.91 kHz)
1	1	0	$f_{CC}/2^{6}$ (15.6 kHz)
1	1	1	$f_{CC}/2^{4}$ (62.5 kHz)
Other than above			Not to be set

Timer start indication bit

TM03	When 1 is written into the bit, the counter and IRQT0 flag are cleared. If bit 2 is set to 1, count operation is started.

Operation mode

TM02	Count operation
0	Stop (retention of count contents)
1	Count operation

Figure 5-31. Timer Counter Mode Register (Channel 1) Format

Address	7	6	5	4	3	2	1	0	Symbol
FA8H	—	TM16	TM15	TM14	TM13	TM12	—	—	TM1

Count pulse (CP) select bit

When fcc = 2 MHz

TM16	TM15	TM14	Count pulse (CP)
0	0	0	Rising edge of INTW (overflow output for clock timer)
1	0	0	fcc/2^{12} (488 Hz)
1	0	1	fcc/2^{10} (1.95 kHz)
1	1	0	fcc/2^{8} (7.81 kHz)
1	1	1	fcc/2^{6} (31.3 kHz)
Other than above			Not to be set

When fcc = 1 MHz

TM16	TM15	TM14	Count pulse (CP)
0	0	0	Rising edge of INTW (overflow output for clock timer)
1	0	0	fcc/2^{12} (244 Hz)
1	0	1	fcc/2^{10} (977 Hz)
1	1	0	fcc/2^{8} (3.91 kHz)
1	1	1	fcc/2^{6} (15.6 kHz)
Other than above			Not to be set

Timer start indication bit

TM13	When 1 is written into the bit, the counter and IRQT1 flag are cleared. If bit 2 is set to 1, count operation is started.

Operation mode

TM12	Count operation
0	Stop (retention of count contents)
1	Count operation

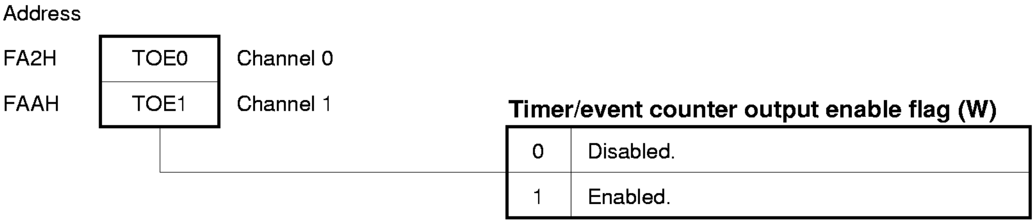
(2) Timer/event counter output enable flag (TOE0, TOE1)

The timer/event counter output enable flag (TOE0, TOE1) controls the output enable/disable to the PTO0 and PTO1 pins in the timer out flip-flop (TOUT flip-flop) status.

The timer out flip-flop is inverted by the match signal sent from the comparator. When bit 3 of the timer/ event counter mode register (TM0, TM1) is set to 1, the timer out flip-flop is cleared to 0.

TOE0, TOE1, and timer out flip-flop are cleared to 0 by a $\overline{\text{RESET}}$ signal generation.

Figure 5-32. Timer/Event Counter Output Enable Flag Format



Address		
FA2H	TOE0	Channel 0
FAAH	TOE1	Channel 1

Timer/event counter output enable flag (W)

0	Disabled.
1	Enabled.

5.5.2 8-bit Timer/Event Counter Mode Operation

It is used as an 8-bit timer/event counter in this mode. It performs an 8-bit programmable interval timer and event counter operation (channel 0 only).

(1) Register setting

The following three registers and one flag are used in the 8-bit timer/event counter mode.

- Timer/event counter mode register (TMn)
- Timer/event counter count register (Tn)
- Timer/event counter modulo register (TMODn)
- Timer/event counter output enable flag (TOEn)

(a) Timer/event counter mode register (TMn)

When the 8-bit timer/event counter mode is used, TMn must be set as shown in Figure 5-33 (For the format of the TMn, see Figures 5-30 and 5-31).

The TMn is manipulated by an 8-bit manipulation instruction. Bit 3 is a timer start indication bit and can be manipulated bit-wise and is automatically cleared to 0 when the timer starts.

The TMn is cleared to 00H when an internal reset signal is generated.

Figure 5-33. Timer/Event Counter Mode Register Setup (1/2)

(a) In the case of timer/event counter (channel 0)

Address	7	6	5	4	3	2	1	0	Symbol
FA0H	—	TM06	TM05	TM04	TM03	TM02	—	—	TM0

Count pulse (CP) selection bit

TM06	TM05	TM04	Count pulse (CP)
0	0	0	TI0 rising edge
0	0	1	TI0 falling edge
1	0	0	fcc/2^{10}
1	0	1	fcc/2^{8}
1	1	0	fcc/2^{6}
1	1	1	fcc/2^{4}
Other than above			Not to be set

Timer start indication bit

TM03	When 1 is written into the bit, the counter and IRQT0 flag are cleared. If bit 2 is set to 1, count operation is started.

Operation mode

TM02	Count operation
0	Stop (retention of count contents)
1	Count operation

Figure 5-33. Timer/Event Counter Mode Register Setup (2/2)

(b) In the case of timer counter (channel 1)

Address	7	6	5	4	3	2	1	0	Symbol
FA8H	—	TM16	TM15	TM14	TM13	TM12	—	—	TM1

Count pulse (CP) selection bit

TM16	TM15	TM14	Count pulse (CP)
0	0	0	Rising edge of INTW (overflow output for clock timer)
1	0	0	fcc/2^{12}
1	0	1	fcc/2^{10}
1	1	0	fcc/2^{8}
1	1	1	fcc/2^{6}
Other than above			Not to be set

Timer start indication bit

TM13	When 1 is written to the bit, the counter and IRQT1 flag are cleared. If bit 2 is set to 1, count operation is started.

Operation mode

TM12	Count operation
0	Stop (retention of count contents)
1	Count operation

(b) Timer/event counter output enable flag (TOEn)

The TOEn is manipulated by a bit manipulation instruction.

The TOEn is cleared to 0 by an internal reset signal.

Figure 5-34. Timer/Event Counter Output Enable Flag Setup

Address		
FA2H	TOE0	Channel 0
FAAH	TOE1	Channel 1

Timer/event counter output enable flag (W)

0	Disabled (outputs the low-level signal).
1	Enabled.

(2) Timer/event counter time setting

[Timer setup time] (cycle) is found by dividing [modulo register contents + 1] by [count pulse (CP) frequency] selected by setting the mode register.

$$T\ (\text{sec}) = \frac{n+1}{f_{CP}} = (n + 1) \cdot (\text{resolution})$$

T (sec) : Timer setup time (seconds)
f_{CP} (Hz) : Count pulse frequency (Hz)
n : Modulo register content ($n \neq 0$)

Once the timer is set, interrupt request signal (IRQTn) is generated at the intervals set in the timer. Table 5-6 lists the resolution and longest setup time (time when FFH is set in the modulo register) for each count pulse to the timer/event counter.

Table 5-6. Resolution and Longest Setup Time

(a) When timer/event counter (channel 0)

Mode register			At 2 MHz		At 1 MHz	
TM06	TM05	TM04	Resolution	Longest setup time	Resolution	Longest setup time
1	0	0	512 µs	131 ms	1024 µs	262 ms
1	0	1	128 µs	32.8 ms	256 µs	65.5 ms
1	1	0	32 µs	8.19 ms	64 µs	16.4 ms
1	1	1	8 µs	2.05 ms	16 µs	4.10 ms

(b) When timer counter (channel 1)

Mode register			At 2 MHz		At 1 MHz	
TM16	TM15	TM14	Resolution	Longest setup time	Resolution	Longest setup time
1	0	0	2048 µs	524 ms	4096 µs	1049 ms
1	0	1	512 µs	131 ms	1024 µs	262 ms
1	1	0	128 µs	32.8 ms	256 µs	65.5 ms
1	1	1	32 µs	8.19 ms	64 µs	16.4 ms

(3) Timer/event counter operation

The timer/event counter operates as follows.

Figure 5-35 shows the configuration of the timer/event counter.

<1> The count pulse (CP) is selected by setting the mode register (TMn) and is input to the count register (Tn).

<2> The Tn is compared with the modulo register (TMODn), and if they are equal, a match signal is generated and the interrupt request flag (IRQTn) is set. At the same time, the timer out flip-flop (TOUT flip-flop) is inverted.

Figure 5-36 is a timing chart of the timer/event counter.

The timer/event counter normally begins operation in the following procedure.

<1> Set a count in the TMODn.

<2> Set the operating mode, count pulse, and start indication in the TMn.

Caution Set a value other than 00H in the modulo register (TMODn).

When using the timer/event counter output pin (PTOn), set the dual function pin P2n as follows.

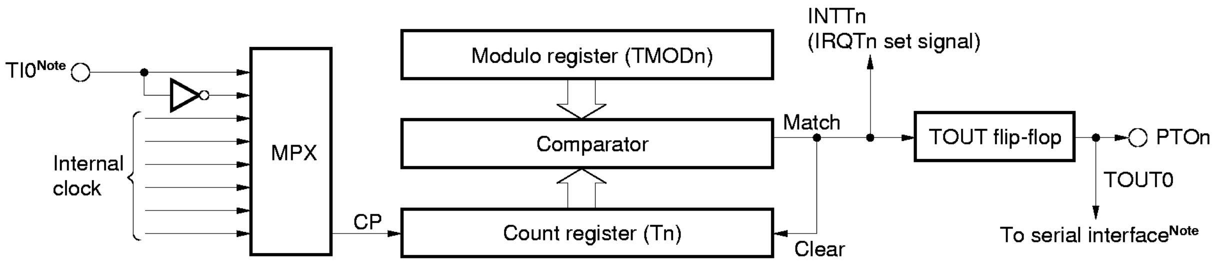
<1> Clear the output latch of P2n.

<2> Set port 2 to the output mode.

<3> Make a status wherein the internal pull-up resistor is not connected in port 2.

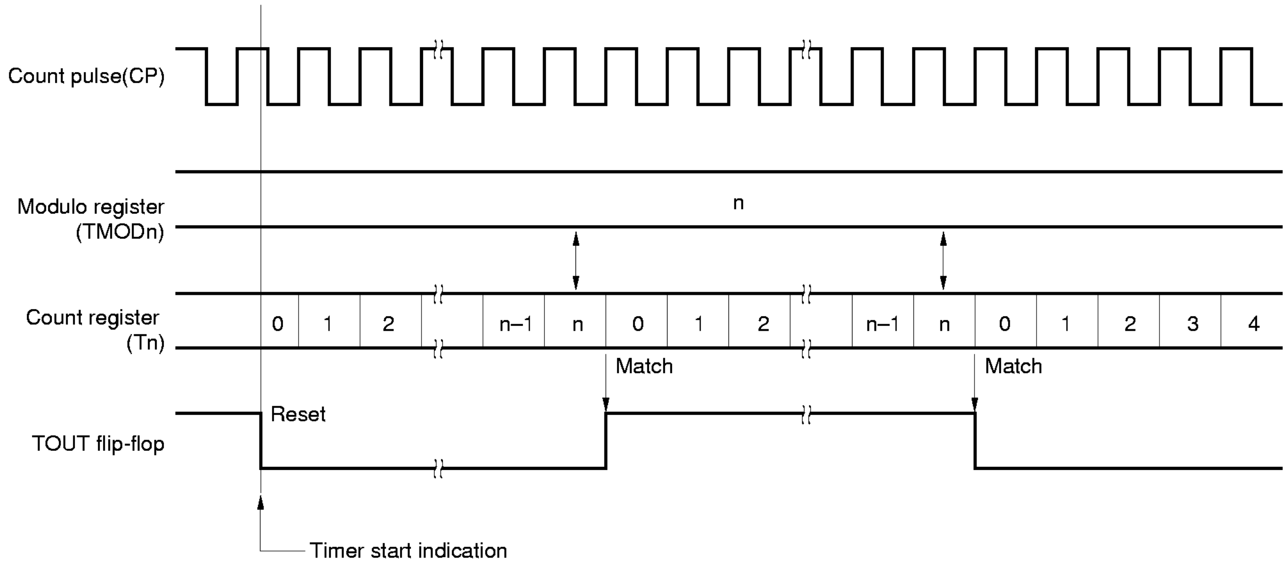
<4> Set the timer/event counter output enable flag (TOEn) to 1.

Figure 5-35. Configuration of Timer/Event Counter



Note Channel 0 of the timer/event counter only.

Figure 5-36. Count Operation Timing



(4) Applications of the timer/event counter

(a) Timer/event counter is used as an interval timer that generates interrupts at intervals of 30 ms.

- The high-order four bits of the mode register are set to 0100B to select maximum set time 131 ms (at 2 MHz).
- The low-order four bits of the mode register are set to 1100B.
- The modulo register is set to the following value:
  30 ms/512 µs = 58.6 ≒ 3BH

<Sample program>

```
SEL     MB15
MOV     XA,#3BH
MOV     TMOD0,XA        ; Set the modulo register
MOV     XA,#01001100B
MOV     TM0,XA          ; Set the mode register and start the timer
EI                      ; Enable an interrupt
EI      IET0            ; Enable a timer interrupt
```

Remark In this application, the TI0 pin can be used as an input pin.

(b) An interrupt is caused when the number of pulses (active high) applied to the TI0 pin reaches 100.

- The high-order four bits of the mode register are set to 0000 to select the rising edge.
- The low-order four bits of the mode register are set to 1100B.
- The modulo register is set to 99 = 100 – 1.

<Sample program>

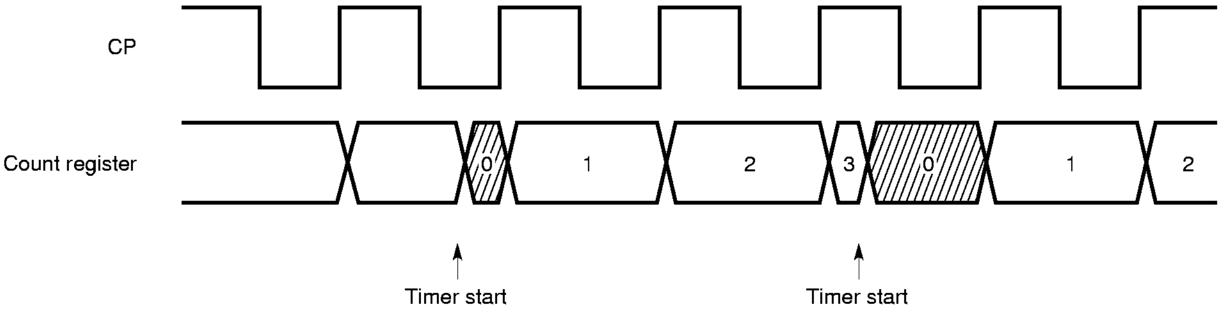
```
SEL    MB15
MOV    XA,#100 – 1
MOV    TMOD0,XA        ; Set the modulo register
MOV    XA,#00001100B
MOV    TM0,XA          ; Set the mode register
EI
EI     IET0            ; Enable INTT0
```

5.5.3 Notes on Timer/Event Counter Applications

(1) Time error at the start of the timer

A maximum error of one count pulse (CP) cycle from a value calculated according to **Section 5.5.2 (2)** occurs in a time period from the start of the timer (bit 3 of the TM0 is set) to the generation of a match signal. This is because the count register T0 is cleared not in phase with the CP as shown in Figure 5-37.

Figure 5-37. Error at the Start of the Timer



(2) Notes on the start of the timer

Usually, when the timer is started (bit 3 of the TM0 is set), the count register T0 and the interrupt request flag (IRQT0) are cleared. However, when the timer is placed in the operation mode, and the setting of IRQT0 and the start of the timer occur at the same time, IRQT0 may not be cleared. This causes no problem if IRQT0 is used for a vectored interrupt. However, if IRQT0 is being tested, a problem arises because IRQT0 is set even if the timer is started. Accordingly, in a situation where the timer is started on such timing that IRQT0 may be set, the timer must be restarted after it is once stopped (bit 2 of the TM0 is cleared to 0), or timer start operation must be performed twice.

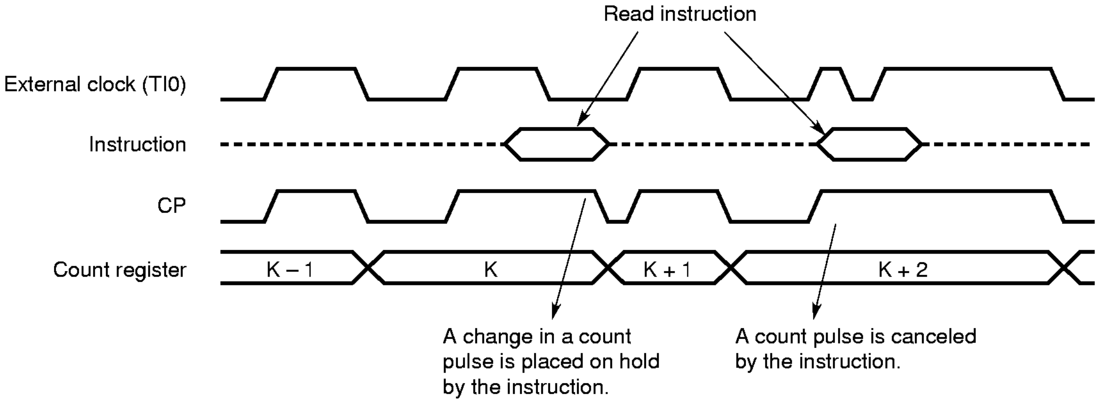
Example The timer is started on such timing that IRQT0 may be set.

```
SEL    MB15
MOV    XA,#0
MOV    TM0,XA    ; Stop the timer
MOV    XA,#4CH
MOV    TM0,XA    ; Restart
or
SEL    MB15
SET1   TM0.3
SET1   TM0.3     ; Restart
```

(3) Error in reading the count register

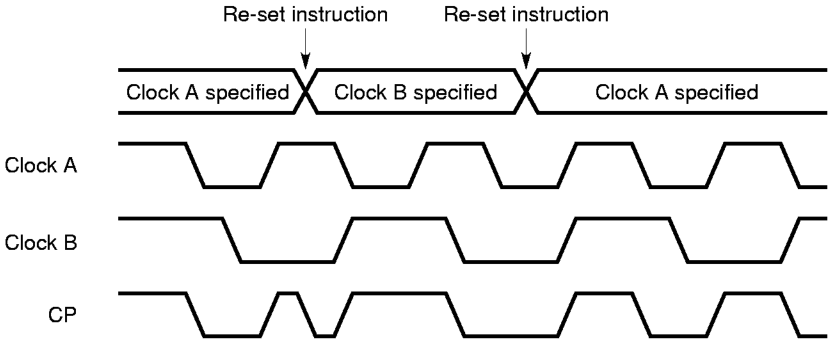
The contents of the count register can be read using an 8-bit data memory manipulation instruction at any time. During operation by such an instruction, all count pulse changes are held not to change the count register. This means that if the count pulse signal source is applied to the TI0 input, as many count pulses as corresponding to the time required to execute the instruction are cut. (When an internal clock is used for the count pulse signal, this problem does not occur because of synchronization with the instruction.)

Accordingly, in an attempt to read the contents of the count register with a count pulse signal applied to TI0, the signal must have a pulse wide enough to avoid incorrect counting even if count pulses are cut. That is, the contents of the count register are held by a read instruction for one machine cycle, so that a signal applied to the TI0 pin must have a pulse wider than that.

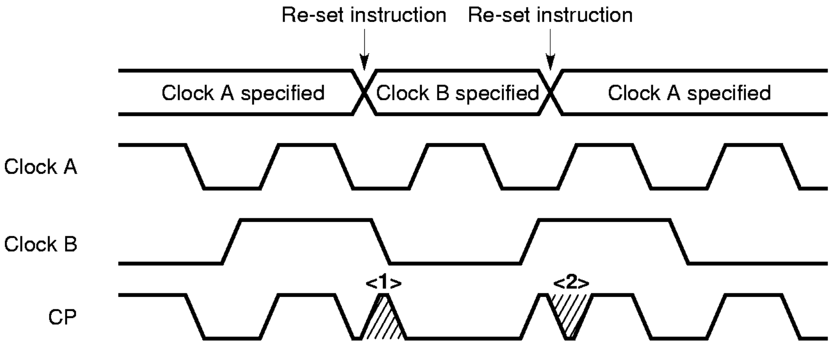

Read instruction
External clock (TI0)
Instruction
CP
Count register
K – 1
K
K + 1
K + 2
A change in a count pulse is placed on hold by the instruction.
A count pulse is canceled by the instruction.

(4) Notes on changing the count pulse

When the count pulse is changed by rewriting the contents of the timer/event counter mode register, this takes effect immediately after the rewrite instruction is executed.

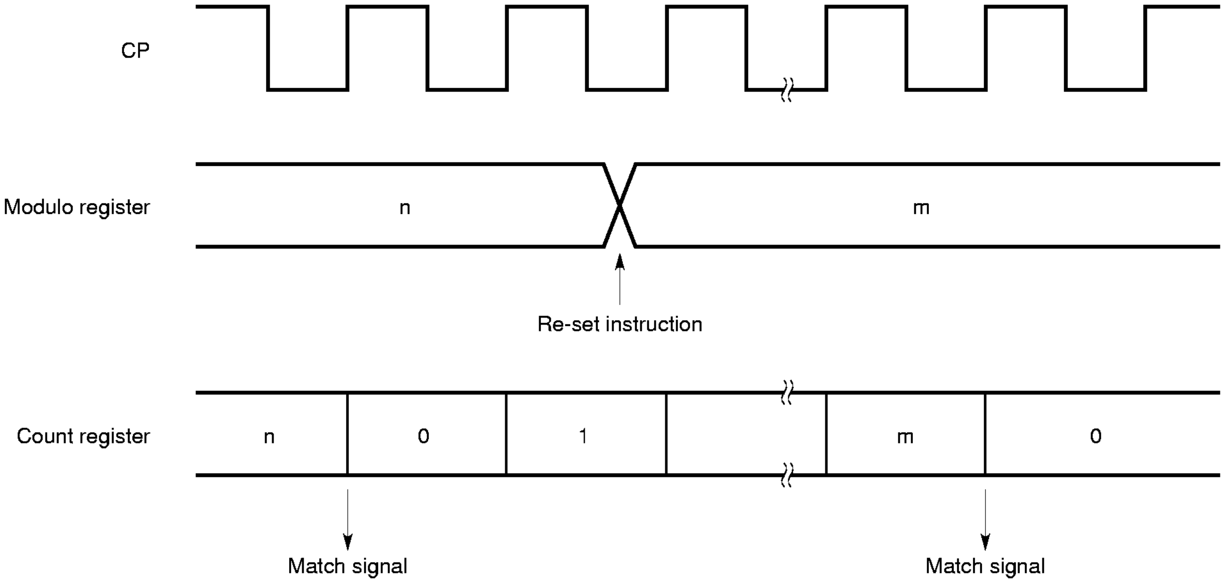


A combination of clocks used for changing count pulse signals can generate a spike (**<1>** or **<2>**) count pulse as shown in the figure below. In this case, an incorrect count operation may occur, or the contents of the count register may be destroyed. So when the count pulse is changed, bit 3 of the timer/event counter mode register must be set to 1, and the timer must be restarted at the same time.

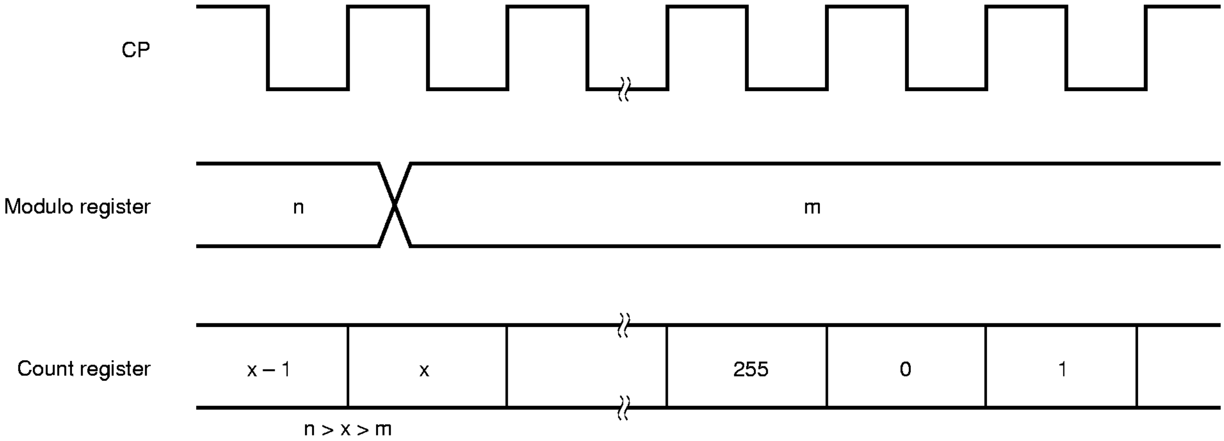


(5) Operation after the modulo register is changed

The contents of the modulo register are changed when an 8-bit data memory manipulation instruction is executed.



If the new value of the modulo register is less than the value of the count register, the count register continues count operation until it overflows, then it restarts count operation from 0. Accordingly, if the new value (m) of the modulo register is less than the value (n) before it is changed, the timer must be restarted after the contents of the modulo register are changed.



5.6 SERIAL INTERFACE

5.6.1 Serial Interface Functions

The µPD750108 contains a clock synchronous 8-bit serial interface, which has four modes.

The functions of the four modes are outlined below.

(1) Operation halt mode

This mode is used when serial transfer is not performed. This mode reduces power consumption.

(2) Three-wire serial I/O mode

In this mode, 8-bit data is transferred through three lines: Serial clock ($\overline{\text{SCK}}$), serial output (SO), and serial input (SI).

The three-wire serial I/O mode allows full-duplex transmission, so data transfer can be performed at higher speed.

The user can choose 8-bit data transfer starting with the MSB or LSB, so devices starting with either the MSB or LSB can be connected.

The three-wire serial I/O mode enables connections to be made with the 75XL series, 78K series, and many other types of peripheral I/O devices.

(3) Two-wire serial I/O mode

In this mode, 8-bit data is transferred through two lines: Serial clock ($\overline{\text{SCK}}$) and serial data bus (SB0 or SB1). By controlling output levels on the two lines by software, communication with multiple devices is enabled.

The output levels of $\overline{\text{SCK}}$ and SB0 (or SB1) can be controlled by software, so the user can match an arbitrary transfer format. This means that a line that has been required for handshaking to connect multiple lines can be eliminated for more efficient input/output port utilization.

(4) Serial bus interface (SBI) mode

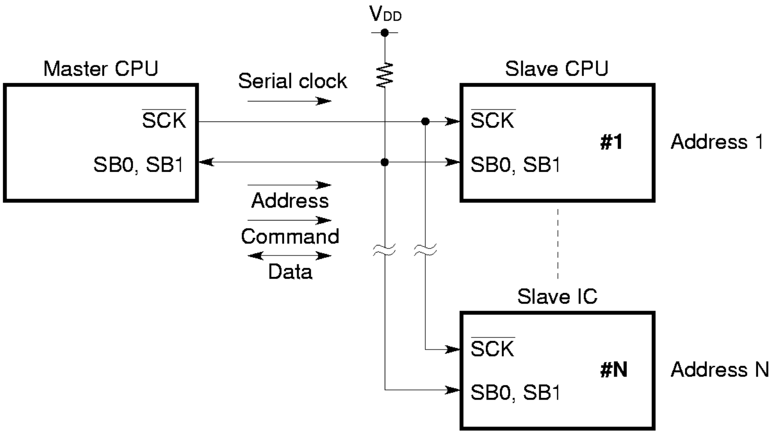
In this mode, communication with multiple devices can be performed using two lines: Serial clock ($\overline{\text{SCK}}$) and serial data bus (SB0 or SB1).

This mode conforms to the NEC serial bus format.

In this mode, the transmitter can output, on the serial data bus, an address for selecting a device subject to serial communication, commands directed to the remote device, and data.

The receiver can identify an address, commands, and data from received data by hardware. This function enables more efficient input/output port utilization as in the case of the two-wire serial I/O mode. In addition, this function can simplify the serial interface control portion of an application program.

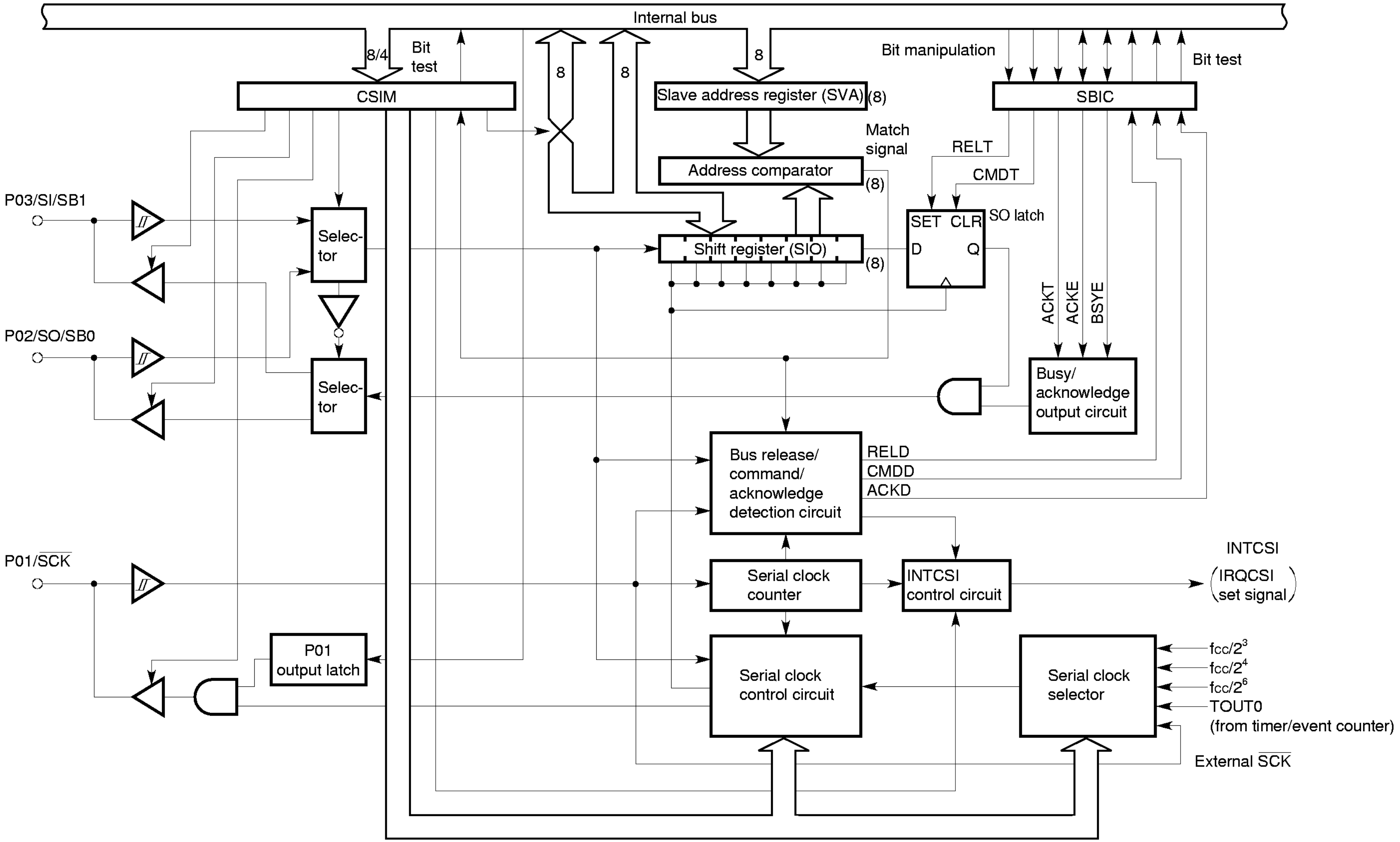
Figure 5-38. Example of the SBI System Configuration



5.6.2 Configuration of Serial Interface

Figure 5-39 shows the block diagram of the serial interface.

Figure 5-39. Block Diagram of the Serial Interface



(1) Serial operation mode register 0 (CSIM)

CSIM is an 8-bit register which specifies a serial interface operation mode, serial clock, wake-up function, and so forth. (See **(1)** in **Section 5.6.3** for details.)

(2) Serial bus interface control register (SBIC)

SBIC is an 8-bit register consisting of bits for controlling the serial bus and flags for indicating the states of input data from the serial bus. SBIC is used mainly in the SBI mode. (See **(2)** in **Section 5.6.3** for details.)

(3) Shift register (SIO)

SIO is an 8-bit register which converts 8-bit serial data to parallel data, and 8-bit parallel data to serial data. SIO performs transfer (shift) in phase with the serial clock. Transfers operations are controlled by writing data to SIO. (See **(3)** in **Section 5.6.3** for details.)

(4) SO latch

SO is a latch to hold the levels of pins SO and SB0, or SI and SB1, which can be controlled directly by software. In the SBI mode, SO is set when the eighth clock of $\overline{\text{SCK}}$ has been output. (See **(2)** in **Section 5.6.3** for details.)

(5) Serial clock selector

The serial clock selector selects the serial clock to be used.

(6) Serial clock counter

The serial clock counter counts the serial clock to be output or input during transfer, and checks whether 8-bit data has been transferred.

(7) Slave address register (SVA) and address comparator

- In the SBI mode

  SVA is used when the μPD750108 is used as a slave device. A slave sets the number assigned to it (slave address) in SVA. The master outputs a slave address to select a particular slave.

  Two data values (a slave address output from the master and the value of SVA) are compared with each other by the address comparator. If a match is found, the slave is selected.
- In the two-wire serial I/O mode or SBI mode

  SVA detects an error when data is transferred with the μPD750108 operating as the master or a slave. (See **(4)** in **Section 5.6.3** for details.)

(8) INTCSI control circuit

The INTCSI control circuit controls interrupt request processing. The circuit issues an interrupt request (INTCSI), and set an interrupt request flag (IRQCSI) in the following cases. (See **Figure 6-1**.)

- In the three-wire or two-wire serial I/O mode

  An interrupt request is issued whenever eight serial clocks are counted.
- In the SBI mode

  When WUP7**Note** = 0, an interrupt request is issued whenever eight serial clocks are counted. When WUP = 1, an interrupt request is issued when values of SVA and SIO match after an address is received.

Note WUP: Wake-up function specification bit (bit 5 of CSIM)

(9) Serial clock control circuit

The serial clock control circuit controls the serial clock to be supplied to the shift register, or controls the clock to be output to the $\overline{\text{SCK}}$ pin when the internal system clock is used.

(10) Busy/acknowledge output circuit and bus release/command/acknowledge detection circuit

The busy/acknowledge output circuit and bus release/command/acknowledge detection circuit output and detect control signals generated in the SBI mode.

These circuits do not operate in the three-wire or two-wire serial I/O mode.

(11) P01 output latch

The P01 output latch generates serial clock by software after the eighth serial clock has been output.

When the $\overline{\text{RESET}}$ signal is entered, this latch is set to 1.

To select the internal system clock as the serial clock, set the P01 output latch to 1.

5.6.3 Register Functions

(1) Serial operation mode register (CSIM)

Figure 5-40 shows the format of serial operation mode register (CSIM).

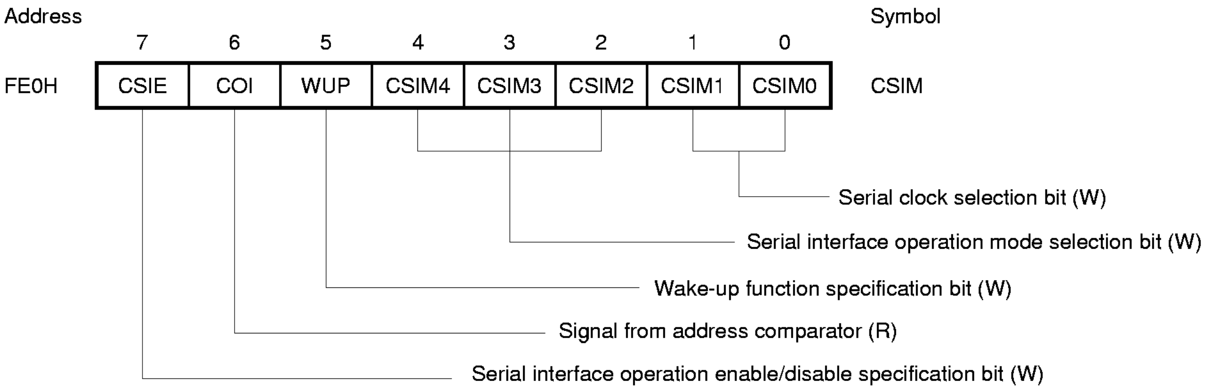
CSIM is an 8-bit register which specifies a serial interface operation mode, serial clock, wake-up function, and so forth.

CSIM is manipulated using an 8-bit memory manipulation instruction. The higher three bits can be manipulated bit by bit. Each bit can be manipulated using its name.

Each bit may or may not allow read and/or write operation (see **Figure 5-40**). Bit 6 allows bit test operation only; any data written to this bit is invalid.

When the $\overline{\text{RESET}}$ signal is generated, all bits are cleared to 0.

Figure 5-40. Format of Serial Operation Mode Register (CSIM) (1/4)


Address
Symbol
7
6
5
4
3
2
1
0
FE0H
CSIE
COI
WUP
CSIM4
CSIM3
CSIM2
CSIM1
CSIM0
CSIM
Serial clock selection bit (W)
Serial interface operation mode selection bit (W)
Wake-up function specification bit (W)
Signal from address comparator (R)
Serial interface operation enable/disable specification bit (W)

Remarks 1. (R) : Read only

2. (W): Write only

Figure 5-40. Format of Serial Operation Mode Register (CSIM) (2/4)

Serial interface operation enable/disable specification bit (W)

		Shift register operation	Serial clock counter	IRQCSI flag	SO/SB0 and SI/SB1 pins
CSIE	0	Shift operation disabled	Cleared	Held	Used only for port 0
	1	Shift operation enabled	Count operation	Can be set.	Used in each mode as well as for port 0

Signal from address comparator (R)

COI[Note]	Condition for being cleared (COI = 0)	Condition for being set (COI = 1)
	When the data in the slave address register (SVA) does not match the data in the shift register	When the data in the slave address register (SVA) matches the data in the shift register

Note COI can be read only before serial transfer is started or after serial transfer is completed. An undefined value may result during transfer.
COI data written by an 8-bit manipulation instruction is ignored.

Wake-up function specification bit (W)

WUP	0	Sets IRQCSI each time serial transfer is completed in each mode.
	1	Used in the SBI mode only to set IRQCSI only when an address received after bus release matches the data in the slave address register (wake-up state). SB0 or SB1 goes to high-impedance state.

Caution **When WUP = 1 is set during $\overline{\text{BUSY}}$ signal output, $\overline{\text{BUSY}}$ is not released. In the SBI mode, the $\overline{\text{BUSY}}$ signal is output until the next falling edge of the serial clock ($\overline{\text{SCK}}$) appears after release of $\overline{\text{BUSY}}$ is directed. Before setting WUP = 1, be sure to confirm that pin SB0 (or SB1) is high after releasing $\overline{\text{BUSY}}$.**

Figure 5-40. Format of Serial Operation Mode Register (CSIM) (3/4)

Serial interface operation mode selection bit (W)

CSIM4	CSIM3	CSIM2	Operation mode	Bit order of shift register	SO pin function	SI pin function
x	0	0	3-wire serial I/O mode	SIO_{7-0} <—> XA (Transfer start with MSB)	SO/P02 (CMOS output)	SI/P03 (Input)
		1		SIO_{0-7} <—> XA (Transfer starting with LSB)		
0	1	0	SBI mode	SIO_{7-0} <—> XA (Transfer starting with MSB)	SB0/P02 (N-ch open-drain I/O)	P03 input
1					P02 input	SB1/P03 (N-ch open-drain I/O)
0	1	1	2-wire serial I/O mode	SIO_{7-0} <—> XA (Transfer starting with MSB)	SB0/P02 (N-ch open-drain I/O)	P03 input
1					P02 input	SB1/P03 (N-ch open-drain I/O)

Remark x: Don’t care

Serial clock selection bit (W)

CSIM1	CSIM0	Serial clock			$\overline{\text{SCK}}$ pin mode
		3-wire serial I/O mode	SBI mode	2-wire serial I/O mode	
0	0	Input clock externally applied to $\overline{\text{SCK}}$ pin			Input
0	1	Timer/event counter output (TOUT0)			Output
1	0	$f_{CC}/2^4$ (125 kHz: during 2-MHz operation, 62.5 kHz: during 1-MHz operation)		$f_{CC}/2^6$ (31.3 kHz: during 2-MHz operation, 15.6 kHz: during 1-MHz operation)	
1	1	$f_{CC}/2^3$ (250 kHz: during 2-MHz operation, 125 kHz: during 1-MHz operation)			

Remarks 1. Each mode can be selected using CSIE, CSIM3, and CSIM2.

CSIE	CSIM3	CSIM2	Operation mode
0	x	x	Operation halt mode
1	0	x	Three-wire serial I/O mode
1	1	0	SBI mode
1	1	1	Two-wire serial I/O mode

Figure 5-40. Format of Serial Operation Mode Register (CSIM) (4/4)

Remarks 2. The P01/$\overline{\text{SCK}}$ pin assumes any of the following states according to the state of CSIE, CSIM1, and CSIM0:

CSIE	CSIM1	CSIM0	P01/$\overline{\text{SCK}}$ pin state
0	0	0	Input port
1	0	0	High impedance
0	0	1	High level output
0	1	0	
0	1	1	
1	0	1	Serial clock output (High level output)
1	1	0	
1	1	1	

3. When clearing CSIE during serial transfer, use the following procedure:

<1> Disable interrupts by clearing the interrupt enable flag (IECSI).

<2> Clear CSIE.

<3> Clear the interrupt request flag (IRQCSI).

Examples 1. $f_X/2^4$ is selected as the serial clock, serial interrupt IRQCSI, is generated each time serial transfer is completed, and serial transfer is performed in the SBI mode with the SB0 pin used as the serial data bus.

```
SEL     MB15              ; or CLR1  MBE
MOV     XA,#10001010B
MOV     CSIM,XA           ; CSIM <– 10001010B
```

2. Serial transfer dependent on the contents of CSIM is enabled.

```
SEL     MB15              ; or CLR1  MBE
SET1    CSIE
```

(2) Serial bus interface control register (SBIC)

Figure 5-41 shows the format of the serial bus interface control register (SBIC).

SBIC is an 8-bit register consisting of bits for controlling the serial bus and flags for indicating the states of input data from the serial bus. SBIC is used mainly in the SBI mode.

SBIC is manipulated using a bit manipulation instruction. SBIC cannot be manipulated using a 4-bit or 8-bit memory manipulation instruction.

Each bit may or may not allow read and/or write operation (**Figure 5-41**).

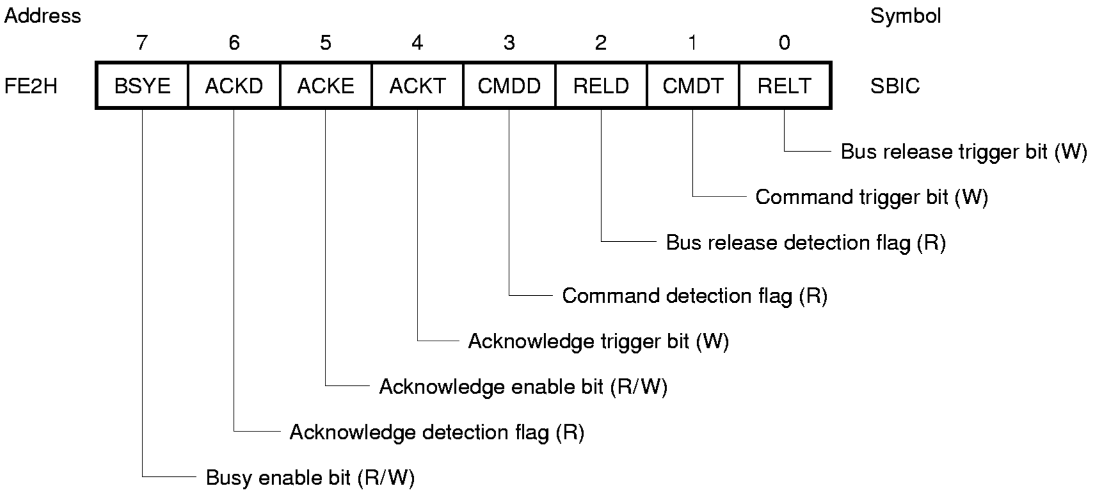
When the $\overline{\text{RESET}}$ signal is generated, all bits are cleared to 0.

Caution Only the following bits can be used in the three-wire and two-wire serial I/O modes:

- **Bus release trigger bit (RELT): Sets the SO latch.**
- **Command trigger bit (CMDT): Clears the SO latch**

.

Figure 5-41. Format of Serial Bus Interface Control Register (SBIC) (1/3)


Address
Symbol
7
6
5
4
3
2
1
0
FE2H
BSYE
ACKD
ACKE
ACKT
CMDD
RELD
CMDT
RELT
SBIC
Bus release trigger bit (W)
Command trigger bit (W)
Bus release detection flag (R)
Command detection flag (R)
Acknowledge trigger bit (W)
Acknowledge enable bit (R/W)
Acknowledge detection flag (R)
Busy enable bit (R/W)

Remarks 1. (R): Read only

2. (W): Write only

3. (R/W): Read/write

Figure 5-41. Format of Serial Bus Interface Control Register (SBIC) (2/3)

Busy enable bit (R/W)

BSYE	0	<1> The busy signal is automatically disabled. <2> Busy signal output is stopped in phase with the falling edge of $\overline{\text{SCK}}$ immediately after clear instruction execution.
	1	The busy signal is output after the acknowledge signal in phase with the falling edge of $\overline{\text{SCK}}$.

Acknowledge detection flag (R)

ACKD	Condition for being cleared (ACKD = 0)	Condition for being set (ACKD = 1)
	<1> The transfer operation is started. <2> The $\overline{\text{RESET}}$ signal is generated.	The acknowledge signal ($\overline{\text{ACK}}$) is detected (in phase with the rising edge of $\overline{\text{SCK}}$).

Acknowledge enable bit (R/W)

ACKE	0	Disables automatic output of the acknowledge signal ($\overline{\text{ACK}}$). (Output by ACKT is possible.)	
	1	When set before transfer	$\overline{\text{ACK}}$ is output in phase with the 9th clock of $\overline{\text{SCK}}$.
		When set after transfer	$\overline{\text{ACK}}$ is output in phase with $\overline{\text{SCK}}$ immediately following the set instruction execution.

Acknowledge trigger bit (W)

ACKT	When set after transfer, $\overline{\text{ACK}}$ is output in phase with the next $\overline{\text{SCK}}$. After $\overline{\text{ACK}}$ signal output, this bit is automatically cleared to 0.

Cautions 1. Never set ACKT before or during serial transfer.

2. ACKT cannot be cleared by software.

3. Before setting ACKT, set ACKE = 0.

Command detection flag (R)

CMDD	Condition for being cleared (CMDD = 0)	Condition for being set (CMDD = 1)
	<1> The transfer start instruction is executed. <2> The bus release signal (REL) <3> The $\overline{\text{RESET}}$ signal is generated. <4> CSIE = 0 (**Figure 5-40**)	The command signal (CMD) is detected.

Figure 5-41. Format of Serial Bus Interface Control Register (SBIC) (3/3)

Bus release detection flag (R)

RELD	Condition for being cleared (RELD = 0)	Condition for being set (RELD = 1)
	<1> The transfer start instruction is executed. **<2>** The RESET signal is generated. **<3>** CSIE = 0 (**Figure 5-40**) **<4>** SVA does not match SIO when an address is received.	The bus release signal (REL) is detected.

Command trigger bit (W)

CMDT	Control bit for command signal (CMD) trigger output. By setting CMDT = 1, the SO latch is cleared. Then the CMDT bit is automatically cleared to 0.

Caution Never clear SB0 (or SB1) during serial transfer. Be sure to clear SB0 (or SB1) before or after serial transfer

Bus release trigger bit (W)

RELT	Control bit for bus release signal (REL) trigger output. By setting RELT = 1, the SO latch is set to 1. Then the RELT bit is automatically cleared to 0.

Caution Never clear SB0 (or SB1) during serial transfer. Be sure to clear SB0 (or SB1) before or after serial transfer.

Examples 1. A command signal is output.

```
SEL    MB15       ; or CLR1  MBE
SET1   CMDT
```

2. RELD and CMDD are tested to identify the types of received data and the types of processing accordingly. By setting WUP = 1, this interrupt routine is processed only when an address match is found.

```
SEL    MB15
SKF    RELD       ; RELD test
BR     !ADRS
SKT    CMDD       ; CMDD test
BR     !DATA
BR     !CMD
CMD: ............ ; Command analysis
DATA:............ ; Data processing
ADRS: ........... ; Address decode
```

(3) Shift register (SIO)

Figure 5-42 shows the configuration of peripheral hardware of shift register. SIO is an 8-bit register which performs parallel-serial conversion and serial transfer (shift) operation in phase with the serial clock. Serial transfer is started by writing data to SIO.

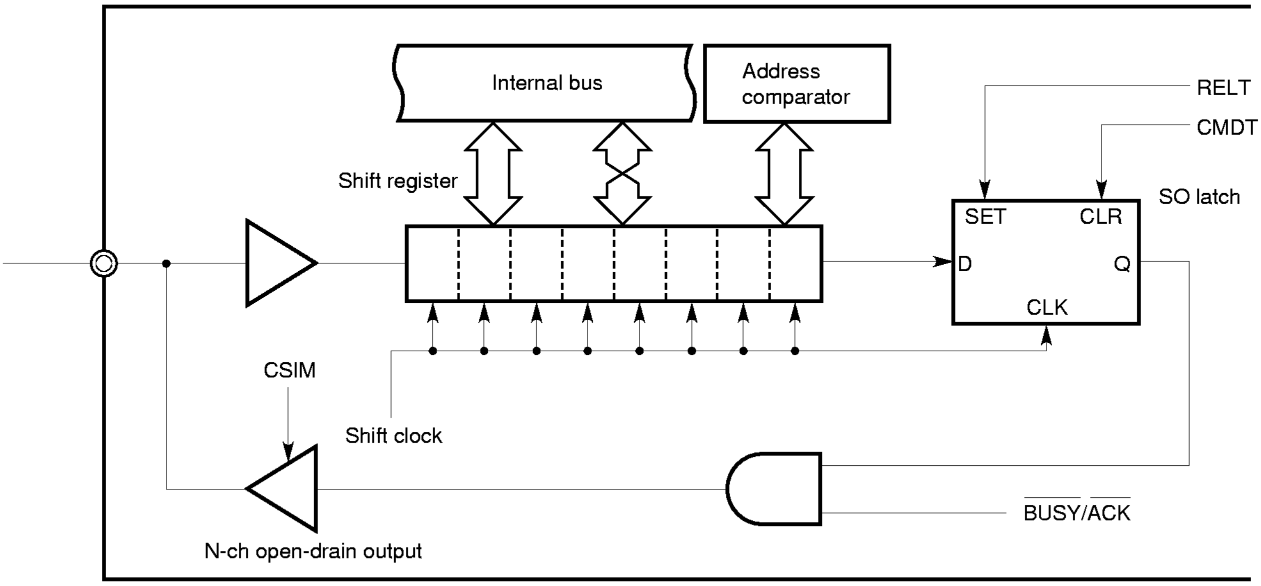
In transmission, data written to SIO is output on the serial output (SO) or serial data bus (SB0 or SB1). In receive operation, data is read from the serial input (SI) or SB0 or SB1 into SIO.

Data can be read from or written to SIO by using an 8-bit manipulation instruction.

When the $\overline{\text{RESET}}$ signal is generated during operation, the value of SIO is undefined. When the $\overline{\text{RESET}}$ signal is generated in the standby mode, the value of SIO is preserved.

Shift operation is stopped after 8-bit send or receive operation is completed.

Figure 5-42. Peripheral Hardware of Shift Register



The timing for reading SIO and start of serial transfer (writing to SIO) is as follows:

- When the serial interface operation enable/disable bit (CSIE) = 1. However, the case where CSIE is set to 1 after data is written to the shift register is excluded.
- When the serial clock is masked after 8-bit serial transfer
- $\overline{\text{SCK}}$ is high.

When reading from or writing to SIO, make sure that $\overline{\text{SCK}}$ is high.

In the two-wire serial I/O mode and SBI mode, the pins specified for the data bus are used for both input and output. Because the configuration of output pins is N-ch open-drain, write FFH in SIO for devices that are to receive data.

(4) Slave address register (SVA)

The slave address register (SVA) is an 8-bit register for a slave to set its slave address (number assigned to it).

SVA is manipulated using an 8-bit manipulation instruction.

When the $\overline{\text{RESET}}$ signal is generated, the value of SVA is undefined. However, the value of SVA is preserved when the $\overline{\text{RESET}}$ signal is generated in the standby mode.

SVA has the following two functions:

(a) Slave address detection

[In the SBI mode]
SVA is used when the µPD750108 is connected as a slave device to the serial bus. SVA is an 8-bit register for a slave to set its slave address (number assigned to it). The master outputs a slave address to the connected slaves to select a particular slave. Two data values (a slave address output from the master and the value of SVA) are compared with each other by the address comparator. If a match is found, the slave is selected.

At this time, bit 6 (COI) of serial operation mode register (CSIM) is set to 1.

If a match with received address data is not found, the bus release detection flag (RELD) is cleared to 0. When WUP = 1 (wake-up state detection), IRQCSI is set only when a match is found. With this interrupt request, the µPD750108 can be informed of a communication request transmitted from the master.

(b) Error detection

[In the two-wire serial I/O mode or SBI mode]
SVA detects an error when addresses, commands, or data is transferred with the µPD750108 operating as the master or when data is transferred with the µPD750108 operating as a slave. (For details, see **(6)** in **Section 5.6.6** and **(8)** in **Section 5.6.7**.)

5.6.4 Operation Halt Mode

The operation halt mode is used when serial transfer is not performed. This mode reduces power consumption.

The shift register does not perform shift operation in this mode, so the shift register can be used as a normal 8-bit register.

When the $\overline{\text{RESET}}$ signal is entered, the operation halt mode is set. The P02/SO/SB0 pin and P03/SI/SBI pin function as input-only port pins. The P01/$\overline{\text{SCK}}$ pin can be used as an input port pin by setting the serial operation mode register.

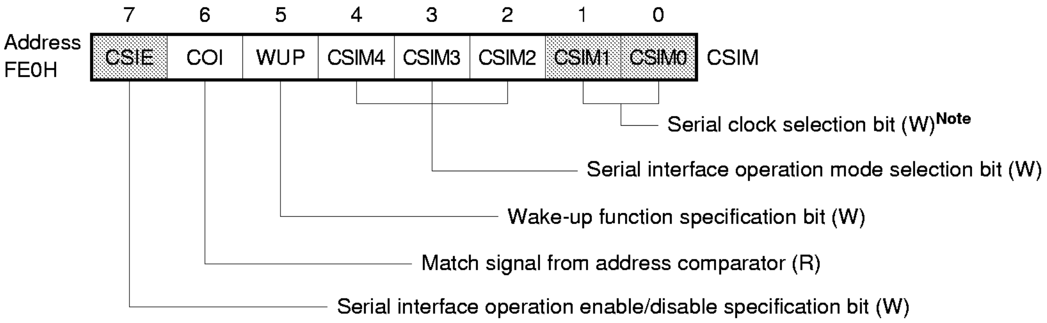
(1) Register setting

To set the operation halt mode, manipulate serial operation mode register (CSIM). (For details on CSIM format, see **(1)** in **Section 5.6.3**.)

CSIM is manipulated with an 8-bit manipulation instruction. Only the CSIE bit of CSIM can be independently manipulated. CSIM can also be manipulated using the name of each bit.

When the $\overline{\text{RESET}}$ signal is entered, CSIM is set to 00H.

In the figure below, hatched portions indicate bits used in the operation halt mode.


7
6
5
4
3
2
1
0
Address
FE0H
CSIE
COI
WUP
CSIM4
CSIM3
CSIM2
CSIM1
CSIM0
CSIM
Serial clock selection bit (W)Note
Serial interface operation mode selection bit (W)
Wake-up function specification bit (W)
Match signal from address comparator (R)
Serial interface operation enable/disable specification bit (W)

Note The status of the P01/$\overline{\text{SCK}}$ pin is selectable.

Remark (R): Read only
(W): Write only

Serial interface operation enable/disable specification bit (W)

		Shift register operation	Serial clock counter	IRQCSI flag	SO/SB0 and SI/SB1 pins
CSIE0	0	Shift operation disabled	Cleared	Held	Used only for port 0

Serial clock selection bit (W)

The P01/$\overline{\text{SCK}}$ pin assumes the following state according to the setting of CSIM0 and CSIM1:

CSIM1	CSIM0	P01/$\overline{\text{SCK}}$ pin state
0	0	High impedance
0	1	High level output
1	0	
1	1	

When clearing CSIE during serial transfer, use the following procedure:

<1> Disable interrupts by clearing the interrupt enable flag (IECSI).
<2> Clear CSIE.
<3> Clear the interrupt request flag (IRQCSI).

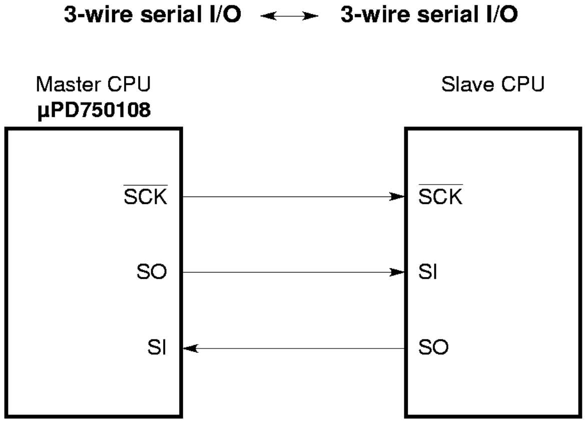
5.6.5 Three-Wire Serial I/O Mode Operations

The three-wire serial I/O mode is compatible with other modes used in the 75XL series, 75X series, μPD7500 series, and 87AD series.

Communication is performed using three lines:

Serial clock ($\overline{\text{SCK}}$), serial output (SO), and serial input (SI).

Figure 5-43. Example of Three-Wire Serial I/O System Configuration



Remark The μPD750108 can also be used as a slave CPU.

(1) Register setting

To set the three-wire serial I/O mode, manipulate the following two registers:

- Serial operation mode register (CSIM)
- Serial bus interface control register (SBIC)

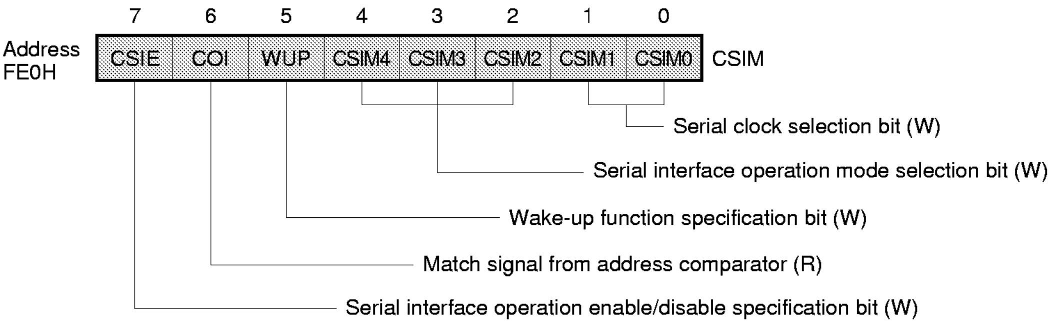
(a) Serial operation mode register (CSIM)

To use the three-wire serial I/O mode, set CSIM as shown below. (For details on CSIM format, see **(1)** in **Section 5.6.3**.)

CSIM0 is manipulated using an 8-bit manipulation instruction. Bits 7, 6, and 5 of CSIM can be manipulated bit by bit.

When the $\overline{\text{RESET}}$ signal is input, CSIM is set to 00H.

In the figure below, hatched portions indicate the bits used in the three-wire serial I/O mode.


7 6 5 4 3 2 1 0
Address FE0H
CSIE COI WUP CSIM4 CSIM3 CSIM2 CSIM1 CSIM0
CSIM
Serial clock selection bit (W)
Serial interface operation mode selection bit (W)
Wake-up function specification bit (W)
Match signal from address comparator (R)
Serial interface operation enable/disable specification bit (W)

Remark (R): Read only
(W): Write only

Serial interface operation enable/disable specification bit (W)

		Shift register operation	Serial clock counter	IRQCSI flag	SO/SB0 and SI/SB1 pins
CSIE	1	Shift operation enabled	Count operation	Can be set	Used in each mode as well as for port 0

Signal from address comparator (R)

COI[Note]	Condition for being cleared (COI = 0)	Condition for being set (COI = 1)
	When the slave address register (SVA) does not match the data of the shift register	When the slave address register (SVA) matches the data of the shift register

Note COI can be read only before serial transfer is started or after serial transfer is completed. An undefined value may be read during transfer. COI data written by an 8-bit manipulation instruction is ignored.

Wake-up function specification bit (W)

WUP	0	Sets IRQCSI each time serial transfer is completed.

Serial interface operation mode selection bit (W)

CSIM4	CSIM3	CSIM2	Shift register sequence	SO pin function	SI pin function
x	0	0	SIO_{7-0} <—> XA (Transfer starting with MSB)	SO/P02 (CMOS output)	SI/P03 (Input)
		1	SIO_{0-7} <—> XA (Transfer starting with LSB)		

Remark x: Don’t care

Serial clock selection bit (W)

CSIM1	CSIM0	Serial clock	$\overline{SCK}$ pin mode
0	0	External clock applied to $\overline{SCK}$ pin	Input
0	1	Timer/event counter output (TOUT0)	Output
1	0	$f_{CC}/2^4$ (125 kHz: during 2-MHz operation, 62.5 kHz: during 1-MHz operation)	
1	1	$f_{CC}/2^3$ (250 kHz: during 2-MHz operation, 125 kHz: during 1-MHz operation)	

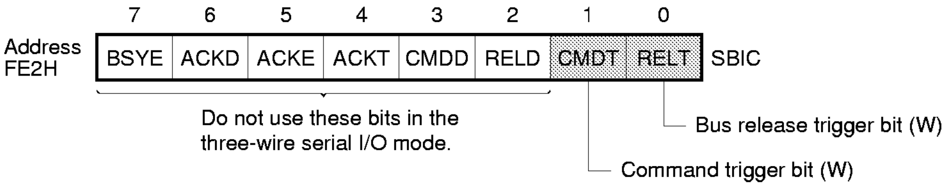
(b) Serial bus interface control register (SBIC)

To use the three-wire serial I/O mode, set SBIC as shown below. (For details on SBIC format, see **(2)** in **Section 5.6.3**.)

SBIC is manipulated using a bit memory manipulation instruction.

When the $\overline{\text{RESET}}$ signal is input, SBIC is set to 00H.

In the figure below, hatched portions indicate the bits used in the three-wire serial I/O mode.



Address	7	6	5	4	3	2	1	0	
FE2H	BSYE	ACKD	ACKE	ACKT	CMDD	RELD	CMDT	RELT	SBIC

Do not use these bits in the three-wire serial I/O mode.

Bus release trigger bit (W)

Command trigger bit (W)

Remark (W): Write only

Command trigger bit (W)

CMDT	Control bit for command signal (CMD) trigger output. By setting CMDT = 1, the SO latch is cleared. Then the CMDT bit is automatically cleared to 0.

Bus release trigger bit (W)

RELT	Control bit for bus release signal (REL) trigger output. By setting RELT = 1, the SO latch is set to 1. Then the RELT bit automatically cleared to 0.

Caution Never use bits other than RELT and CMDT in the three-wire serial I/O mode.

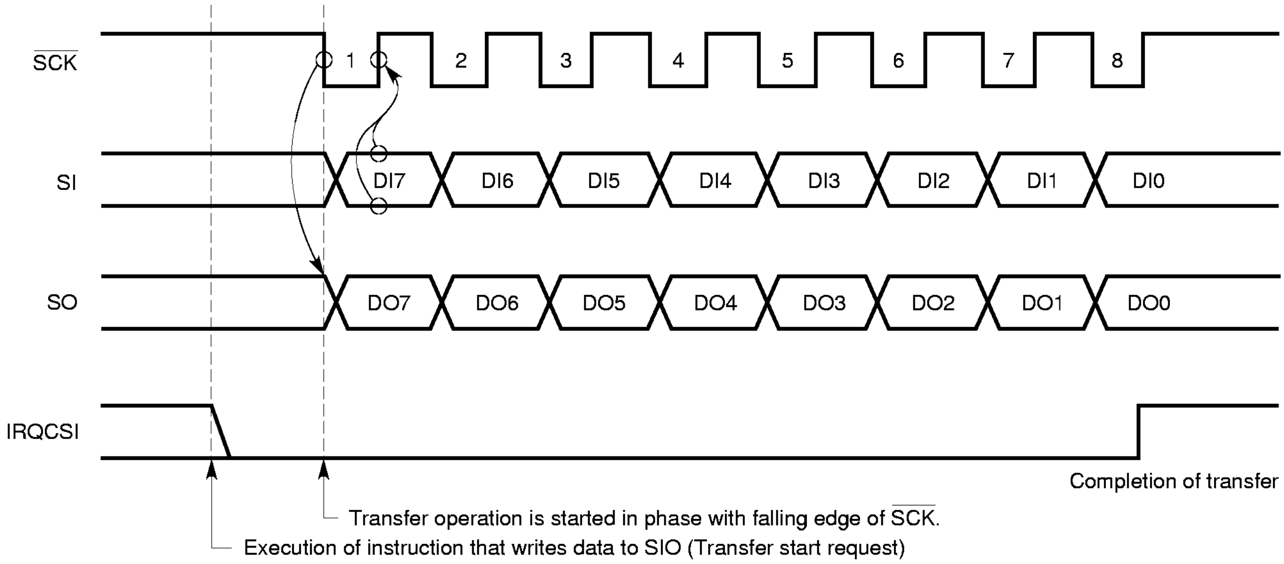
(2) Communication operation

The three-wire serial I/O mode transfers data, with eight bits as one block. Data is transferred bit by bit in phase with the serial clock.

The shift register performs shift operation on the falling edge of the serial clock ($\overline{\text{SCK}}$). Send data is latched on the SO latch, and is output on the SO pin. Receive data applied to the SI pin is latched in the shift register on the rising edge of $\overline{\text{SCK}}$.

When eight bits have been transferred, shift register operation automatically terminates setting the interrupt request flag (IRQCSI).

Figure 5-44. Timing of Three-Wire Serial I/O Mode


SCK
SI
SO
IRQCSI
1
2
3
4
5
6
7
8
DI7
DI6
DI5
DI4
DI3
DI2
DI1
DI0
DO7
DO6
DO5
DO4
DO3
DO2
DO1
DO0
Completion of transfer
Transfer operation is started in phase with falling edge of SCK.
Execution of instruction that writes data to SIO (Transfer start request)

The SO pin becomes a CMOS output and outputs the state of the SO latch. So the output state of the SO pin can be manipulated by setting the RELT bit and CMDT bit.

However, this manipulation must not be performed during serial transfer.

The output level of the SCK pin can be controlled by manipulating the P01 output latch in the output mode (internal system clock mode). (See **Section 5.6.8**.)

(3) Serial clock selection

To select the serial clock, manipulate bits 0 and 1 of serial operation mode register 0 (CSIM). The serial clock can be selected out of the following four clocks:

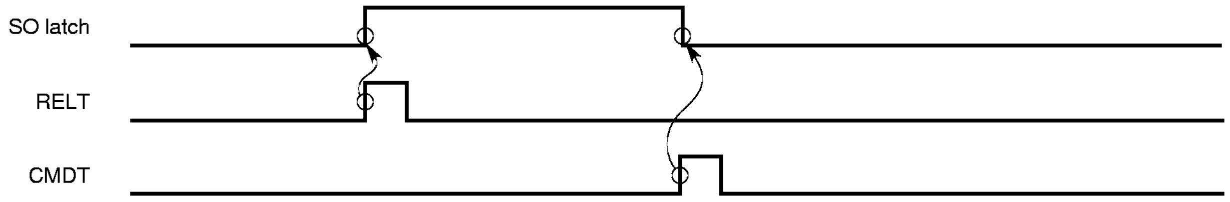
Table 5-7. Serial Clock Selection and Application (In the Three-Wire Serial I/O Mode)

Mode register		Serial clock		Timing for shift register R/W and start of serial transfer	Application
CSIM 1	CSIM 0	Source	Masking of serial clock		
0	0	External SCK	Automatically masked when 8-bit data transfer is completed	<1> In the operation halt mode (CSIE = 0) <2> When the serial clock is masked after 8-bit transfer <3> When SCK is high	Slave CPU
0	1	TOUT flip-flop			Half-duplex asynchronous transfer (software control)
1	0	$f_{CC}/2^4$			Middle-speed serial transfer
1	1	$f_{CC}/2^3$			High-speed serial transfer

(4) Signals

Figure 5-45 shows operations of RELT and CMDT.

Figure 5-45. Operations of RELT and CMDT



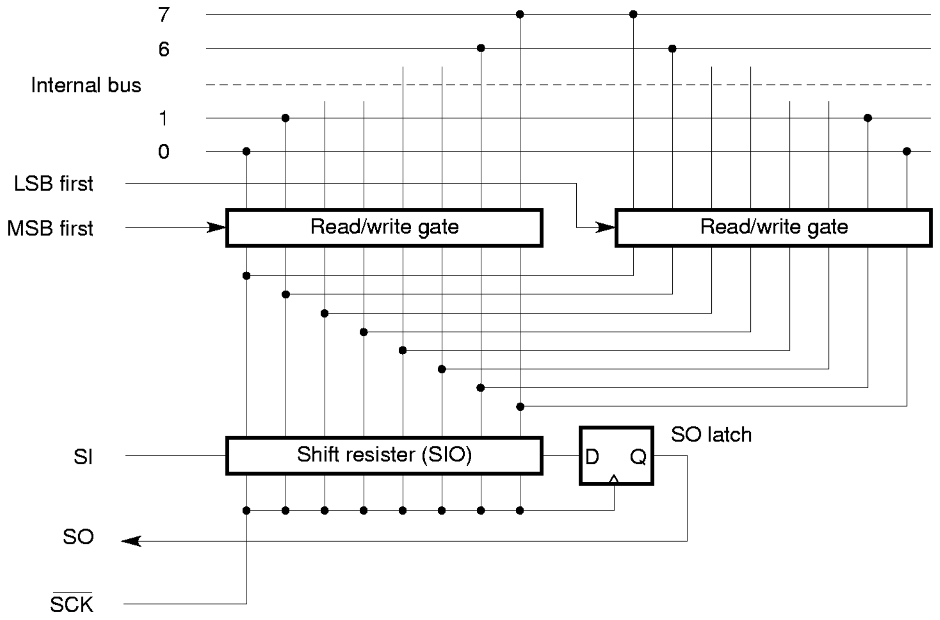
(5) Switching between MSB and LSB as the first transfer bit

The three-wire serial I/O mode has a function that can switch between the MSB and LSB as the first bit of transfer.

Figure 5-46 shows the configuration of shift register (SIO) and internal bus. As shown in Figure 5-46, read or write operation can be performed by switching between the MSB and LSB.

This switching can be specified using bit 2 of serial operation mode register (CSIM).

Figure 5-46. Transfer Bit Switching Circuit



The first bit is switched by changing the order of data bits written to shift register (SIO). The shift operation order of SIO is always the same.

Accordingly, the first bit must be switched between the MSB and LSB before writing data to the shift register.

(6) Transfer start

Serial transfer is started by writing transfer data into shift register (SIO), provided that the following two conditions are satisfied:

- The serial interface operation enable/disable specification bit (CSIE) is set to 1.
- The internal serial clock is not operating after 8-bit serial transfer, or $\overline{\text{SCK}}$ is high.

Caution Setting CSIE to 1 after writing data to the shift register does not start transfer.

When eight bits have been transferred, serial transfer automatically terminates setting the interrupt request flag (IRQCSI).

Example To transfer the RAM data specified with the HL register to SIO, load the SIO data to the accumulator and start serial transfer:

```
MOV  XA,@HL     ; Fetch transmit data from RAM
SEL  MB15       ; or CLR1  MBE
XCH  XA,SIO     ; Exchange transmit data and receive data, and start transfer
```

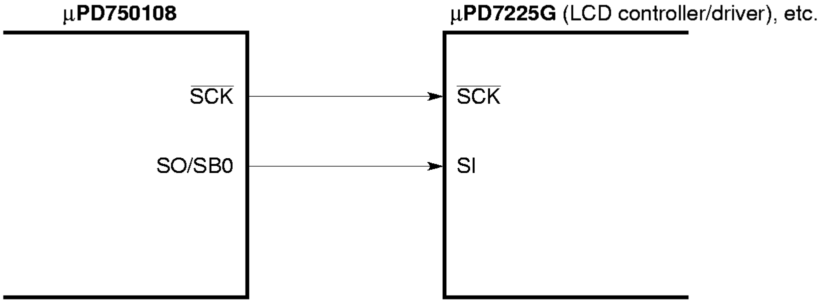
(7) Application of the three-wire serial I/O mode

(a) Data is transferred starting with the MSB on a transfer clock of 62.5 kHz (during 1-MHz operation). (Master operation)

<Sample program>

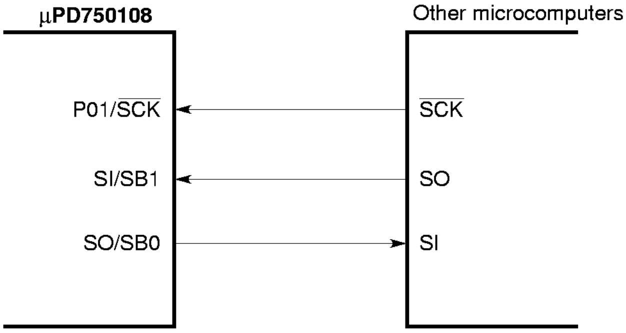
```
CLR1 MBE
MOV  XA,#10000010B
MOV  CSIM,XA            ; Set transfer mode
MOV  XA,TDATA           ; TDATA is transfer data storage address
MOV  SIO,XA             ; Set transfer data, and start transfer
```

Caution A second or subsequent transfer can be started by setting data in SIO (MOV SIO,XA or XCH XA,SIO).



In this case, the SI/SBI pin on the μPD750108 can be used as an input.

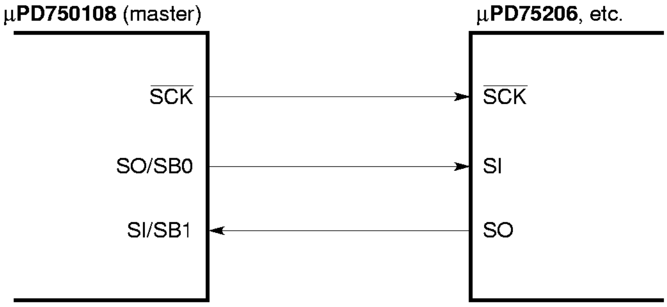
(b) Data is transmitted and received starting with the LSB on an external clock (slave operation).
(In this case, the function of inverting the MSB/LSB is used for shift register read/write operation.)



<Sample program>

```
          Main routine
          CLR1    MBE
          MOV     XA,#84H
          MOV     CSIM,XA       ; Serial operation halt, MSB/LSB invert mode, external clock
          MOV     XA,TDATA
          MOV     SIO,XA        ; Set transfer data, and start transfer
          EI      IECSI
          EI

Interrupt routine (MBE = 0)
          MOV     XA,TDATA
          XCH     XA,SIO        ; Start to transfer receive data and transmit data
          MOV     RDATA,XA      ; Save receive data
          RETI
```

(c) Data is transmitted and received by using a transfer clock of 125 kHz (during 1-MHz operation).



<Sample program> (master side):

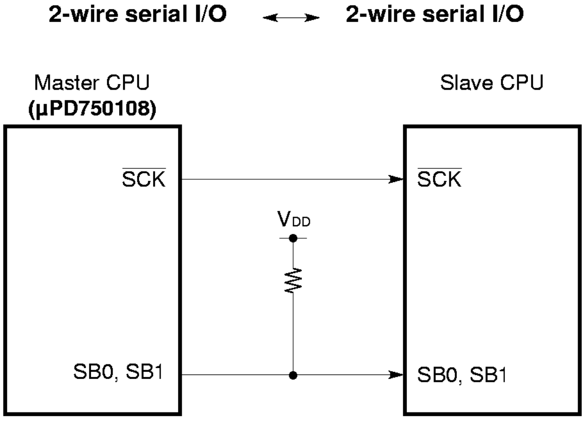
```
          CLR1      MBE
          MOV       XA,#10000011B
          MOV       CSIM,XA         ; Set transfer mode
          MOV       XA,TDATA
          MOV       SIO,XA          ; Set transfer data, and start transfer
          .
          .
          .
LOOP :    SKTCLR    IRQCSI          ; Test IRQCSI
          BR        LOOP
          MOV       XA,SIO          ; Read in receive data
```

5.6.6 Two-Wire Serial I/O Mode

The two-wire serial I/O mode can be made compatible with any communication format by programming.

In this mode, communication is basically performed using two lines: Serial clock ($\overline{\mathrm{SCK}}$) and serial data input/output (SB0 or SB1).

Figure 5-47. Example of Two-Wire Serial I/O System Configuration



Remark The µPD750108 can also be used as a slave CPU.

(1) Register setting

To set the two-wire serial I/O mode, manipulate the following two registers:

- Serial operation mode register (CSIM)
- Serial bus interface control register (SBIC)

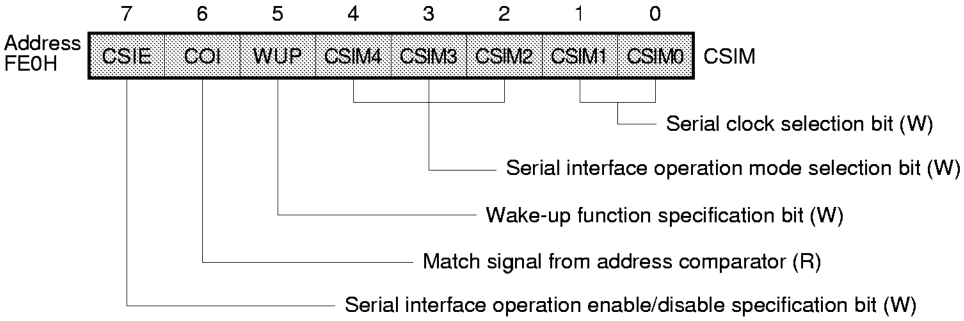
(a) Serial operation mode register (CSIM)

To use the two-wire serial I/O mode, set CSIM as shown below. (For details on CSIM format, see **(1)** in **Section 5.6.3**.)

CSIM is manipulated using an 8-bit manipulation instruction. Bits 7, 6, and 5 of CSIM can be manipulated bit by bit.

When the $\overline{\text{RESET}}$ signal is input, CSIM is set to 00H.

In the figure below, hatched portions indicate the bits used in the two-wire serial I/O mode.


7
6
5
4
3
2
1
0
Address
FE0H
CSIE
COI
WUP
CSIM4
CSIM3
CSIM2
CSIM1
CSIM0
CSIM
Serial clock selection bit (W)
Serial interface operation mode selection bit (W)
Wake-up function specification bit (W)
Match signal from address comparator (R)
Serial interface operation enable/disable specification bit (W)

Remark (R): Read only
(W): Write only

Serial interface operation enable/disable specification bit (W)

		Shift register operation	Serial clock counter	IRQCSI flag	SO/SB0 and SI/SB1 pins
CSIE	1	Shift operation enabled	Count operation	Can be set	Used in each mode as well as for port 0

Signal from address comparator (R)

COI[Note]	Condition for being cleared (COI = 0)	Condition for being set (COI = 1)
	When the slave address register (SVA) does not match the data of the shift register	When the slave address register (SVA) matches the data of the shift register

Note COI can be read only before serial transfer is started or after serial transfer is completed. An undefined value may be read during transfer. COI data written by an 8-bit manipulation instruction is ignored.

Wake-up function specification bit (W)

WUP	0	Sets IRQCSI each time serial transfer is completed.

Serial interface operation mode selection bit (W)

CSIM4	CSIM3	CSIM2	Shift register sequence	SO pin function	SI pin function
0	1	1	SIO_{7-0} <—> XA (Transfer starting with MSB)	SB0/P02 (N-ch open-drain I/O)	P03 input
1				P02 input	SB1/P03 (N-ch open-drain I/O)

Serial clock selection bit (W)

CSIM1	CSIM0	Serial clock	$\overline{\text{SCK}}$ pin mode
0	0	External clock applied to $\overline{\text{SCK}}$ pin	Input
0	1	Timer/event counter output (TOUT0)	Output
1	0	$f_{CC}/2^6$ (31.3 kHz: during 2-MHz operation, 15.6 kHz: during 1-MHz operation)	
1	1		

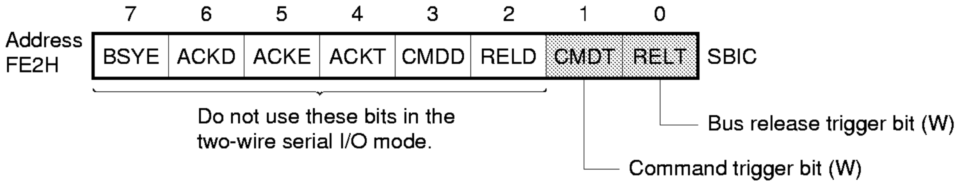
(b) Serial bus interface control register (SBIC)

To use the two-wire serial I/O mode, set SBIC as shown below. (For details on SBIC format, see **(2)** in **Section 5.6.3**.)

SBIC is manipulated using a bit manipulation instruction.

When the $\overline{\text{RESET}}$ signal is input, SBIC is set to 00H.

In the figure below, the hatched portions indicate the bits used in the two-wire serial I/O mode.


7
6
5
4
3
2
1
0
Address
FE2H
BSYE
ACKD
ACKE
ACKT
CMDD
RELD
CMDT
RELT
SBIC
Do not use these bits in the two-wire serial I/O mode.
Bus release trigger bit (W)
Command trigger bit (W)

Remark (W): Write only

Command trigger bit (W)

CMDT	Control bit for command signal (CMD) trigger output. By setting CMDT = 1, the SO latch is cleared to 0. Then the CMDT bit is automatically cleared to 0.

Bus release trigger bit (W)

RELT	Control bit for bus release signal (REL) trigger output. By setting RELT = 1, the SO latch is set to 1. Then the RELT bit automatically cleared to 0.

Caution Never use bits other than RELT and CMDT in the two-wire serial I/O mode.

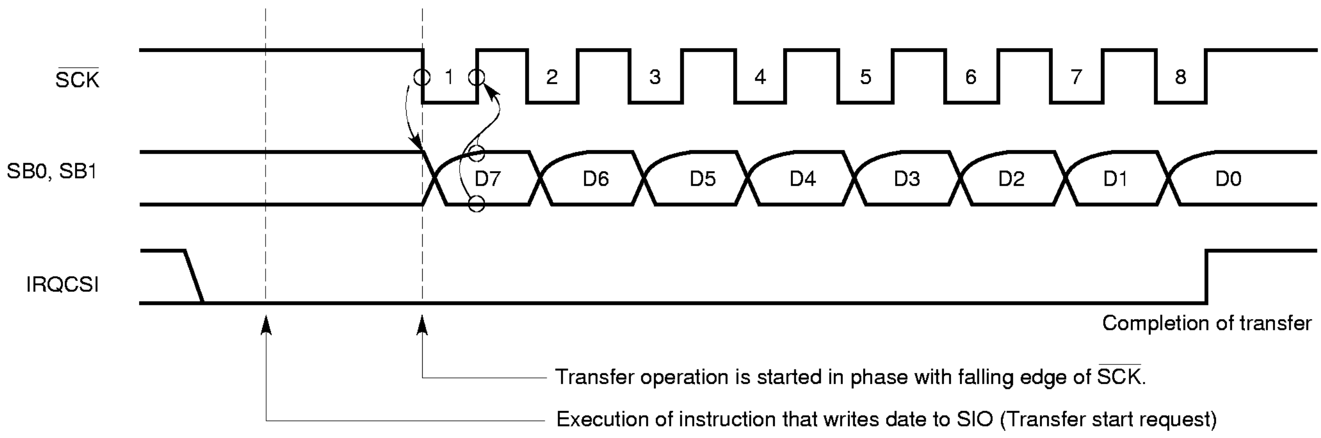
(2) Communication operation

The two-wire serial I/O mode transfers data, with eight bits as one block. Data is transferred bit by bit in phase with the serial clock.

The shift register performs shift operation on the falling edge of the serial clock ($\overline{\text{SCK}}$). Transmit data is latched on the SO latch, and is output on the SB0/P02 pin or SB1/P03 pin starting with the MSB. Receive data applied to the SB0 pin or SB1 pin is latched in the shift register on the rising edge of $\overline{\text{SCK}}$.

When eight bits have been transferred, shift register operation automatically terminates setting the interrupt request flag (IRQCSI).

Figure 5-48. Timing of Two-Wire Serial I/O Mode


SCK
SB0, SB1
IRQCSI
1
2
3
4
5
6
7
8
D7
D6
D5
D4
D3
D2
D1
D0
Completion of transfer
Transfer operation is started in phase with falling edge of SCK.
Execution of instruction that writes date to SIO (Transfer start request)

The SB0 (or SB1) pin becomes an N-ch open-drain I/O when specified as the serial data bus, so the voltage level on that pin must be pulled up externally.

The state of the SO latch is output on the SB0 (or SB1) pin, so the SB0 (or SB1) pin output states can be controlled by setting the RELT or CMDT bit.

However, this operation must not be performed during serial transfer.

The output state of the $\overline{\text{SCK}}$ pin can be controlled by manipulating the P01 output latch in the output mode (internal system clock mode). (See **Section 5.6.8**.)

(3) Serial clock selection

To select the serial clock, manipulate bits 0 and 1 of serial operation mode register (CSIM). The serial clock can be selected out of the following four clocks:

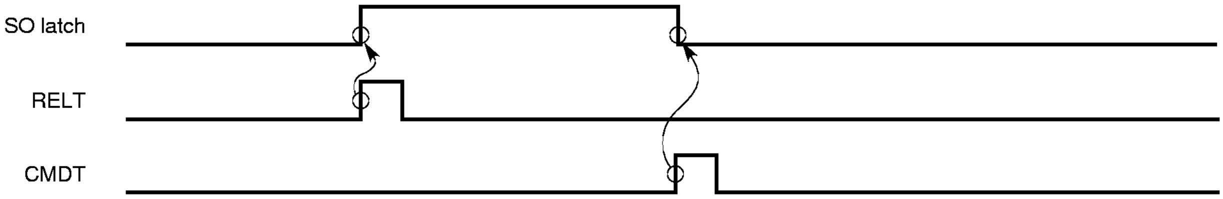
Table 5-8. Serial Clock Selection and Application (In the Two-Wire Serial I/O Mode)

Mode register		Serial clock		Timing for shift register R/W and start of serial transfer	Application
CSIM 1	CSIM 0	Source	Masking of serial clock		
0	0	External $\overline{SCK}$	Automatically masked when 8-bit data transfer is completed	<1> In the operation halt mode (CSIE = 0) <2> When the serial clock is masked after 8-bit transfer <3> When $\overline{SCK}$ is high	Slave CPU
0	1	TOUT flip-flop			Arbitrary-speed serial transfer
1	0	$f_{CC}/2^6$			Low-speed serial transfer
1	1				

(4) Signals

Figure 5-49 shows operations of RELT and CMDT.

Figure 5-49. Operations of RELT and CMDT



(5) Transfer start

Serial transfer starts by writing transfer data into shift register (SIO), provided that the following two conditions are satisfied:

- The serial interface operation enable/disable specification bit (CSIE) is set to 1.
- The internal serial clock is not operating after 8-bit serial transfer, or $\overline{SCK}$ is high.

Cautions 1. Setting CSIE to 1 after writing data to the shift register does not start transfer.

2. When data is received, the N-ch transistor must be turned off, so FFH must be written to SIO beforehand.

When eight bits have been transferred, serial transfer automatically terminates setting the interrupt request flag (IRQCSI).

(6) Error detection

In the two-wire serial I/O mode, the state of serial bus SB0 or SB1 being used for communication is loaded into the shift register (SIO) of the transmitting device. So a transmission error can be detected by the methods described below.

(a) Comparing SIO data before start of transmission with SIO data after start of transmission

With this method, the occurrence of a transmission error is assumed when two SIO values disagree with each other.

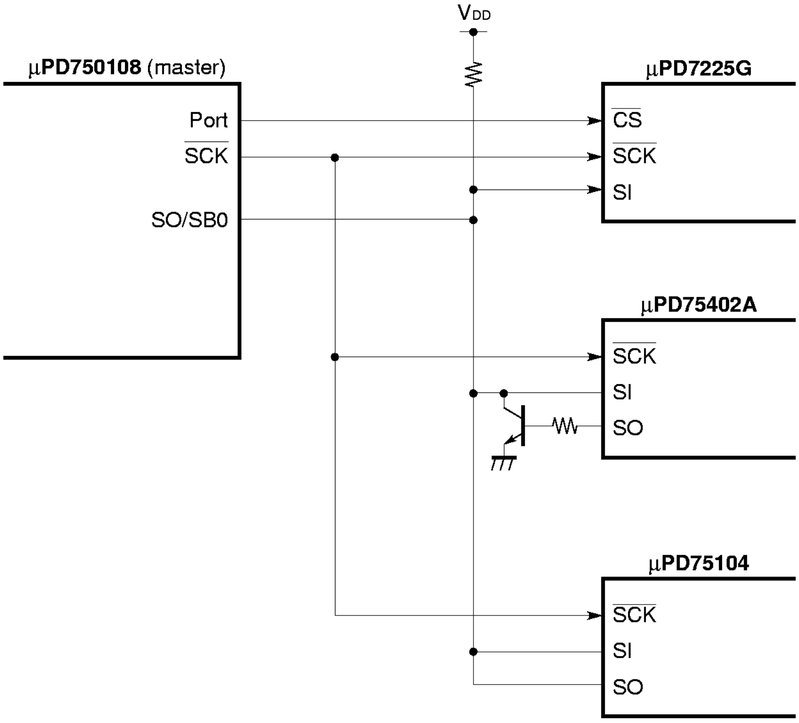
(b) Using the slave address register (SVA)

Transmit data is set in SVA as well before the data is transmitted. On completion of transmission, the COI bit (match signal from the address comparator) of serial operation mode register (CSIM) is tested. If the result is 1, the transmission is regarded as successful. If the result is 0, the occurrence of a transmission error is assumed.

(7) Application of two-wire serial I/O mode

A serial bus is configured, and multiple devices are connected to it.

Example A system is configured with a µPD750108 as the master to which a µPD75104, µPD75402A, and µPD7225G are connected as slaves.



To configure the bus as shown above, connect the SI pin and SO pin. Then, writes FFH to the shift register to make the SO pin high except when serial data is output, and free the bus by setting off the output buffer. The SO pin of the µPD75402A cannot go into a high-impedance state, so that a transistor must be connected as shown in the figure to make open collector output appear on the pin. When data is input, 00H must be set beforehand in the shift register to set the transistor off.

The timing of data output by each microcomputer must be predetermined.

The µPD750108, which is the master microcomputer, outputs a serial clock, and all slave microcomputers operate with an external clock.

5.6.7 SBI Mode Operation

The SBI (serial bus interface) is a high-speed serial interface that conforms to the NEC serial bus format.

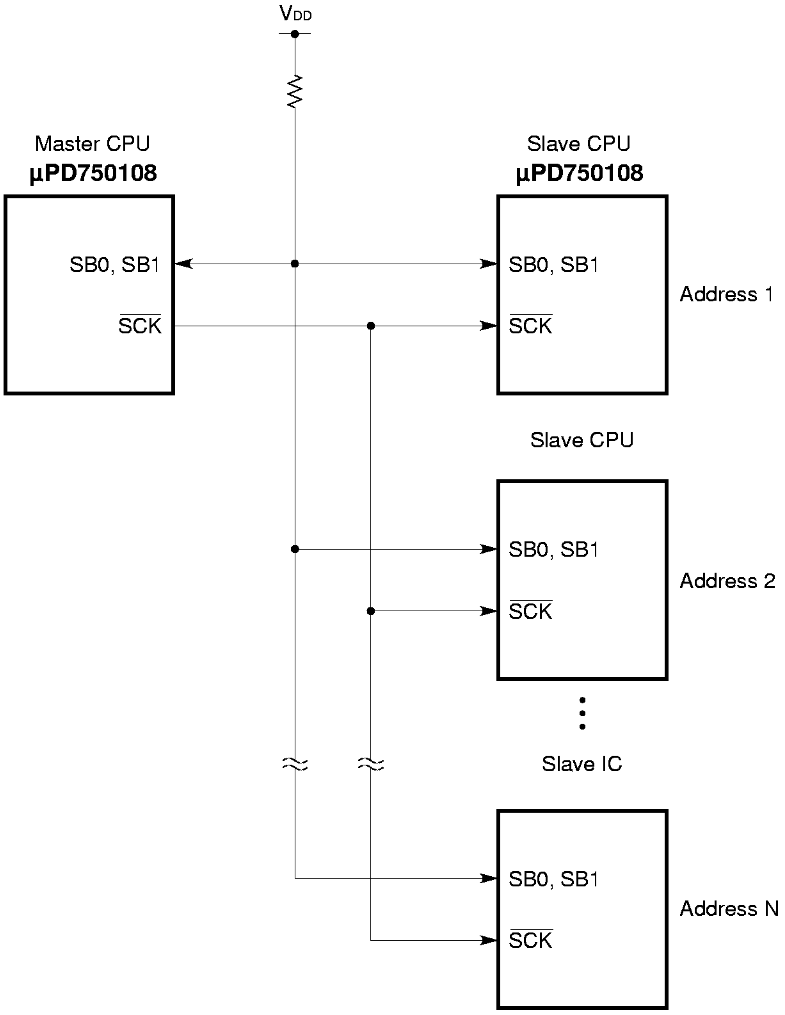
To allow communication with multiple devices on a single-master and high-speed serial bus using two signal lines, the SBI has a bus configuration function added to the clock synchronous serial I/O method. So the SBI can reduce ports and wires on boards when multiple microcomputers and peripheral ICs are used to configure a serial bus.

The master can output, on the serial data bus, an address for selecting a device subject to serial communication, commands directed to the remote device, and data. A slave can identify an address, commands, and data from received data by hardware. This function simplifies the serial interface control portion of an application program.

The SBI function is available with devices such as the 75X series, 75XL series, and 78K series 8-/16-bit single chip microcomputers.

Figure 5-50 is an example of the SBI system configuration when the CPU with a serial interface conforming to SBI or peripheral ICs are used.

Figure 5-50. Example of SBI System Configuration



Cautions 1. In the SBI mode, the serial data bus pin SB0 (or SB1) is an open-drain output. So the serial data bus line is placed in the wired OR state. A pull-up resistor is required for the serial data bus line.

2. To switch between the master and slave, a pull-up resistor is required also for the serial clock line ($\overline{\text{SCK}}$) because $\overline{\text{SCK}}$ input/output switching is performed between the master and slave asynchronously.

(1) SBI functions

Conventional serial I/O methods provide only data transfer functions. Therefore, many ports and wires are required to identify chip select signals, commands, and data, and to detect busy states, when the serial bus is configured with multiple devices. Also, these processes are too burdensome to be controlled by software.

The SBI method can configure a serial bus with two signal lines: Serial clock $\overline{\text{SCK}}$ and serial data bus (SB0 or SB1). For this reason, the number of ports on a microcomputer can be reduced and the wiring on a circuit board can be simplified.

SBI functions are described below.

(a) Address/command/data identification function

Serial data is classified into three types: Address, command, and data.

(b) Address-based chip select function

The master selects a chip for a slave by address transfer.

(c) Wake-up function

A slave can easily check address reception (for chip select identification) with the wake-up function. This function can be set or released by software.

When the wake-up function is set, an interrupt (IRQCSI) is generated when a match address is received. For this reason, in communication with multiple devices, a CPU other than a selected slave can operate independently of serial communication.

(d) Acknowledge signal ($\overline{\text{ACK}}$) control function

The acknowledge signal, which is used to confirm the reception of serial data, can be controlled.

(e) Busy signal ($\overline{\text{BUSY}}$) control function

The busy signal, which is used to post the busy state of a slave, can be controlled.

(2) SBI definition

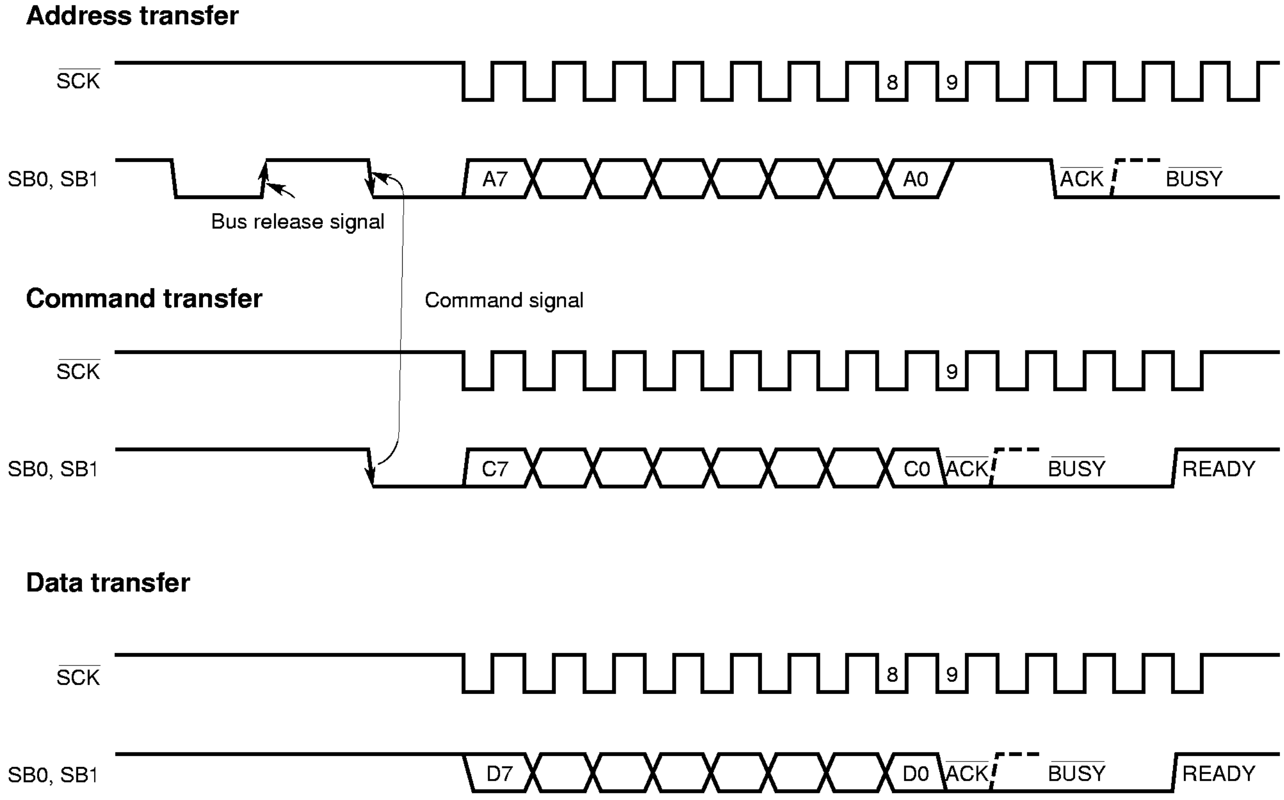
The format of serial data and signal used in the SBI mode are described below.

Serial data to be transferred in the SBI mode is classified into three types: Address, command, and data.

Serial data forms one frame as shown below.

Figure 5-51 is a timing chart for transferring address, command, and data.

Figure 5-51. Timing of SBI Transfer

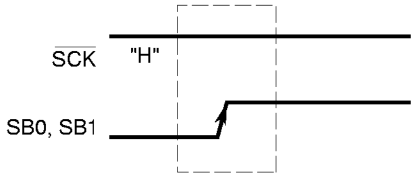


The bus release signal and command signal are output by the master. $\overline{\text{BUSY}}$ is output by a slave. $\overline{\text{ACK}}$ is output by either the master or a slave. (Normally, the device which received 8-bit data outputs $\overline{\text{ACK}}$.) The master continues to output the serial clock from when 8-bit data transfer starts to when $\overline{\text{BUSY}}$ is released.

(a) Bus release signal (REL)

When the $\overline{\text{SCK}}$ line is high (the serial clock is not output), the SB0 (or SB1) line changes from low to high. This signal is called the bus release signal, and is output by the master.

Figure 5-52. Bus Release Signal

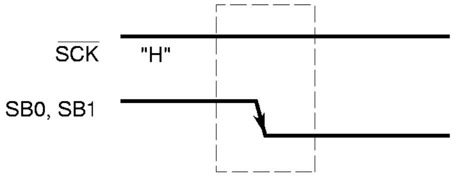


This signal indicates that the master is to send an address to a slave. Slaves contain hardware to detect the bus release signal.

(b) Command signal (CMD)

When the $\overline{\text{SCK}}$ line is high (the serial clock is not output), the SB0 (or SB1) line changes from high to low. This signal is called the command signal, which is output by the master.

Figure 5-53. Command Signal

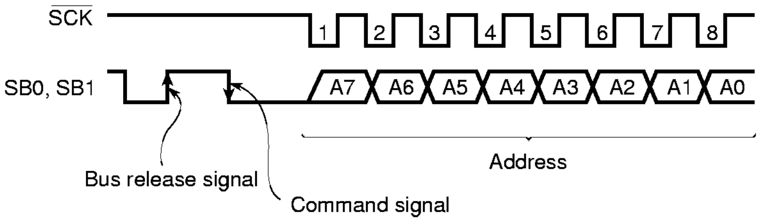


Slaves contain hardware to detect the command signal.

(c) Address

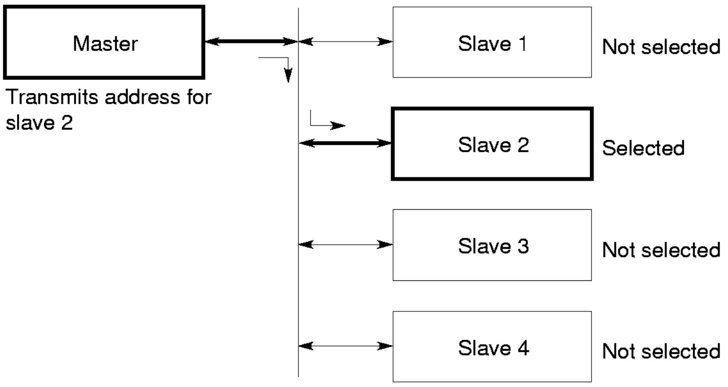
An address is 8-bit data and is output by the master to connected slaves to select a particular slave.

Figure 5-54. Address



The 8-bit data following the bus release signal or command signal is defined as an address. A slave detects the condition for the addresses by hardware, and checks whether the 8-bit data matches the number assigned to the slave (slave address). If the 8-bit data matches the slave address, that slave is selected. The selected slave continues to communicate with the master until disconnection is directed by the master.

Figure 5-55. Slave Selection Using an Address



(d) Command and data

The master sends commands to the slave selected by sending an address. The master also transfers data to or from the slave.

Figure 5-56. Command

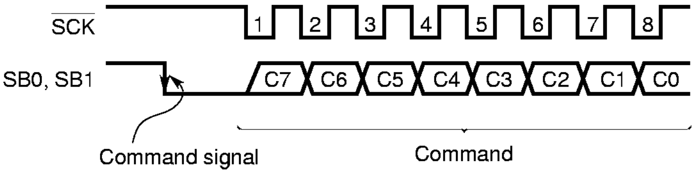
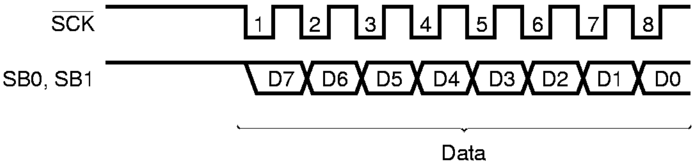


Figure 5-57. Data

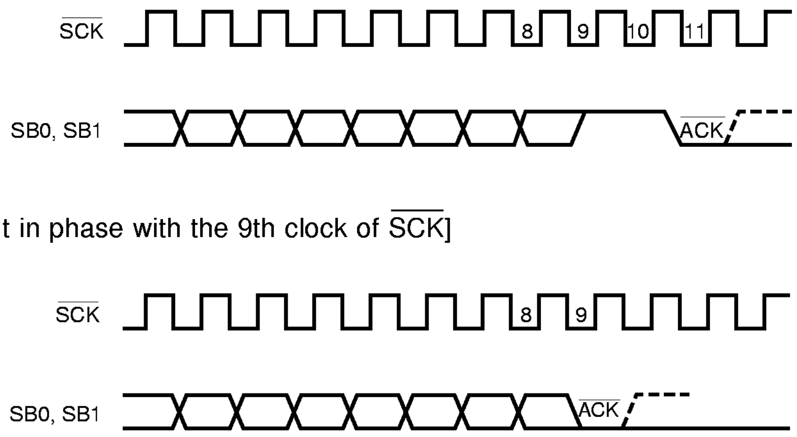


The 8-bit data following the command signal is defined as a command. The 8-bit data without the command signal is defined as data. The usage of commands or data can be selected optionally according to the communication specifications.

(e) Acknowledge signal ($\overline{\text{ACK}}$)

The acknowledge signal confirms the reception of serial data between the transmitter and the receiver.

Figure 5-58. Acknowledge Signal



[When output in phase with the 11th clock of $\overline{\text{SCK}}$]

[When output in phase with the 9th clock of $\overline{\text{SCK}}$]

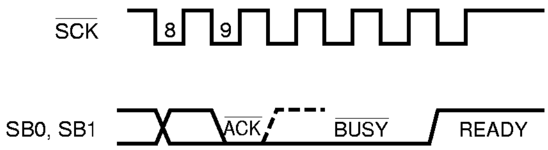
The acknowledge signal is a one-shot pulse output in phase with the falling edge of $\overline{\text{SCK}}$ after 8-bit data transfer. This signal may be synchronized with any clock of $\overline{\text{SCK}}$.

The transmitter checks if the receiver returns the acknowledge signal after 8-bit data transfer. If the acknowledge signal is not returned after a specified period of time, the transmitter can assume that the reception failed.

(f) Busy signal ($\overline{\text{BUSY}}$) and ready signal (READY)

The busy signal informs the master that a slave is getting ready for data transfer.
The ready signal informs the master that a slave is ready for data transfer.

Figure 5-59. Busy and Ready Signals



In the SBI mode, a slave notifies the master of the busy state by changing SB0 (or SB1) from high to low.
The busy signal is output following the acknowledge signal output by the master or a slave. The busy signal is set and released in phase with the falling edge of $\overline{\text{SCK}}$. The master automatically terminates output of serial clock $\overline{\text{SCK}}$ when the busy signal is released.
The master can transfer the next data when the busy signal is released and a slave enters the state in which the ready signal is to be output.

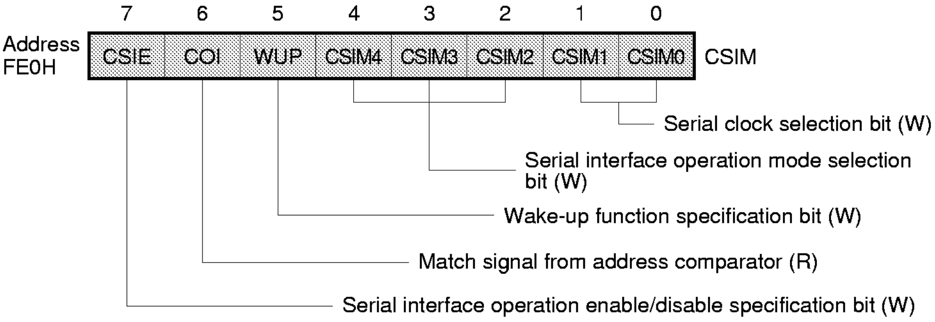
(3) Register setting

To set the SBI mode, manipulate the following two registers:

- Serial operation mode register (CSIM)
- Serial bus interface control register (SBIC)

(a) Serial operation mode register (CSIM)

To use the SBI mode, set CSIM as shown below. (For details on CSIM format, see **(1)** in **Section 5.6.3**.)
CSIM is manipulated using an 8-bit manipulation instruction. Bits 7, 6, and 5 of CSIM can be manipulated bit by bit.
When the $\overline{\text{RESET}}$ signal is input, CSIM is set to 00H.
In the figure below, hatched portions indicate the bits used in the SBI mode.


7
6
5
4
3
2
1
0
Address FE0H
CSIE
COI
WUP
CSIM4
CSIM3
CSIM2
CSIM1
CSIM0
CSIM
Serial clock selection bit (W)
Serial interface operation mode selection bit (W)
Wake-up function specification bit (W)
Match signal from address comparator (R)
Serial interface operation enable/disable specification bit (W)

Remark (R): Read only
(W): Write only

Serial interface operation enable/disable specification bit (W)

		Shift register operation	Serial clock counter	IRQCSI flag	SO/SB0 and SI/SB1 pins
CSIE	1	Shift operation enabled	Count operation	Can be set	Used in each mode as well as for port 0

Signal from address comparator (R)

COI[Note]	Condition for being cleared (COI = 0)	Condition for being set (COI = 1)
	When the slave address register (SVA) does not match the data of the shift register	When the slave address register (SVA) matches the data of the shift register

Note COI can be read only before serial transfer is started or after serial transfer is completed. An undefined value may be read during transfer.
COI data written by an 8-bit manipulation instruction is ignored.

Wake-up function specification bit (W)

WUP	0	Sets IRQCSI each time serial transfer is completed in each mode.
	1	Used in the SBI mode only to set IRQCSI only when an address received after bus release matches the data in the slave address register (wake-up state). SB0 or SB1 goes to high-impedance state.

Caution When WUP = 1 is set during $\overline{\text{BUSY}}$ signal output, $\overline{\text{BUSY}}$ is not released. In the SBI mode, the $\overline{\text{BUSY}}$ signal is output until the next falling edge of the serial clock ($\overline{\text{SCK}}$) appears after release of $\overline{\text{BUSY}}$ is directed. Before setting WUP = 1, be sure to confirm that the SB0 (or SB1) pin is high after releasing $\overline{\text{BUSY}}$.

Serial interface operation mode selection bit (W)

CSIM4	CSIM3	CSIM2	Shift register sequence	SO pin function	SI pin function
0	1	0	$\text{SIO}_{7\text{-}0}$ <—> XA (Transfer starting with MSB)	SB0/P02 (N-ch open-drain I/O)	P03 input
1				P02 input	SB1/P03 (N-ch open-drain I/O)

Serial clock selection bit (W)

CSIM1	CSIM0	Serial clock	$\overline{\text{SCK}}$ pin mode
0	0	External clock applied to $\overline{\text{SCK}}$ pin	Input
0	1	Timer/event counter output (TOUT0)	Output
1	0	$f_{CC}/2^4$ (125 kHz: during 2-MHz operation, 62.5 kHz: during 1-MHz operation)	
1	1	$f_{CC}/2^3$ (250 kHz: during 2-MHz operation, 125 kHz: during 1-MHz operation)	

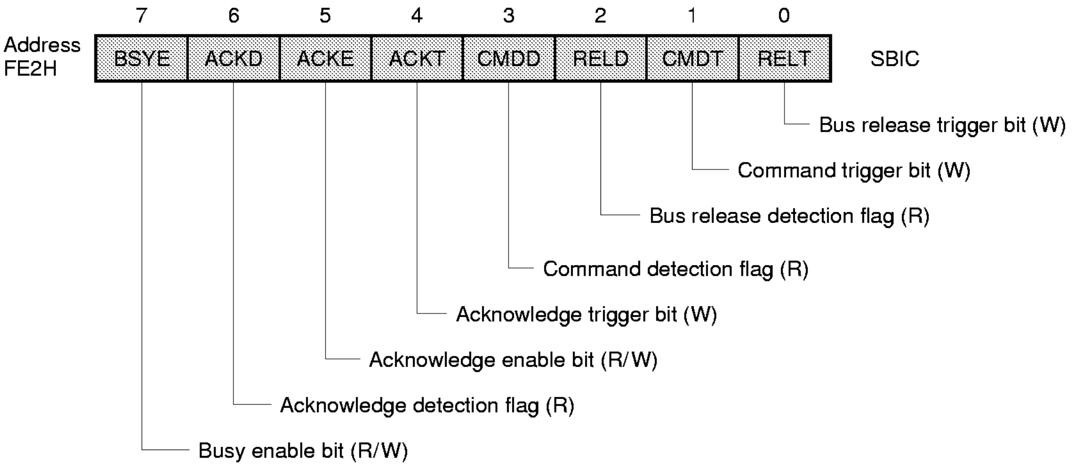
(b) Serial bus interface control register (SBIC)

To use the SBI mode, set SBIC as shown below. (For details on SBIC format, see **(2)** in **Section 5.6.3**.)

SBIC is manipulated using a bit manipulation instruction.

When the $\overline{\text{RESET}}$ signal is input, SBIC is set to 00H.

In the figure below, hatched portions indicate the bits used in the SBI mode.


7
6
5
4
3
2
1
0
Address
FE2H
BSYE
ACKD
ACKE
ACKT
CMDD
RELD
CMDT
RELT
SBIC
Bus release trigger bit (W)
Command trigger bit (W)
Bus release detection flag (R)
Command detection flag (R)
Acknowledge trigger bit (W)
Acknowledge enable bit (R/W)
Acknowledge detection flag (R)
Busy enable bit (R/W)

Remark (R): Read only
(W): Write only
(R/W): Read/write

Busy enable bit (R/W)

BSYE	0	<1> The busy signal is automatically disabled. <2> Busy signal output is stopped in phase with the falling edge of $\overline{\text{SCK}}$ immediately after clear instruction execution.
	1	The busy signal is output after the acknowledge signal in phase with the falling edge of $\overline{\text{SCK}}$.

Acknowledge detection flag (R)

ACKD	Condition for being cleared (ACKD = 0)	Condition for being set (ACKD = 1)
	<1> The transfer operation is started. <2> The $\overline{\text{RESET}}$ signal is entered.	The acknowledge signal ($\overline{\text{ACK}}$) is detected (in phase with the rising edge of $\overline{\text{SCK}}$).

Acknowledge enable bit (R/W)

ACKE	0	Disables automatic output of the acknowledge signal. (Output by ACKT is possible.)	
	1	When set before transfer	$\overline{\text{ACK}}$ is output in phase with the 9th clock of $\overline{\text{SCK}}$.
		When set after transfer	$\overline{\text{ACK}}$ is output in phase with $\overline{\text{SCK}}$ immediately following the set instruction execution.

Acknowledge trigger bit (W)

ACKT	When set after transfer, $\overline{\text{ACK}}$ is output in phase with the next $\overline{\text{SCK}}$. After $\overline{\text{ACK}}$ signal output, this bit is automatically cleared to 0.

Cautions 1. Never set ACKT to 1 before or during serial transfer.

2. ACKT cannot be cleared by software.

3. Before setting ACKT, set ACKE = 0.

Command detection flag (R)

CMDD	Condition for being cleared (CMDD = 0)	Condition for being set (CMDD = 1)
	<1> The transfer start instruction is executed. <2> The bus release signal (REL) <3> The $\overline{\text{RESET}}$ signal is entered. <4> CSIE = 0 (**Figure 5-40**)	The command signal (CMD) is detected.

Bus release detection flag (R)

RELD	Condition for being cleared (RELD = 0)	Condition for being set (RELD = 1)
	<1> The transfer start instruction is executed. <2> The $\overline{\text{RESET}}$ signal is entered. <3> CSIE = 0 (**Figure 5-40**) <4> SVA does not match SIO when an address is received.	The bus release signal (REL) is detected.

Command trigger bit (W)

CMDT	Control bit for command signal (CMD) trigger output. By setting CMDT = 1, the SO latch is cleared to 0. Then the CMDT bit is automatically cleared to 0.

Caution Never set SB0 (or SB1) during serial transfer. Be sure to set SB0 (or SB1) before or after serial transfer.

Bus release trigger bit (W)

RELT	Control bit for bus release signal (REL) trigger output. By setting RELT = 1, the SO latch is set to 1. Then the RELT bit automatically cleared to 0.

Caution Never set SB0 (or SB1) during serial transfer. Be sure to set SB0 (or SB1) before or after serial transfer.

(4) Serial clock selection

To select the serial clock, manipulate bits 0 and 1 of serial operation mode register (CSIM). The serial clock can be selected out of the following four clocks:

Table 5-9. Serial Clock Selection and Application (In the SBI Mode)

Mode register		Serial clock		Timing for shift register R/W and start of serial transfer	Application
CSIM 1	CSIM 0	Source	Masking of serial clock		
0	0	External $\overline{SCK}$	Automatically masked when 8-bit data transfer is completed	<1> In the operation halt mode (CSIE = 0) <2> When the serial clock is masked after 8-bit transfer <3> When $\overline{SCK}$ is high	Slave CPU
0	1	TOUT flip-flop			Arbitrary-speed serial transfer
1	0	$f_{CC}/2^4$			Middle-speed serial transfer
1	1	$f_{CC}/2^3$			High-speed serial transfer

When the internal system clock is selected, $\overline{SCK}$ is internally terminated when the 8th clock has been output, and is externally counted until the slave enters the ready state.

(5) Signals

Figures 5-60 to 5-65 show signals to be generated in the SBI mode and flag operations on the SBIC. Table 5-10 lists signals used in the SBI mode.

Figure 5-60. Operations of RELT, CMDT, RELD, and CMDD (Master)

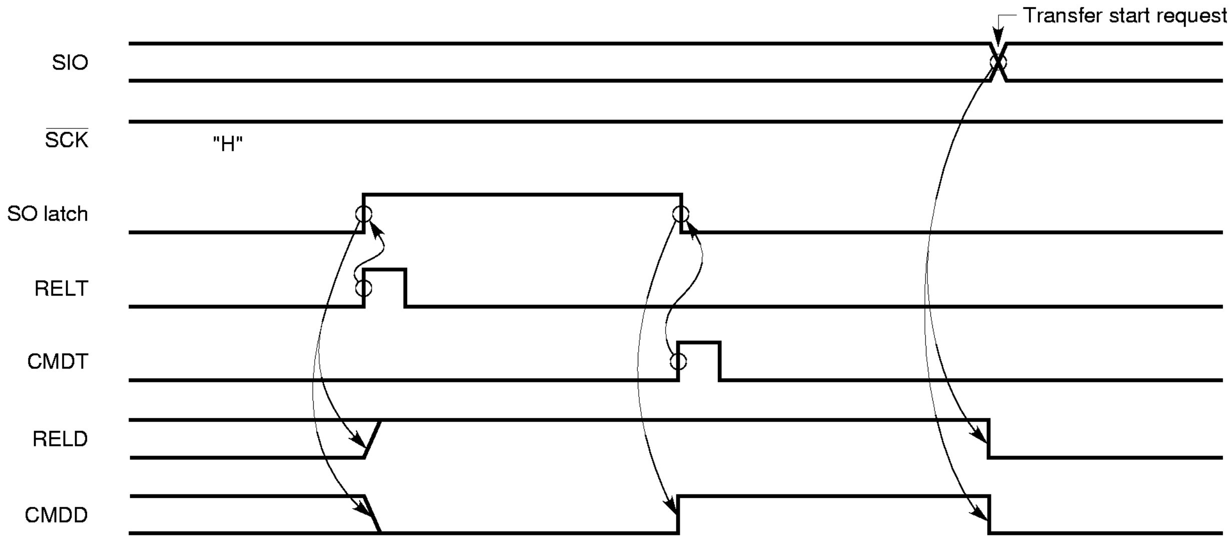


Figure 5-61. Operations of RELT, CMDT, RELD, and CMDD (Slave)

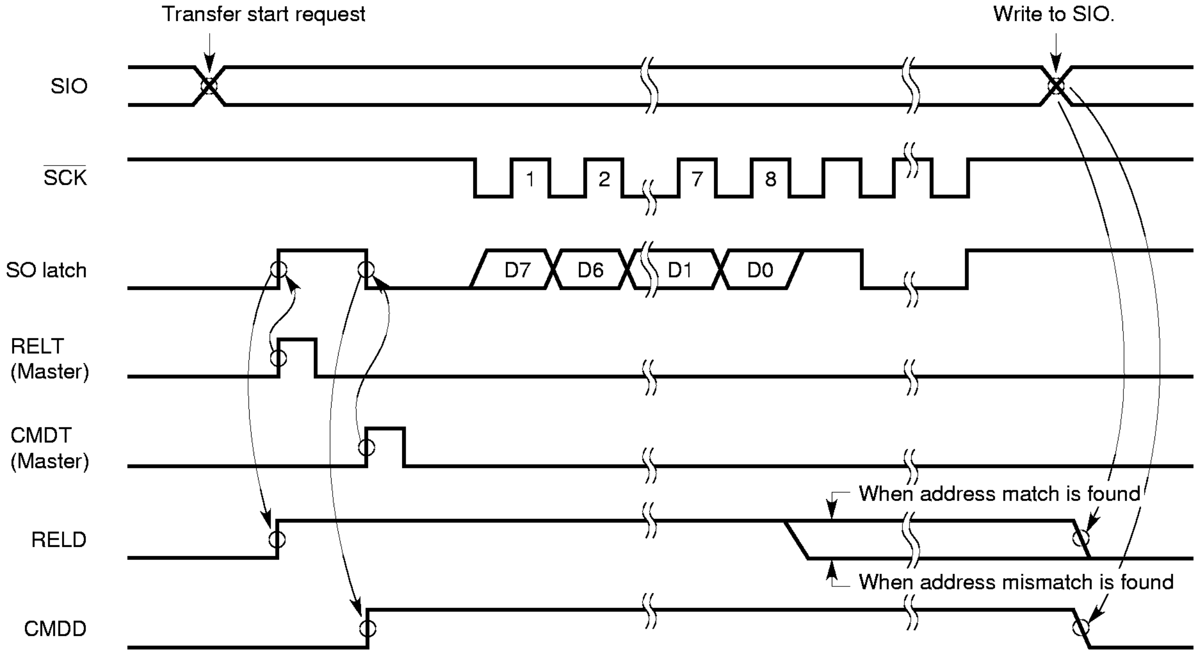
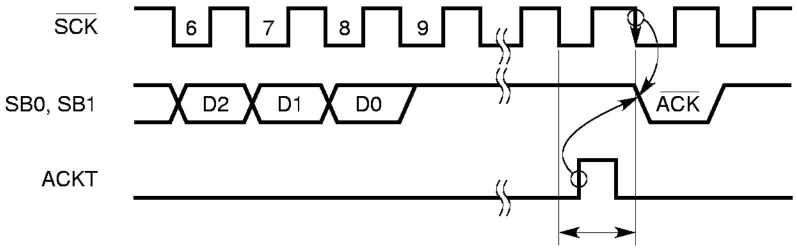


Figure 5-62. Operation of ACKT

When ACKT is set after transfer completion

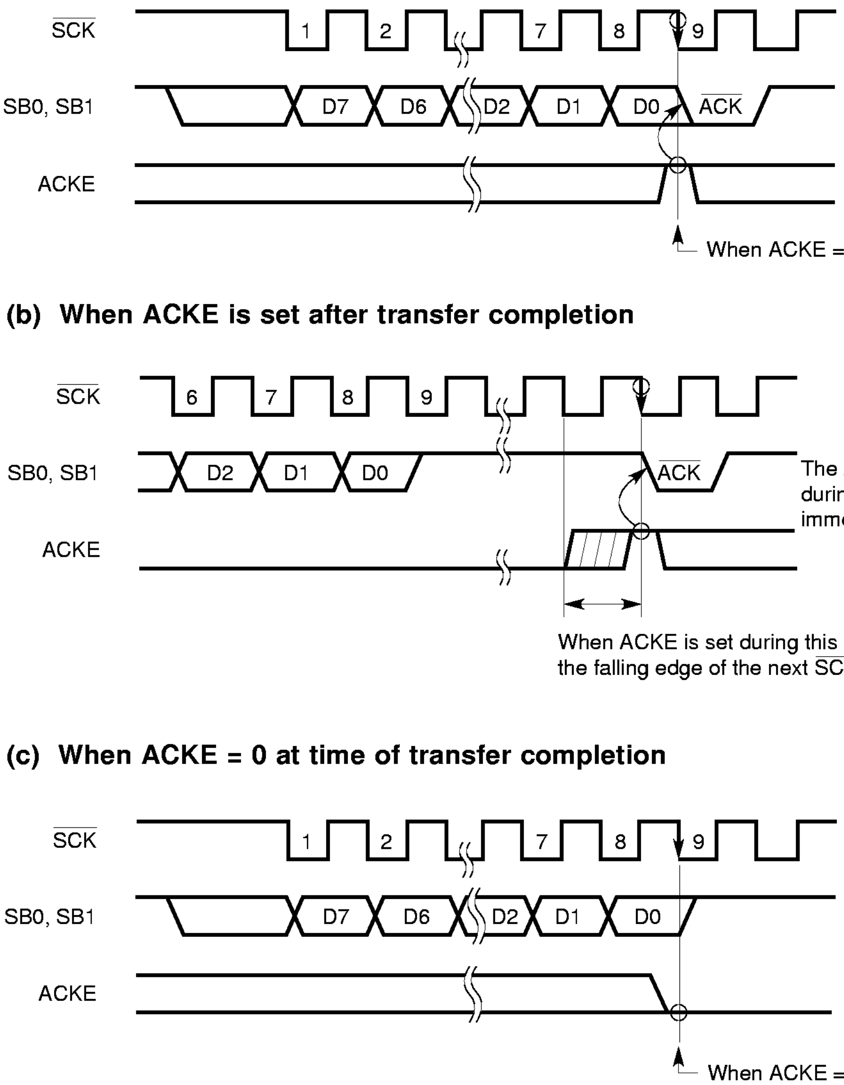


$\overline{\text{ACK}}$ signal is output during the first clock cycle immediately after ACKT is set.

When set during this period

Caution Do not set the ACKT until the transfer is completed.

Figure 5-63. Operation of ACKE (1/2)



(a) When ACKE = 1 at time of transfer completion

The $\overline{\text{ACK}}$ signal is output during the ninth clock cycle.

When ACKE = 1 at this point

(b) When ACKE is set after transfer completion

The $\overline{\text{ACK}}$ signal is output during the first clock cycle immediately after ACKE is set.

When ACKE is set during this period and ACKE = 1 at the falling edge of the next $\overline{\text{SCK}}$

(c) When ACKE = 0 at time of transfer completion

The $\overline{\text{ACK}}$ signal is not output.

When ACKE = 0 at this point

Figure 5-63. Operation of ACKE (2/2)

(d) When ACKE = 1 period is too short

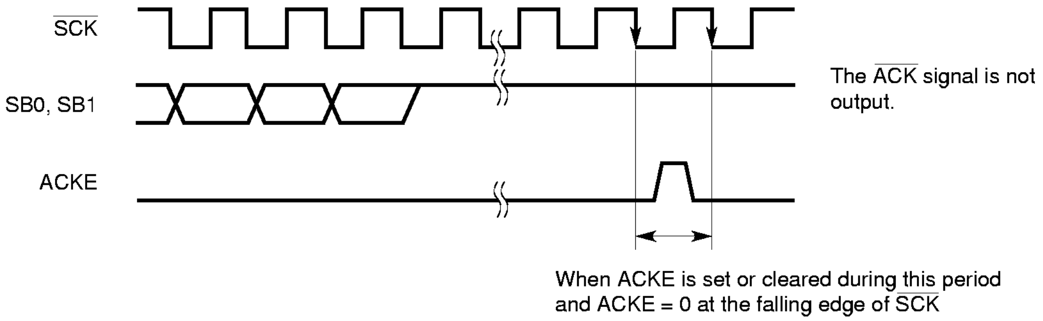
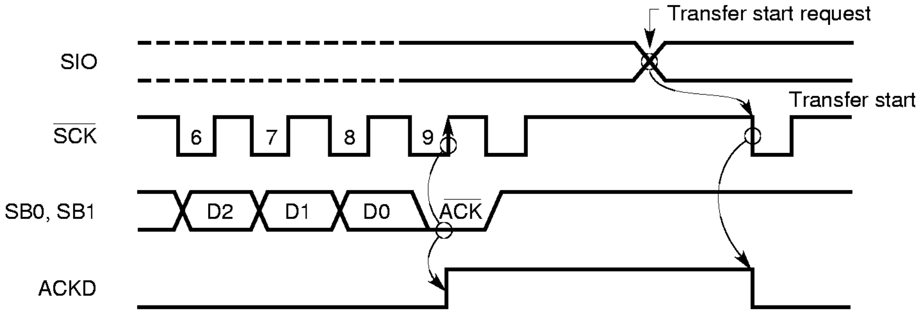

SCK
SB0, SB1
ACKE
The ACK signal is not output.
When ACKE is set or cleared during this period and ACKE = 0 at the falling edge of SCK

Figure 5-64. Operation of ACKD (1/2)

(a) When ACK signal is output during the ninth SCK clock



(b) When ACK signal is output after the ninth SCK clock

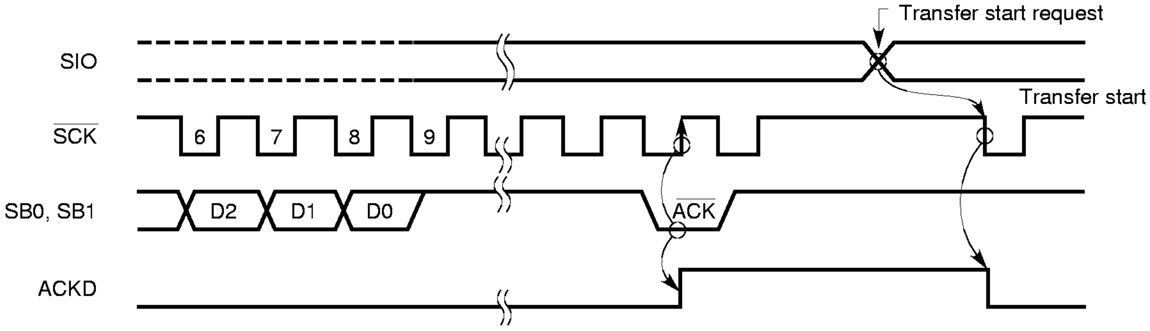


Figure 5-64. Operation of ACKD (2/2)

(c) Clear timing for case where start of transfer is directed during $\overline{\text{BUSY}}$

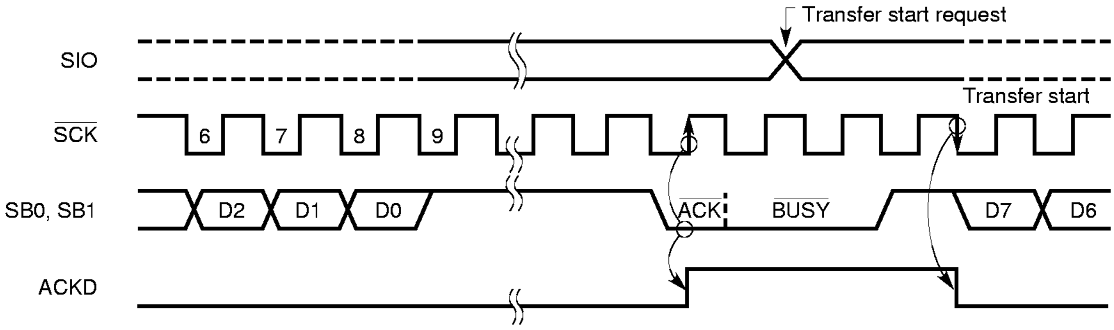


Figure 5-65. Operation of BSYE

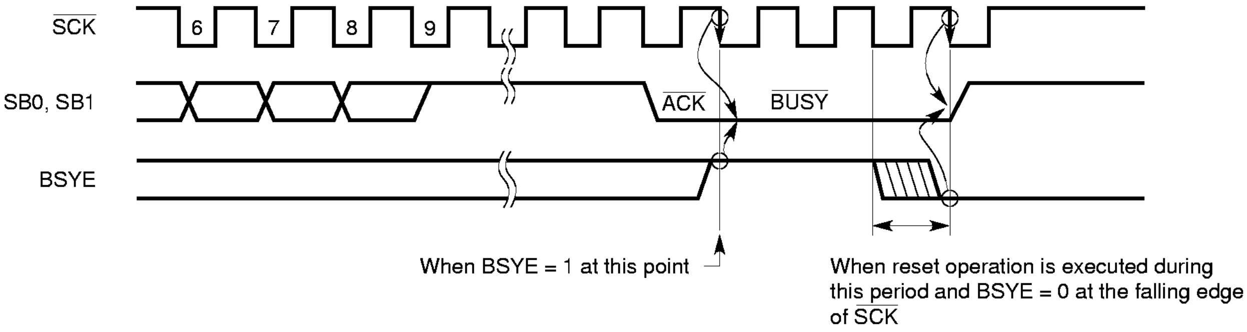

SCK
6
7
8
9
SB0, SB1
ACK
BUSY
BSYE
When BSYE = 1 at this point
When reset operation is executed during this period and BSYE = 0 at the falling edge of SCK

Table 5-10. Various Signals Used in the SBI Mode (1/2)

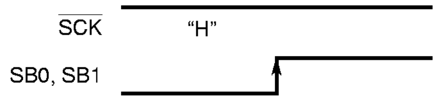
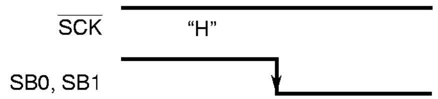
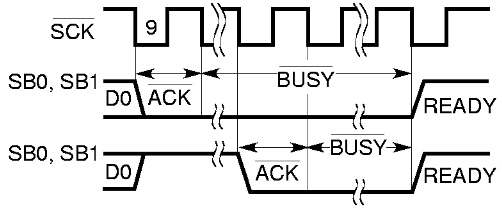
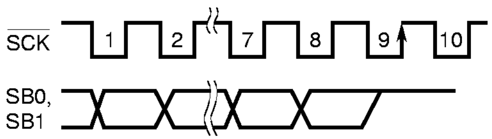
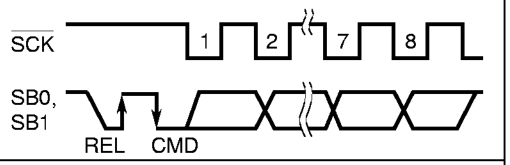
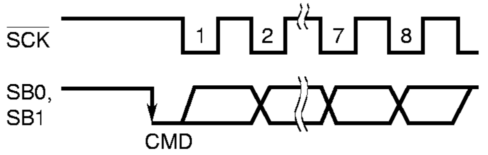
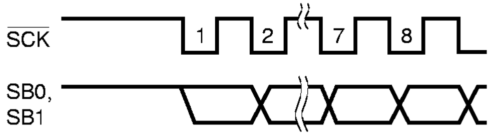
Signal name	Output device	Definition	Timing chart	Condition for output	Flag operation	Meaning of signal
Bus release signal (REL)	Master	Rising edge of SB0 or SB1 when $\overline{SCK}$ = 1		• RELT is set.	• RELD is set. • CMDD is cleared.	Indicates that CMD signal follows and data transmitted is address data.
Command signal (CMD)	Master	Falling edge of SB0 or SB1 when $\overline{SCK}$ = 1		• CMDT is set.	• CMDD is set.	i) Data transmitted after REL signal output is address. ii) Data transmitted, with REL signal not being output, is command.
Acknowledge signal ($\overline{ACK}$)	Master/ slave	Low level signal output on SB0 or SB1 during one $\overline{SCK}$ clock cycle after serial reception is completed	[Synchronous $\overline{BUSY}$ output]  $\overline{SCK}$ 9 SB0, SB1 D0 $\overline{ACK}$ $\overline{BUSY}$ READY SB0, SB1 D0 $\overline{ACK}$ $\overline{BUSY}$ READY	<1> ACKE = 1 <2> ACKT is set.	• ACKD is set.	Indicates completion of reception.
Busy signal ($\overline{BUSY}$)	Slave	[Synchronous $\overline{BUSY}$ signal] Low level signal output on SB0 or SB1 after acknowledge signal		• BSYE = 1	–	Indicates that serial reception is disabled because processing is in progress.
Ready signal (READY)	Slave	High level signal output on SB0 or SB1 before serial transfer is started or after serial transfer is completed		<1> BSYE = 0 <2> Execution of instruction to write data to SIO (Transfer start request)	–	Indicates that serial reception is enabled.

Table 5-10. Various Signals Used in the SBI Mode (2/2)

Signal name	Output device	Definition	Timing chart	Condition for output	Flag operation	Meaning of signal
Serial clock ($\overline{\text{SCK}}$)	Master	Synchronous clock for outputting address/ command/data, $\overline{\text{ACK}}$ signal, synchronous $\overline{\text{BUSY}}$ signal, and so on. Address/command/data is output during first 8 clock cycles.	 $\overline{\text{SCK}}$ 1 2 7 8 9 10 SB0, SB1	Execution of instruction to write data to SIO when CSIE = 1 (Serial transfer start request)[Note 2]	IRQCSI is set (on rising edge of 9th clock of $\overline{\text{SCK}}$)[Note 1]	Timing of signal output on serial data bus
Address (A7 - A0)	Master	8-bit data transferred in phase with $\overline{\text{SCK}}$ after REL signal and CMD signal output	 $\overline{\text{SCK}}$ 1 2 7 8 SB0, SB1 REL CMD			Address of slave device on serial bus
Command (C7 - C0)	Master	8-bit data transferred in phase with $\overline{\text{SCK}}$ after only CMD signal is output, with REL signal not being output	 $\overline{\text{SCK}}$ 1 2 7 8 SB0, SB1 CMD			Directions and messages to slave device
Data (D7 - D0)	Master/ slave	8-bit data transferred in phase with $\overline{\text{SCK}}$, with neither REL signal nor CMD signal being output				Numeric processed by slave or master device

Notes 1. When WUP = 0, IRQCSI is always set on the ninth rising edge of the $\overline{\text{SCK}}$ signal.
When WUP = 1, IRQCSI is set on the ninth rising edge of $\overline{\text{SCK}}$ only when the received address matches the value held in the slave address register (SVA).

2. In the $\overline{\text{BUSY}}$ state, data transfer is initiated after the READY state is set.

(6) Pin configuration

The configurations of serial clock pin $\overline{\text{SCK}}$ and serial data bus pin (SB0 or SB1) are as follows:

(a) $\overline{\text{SCK}}$: Pin for serial clock I/O

<1> Master : CMOS, push-pull output

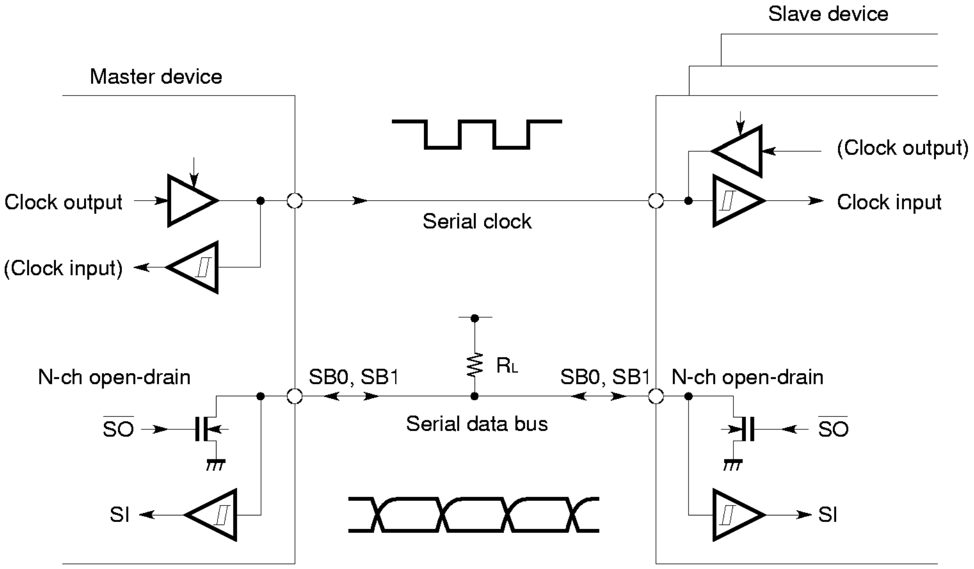
<2> Slave : Schmitt input

(b) SB0, SB1: Pin for serial data I/O

Output to SB0 or SB1 is an N-ch open-drain output and input is Schmitt input for both the master and a slave.

The serial data bus line must be externally pulled up because it has originally an N-ch open-drain output.

Figure 5-66. Pin Configuration



Caution When data is received, the N-ch transistor must be turned off, so FFH must be written to SIO beforehand. The N-ch open-drain output can be turned off at any time during transfer. However, when the wake-up function specification bit (WUP) is set to 1, the N-ch transistor is always off, so there is no need to write FFH to SIO before reception.

(7) Address match detection method

In the SBI mode, communication starts when the master selects a particular slave device by outputting an address.

An address match is detected by hardware. The slave address register (SVA) is available. In the wake-up state (WUP = 1), IRQCSI is set only when the address transmitted by the master and the value held in SVA match.

Cautions 1. Whether a slave is selected is determined by detecting a match for a slave address received after bus release (in the state of RELD = 1).
An address match is detected usually using an address match interrupt (IRQCSI) generated when WUP is set to 1. So detect selection/nonselection state by slave address when WUP is set to 1.

2. When determining whether a slave is selected without using an interrupt when WUP is 0, do not use the address match detection method. Instead, use transfer of commands set in advance in a program.

(8) Error detection

In the SBI mode, the state of serial bus SB0 (or SB1) being used for communication is loaded into the shift register (SIO) of the transmitting device. So a transmission error can be detected by the methods described below.

(a) Comparing SIO data before start of transmission with SIO data after start of transmission

With this method, the occurrence of a transmission error is assumed if two SIO values disagree with each other.

(b) Using the slave address register (SVA)

Transmit data is set in SIO and SVA as well before the data is transmitted. On completion of transmission, the COI bit (match signal from the address comparator) of serial operation mode register (CSIM) is tested. If the result is 1, the transmission is regarded as successful. If the result is 0, the occurrence of a transmission error is assumed.

(9) Communication operation

In the SBI mode, the master usually selects a slave device to communicate with from multiple devices by outputting the address of the slave to the serial bus.

After selecting a device to communicate with, the master exchanges commands and data with the slave device, thus establishing serial communication.

Figures 5-67 to 5-70 show the timing charts of data communication operations.

In the SBI mode, the shift register performs shift operation on the falling edge of the serial clock ($\overline{\text{SCK}}$). Transmit data is held on the SO latch, and is output on the SB0/P02 or SB1/P03 pin starting with the MSB. Receive data applied to the SB0 (or SB1) pin is latched in the shift register on the rising edge of $\overline{\text{SCK}}$.

Figure 5-67. Address Transfer Operation from Master Device to Slave Device (WUP = 1)

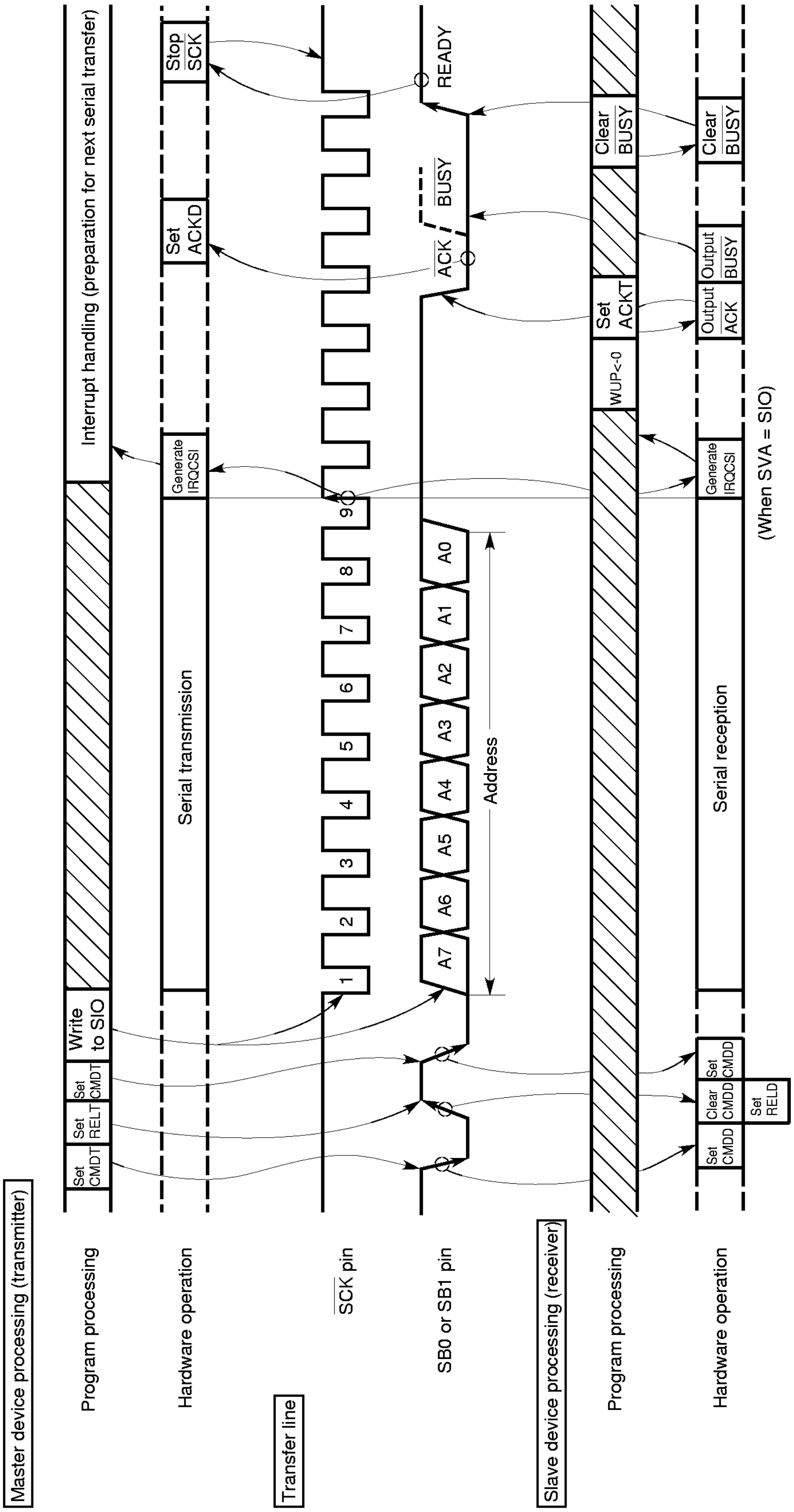


Figure 5-68. Command Transfer Operation from Master Device to Slave Device

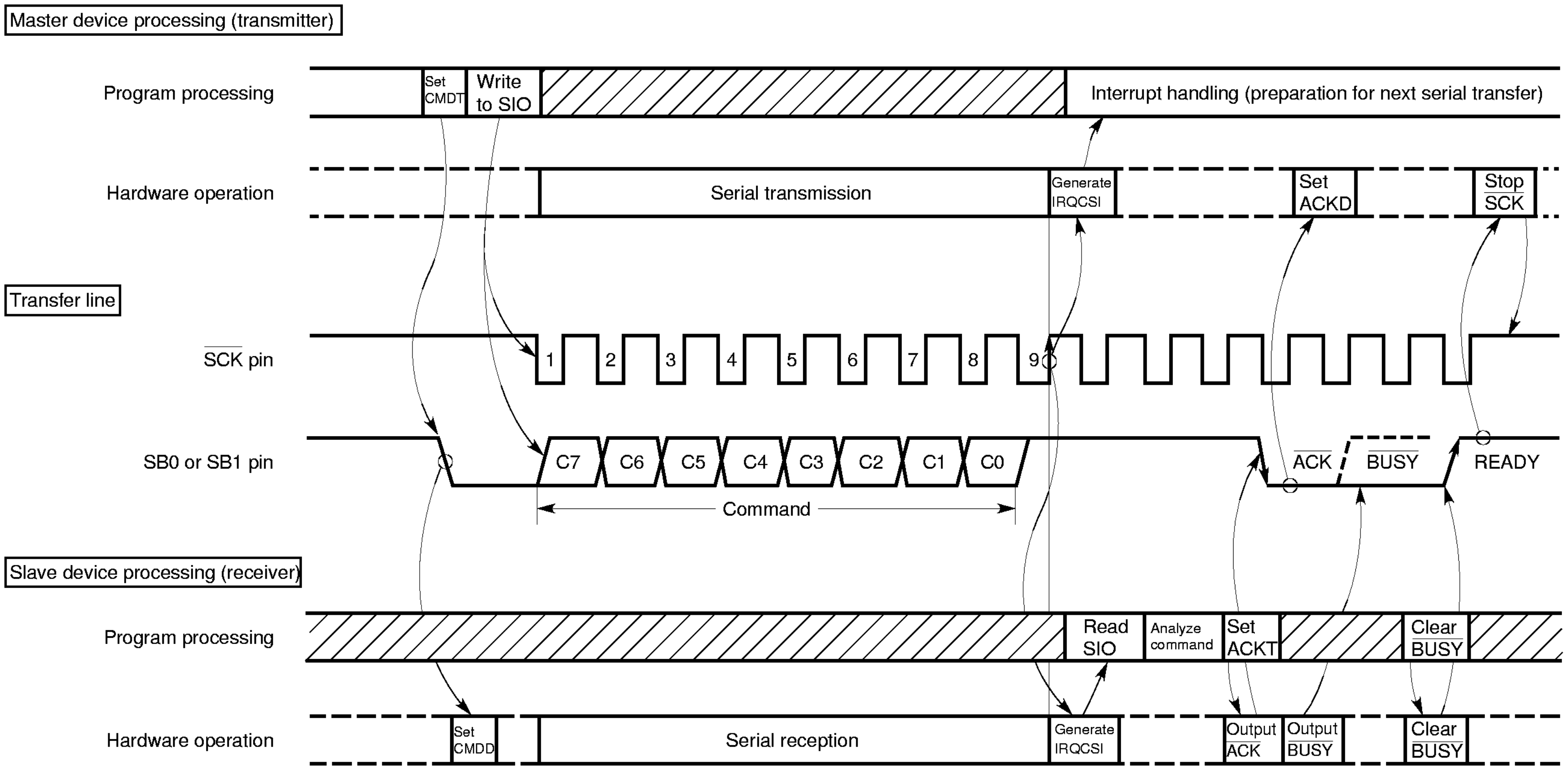


Figure 5-69. Data Transfer Operation from Master Device to Slave Device

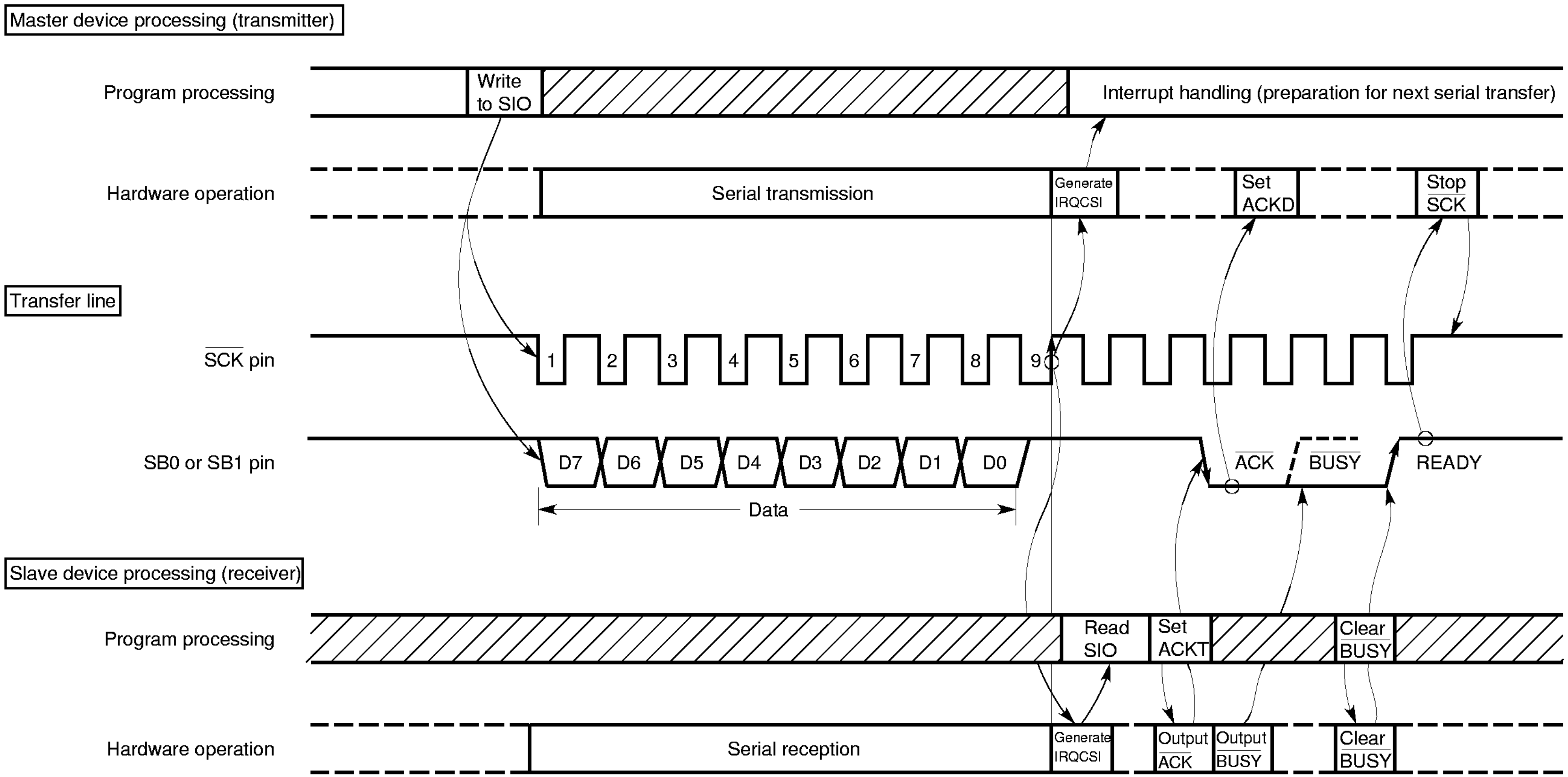
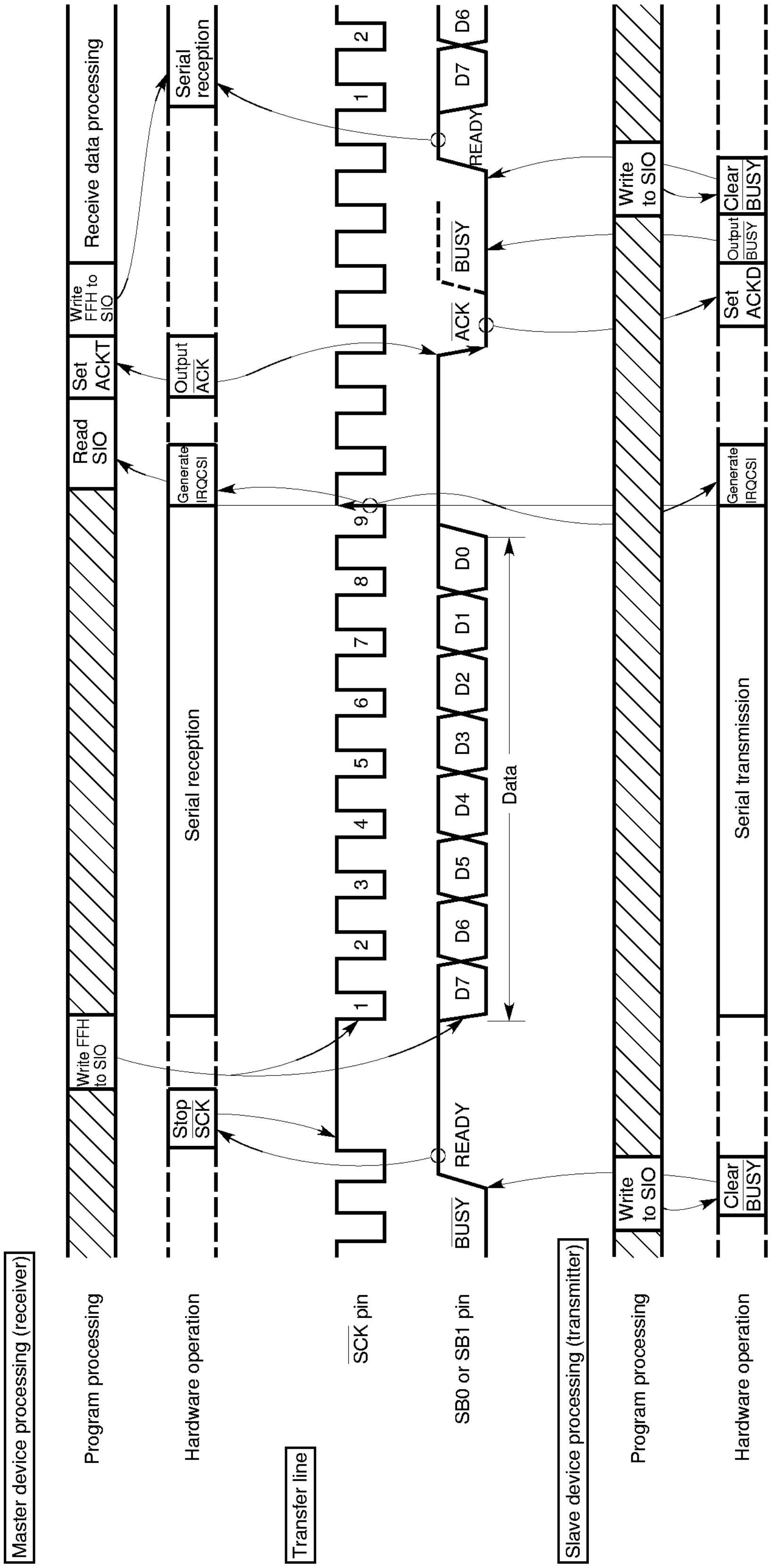


Figure 5-70. Data Transfer Operation from Slave Device to Master Device



(10) Transfer start

Serial transfer is started by writing transfer data in shift register (SIO), provided that the following two conditions are satisfied:

- The serial interface operation enable/disable bit (CSIE) is set to 1.
- The internal serial clock is not operating after 8-bit serial transfer, or $\overline{\text{SCK}}$ is high.

Cautions 1. Transfer cannot be started by setting CSIE to 1 after writing data to the shift register.

2. The N-ch transistor needs to be turned off when data is received. So FFH must be written to SIO beforehand. However, when the wake-up function specification bit (WUP) is set to 1, the N-ch transistor is always off. So FFH need not be written to SIO beforehand for reception.

3. If data is written to SIO when the slave is busy, the data is not lost. Transfer is started when the busy state is released and input to SB0 (or SB1) goes high.

When eight bits have been transferred, serial transfer automatically terminates setting the interrupt request flag (IRQCSI).

Example When RAM data specified by the HL register is transferred to SIO, from which data is loaded into the accumulator at the same time, and serial transfer is started.

```
MOV  XA,@HL  ; Extracts transmit data from RAM
SEL  MB15    ; or CLR1  MBE
XCH  XA,SIO  ; Exchanges transmit data with receive data and starts transfer
```

(11) Notes on the SBI mode

(a) Whether a slave is selected is determined by detecting a match for a slave address received after bus release (in the state of RELD = 1).
An address match is detected usually using, an address match interrupt (IRQCSI) generated when WUP is 1. So detect selection/nonselection state by slave address when WUP is set to 1.

(b) When determining whether a slave is selected without using an interrupt when WUP = 0, do not use the address match detection method. Instead, use transfer of commands set in advance in a program.

(c) When WUP is set to 1 during $\overline{\text{BUSY}}$ signal output, $\overline{\text{BUSY}}$ is not released. In the SBI mode, after release of $\overline{\text{BUSY}}$ is directed, the $\overline{\text{BUSY}}$ signal is output until the next falling edge of the serial clock ($\overline{\text{SCK}}$) appears. Before setting WUP to 1, be sure to confirm that the SB0 (or SB1) pin is high after releasing $\overline{\text{BUSY}}$.

(12) SBI mode

This section describes an example of application which performs serial data communication in the SBI mode. In the example, the µPD750108 can be used as either the master CPU or a slave CPU on the serial bus.

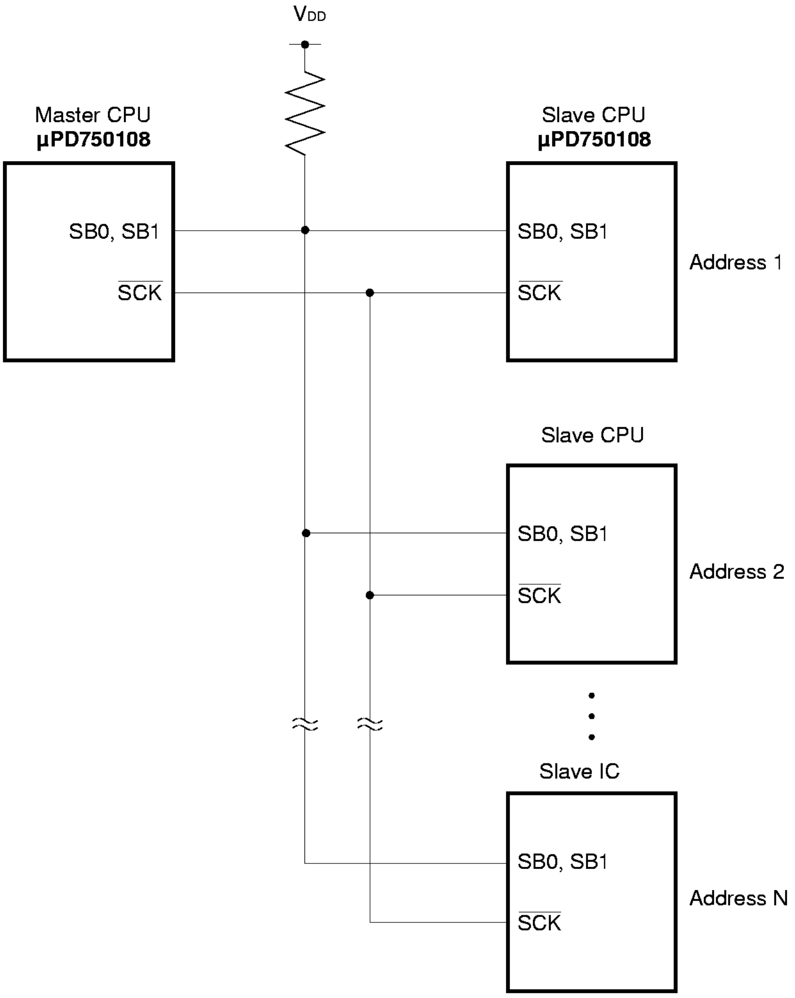
The master can be switched to another CPU with a command.

(a) Serial bus configuration

In the serial bus configuration used for the example of this section, a µPD750108 is connected to the bus line as a device on the serial bus.

Two pins on the µPD750108 are used: serial data bus SB0 (or SB1) and serial clock $\overline{\text{SCK}}$ (P01). Figure 5-71 shows an example of the serial bus configuration.

Figure 5-71. Example of Serial Bus Configuration



(b) Explanation of commands

(i) Types of commands

This example uses the following commands:

<1> READ command : Transfers data from slave to master.
<2> WRITE command : Transfers data from master to slave.
<3> END command : Informs slave of WRITE command completion.
<4> STOP command : Informs slave of WRITE command interruption.
<5> STATUS command : Reads slave status.
<6> $\overline{\text{RESET}}$ command : Sets currently selected slave as non-selected slave.
<7> CHGMST command: Passes master authority to slave.

(ii) Protocol

The following protocol is used for communication between the master and slaves.

<1> The address of a slave with which the master intends to communicate is transmitted to select the slave (chip select). This starts communication.
The slave that has received the address returns $\overline{\text{ACK}}$ to engage in communication with the master (The state of the slave is changed from the non-selected state to selected state).

<2> Commands and data are transferred between the master and the slave selected in <1>.
Command and data are transferred between the master and the selected slave on a one-to-one basis, so the other slaves must be placed in the non-selected state.

<3> Communication is completed when the selected slave is placed in the non-selected state.
This state is caused in the following cases:

- The selected slave is placed in the non-selected state when the slave receives a $\overline{\text{RESET}}$ command from the master.
- The device that is switched from the master to a slave with a CHGMST command is placed in the non-selected state.

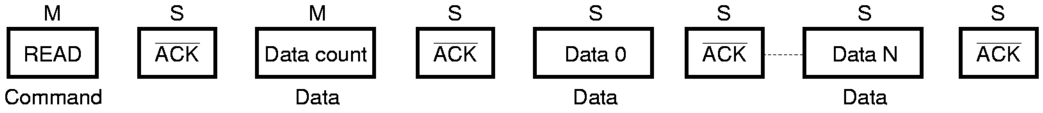
(iii) Command format

The transfer format of each command is described below.

<1> READ command

The READ command reads data from a slave. One to 256 bytes of data can be read. The data length is specified in a parameter by the master. When 00H is specified as the data length, the 256-byte data transfer is assumed.

Figure 5-72. Transfer Format of the READ Command

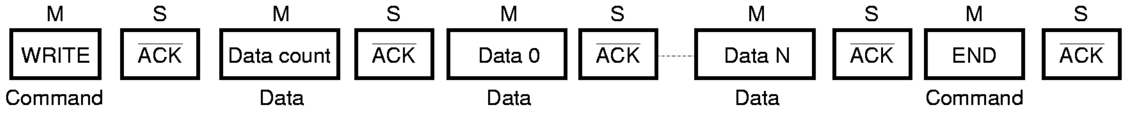


Remark M: Output by the master
S: Output by the slave

When the slave receives a transmission data count, if it has data enough for transmitting the specified number of bytes of data, the slave returns $\overline{\text{ACK}}$. If the slave does not have enough data for transmission, an error occurs; $\overline{\text{ACK}}$ is not returned in this case.
The master sends $\overline{\text{ACK}}$ to the slave each time it receives one byte.

<2> **WRITE command, END command, STOP command**
These commands write data to a slave. One to 256 bytes of data can be written. The data length is specified in a parameter by the master. When 00H is specified as the data length, the 256-byte data transfer is assumed.

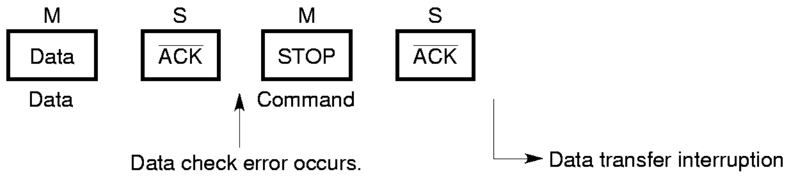
Figure 5-73. Transfer Format of the WRITE and END Commands



Remark M: Output by the master
S: Output by the slave

If the slave has an enough area for storing receive data of the specified length, the slave returns $\overline{\text{ACK}}$. If the slave does not have an enough area, an error occurs; $\overline{\text{ACK}}$ is not returned in this case.
The master transmits an END command when all data have been transferred. The END command informs the slave that all data have been transferred correctly.
The slave accepts an END command even before data reception is uncompleted. In this case, the data received just before the acceptance of the END command becomes valid.
The master compares the contents of SIO before transfer with the contents of SIO after transfer to check whether the data has been output onto the bus correctly. If the contents of SIO disagree with each other, the master interrupts data transfer by transmitting a STOP command.

Figure 5-74. Transfer Format of the STOP Command



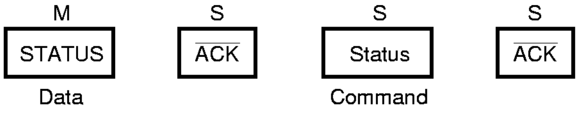
Remark M: Output by the master
S: Output by the slave

When the slave receives a STOP command, the slave invalidates the most recently received one byte.

<3> STATUS command

The STATUS command reads the status of the current slave.

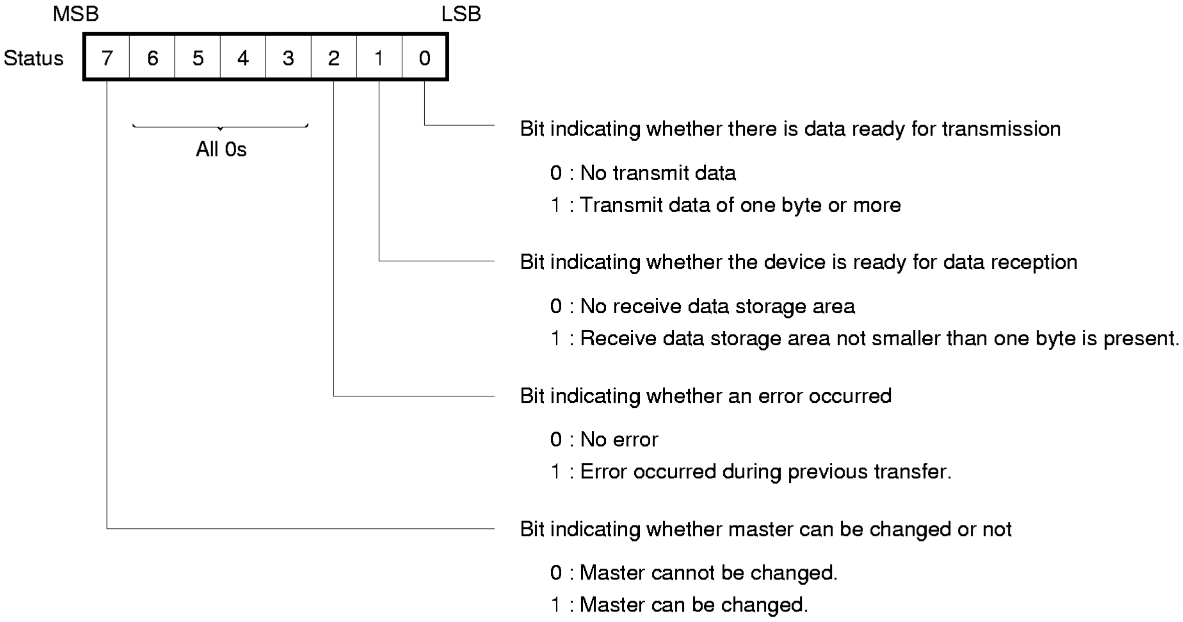
Figure 5-75. Transfer Format of the STATUS Command



Remark M: Output by the master
S: Output by the slave

The slave returns the status in the format shown in **Figure 5-78**.

Figure 5-76. Status Format of the STATUS Command

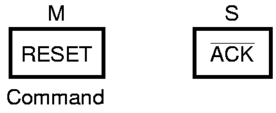

MSB
LSB
Status
7
6
5
4
3
2
1
0
All 0s
Bit indicating whether there is data ready for transmission
0 : No transmit data
1 : Transmit data of one byte or more
Bit indicating whether the device is ready for data reception
0 : No receive data storage area
1 : Receive data storage area not smaller than one byte is present.
Bit indicating whether an error occurred
0 : No error
1 : Error occurred during previous transfer.
Bit indicating whether master can be changed or not
0 : Master cannot be changed.
1 : Master can be changed.

When the master receives a status, it returns $\overline{\text{ACK}}$ to the current slave.

<4> RESET command

The $\overline{\text{RESET}}$ command changes the currently selected slave to a non-selected slave. When a $\overline{\text{RESET}}$ command is transmitted, any slave can be placed in the non-selected state.

Figure 5-77. Transfer Format of the RESET Command

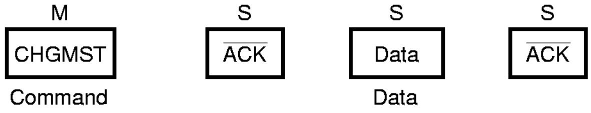


Remark M: Output by the master
S: Output by the slave

<5> CHGMST command

The CHGMST command passes the master authority to the currently selected slave.

Figure 5-78. Transfer Format of the CHGMST Command



Remark M: Output by the master
S: Output by the slave

When the slave receives a CHGMST command, the slave returns one of the following data to the master after checking whether the slave can receive the master authority:

- 0FFH: Master changeable
- 00H: Master not changeable

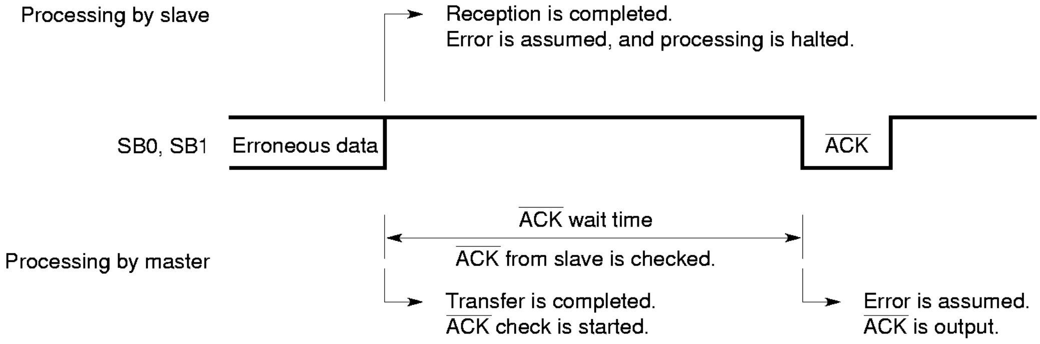
The slave compares the contents of SIO before transfer with the contents of SIO after transfer. If the contents of SIO disagree with each other, an error occurs; $\overline{\text{ACK}}$ is not returned in this case.
If the master receives 0FFH, the master returns $\overline{\text{ACK}}$ to the slave, and starts to operate as a slave. The slave which transmitted 0FFH starts to operate as the master when it receives $\overline{\text{ACK}}$.

(iv) Error occurrence

If a communication error occurs, the operation described below is performed.
The slave reports the occurrence of an error by not returning $\overline{\text{ACK}}$ to the master. If an error occurs during reception of data, the slave sets the status bit for indicating error occurrence, and cancels all command processing being performed.
When the transmission of one byte is completed, the master checks for $\overline{\text{ACK}}$ from the slave.

If $\overline{\text{ACK}}$ is not returned from the slave within a predetermined period after transmission completion, the occurrence of an error is assumed; the master outputs the $\overline{\text{ACK}}$ signal as a dummy.

Figure 5-79. Master and Slave Operation in Case of Error


Processing by slave
Reception is completed.
Error is assumed, and processing is halted.
SB0, SB1
Erroneous data
$\overline{\text{ACK}}$
$\overline{\text{ACK}}$ wait time
Processing by master
$\overline{\text{ACK}}$ from slave is checked.
Transfer is completed.
$\overline{\text{ACK}}$ check is started.
Error is assumed.
$\overline{\text{ACK}}$ is output.

The following errors may occur:

- Error that may occur on the slave side
  <1> Invalid command transfer format
  <2> Reception of an undefined command
  <3> Insufficient number of transfer data bytes for a READ command
  <4> Insufficient area to contain data for a WRITE command
  <5> Change in data during transmission of a READ, STATUS, or CHGMST command

  If any of the above types of errors occurs, $\overline{\text{ACK}}$ is not returned.

- Error that may occur on the master side

  If data transmitted with a WRITE command changes during transmission, the master transmits a STOP command to the slave.

5.6.8 Manipulation of $\overline{\text{SCK}}$ Pin Output

The $\overline{\text{SCK}}$/P01 pin has a built-in output latch, so that this pin allows static output by software manipulation in addition to normal serial clock output.

The number of $\overline{\text{SCK}}$ pulses can be software-set arbitrarily by manipulating the P01 output latch. (The SO/SB0/P02 or SI/SB1/P03 pin is controlled by manipulating the RELT and CMDT bits of SBIC.)

The procedure for manipulating $\overline{\text{SCK}}$/P01 pin output is explained below.

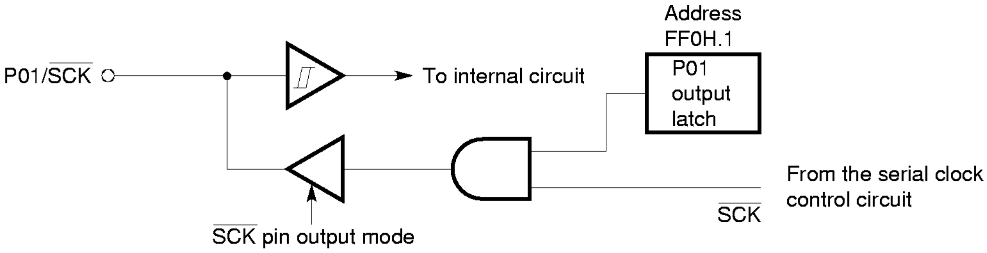
<1> Set serial operation mode register (CSIM) ($\overline{\text{SCK}}$ pin: output mode). When serial transfer is halted, $\overline{\text{SCK}}$ from the serial clock control circuit is set to 1.

<2> Manipulate the P01 output latch by using a bit manipulation instruction.

Example To output one $\overline{\text{SCK}}$/P01 pin clock cycle by software

```
SEL   MB15            ; or CLR1  MBE
MOV   XA,#10000011B   ; SCK (fCC/2^3), output mode
MOV   CSIM,XA
CLR1  0FF0H.1         ; SCK/P01 <- 0
SET1  0FF0H.1         ; SCK/P01 <- 1
```

Figure 5-80. $\overline{\text{SCK}}$/P01 Pin Circuit Configuration



The P01 output latch is mapped to bit 1 of address FF0H. A $\overline{\text{RESET}}$ signal sets the P01 output latch to 1.

Cautions 1. During normal serial transfer, the P01 output latch must be set to 1.

2. The P01 output latch cannot be addressed by specifying PORT0.1 (as described below). The address of the latch (0FF0H.1) must be coded in the operand of an instruction directly. However, MBE = 0 (or MBE = 1, MBS = 15) must be specified before the instruction is executed.

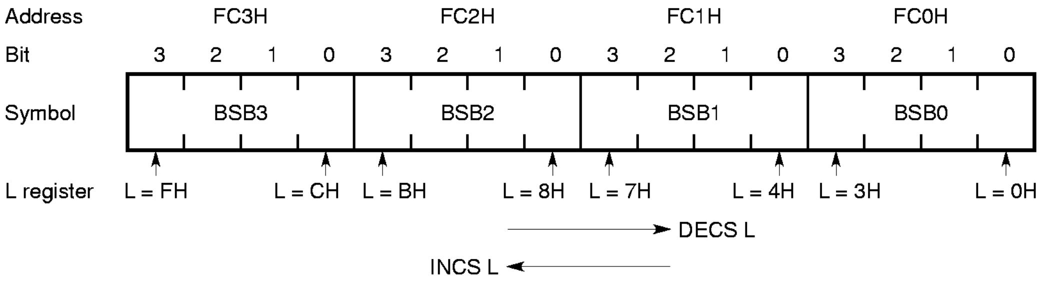
```
CLR1  PORT0.1 }
SET1  PORT0.1 }   Not allowed
CLR1  0FF0H.1 }
SET1  0FF0H.1 }   Allowed
```

5.7 BIT SEQUENTIAL BUFFER: 16-BIT

The bit sequential buffer (BSB) is special data memory for bit manipulations. In particular, the buffer allows bit manipulations to be performed very easily by sequentially changing address and bit specifications. So the buffer is useful in processing long data bit by bit.

This data memory consists of 16 bits, and allows pmem.@L addressing with a bit manipulation instruction. This addressing uses the L register for indirect bit specification. In this case, only by incrementing or decrementing the L register in a program loop, the bit to be manipulated can be sequentially shifted for continued processing.

Figure 5-81. Format of the Bit Sequential Buffer



Remarks 1. With pmem.@L addressing, bit specification is shifted according to the L register.

2. With pmem.@L addressing, BSB can be manipulated at any time regardless of MBE/MBS specification.

Data can also be manipulated by direct addressing. The buffer can be used for applications such as continuous 1-bit data input or output operations by combining direct 1-bit, 4-bit, and 8-bit addressing with pmem.@L addressing. In 8-bit manipulation, the higher eight bits or lower eight bits are manipulated by specifying BSB0 or BSB2.

Example To output 16-bit data of BUFF1 and BUFF2 serially from bit 0 of port 3:

```
          CLR1    MBE
          MOV     XA,BUFF1
          MOV     BSB0,XA     ; Set BSB0 and BSB1
          MOV     XA,BUFF2
          MOV     BSB2,XA     ; Set BSB2 and BSB3
          MOV     L,#0
LOOP0:    SKT     BSB0, @L    ; Tests the specification bit of BSB
          BR      LOOP1
          NOP                 ; Dummy (For timing adjustment)
          SET1    PORT3. 0    ; Sets bit 0 of port 3
          BR      LOOP2
LOOP1:    CLR1    PORT3. 0    ; Clears bit 0 of port 3
          NOP                 ; Dummy (For timing adjustment)
          NOP
LOOP2:    INCS    L           ; L <- L + 1
          BR      LOOP0
          RET
```

CHAPTER 6 INTERRUPT AND TEST FUNCTIONS

The µPD750108 has seven vectored interrupt sources and two test inputs, allowing a wide range of applications.

In addition, the interrupt control circuitry of the µPD750108 has the following features for very high-speed interrupt processing.

(1) Interrupt functions

(a) Hardware controlled vectored interrupt function which can control whether or not to accept an interrupt using the interrupt flag (IExxx) and interrupt master enable flag (IME).

(b) The interrupt start address can be set arbitrarily.

(c) Multiple interrupt function which can specify the priority by the interrupt priority specification register (IPS)

(d) Test function of an interrupt request flag (IRQxxx)
(The software can confirm that an interrupt occurred.)

(e) Release of the standby mode (Interrupts released by an interrupt enable flag can be selected.)

(2) Test functions

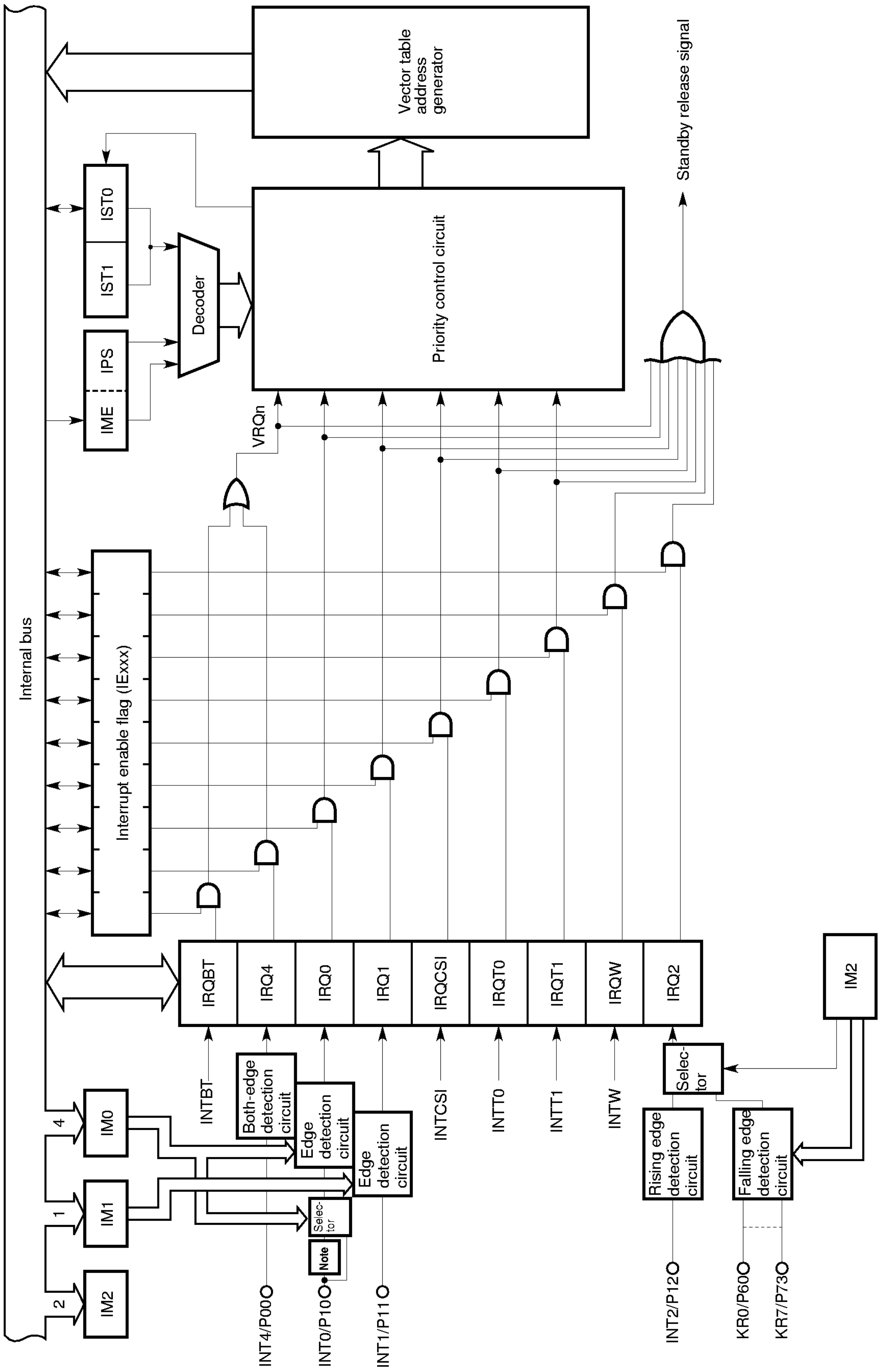
(a) Whether test request flags (IRQxxx) are issued can be checked with software.

(b) Release of the standby mode (A test source to be released can be selected with test enable flags.)

6.1 CONFIGURATION OF THE INTERRUPT CONTROL CIRCUIT

Figure 6-1 shows the configuration of the interrupt control circuit. Each hardware item is mapped to a data memory space.

Figure 6-1. Block Diagram of Interrupt Control Circuit



Note Noise eliminator (when the noise eliminator is selected, standby mode cannot be released.)

6.2 TYPES OF INTERRUPT SOURCES AND VECTOR TABLES

Table 6-1 lists the types of interrupt sources, and Figure 6-2 shows vector tables.

Table 6-1. Interrupt Sources

Interrupt source signal		In/out	Interrupt priority[Note]	Vectored interrupt request (vector table address)
INTBT	[Reference time interval signal from basic interval timer/wactchdog timer]	In	1	VRQ1 (0002H)
INT4	[Detection of both rising and falling edges]	Out		
INT0	[Rising/falling edge detection specification]	Out	2	VRQ2 (0004H)
INT1		Out	3	VRQ3 (0006H)
INTCSI	[Serial data transfer completion signal]	In	4	VRQ4 (0008H)
INTT0	[Match signal between the count register of timer/event counter 0 and modulo register]	In	5	VRQ5 (000AH)
INTT1	[Match signal between the count register of timer counter 1 and modulo register]	In	6	VRQ6 (000CH)

Note The interrupt priority is used to determine the priority when two or more interrupts are simultaneously generated.

Figure 6-2. Interrupt Vector Table

Address			
0000H	MBE	RBE	Internal reset start address (high-order 6 bits)
			Internal reset start address (low-order 8 bits)
0002H	MBE	RBE	INTBT/INT4 start address (high-order 6 bits)
			INTBT/INT4 start address (low-order 8 bits)
0004H	MBE	RBE	INT0 start address (high-order 6 bits)
			INT0 start address (low-order 8 bits)
0006H	MBE	RBE	INT1 start address (high-order 6 bits)
			INT1 start address (low-order 8 bits)
0008H	MBE	RBE	INTCSI start address (high-order 6 bits)
			INTCSI start address (low-order 8 bits)
000AH	MBE	RBE	INTT0 start address (high-order 6 bits)
			INTT0 start address (low-order 8 bits)
000CH	MBE	RBE	INTT1 start address (high-order 6 bits)
			INTT1 start address (low-order 8 bits)

The column of interrupt priority in Table 6-1 indicates a priority assigned when multiple interrupt requests occur concurrently or are held.

A vector table contains interrupt processing start addresses and MBE and RBE setting values during interrupt processing. An assembler pseudo instruction (VENTn) is used to set a vector table.

Example A vector table is set for INTBT/INT4.

VENT1	MBE = 0, RBE = 0,	GOTOBT
Vector table at address 0002	MBE·RBE setting value in interrupt service routine	Symbol for indicating an interrupt service routine start address

Caution The vector table specified by VENTn (n = 1 to 6) is located at address 2n.

Example Vector tables are set for INTBT/INT4 and INTT0.

```
VENT1   MBE = 0, RBE = 0, GOTOBT
VENT5   MBE = 0, RBE = 1, GOTOT0
```

6.3 VARIOUS DEVICES TO CONTROL INTERRUPT FUNCTIONS

(1) Interrupt request flags and interrupt enable flags

The following seven interrupt request flags (IRQxxx) corresponding to the interrupt sources are provided.

INT0 interrupt request flag (IRQ0)
INT1 interrupt request flag (IRQ1)
INT4 interrupt request flag (IRQ4)
BT interrupt request flag (IRQBT)
Serial interface interrupt request flag (IRQCSI)
Timer/event counter interrupt request flag (IRQT0)
Timer counter interrupt request flag (IRQT1)

An interrupt request flag is set to 1 by an interrupt request, and is automatically cleared to 0 when interrupt processing is performed. However, IRQBT and IRQ4 are cleared in a different way because these flags share a vector address. (See **Section 6.6**.)
The following seven interrupt enable flags (IExxx) corresponding to the interrupt request flags are provided.

INT0 interrupt enable flag (IE0)
INT1 interrupt enable flag (IE1)
INT4 interrupt enable flag (IE4)
BT interrupt enable flag (IEBT)
Serial interface interrupt enable flag (IECSI)
Timer/event counter interrupt enable flag (IET0)
Timer counter interrupt enable flag (IET1)

An interrupt enable flag set to 1 enables the corresponding interrupt, and an interrupt enable flag set to 0 disables the corresponding interrupt.

When an interrupt request flag and the interrupt enable flag are set to 1, a vectored interrupt request (VRQn) occurs. This condition is also used to release a standby mode.

A bit manipulation instruction or 4-bit memory manipulation instruction is used to manipulate an interrupt request flag and interrupt enable flag. A bit manipulation instruction allows direct manipulation regardless of MBE setting. An interrupt enable flag can be manipulated using an EI IExxx instruction or DI IE instruction. The SKTCLR instruction is usually used to test an interrupt request flag.

Example

```
EI       IE0          ; Enable INT0
DI       IE1          ; Disable INT1
SKTCLR   IRQCSI       ; Skip and clear IRQCSI when it is set to 1.
```

When an interrupt request flag is set using an instruction, even if there is no interrupt request, a vectored interrupt is executed in the same way as when an interrupt is requested.

Inputting a $\overline{\text{RESET}}$ signal clears the interrupt request and interrupt enable flags to 0, disabling all interrupts.

Table 6-2. Set Signals for Interrupt Request Flags

Interrupt request flag	Set signals for interrupt request flags	Interrupt enable flag
IRQBT	Set by a reference time interval signal from the basic interval timer/watchdog timer.	IEBT
IRQ4	Set by a detected rising or falling edge of an INT4/P00 pin input signal.	IE4
IRQ0	Set by a detected edge of an INT0/P10 pin input signal. The detection edge is specified by the INT0 edge detection mode register (IM0).	IE0
IRQ1	Set by a detected edge of an INT1/P11 pin input signal. The detection edge is specified by the INT1 edge detection mode register (IM1).	IE1
IRQCSI	Set by a serial data transfer completion signal for the serial interface.	IECSI
IRQT0	Set by a match signal from timer/event counter 0.	IET0
IRQT1	Set by a match signal from the timer counter.	IET1

(2) Interrupt priority specification register (IPS)

The interrupt priority specification register selects an interrupt with a higher priority from multiple interrupts using the low-order three bits.

Bit 3, interrupt master enable flag (IME), specifies whether to disable all interrupts.

The IPS is set using a 4-bit memory manipulation instruction. Bit 3 is set by an EI instruction and reset by a DI instruction.

When changing the low-order three bits of the IPS, interrupts must be disabled (IME = 0) beforehand.

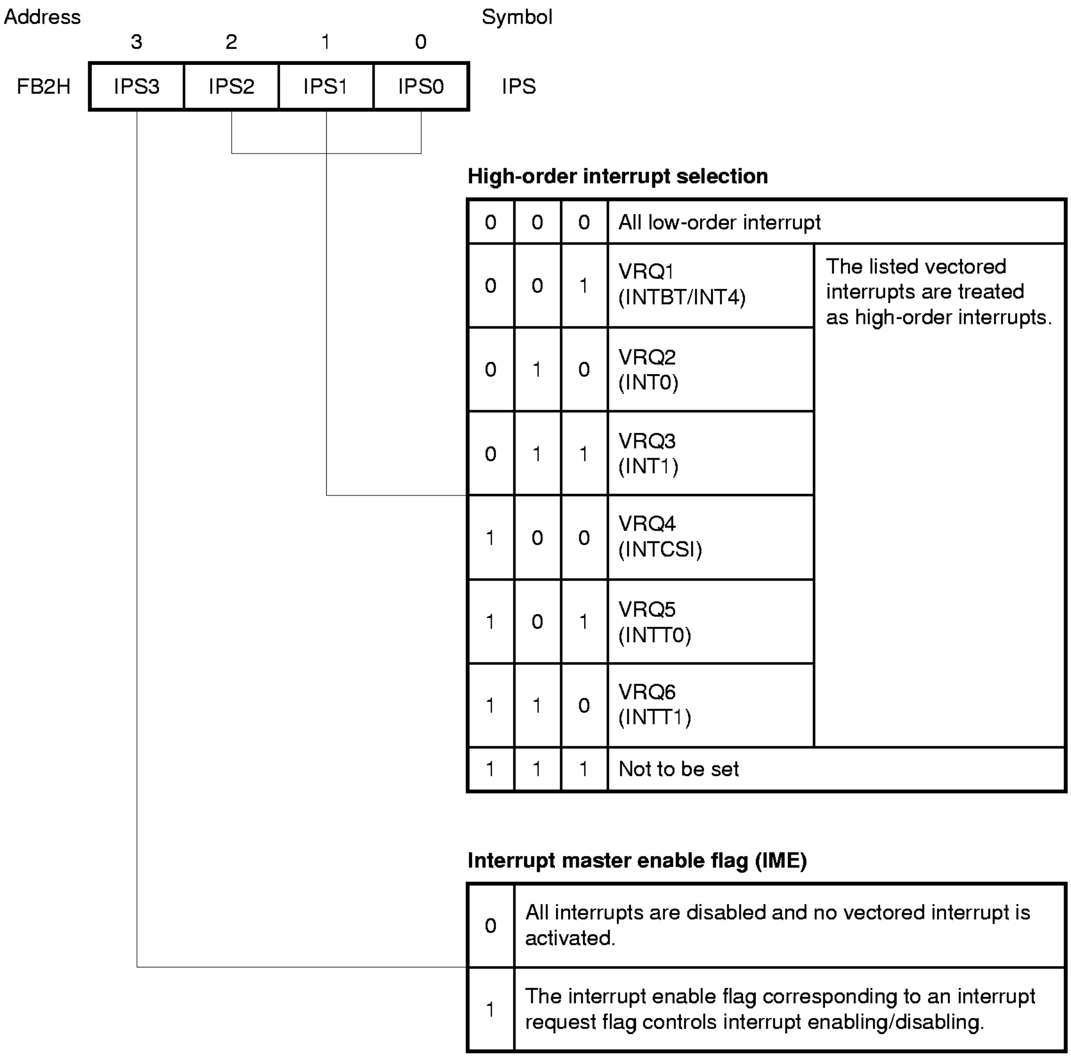
Example

```
DI                  ; Disable interrupts
CLR1    MBE
MOV     A,#1011B
MOV     IPS,A       ; Assign a higher priority to INT1, then enable interrupts.
```

A $\overline{\text{RESET}}$ signal clears all bits to 0.

Caution Disable interrupts before setting the IPS.

Figure 6-3. Interrupt Priority Specification Register



Address	3	2	1	0	Symbol
FB2H	IPS3	IPS2	IPS1	IPS0	IPS

High-order interrupt selection

IPS2	IPS1	IPS0		
0	0	0	All low-order interrupt	
0	0	1	VRQ1 (INTBT/INT4)	The listed vectored interrupts are treated as high-order interrupts.
0	1	0	VRQ2 (INT0)	
0	1	1	VRQ3 (INT1)	
1	0	0	VRQ4 (INTCSI)	
1	0	1	VRQ5 (INTT0)	
1	1	0	VRQ6 (INTT1)	
1	1	1	Not to be set	

Interrupt master enable flag (IME)

IPS3	
0	All interrupts are disabled and no vectored interrupt is activated.
1	The interrupt enable flag corresponding to an interrupt request flag controls interrupt enabling/disabling.

(3) Configurations of the INT0, INT1, and INT4 circuits

(a) As shown in Figure 6-4 (a), the INT0 circuit accepts an external interrupt at its rising or falling edge. The edge to be detected can be selected.

The INT0 circuit has a noise elimination function (see **Figure 6-5**), called a noise eliminator, using a sampling clock, which removes pulses shorter than two sampling clock cycles**Note** as noise. The INT0 circuit may accept pulses which are longer than one sampling clock cycle and shorter than two cycles as interrupts depending on the sampling timing (see **Figure 6-4 (a)**). The circuit is sure to accept pulses equal to or longer than two sampling clock cycles as interrupts.

The INT0 pin is supplied with sampling clock Φ or $f_{CC}/64$, whichever is selected by bit 3 (IM03) of the INT0 edge detection mode register (IM0).

Bit 0 (IM00) and bit 1 (IM01) of the INT0 edge detection mode register (IM0) are used to select a detection edge.

Figure 6-6 (a) shows the format of IM0. A 4-bit memory manipulation instruction is used to set IM0. A $\overline{\text{RESET}}$ signal clears all bits to 0, and a rising edge is specified to be detected.

Note When the frequency of a sampling clock is Φ, these cycles are equal to $2t_{CY}$. When the frequency of a sampling clock is $f_{CC}/64$, these cycles are equal to $128/f_{CC}$.

Cautions 1. Input a pulse wider than two sampling clock cycles to the INT0/P10 pin. Otherwise, the pulse is suppressed as noise by a noise eliminator when the pin is used as a port.

2. When the noise eliminator is selected (IM02 is set to 0), INT0 does not operate in standby mode because INT0 requires a clock for sampling. Do not select the noise eliminator when using INT0 to release standby mode (set IM02 to 1).

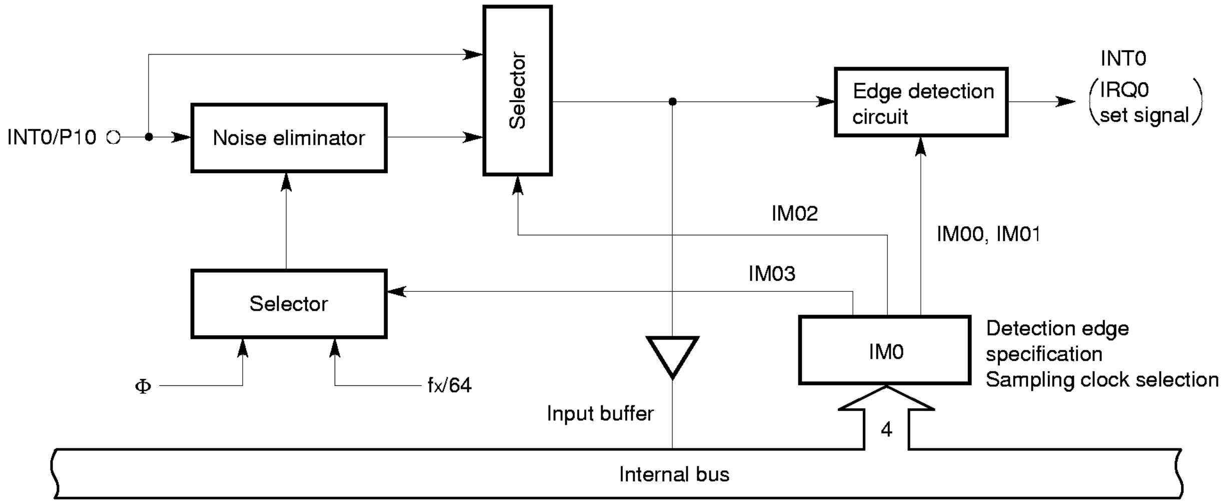
(b) As shown in Figure 6-4 (b), the INT1 circuit accepts an external interrupt at its rising or falling edge. The INT1 edge detection mode register (IM1) is used to select a detection edge.

Figure 6-6 (b) shows the format of IM1. A bit manipulation instruction is used to set IM1. A $\overline{\text{RESET}}$ signal clears all bits to 0, and a rising edge is specified to be detected.

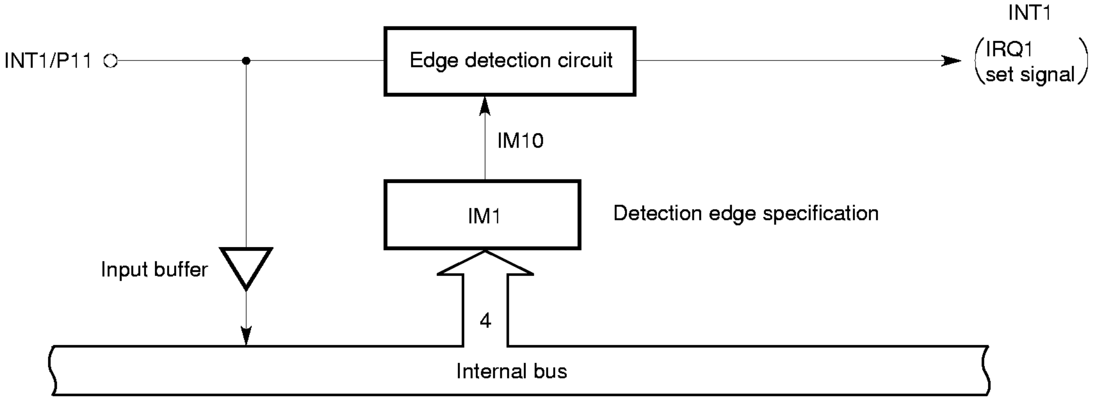
(c) As shown in Figure 6-4 (c), the INT4 circuit accepts an external interrupt at its rising and falling edges.

Figure 6-4. Configurations of the INT0, INT1, and INT4 Circuits

(a) Configuration of the INT0 circuit



(b) Configuration of the INT1 circuit



(c) Configuration of the INT4 circuit

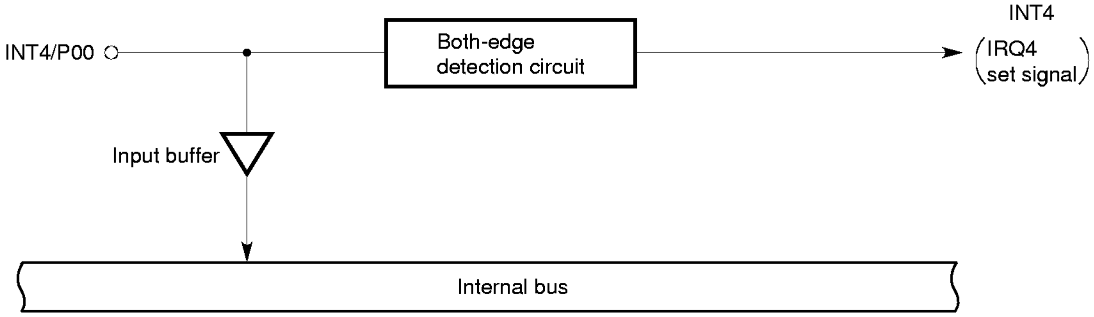
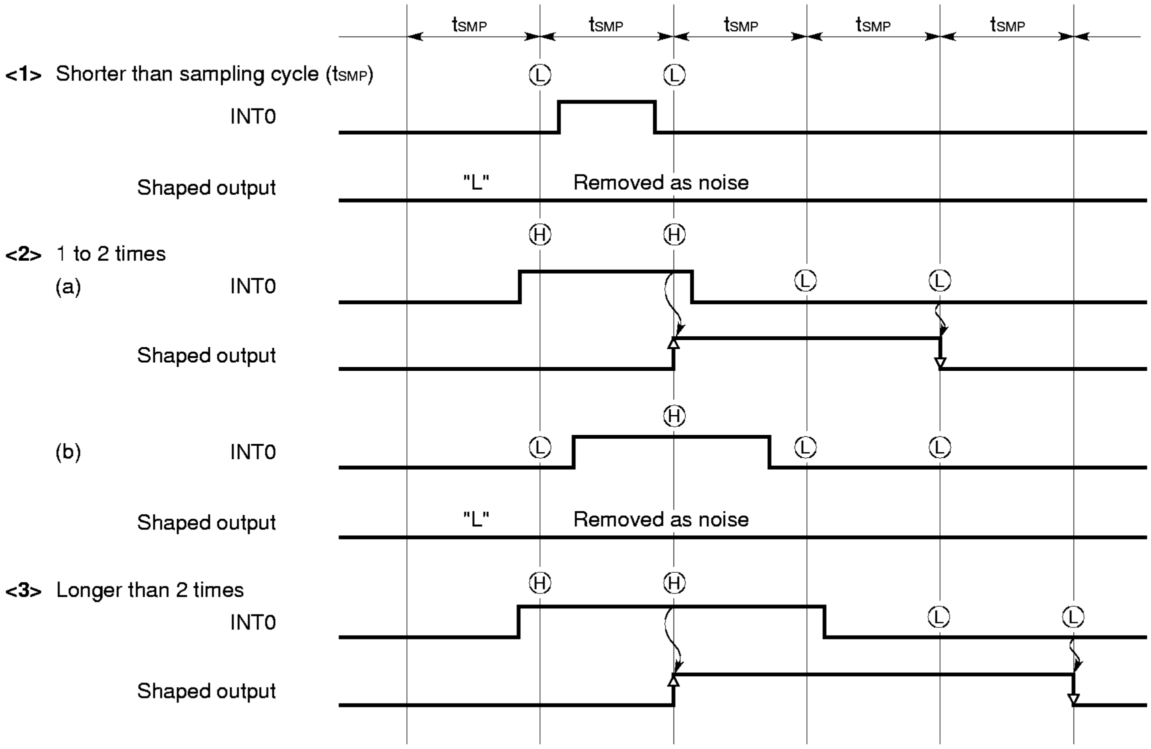


Figure 6-5. I/O Timing of a Noise Eliminator



Remark $t_{SMP} = t_{CY}$ or $64/f_{CC}$

Figure 6-6. Format of Edge Detection Mode Registers

(a) INT0 edge detection mode register (IM0)

Address	3	2	1	0	Symbol
FB4H	IM03	IM02	IM01	IM00	IM0

IM01	IM00	Detection edge specification
0	0	Specifies rising edge.
0	1	Specifies falling edge.
1	0	Specifies both rising and falling edges.
1	1	Ignored (No interrupt request flag is set.)

IM02	Noise eliminator selection bit	Sampling	Standby mode
0	Selects a noise eliminator.	Enabled	Cannot be released
1	Does not select a noise eliminator.	Disabled	Can be released

IM03	Sampling clock
0	Φ (2, 4, 8, 32 µs at 2 MHz, 4, 8, 16, 64 µs at 1 MHz)
1	f_{CC}/64 (32 µs at 2 MHz, 64 µs at 1 MHz)

(b) INT1 edge detection mode register (IM1)

Address	3	2	1	0	Symbol
FB5H	0	0	0	IM10	IM1

IM10	Detection edge specification
0	Specifies rising edge.
1	Specifies falling edge.

Caution Changing the edge detection mode register may set an interrupt request flag. So, disable the interrupts before changing the edge detection mode register. Then clear the interrupt request flag with a CLR1 instruction and enable the interrupts. When f_{CC}/64 is selected as a sampling clock pulse in changing IM0, wait for 16 machine cycles after changing the mode register and clear the interrupt request flag.

(4) Interrupt status flags

The interrupt status flags (IST0 and IST1), which are contained in the PSW, indicate the status of processing currently executed by the CPU.

By using the content of these flags, the interrupt priority control circuit controls multiple interrupts as indicated in Table 6-3.

A 4-bit manipulation instruction or bit manipulation instruction can be used to set and reset IST0 and IST1, so that multiple interrupts are enabled by changing the current status of execution. IST0 and IST1 can be manipulated on a single-bit basis at any time regardless of MBE setting.

Before IST0 or IST1 is manipulated, the DI instruction must be executed to disable interrupts, then the EI instruction must be executed to enable interrupts.

IST1 and IST0 as well as the other PSW bits are saved in the stack memory when an interrupt is accepted and the status of IST0 and IST1 changes to a status one level higher. When a RETI instruction is executed, the former values of IST1 and IST0 are resumed.

Inputting a $\overline{\text{RESET}}$ signal clears the content of the flag to 0.

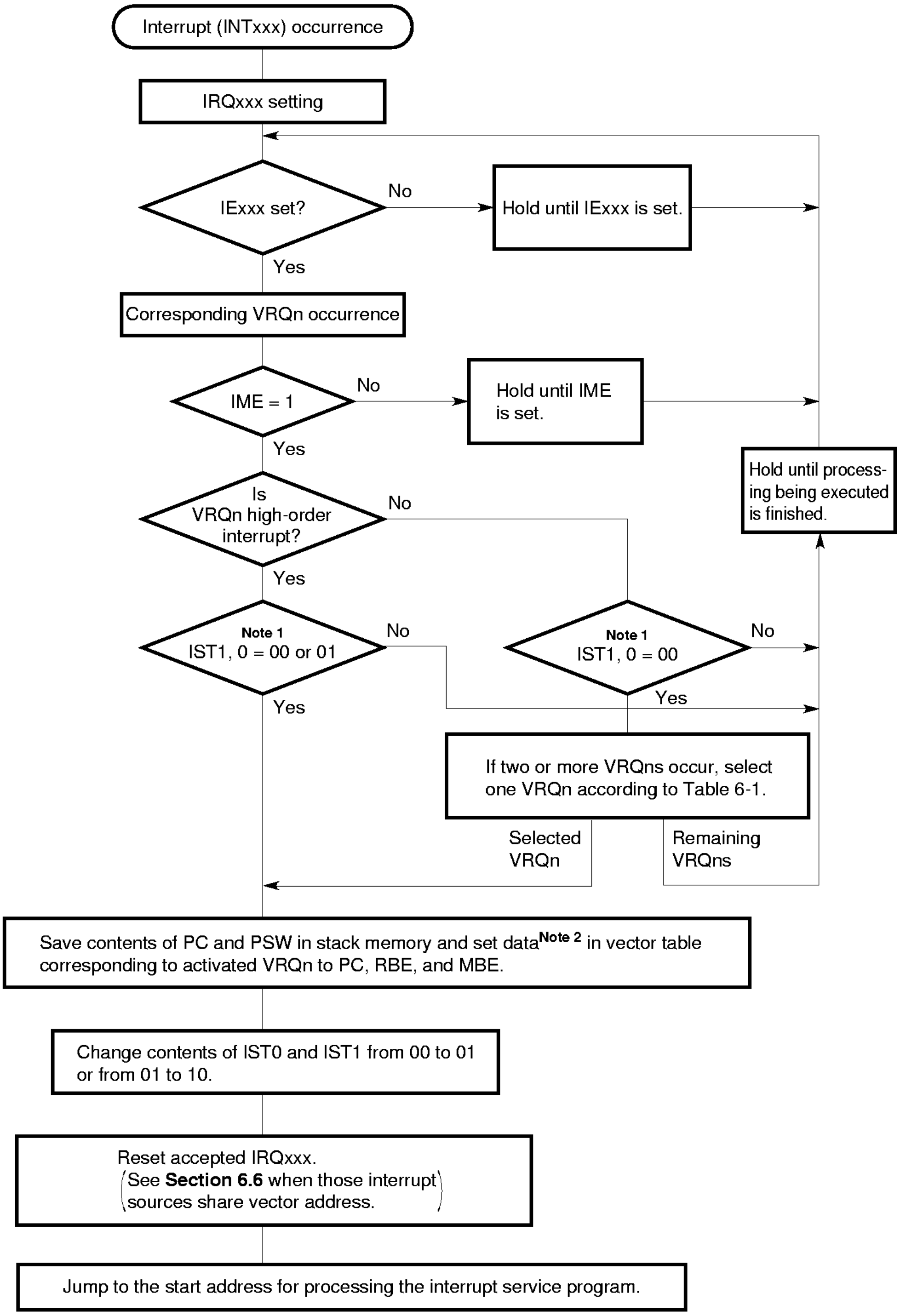
Table 6-3. Interrupt Processing Statuses of IST0 and IST1

IST1	IST0	Processing status	CPU operation	Interrupts that can be accepted	After acceptance	
					IST1	IST0
0	0	Status 0	Is processing the normal program.	All	0	1
0	1	Status 1	Is processing a low- or high-order interrupt.	Only high-order interrupts	1	0
1	0	Status 2	Is processing a high-order interrupt.	No	—	—
1	1	Not to be set				

6.4 INTERRUPT SEQUENCE

When an interrupt occurs, it is processed using the procedure shown in Figure 6-7.

Figure 6-7. Interrupt Sequence


Interrupt (INTxxx) occurrence
IRQxxx setting
IExxx set?
No
Hold until IExxx is set.
Yes
Corresponding VRQn occurrence
IME = 1
No
Hold until IME is set.
Yes
Hold until processing being executed is finished.
Is VRQn high-order interrupt?
No
Yes
Note 1
IST1, 0 = 00 or 01
No
Note 1
IST1, 0 = 00
No
Yes
Yes
If two or more VRQns occur, select one VRQn according to Table 6-1.
Selected VRQn
Remaining VRQns
Save contents of PC and PSW in stack memory and set data Note 2 in vector table corresponding to activated VRQn to PC, RBE, and MBE.
Change contents of IST0 and IST1 from 00 to 01 or from 01 to 10.
Reset accepted IRQxxx.
(See Section 6.6 when those interrupt sources share vector address.)
Jump to the start address for processing the interrupt service program.

Notes 1. IST0 and IST1 are the interrupt status flags (bits 3 and 2 of the PSW). (See **Table 6-3**.)

2. An interrupt service program start address and MBE and RBE setting values at the start of interrupt are stored in each vector table.

6.5 MULTIPLE INTERRUPT PROCESSING CONTROL

The µPD750108 can handle multiple interrupts by either of the following methods.

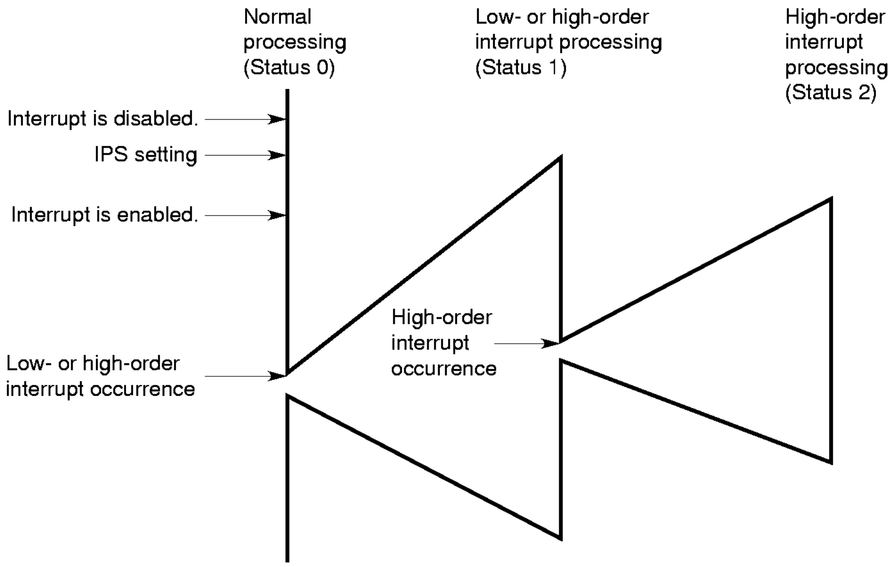
(1) Multiple interrupt processing by a high-order interrupt

In this method, the µPD750108 selects an interrupt source among multiple interrupt sources, enabling double interrupt processing.

That is, the high-order interrupt specified by the interrupt priority specification register (IPS) is enabled when the processing status is 0 or 1. Other interrupts (interrupts lower than the specified high-order interrupt) are enabled only when the status is 0. (See **Figure 6-8** and **Table 6-3**.)

When only one interrupt is used as a level-two interrupt, using this method saves the user the trouble of enabling or disabling interrupts during an interrupt processing, and holds down the number of nesting levels to two.

Figure 6-8. Multiple Interrupt Processing by a High-Order Interrupt



Normal processing (Status 0)
Low- or high-order interrupt processing (Status 1)
High-order interrupt processing (Status 2)
Interrupt is disabled.
IPS setting
Interrupt is enabled.
Low- or high-order interrupt occurrence
High-order interrupt occurrence

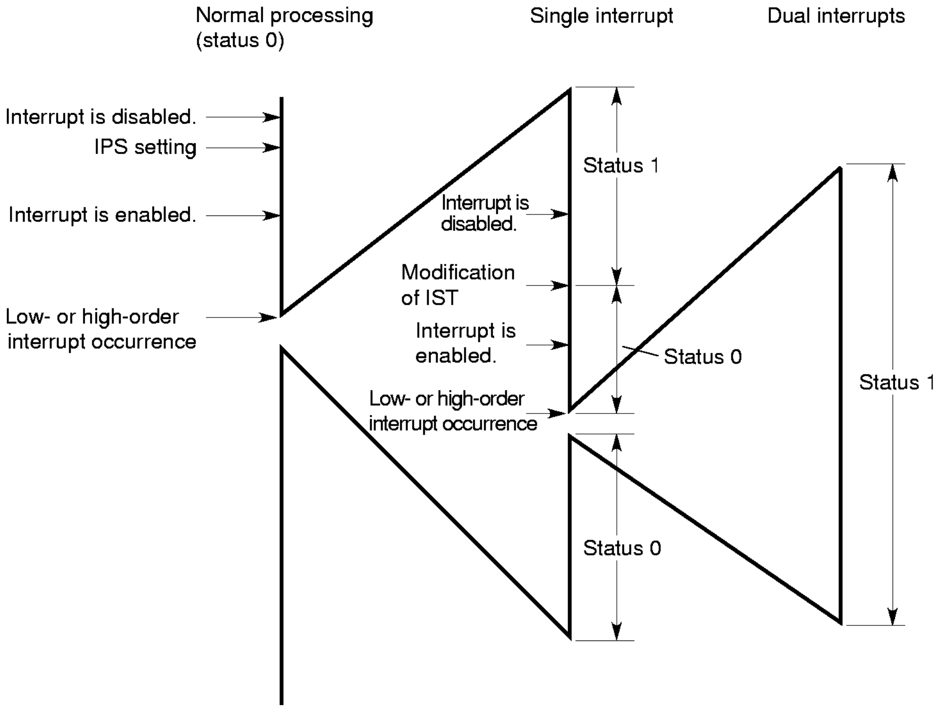
(2) Multiple interrupt processing by changing the interrupt status flags

Changing the interrupt status flags with the program causes multiple interrupts to be enabled. That is, when the interrupt processing program changes both IST1 and IST0 to 0 (status 0), multiple interrupt processing is enabled.

This method is used when two or more interrupts are to be enabled at a time or when the processing of three or more interrupts is to be performed.

When changing IST1 and IST0, interrupts must be disabled beforehand with a DI instruction.

Figure 6-9. Multiple Interrupt Processing by Changing the Interrupt Status Flags



6.6 PROCESSING OF INTERRUPTS SHARING A VECTOR ADDRESS

Interrupt sources INTBT and INT4 share a vector table, so an interrupt source is selected as described below.

(1) Using only one interrupt

The interrupt enable flag for desired one of the two interrupt sources sharing a vector table is set to 1, and the interrupt enable flag for the other is cleared to 0. In this case, the enabled (IExxx = 1) interrupt source causes an interrupt request. When the interrupt request is accepted, the interrupt request flag is reset.

(2) Using both interrupts

The interrupt enable flags corresponding to the two interrupt sources are both set to 1. In this case, the logical sum of the interrupt request flags for the two interrupt sources is used as an interrupt request. In this case, even if an interrupt request or interrupt requests caused by the setting of one or both of the interrupt request flags are accepted, the interrupt request flag or flags are not reset.

Accordingly, which of the two interrupt sources caused the interrupt needs to be determined using the interrupt service routine. For this determination, the DI instruction is to be executed at the start of the interrupt service routine, and the interrupt request flags are checked with the SKTCLR instruction.

If both the request flags are set when this request flag is tested or cleared, the interrupt request remains even if one of the request flags is cleared. If this interrupt is selected as having the higher priority, nesting processing is started by the remaining interrupt request.

Consequently, the interrupt request not tested is processed first. If the selected interrupt has the lower priority, the remaining interrupt is kept pending and therefore, the interrupt request tested is processed first. Therefore, an interrupt sharing a vector address with another interrupt is identified differently, depending whether it has the higher priority, as shown in Table 6-4.

Table 6-4. Identifying Interrupt Sharing Vector Table Address

With higher priority	Interrupt is disabled and interrupt request flag of interrupt that takes precedence is tested
With lower priority	Interrupt request flag of interrupt that takes precedence is tested

Examples 1. To use both INTBT and INT4 as having the higher priority and give priority to INT4

```
           DI
           SKTCLR     IRQ4       ;  IRQ4 = 1 ?
           BR         VSUBBT     ⎫
               :                 ⎪  Processing routine
           EI                    ⎬  of INT4
           RETI                  ⎪
               :                 ⎭

VSUBBT:    CLR1       IRQBT      ⎫
               :                 ⎪
               :                 ⎬  Processing routine
               :                 ⎪  of INTBT
           EI                    ⎭
           RETI
```

2. To use both INTBT and INT4 as having the lower priority and give priority to INT4

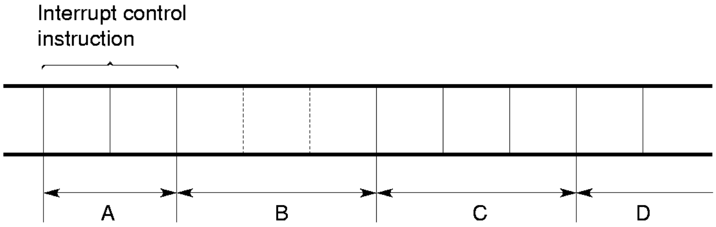
```
           SKTCLR     IRQ4       ;  IRQ4 = 1 ?
           BR         VSUBBT     ⎫
               :                 ⎪
               :                 ⎬  Processing routine
               :                 ⎪  of INT4
           RETI                  ⎭


VSUBBT:    CLR1       IRQBT      ⎫
               :                 ⎪
               :                 ⎬  Processing routine
               :                 ⎪  of INTBT
           RETI                  ⎭
```

6.7 MACHINE CYCLES FOR STARTING INTERRUPT PROCESSING

With the μPD750108 series, the following machine cycles are used to start the execution of the interrupt service routine after an interrupt request flag (IRQn) is set.

(1) When IRQn is set during execution of an interrupt control instruction

When IRQn is set during execution of an interrupt control instruction, an instruction preceded by that instruction is executed, and an interrupt processing of three machine cycles is executed, then the interrupt service routine is started.



A: IRQn is set.
B: The next instruction is executed (1 to 3 machine cycles according to the instruction).
C: Interrupt processing (3 machine cycles)
D: Interrupt service routine is executed.

Cautions 1. When interrupt control instructions are contiguous these interrupt control instructions are executed up to the last one. An instruction preceded by the interrupt control instruction executed last is executed, and an interrupt processing of three machine cycles is executed, then the interrupt service routine is started.

2. When a DI instruction is executed in the period during which IRQn is set (A in the figure), or in the immediately following period, the interrupt request of the set IRQn is held until an EI instruction is executed.

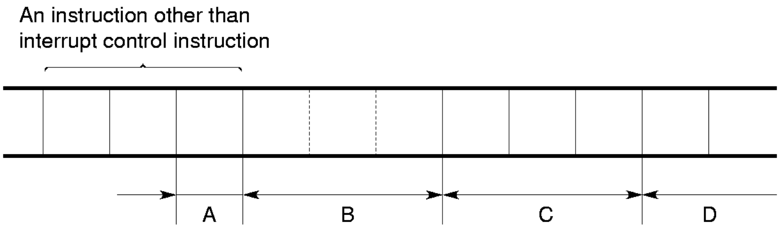
Remarks 1. An interrupt control instruction manipulates hardware (address FBxH in data memory) which handles interrupt processings. There are two types of interrupt control instruction, a DI instruction and an EI instruction.

2. Three machine cycles required for the interrupt processing include the time to manipulate the stack when an interrupt is accepted.

(2) When IRQn is set during an instruction other than that described in (1)

(a) When IRQn is set at the last machine cycle of the instruction being executed

In this case, an instruction preceded by the instruction being executed is executed, and an interrupt processing of three machine cycles is executed, then the interrupt service routine is started.

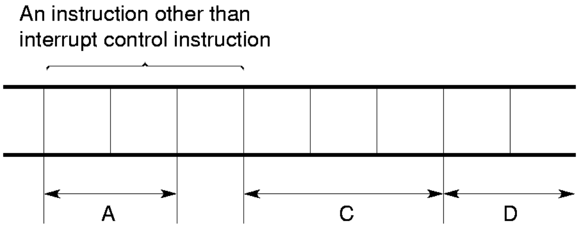


A: IRQn is set.
B: The next instruction is executed (1 to 3 machine cycles to the instruction).
C: Interrupt processing (3 machine cycles)
D: Interrupt service routine is executed.

Caution When one or more interrupt control instructions follow, an instruction preceded by the interrupt control instructions is executed, and an interrupt processing of three machine cycles is executed, then the interrupt service routine is started. When an instruction to be executed after setting IRQn is a DI instruction, the interrupt request of the set IRQn is held.

(b) When IRQn is set earlier than the last machine cycle of the instruction being executed

In this case, after executing the instruction being executed, an interrupt processing of three machine cycles is executed, then the interrupt service routine is started.



A: IRQn is set.
C: Interrupt processing (3 machine cycles)
D: Interrupt service routine is executed.

6.8 EFFECTIVE USE OF INTERRUPTS

The interrupt function can be used more effectively in the ways described below.

(1) MBE = 0 is set for the interrupt service routine

By allocating addresses 00H to 7FH as data memory used by the interrupt service routine and specifying MBE = 0 in an interrupt vector table, the user can code a program without being concerned with a memory bank.

If a program must use memory bank 1 for some reason, save the memory bank select register using the PUSH BS instruction before selecting memory bank 1.

(2) Use different register banks for the normal routine and interrupt routine.

The normal routine uses register banks 2 and 3 with RBE = 1 and RBS = 2. If the interrupt routine is for one nested interrupt, use register bank 0 with RBE = 0, so that you do not have to save or restore the registers. When two or more interrupts are nested, set RBE to 1, save the register bank by using the PUSH BS instruction, and set RBS to 1 to select register bank 1.

(3) Use of a software interrupt for debugging

Setting an interrupt request flag using an instruction has the same effect as the occurrence of an interrupt. Debug operation for irregular interrupts or concurrently occurring interrupts can be performed more efficiently by setting the interrupt request flags using an instruction.

6.9 INTERRUPT APPLICATIONS

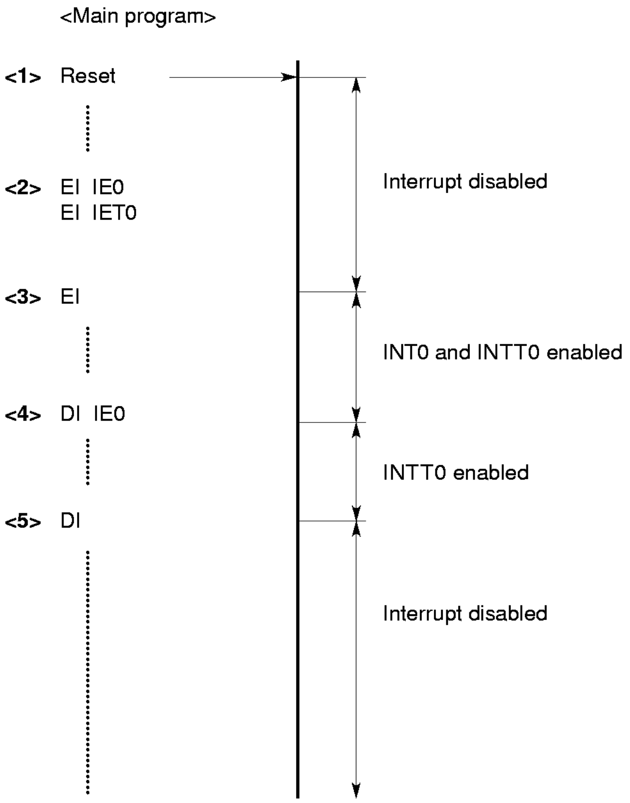
To use the interrupt function, a main program must:

(a) Set a desired interrupt enable flag (using the EI IExxx instruction)
(b) Select an active edge when INT0 or INT1 is used (set IM0 or IM1)
(c) To use nesting (of an interrupt with the higher priority), set IPS (IME can be set at the same time).
(d) Set the interrupt master enable flag (IME) using the EI instruction

In the interrupt routine, MBE and RBE are set by the vector table. However, when the interrupt specified as having the higher priority is processed, the register bank must be saved and set.

To return from the interrupt routine, use the RETI instruction.

(1) Interrupt enable/disable



<1> A $\overline{\text{RESET}}$ signal disables all interrupts.

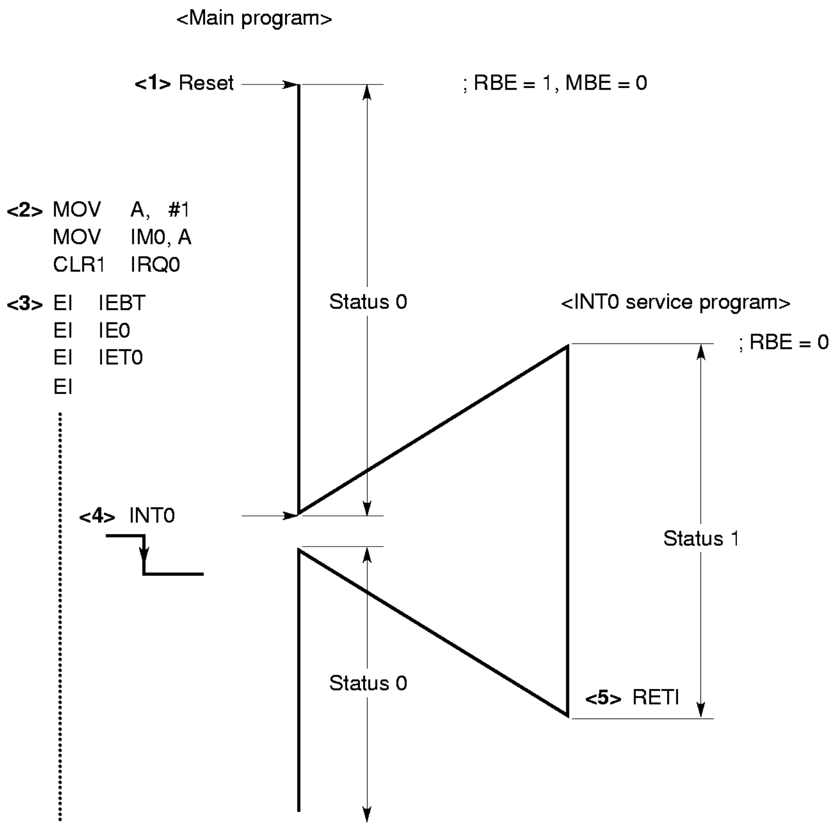
<2> Interrupt enable flags are set by the EI IExxx instruction. At this stage, all interrupts are disabled.

<3> The interrupt master enable flag is set by the EI instruction. At this stage, INT0 and INTT0 are enabled.

<4> An interrupt enable flag is cleared by the DI IExxx instruction to disable INT0.

<5> The DI instruction disables all interrupts.

(2) Example of using INTBT, INT0 (falling edge active), and INTT0 without multiple interrupt processing



<1> A $\overline{\text{RESET}}$ signal disables all interrupts, setting status 0.

<2> INT0 is set to be falling edge active.

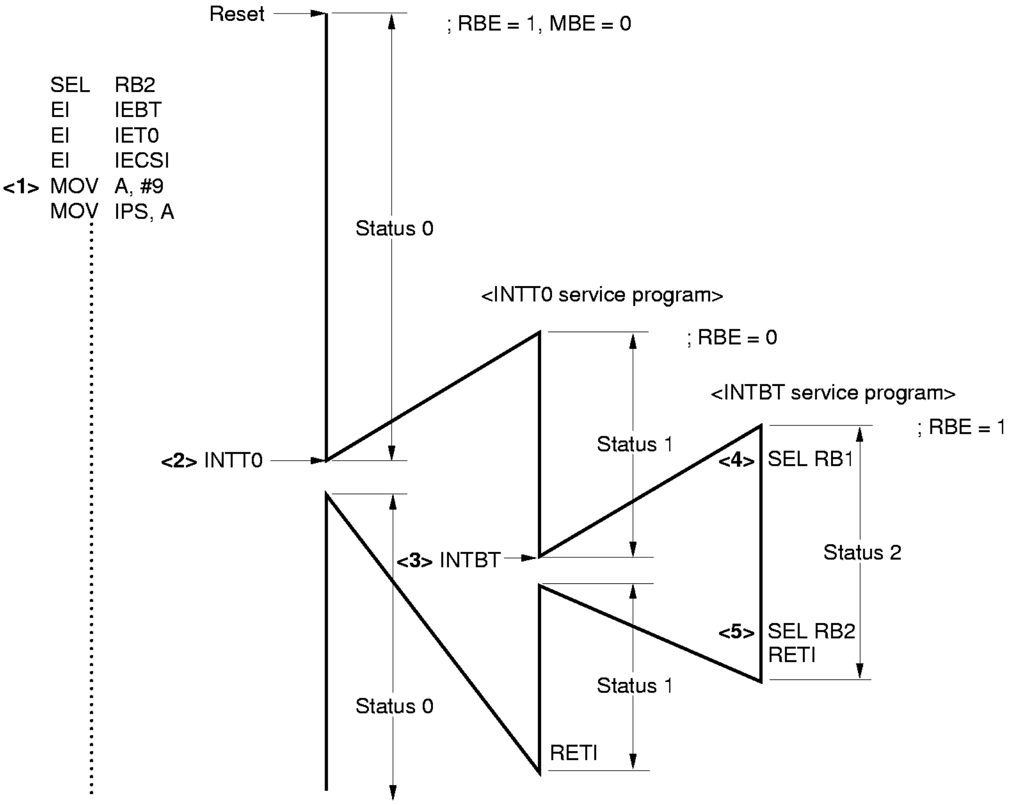
<3> Interrupts are enabled by the EI and EI IExxx instructions.

<4> On the falling edge of INT0, the INT0 interrupt service program is started, status is set to 1, and all interrupts are disabled.

<5> Control is returned from the interrupts by the RETI instruction, status 0 is set again, and interrupts are enabled.

Remark If all the interrupts are used as having the lower priority as shown in this example, saving or restoring the register bank is not necessary if RBE = 1 and RBS = 2 for the main program and register banks 2 and 3 are used, and RBE = 0 for the interrupt service program and register banks 0 and 1 are used.

(3) Nesting of interrupts with higher priority (INTBT has higher priority and INTT0 and INTCSI have lower priority)



<1> INTBT is specified as having the higher priority by setting of IPS, and the interrupt is enabled at the same time.

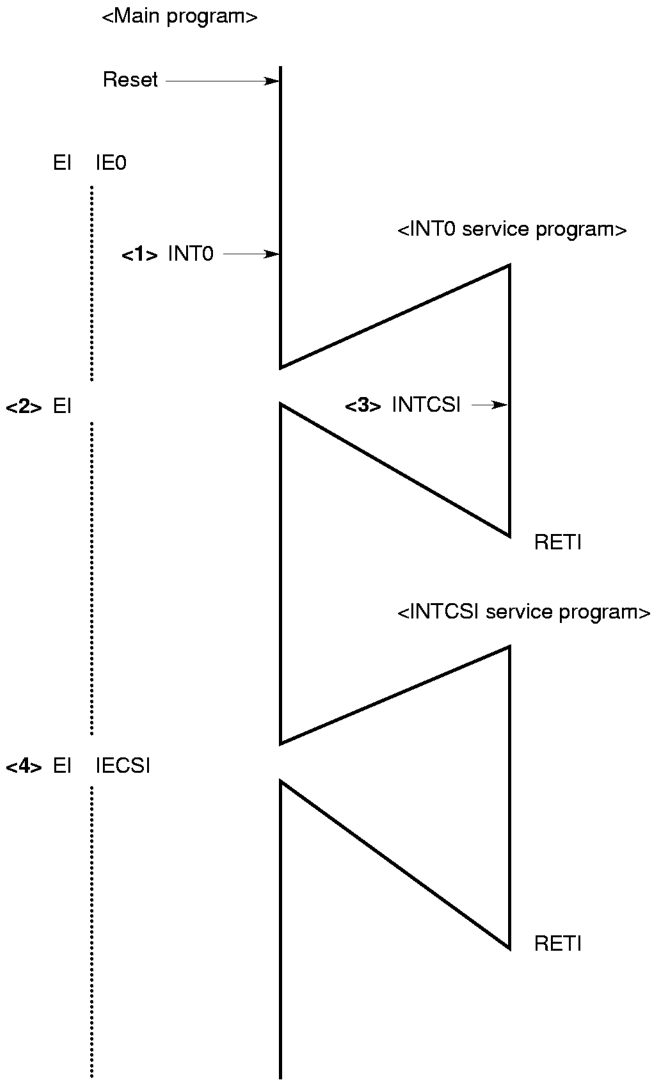
<2> INTT0 service program is started when INTT0 with the lower priority occurs. Status 1 is set and the other interrupts with the lower priority are disabled. RBE = 0 to select register bank 0.

<3> INTBT with the higher priority occurs. The level-two interrupts occurs. The status is changed to 0 and all the interrupts are disabled.

<4> RBE = 1 and RBS = 1 to select register bank 1 (only the registers used may be saved by the PUSH instruction).

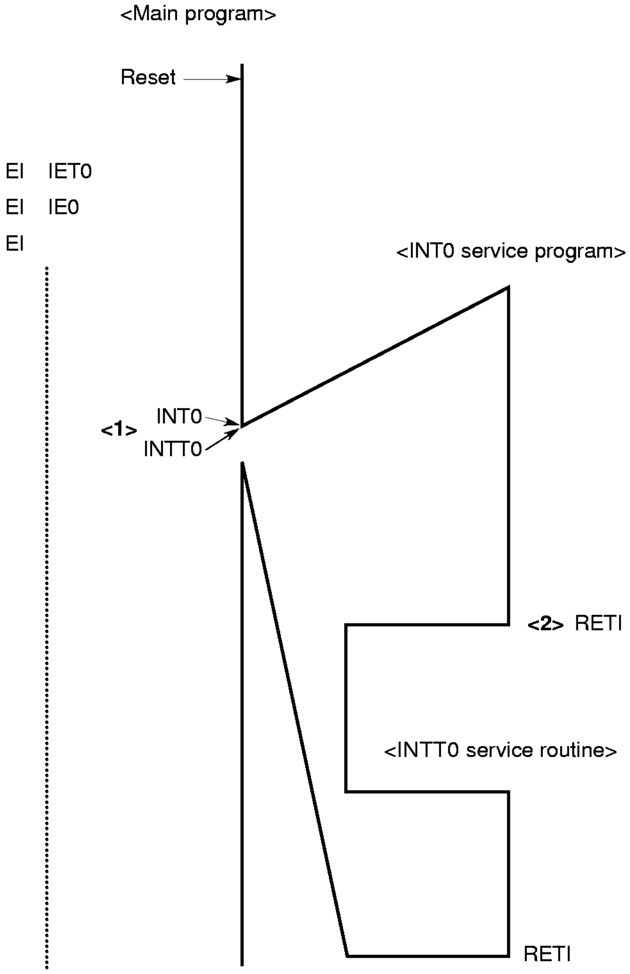
<5> RBS is returned to 2, and execution returns to the main program. The status is returned to 1.

(4) Execution of held interrupts (interrupt requests when interrupts are disabled)



<1> If INT0 is set when interrupts are disabled, the interrupt request flag is held.
<2> When the interrupt is enabled by the EI instruction, the INT0 interrupt service program starts.
<3> Same as <1>
<4> When the held INTCSI flag is enabled, the INTCSI interrupt service program starts.

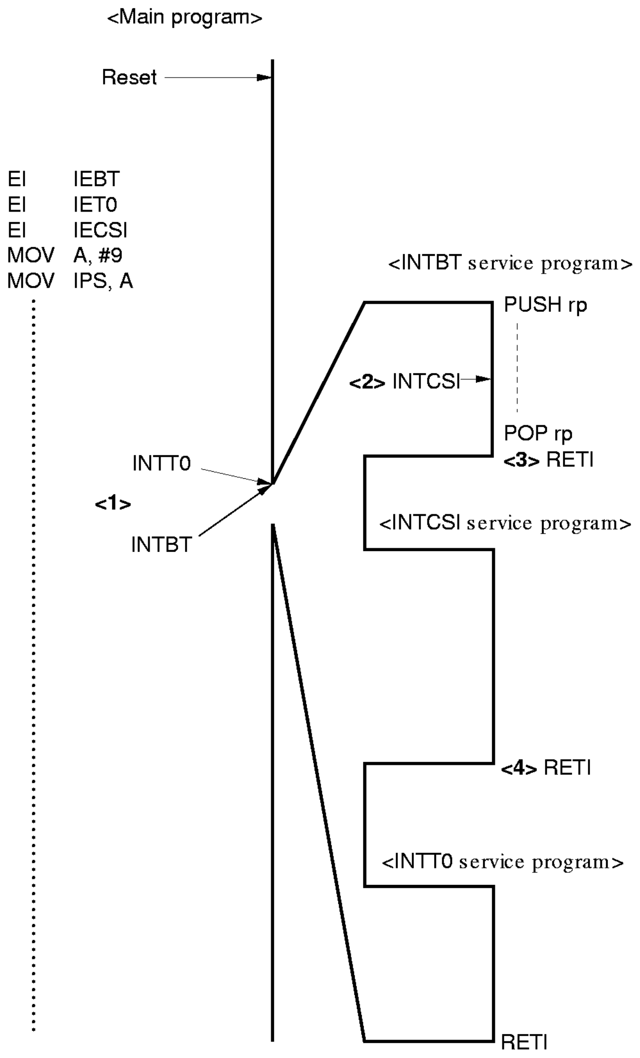
(5) Execution of held interrupts – two interrupts with lower priority occur concurrently –



<1> When INT0 and INTT0 with the lower priority occur concurrently (during execution of the same instruction), INT0, with a higher priority, is executed first. (INTT0 is held.)

<2> When the INT0 interrupt service program has been executed, the RETI instruction is executed to start the interrupt service program for INTT0, which has been held.

(6) Executing pending interrupt – interrupt occurs during interrupt processing (INTBT has higher priority and INTT0 and INTCSI have lower priority) –



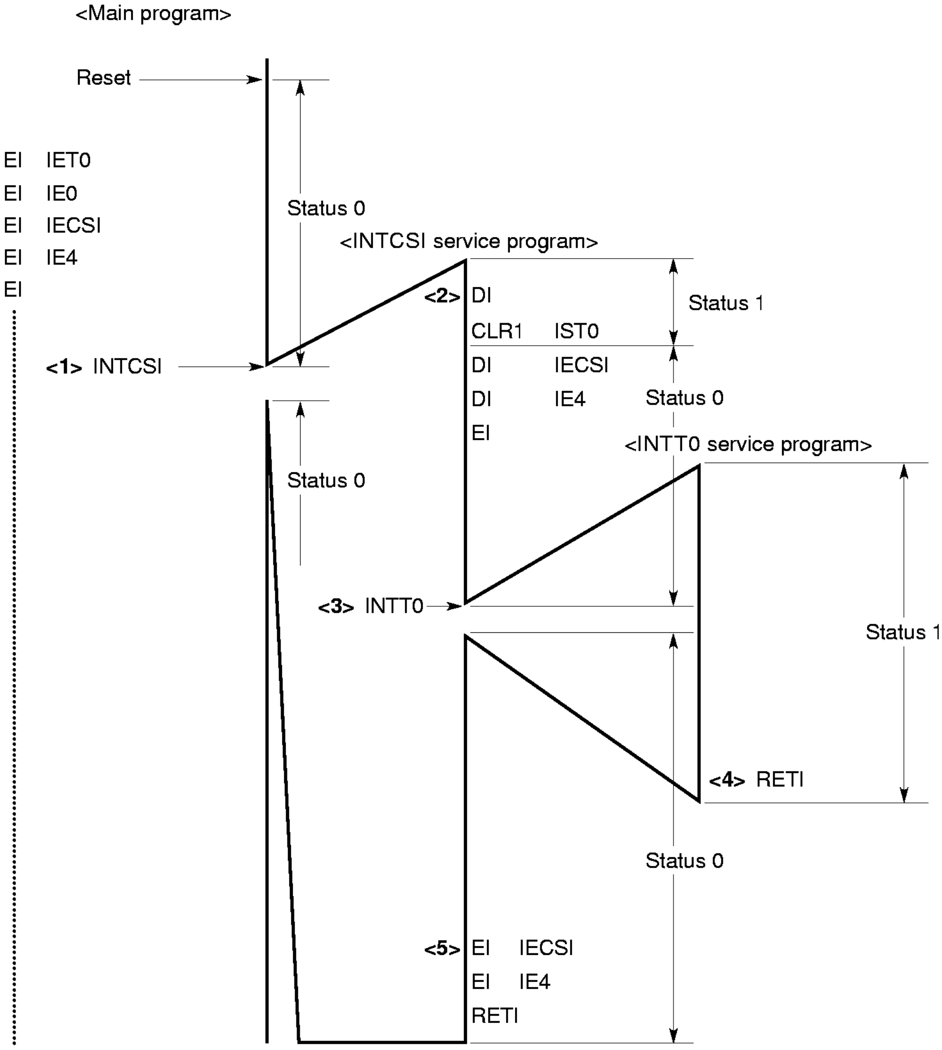
<1> When INTBT with the higher priority and INTT0 with the lower priority occur at the same time, the processing of the interrupt with the higher priority is started (if there is no possibility that an interrupt with the higher priority occurs while another interrupt with the higher priority is processed, DI IExx is not necessary).

<2> When an interrupt with the lower priority occurs while the interrupt with the higher priority is executed, the interrupt with the lower priority is kept pending.

<3> When the interrupt with the higher priority has been processed, INTCSI with the higher priority of the pending interrupts is executed.

<4> When the processing of INTCSI has been completed, the pending INTT0 is processed.

(7) Enabling of level-two interrupts (enabling level-two INTT0 and INT0 interrupts with INTCSI and INT4 handled as level-one interrupts)



<1> When an INTCSI interrupt not allowed to be a level-two interrupt occurs, the INTCSI service program starts, and status 1 is set.

<2> Status 0 is set by clearing IST0. INTCSI and INT4 not allowed to be level-two interrupts are disabled.

<3> When INTT0 allowed to be a level-two interrupt occurs, the level-two interrupt is executed, and status 1 is set to disable all interrupts.

<4> When INTT0 processing is completed, status 0 is set again.

<5> INTCSI and INT4 which have been disabled are enabled, then control returns.

6.10 TEST FUNCTION

6.10.1 Test Sources

The μPD750108 has two test sources. INT2 provides two types of edge-detection-test inputs.

Table 6-5. Test Source

Test source	Internal/external
INT2 (detection of the rising edge of the signal input to the INT2 pin or that of the first falling edge of the signals input to KR0 to KR7)	External
INTW (signal from clock timer)	Internal

6.10.2 Hardware to Control Test Functions

(1) Test request flags, test enable flags

Test request flags (IRQxxx) are set to 1 when the corresponding test requests (INTxxx) are issued. Clear the test request flags to 0 with the software once the test processing has been executed.

Test enable flags (IExxx) correspond to test request flags. The test enable flags enable the standby release signal when they are set to 1. They disables the standby release signal when they are set to 0. When both a test request flag and the corresponding test enable flag are set to 1, the standby release signal is generated.

Table 6-6 shows the signals which set test request flags.

Table 6-6. Signals Setting Test Request Flags

Test request flag	Signals setting test request flags	Test enable flag
IRQW	Signal from the clock timer.	IEW
IRQ2	Detection of the rising edge of INT2/P12 pin input signal or the first falling edge of the signals input to the KR0/P60 to KR7/P73 pins. The detection edge is selected with the INT2 edge detection mode register (IM2).	IE2

(2) INT2 and key interrupt (KR0 to KR7) hardware

Figure 6-10 shows the configuration of INT2 and KR0 to KR7.

The IRQ2 set signal is output in either of the following edge detection modes, which is selected with the INT2 edge detection mode register (IM2).

(a) Detection of a rising edge on the INT2 input pin

IRQ2 is set when a rising edge is detected on the INT2 input pin.

(b) Detection of a falling edge on any of the KR0 to KR7 input pins (key interrupt)

One of the pins KR0 to KR7 is selected to be used for interrupt input with the INT2 edge detection mode register (IM2). When a falling edge of one of input signals applied to the selected pin is detected, IRQ2 is set.

Figure 6-11 shows the format of IM2. IM2 is set using a 4-bit manipulation instruction. When the $\overline{\text{RESET}}$ signal is generated, all bits are cleared to 0, and the rising edge on INT2 is specified.

Figure 6-10. Block Diagram of the INT2 and KR0 to KR7 Circuits

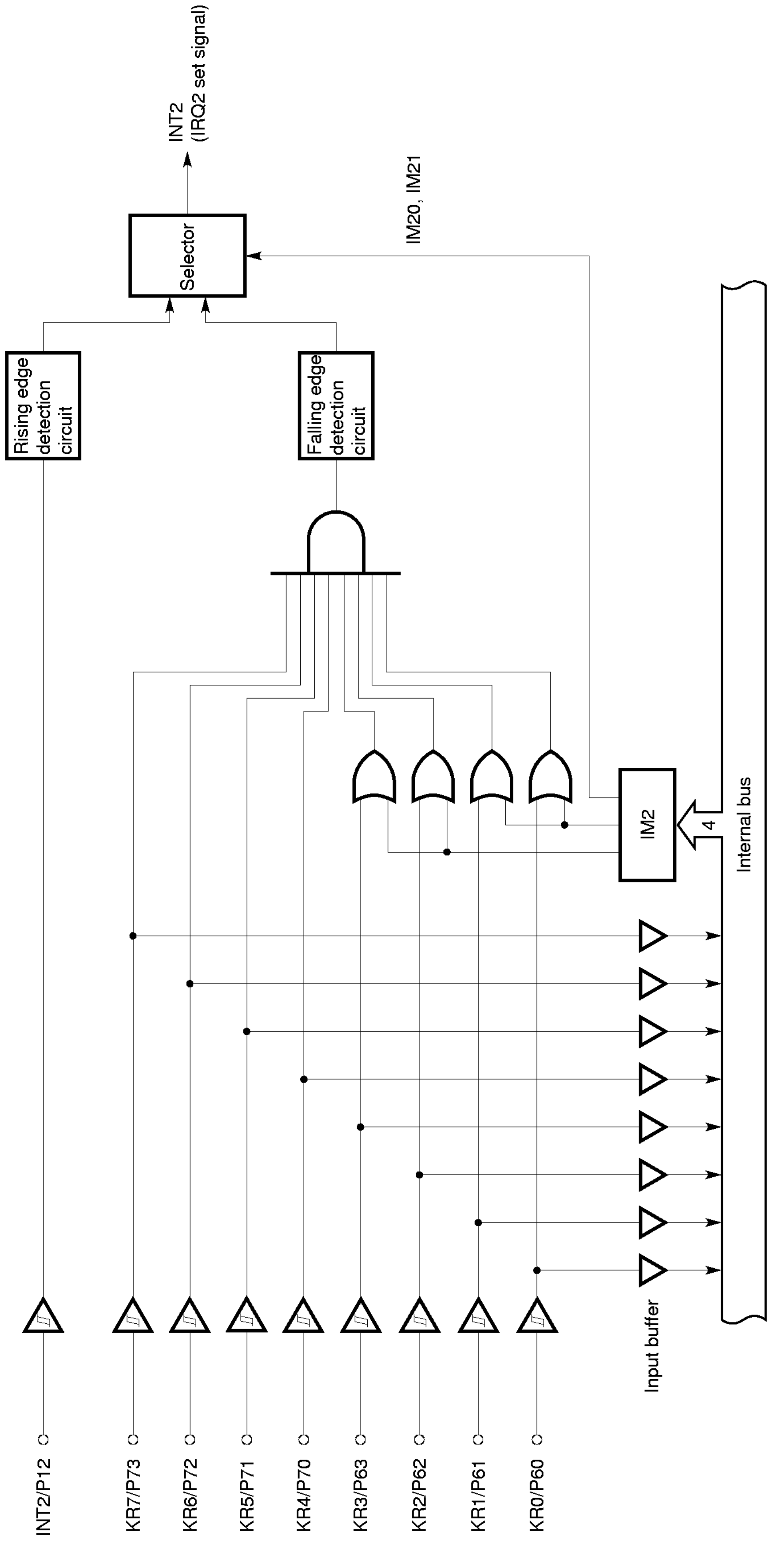
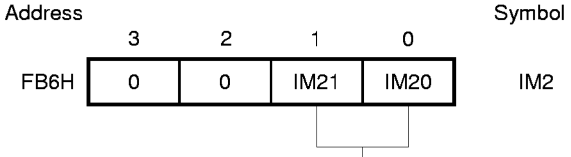


Figure 6-11. Format of INT2 Edge Detection Mode Register (IM2)



IM21	IM20	INT2 interrupt source	Interrupt input pin
0	0	Specifies rising edge of INT2 pin input.	INT2 (1)
0	1	Specifies falling edge of any of KRx pin inputs.	KR4 - KR7 (4)
1	0		KR2 - KR7 (6)
1	1		KR0 - KR7 (8)

Cautions 1. When the edge detection mode register is modified, test request flags may be set in some cases. So, disable test inputs before modifying the edge detection mode register. Then, clear the test request flags using a CLR1 instruction before enabling test inputs.

2. When a low-level signal is applied to any of the pins subjected to falling edge detection, IRQ2 is not set when a falling edge is detected on another pin.

[MEMO]

CHAPTER 7 STANDBY FUNCTION

The µPD750108 provides a standby function to reduce the power consumption by the system. The standby function is available in the two modes: the STOP mode and HALT mode.

Differences between these two modes are as follows:

(1) STOP mode

In the STOP mode, the main system clock oscillator is stopped, and the entire system stops. The current used by the CPU is reduced to quite a low level.

In addition, the contents of data memory can be preserved with a low supply voltage of down to V_{DD} = 1.8V, that is, this mode is effective to retain data memory with a very low current.

The wait time applied when STOP mode is released by an interrupt request can be specified as $2^9/f_{CC}$ or no wait, by using a mask option. To start processing immediately upon the detection of an interrupt request, select no wait. The µPD75P0116, however, does not have a mask option and its wait time is fixed to $2^9/f_{CC}$.

If $2^9/f_{CC}$ has been selected and processing must be started immediately upon the detection of an interrupt request, select HALT mode.

(2) HALT mode

In the HALT mode, the CPU clock is stopped, but the oscillation of the system clock oscillator continues. In this mode, the system uses more current than in the STOP mode. However, the HALT mode is suitable for starting processing immediately after an interrupt request or for intermittent operations such as watch operation.

In either mode, all contents of the registers, flags, and data memory that are present immediately before the standby mode is set are preserved. In addition, the states of the output latches of the I/O ports and the states of the output buffers are also preserved, so that the states of the I/O ports are to be processed to minimize the power consumption of the entire system.

Cautions 1. The STOP mode can be used only for the main system clock. (Subsystem clock generation cannot be terminated.) The HALT mode can be used for either the main system clock or the subsystem clock.

2. If the STOP mode is set when main system clock f_{CC} is used for clock timer operation, the clock stops operating. For continued operation, the clock must be changed to subsystem clock f_{XT} before the STOP mode is set.

3. A lower power consumption and lower-voltage operation are enabled by switching standby modes or switching CPU and system clocks. However, a switching time as described in Section 5.2.3 is required before operation is started with a new clock after the clock is selected with the control register. For this reason, when the clock switching function is used together with a standby mode, the standby mode must be set after a time needed for switching elapses.

4. Configure I/O ports for minimum power consumption in the stand by mode. Be sure to connect signals which are high or low to input ports.

7.1 SETTING OF STANDBY MODES AND OPERATION STATUS

Table 7-1. Operation Statuses in the Standby Mode

Item \ Mode		STOP mode	HALT mode
Instruction for setting		STOP instruction	HALT instruction
System clock for setting		Can be set only when operating on the main system clock	Can be set either with the main system clock or the subsystem clock
Operation status	Clock oscillator	Only the main system clock stops its operation	Only the CPU clock Φ stops its operation (oscillation continues)
	Basic interval timer/watchdog timer	Does not operate	Can operate only at main system clock oscillation. (IRQBT is set at reference time intervals.)
	Serial interface	Can operate only when the external $\overline{\text{SCK}}$ input is selected for the serial clock	Can operate only when external $\overline{\text{SCK}}$ input is selected as the serial clock or at main system clock oscillation.
	Timer/event counter	Can operate only when the TI0 pin input is selected for the count clock	Can operate only when TI0 pin input is specified as the count clock or at main system clock oscillation.
	Timer counter	Does not operate	Can operate**Note 1**
	Clock timer	Can operate when f_{XT} is selected as the count clock	Can operate
	External interrupt	INT1, INT2, and INT4 can operate. Only INT0 cannot operate.**Note 2**	
	CPU	Does not operate	
Release signal		An interrupt request signal from hardware whose operation is enabled by the interrupt enable flag or the generation of a $\overline{\text{RESET}}$ signal	

Notes 1. Operation is possible only when the main system clock operates.

2. Operation is possible only when the noise eliminator is not selected by bit 2 of the edge detection mode register (IM0) (when IM02 = 1).

A STOP instruction is used to set the STOP mode, and a HALT instruction is used to set the HALT mode. (A STOP instruction sets bit 3 of PCC, and a HALT instruction sets bit 2 of PCC.)

STOP instruction or HALT instruction must always be followed by an NOP instruction.

When changing a CPU operation clock pulse with the low-order two bits of PCC, a time lag may occur from the time when PCC is rewritten as shown in **Table 5-5** to the time when the CPU clock signal is changed. When changing an operation clock pulse before the standby mode or a CPU clock signal after the standby mode is released, it is necessary to rewrite PCC and set the standby mode after as many machine cycles as required to change the CPU clock pulse have elapsed.

In a standby mode, the contents of all registers and data memory that are stopped during the standby mode, including general registers, flags, mode registers, and output latches, are retained.

Caution Reset all the interrupt request flags before setting the standby mode. If an interrupt source whose interrupt request flag and interrupt enable flag are both set exists, the initiated standby mode is released immediately after it is set (see Figure 6-1). When the STOP mode is set, however, the μPD750108 enters the HALT mode immediately after the STOP instruction is executed, then returns to the operation mode after the specified wait time[Note] **has elapsed.**

Note Either of the following can be selected by using a mask option:

$2^9/f_{CC}$ (256 μs at 2 MHz, 512 μs at 1 MHz)

No wait

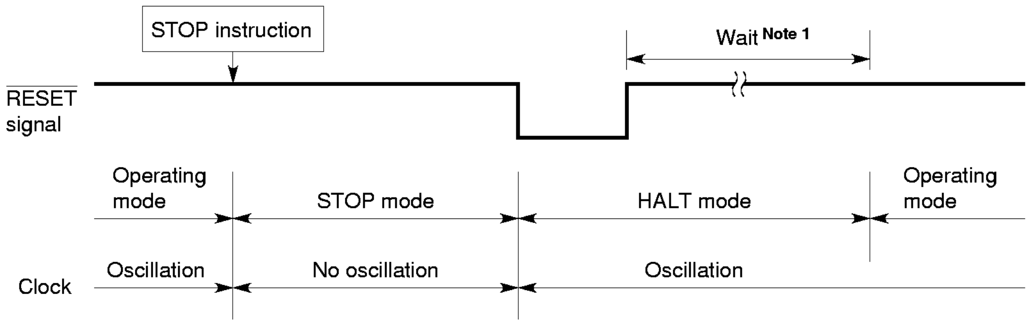
The μPD75P0116, however, does not have a mask option. Its wait time is fixed to $2^9/f_{CC}$.

7.2 RELEASE OF THE STANDBY MODES

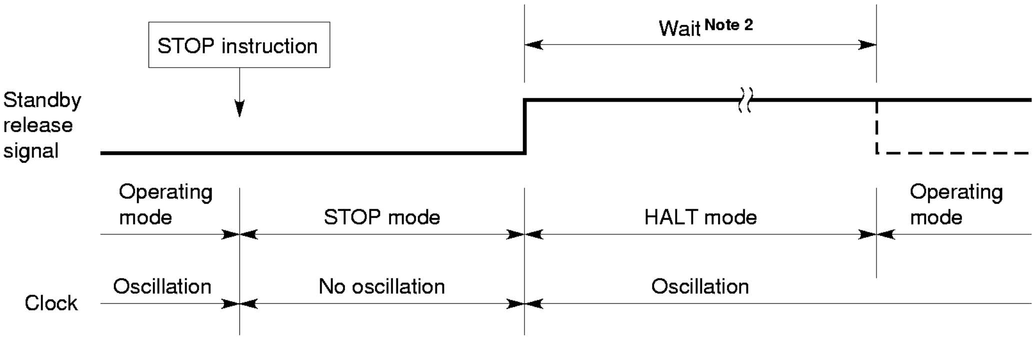
The STOP mode and HALT mode are released by a $\overline{\text{RESET}}$ signal or the generation of an interrupt request signal that is enabled with the interrupt enable flag. Figure 7-1 shows how the STOP and HALT modes are released.

Figure 7-1. Standby Mode Release Operation (1/2)

(a) Release of the STOP mode by $\overline{\text{RESET}}$ signal



(b) Release of the STOP mode by the occurrence of an interrupt



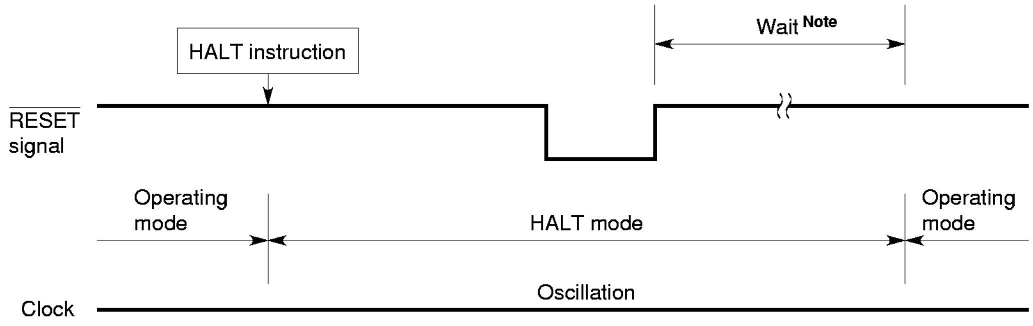
Notes 1. 56/f_{CC} (28 µs at 2 MHz, 56 µs at 1 MHz)

2. Either of the following can be selected by using a mask option:
$2^9/f_{CC}$ (256 µs at 2 MHz, 512 µs at 1 MHz)
No wait
The µPD75P0116, however, does not have a mask option. Its wait time is fixed to $2^9/f_{CC}$.

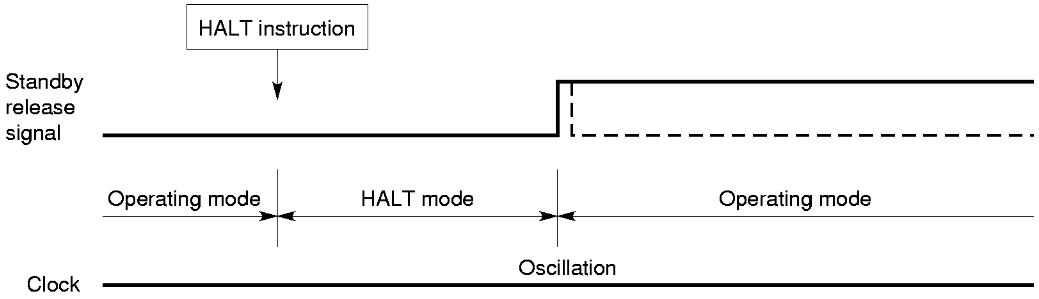
Remark The dashed line indicates the case where the interrupt request that releases the standby mode is accepted.

Figure 7-1. Standby Mode Release Operation (2/2)

(c) Release of the HALT mode by $\overline{\text{RESET}}$ signal



(d) Release of the HALT mode by the occurrence of an interrupt



Note 56/f_{CC} (28 µs at 2 MHz, 56 µs at 1 MHz)

Remark The dashed line indicates the case where the interrupt request that releases the standby mode is accepted.

7.3 OPERATION AFTER A STANDBY MODE IS RELEASED

(1) If a standby mode is released by a $\overline{\text{RESET}}$ signal, normal reset operation is performed.

(2) If a standby mode is released by the occurrence of an interrupt request, the contents of the interrupt master enable flag (IME) determines whether to perform a vectored interrupt when the CPU resumes instruction execution.

(a) When IME = 0

If a standby mode is released, execution restarts with the instruction immediately following the instruction used to set the standby mode.
The interrupt request flag is held.

(b) When IME = 1

If a standby mode is released, a vectored interrupt is executed after the two instructions are executed. However, if the standby mode is released by INT2 or INTW (testable input), no vectored interrupt occurs, and the same processing as (a) above is performed.

7.4 SELECTION OF A MASK OPTION

For the standby function of the μPD750108, the wait time applied when STOP mode is released by an interrupt can be set to either of the following by using a mask option:

<1> $2^9/f_{CC}$ (256 μs at 2 MHz, 512 μs at 1 MHz)
<2> No wait

The μPD75P0116, however, does not have a mask option. Its wait time is fixed to $2^9/f_{CC}$.

7.5 APPLICATIONS OF THE STANDBY MODES

When the standby modes are used, the following steps are used.

<1> Detect a standby mode setting factor such as power removal on an interrupt input or port input. (INT4 is useful for power removal detection.)

<2> Configure I/O ports for minimum current drain.

<3> Specify interrupts for releasing a standby mode. (INT4 is useful. All interrupt enable flags not used for release are to be cleared.)

<4> Specify an operation to be performed after release. (IME is to be manipulated according to whether interrupt processing is performed or not.)

<5> Specify a CPU clock to be used after release. (If the CPU clock is changed, required machine cycles must elapse before the standby mode is set.)

<6> Select a wait time to be used when a standby mode is released.

<7> Set a standby mode using a STOP or HALT instruction.

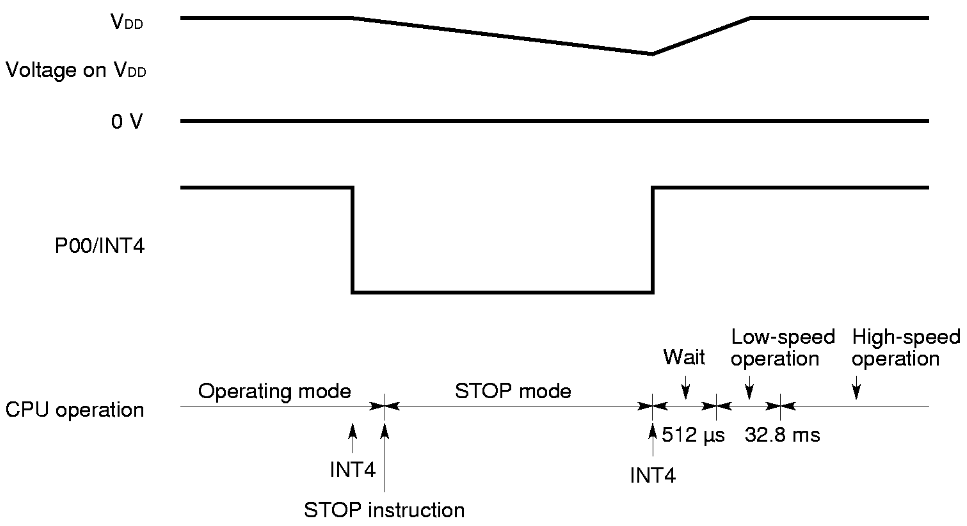
A standby mode when combined with the system clock switch function enables a lower power consumption and lower-voltage operation.

(1) Application of the STOP mode (at f_{CC} = 1 MHz)

<Use of the STOP mode under the following conditions>

- The STOP mode is set on the falling edge of INT4, and is released on the rising edge of INT4. (INTBT is not used.)
- All I/O ports have a high impedance.
- The INT0 and INTT0 interrupts are used for the program, but are not used to release the STOP mode.
- After the STOP mode is released, interrupts are enabled.
- After the STOP mode is released, operation is started using the lowest-speed CPU clock.
- The wait time applied when the STOP mode is released is set to 512 μs by using a mask option.
- After the STOP mode is released, another wait time of 32.8 ms is used for stable power supply operation. The P00/INT4 pin is checked twice to remove chattering.

<Timing chart>



<Sample program>

(INT4 service program, MBE = 0)

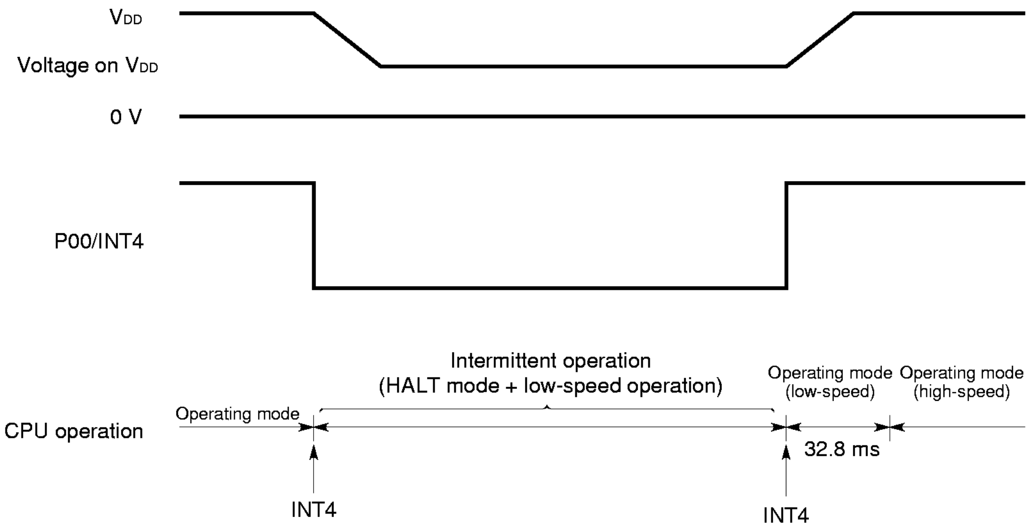
```
VSUB4:     SKT      PORT0.0       ; P00 = 1?
           BR       PDOWN         ; Power-down
           MOV      A,#1101B
           MOV      BTM,A         ; Wait time = 32.8 ms
WAIT:      SKT      IRQBT         ; Wait for 512 µs.
           BR       WAIT
           SKT      PORT0.0       ; Chattering check
           BR       PDOWN
           MOV      A,#0011B
           MOV      PCC,A         ; Set high-speed mode.
         ┌ MOV      XA.#xxH    ┐  ; Set port mode register.
         └ MOV      PMGm,XA    ┘
           EI       IE0
           EI       IET0
           RETI
PDOWN:     MOV      A,#0          ; Lowest-speed mode
           MOV      PCC,A
           MOV      XA,#00H
           MOV      PMGA,XA       ; I/O port high impedance
           MOV      PMGB,XA
           DI       IE0           ; Disable INT0 and INTT0
           DI       IET0
           STOP                   ; Set STOP mode.
           NOP
           RETI
```

(2) Application of the HALT mode (at f_{CC} = 1 MHz)

<Intermittent operation under the following conditions>

- The main system clock is switched to the subsystem clock on the falling edge of INT4.
- The oscillation of the main system clock is stopped, and HALT mode is set.
- In the standby mode, intermittent operation is performed at intervals of 0.5 s.
- The subsystem clock is switched back to the main system clock on the rising edge of INT4.
- INTBT is not used.
- After the STOP mode is released, another wait time of 32.8 ms is used for stable power supply operation. The P00/INT4 pin is checked twice to remove chattering.

<Timing chart>



<Sample program>

```
(Initialization)
                MOV       A,#0011B
                MOV       PCC,A          ; High-speed mode
                MOV       XA,#05
                MOV       WM,XA          ; Subsystem clock
                EI        IE4
                EI        IEW
                EI                       ; Enable interrupt
(Main routine)
                SKT       PORT0.0        ; Power normal?
                HALT                     ; Power-down mode
                NOP                      ; Power normal?
                SKTCLR    IRQW           ; Flag set for 0.5 second?
                BR        MAIN           ; NO
                CALL      WATCH          ; Clock subroutine
MAIN:           .
                .
                .
                .
                .
                .
                .
                .
                .
(INT4 service routine)
VINT4:          SKT       PORT0.0        ; Power normal?  MBE = 0
                BR        PDOWN
                CLR1      SCC.3          ; Start main system clock oscillation
                MOV       A,#0DH
                MOV       BTM,A
WAIT1:          SKT       IRQBT          ; Wait for 32.8 ms
                BR        WAIT1
                SKT       PORT0.0        ; Chattering check
                BR        PDOWN
                CLR1      SCC.0          ; Switch to main system clock
                RETI
PDOWN:          SET1      SCC.0          ; Switch to subsystem clock
                MOV       A,#0AH
WAIT2:          INCS      A              ; Wait for 15 machine cycles
                BR        WAIT2
                SET1      SCC.3          ; Stop main system clock oscillation
                RETI
```

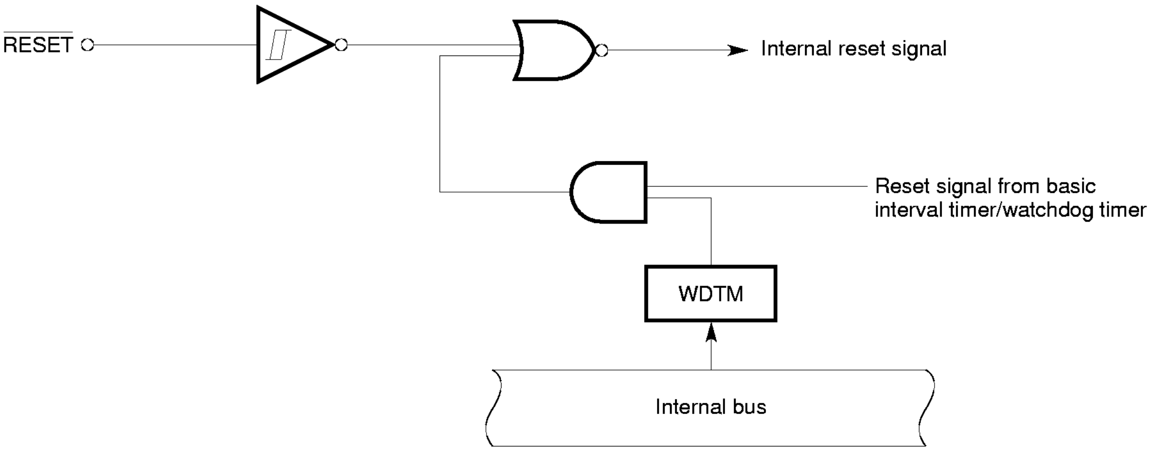
Caution **Before the system clock is changed from the main system clock to the subsystem clock, a wait time sufficient for stable subsystem clock generation is required.**

[MEMO]

CHAPTER 8 RESET FUNCTION

The μPD750108 is reset with the external reset signal ($\overline{\text{RESET}}$) or the reset signal received from the basic interval timer/watchdog timer. When either reset signal is input, the internal reset signal is generated. Figure 8-1 shows the configuration of the reset circuit.

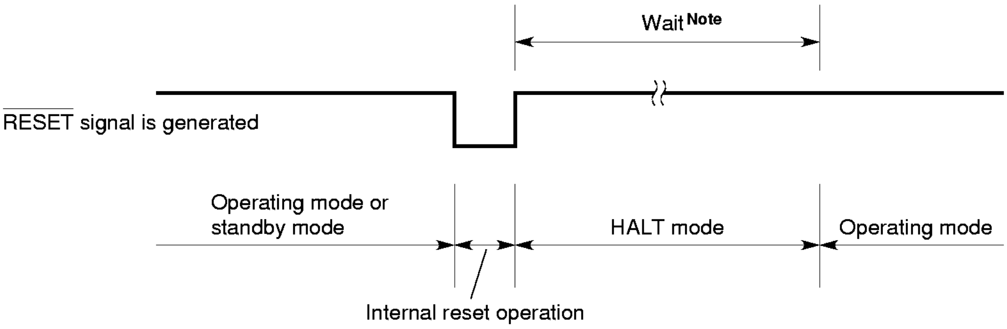
Figure 8-1. Configuration of Reset Functions



$\overline{\text{RESET}}$
Internal reset signal
Reset signal from basic interval timer/watchdog timer
WDTM
Internal bus

When the $\overline{\text{RESET}}$ signal is generated, all hardware is initialized as indicated in Table 8-1. Figure 8-2 shows the reset operation timing.

Figure 8-2. Reset Operation by Generation of $\overline{\text{RESET}}$ Signal



Wait Note
$\overline{\text{RESET}}$ signal is generated
Operating mode or standby mode
HALT mode
Operating mode
Internal reset operation

Note 56/f_{CC} (28 μs at 2 MHz, 56 μs at 1 MHz).

Table 8-1. Status of the Hardware after a Reset (1/2)

Hardware		Generation of a $\overline{\text{RESET}}$ signal in a standby mode	Generation of a $\overline{\text{RESET}}$ signal during operation
Program counter (PC)	μPD750104	4 low-order bits at address 0000H in program memory are set in PC bits 11 to 8, and the data at address 0001H are set in PC bits 7 to 0.	4 low-order bits at address 0000H in program memory are set in PC bits 11 to 8, and the data at address 0001H are set in PC bits 7 to 0.
	μPD750106, μPD750108	5 low-order bits at address 0000H in program memory are set in PC bits 12 to 8, and the data at address 0001H are set in PC bits 7 to 0.	5 low-order bits at address 0000H in program memory are set in PC bits 12 to 8, and the data at address 0001H are set in PC bits 7 to 0.
	μPD75P0116	6 low-order bits at address 0000H in program memory are set in PC bits 13 to 8, and the data at address 0001H are set in PC bits 7 to 0.	6 low-order bits at address 0000H in program memory are set in PC bits 13 to 8, and the data at address 0001H are set in PC bits 7 to 0.
PSW	Carry flag (CY)	Held	Undefined
	Skip flags (SK0 to SK2)	0	0
	Interrupt status flags (IST0, IST1)	0	0
	Bank enable flags (MBE, RBE)	Bit 6 at address 0000H in program memory is set in RBE, and bit 7 is set in MBE.	Bit 6 at address 0000H in program memory is set in RBE, and bit 7 is set in MBE.
Stack pointer (SP)		Undefined	Undefined
Stack bank selection register (SBS)		1000B	1000B
Data memory (RAM)		Held[Note]	Undefined
General registers (X, A, H, L, D, E, B, C)		Held	Undefined
Bank selection register (MBS, RBS)		0, 0	0, 0
Basic interval timer/watch-dog timer	Counter (BT)	00H	00H
	Mode register (BTM)	0	0
	Watchdog timer enable flag (WDTM)	0	0
Timer/ event counter	Counter (T0)	0	0
	Modulo register (TMOD0)	FFH	FFH
	Mode register (TM0)	0	0
	TOE0, TOUT flip-flop	0, 0	0, 0
Timer counter	Counter (T1)	0	0
	Modulo registers (TMOD1)	FFH	FFH
	Mode register (TM1)	0	0
	TOE1, TOUT flip-flop	0, 0	0, 0
Clock timer	Mode register (WM)	0	0
Serial interface	Shift register (SIO)	Held	Undefined
	Operation mode register (CSIM)	0	0
	SBI control register (SBIC)	0	0
	Slave address register (SVA)	Held	Undefined

Note Data of address 0F8H to 0FDH of the data memory becomes undefined when the $\overline{\text{RESET}}$ signal is generated.

Table 8-1. Statuses of the Hardware after a Reset (2/2)

Hardware		Generation of a $\overline{\text{RESET}}$ signal in a standby mode	Generation of a $\overline{\text{RESET}}$ signal during operation
Clock generator, clock output circuit	Processor clock control register (PCC)	0	0
	System clock control register (SCC)	0	0
	Clock output mode register (CLOM)	0	0
Sub-oscillator control register (SOS)		0	0
Interrupt	Interrupt request flag (IRQxxx)	Reset (0)	Reset (0)
	Interrupt enable flag (IExxx)	0	0
	Priority selection register (IPS)	0	0
	INT0, INT1, and INT2 mode registers (IM0, IM1, IM2)	0, 0, 0	0, 0, 0
Digital ports	Output buffer	Off	Off
	Output latch	Clear (0)	Clear (0)
	I/O mode registers (PMGA, PMGB, PMGC)	0	0
	Pull-up resistor specification register (POGA, POGB)	0	0
Bit sequential buffers (BSB0 to BSB3)		Held	Undefined

[MEMO]

CHAPTER 9 WRITING TO AND VERIFYING PROGRAM MEMORY (PROM)

The program memory in the µPD75P0116 consists of a one-time PROM (16384 x 8 bits).

Writing to and verifying the contents of the one-time PROM is accomplished by using the pins shown in the table below. Note that address inputs are not used; instead, the address is updated using the clock input from the CL1 pin.

Pin name	Function
V_{PP}	Voltage is applied to this pin when writing to the program memory or verifying its contents (normally V_{DD} electric potential).
CL1, CL2	An address update clock, used when writing to program memory or verifying its contents, is input to the CL1 pin. Leave the CL2 pin open.
MD0 to MD3	Operation mode selection pins used when writing to the program memory or verifying its contents.
P40 to P43 (low-order four bits) P50 to P53 (high-order four bits)	I/O pins for 8-bit data used when writing to the program memory or verifying its contents.
V_{DD}	Power voltage is applied to this pin. During normal operation, 1.8 to 5.5 V should be applied; +6 V should be applied when writing to the program memory or verifying its contents.

Cautions 1. The µPD75P0116CU/GB does not have an erasure window, so the erasing with ultraviolet radiation cannot be performed.

2. Handle the pins not used for writing to or verifying the program memory, as follows:

- **Pins other than XT2: Connect these pins to V_{SS} through pull-down resistors.**
- **XT2 pin: Open**

9.1 OPERATING MODES WHEN WRITING TO AND VERIFYING THE PROGRAM MEMORY

If +6 V is applied to the V_{DD} pin and +12.5 V is applied to the V_{PP} pin, the μPD75P0116 enters program memory write/verify mode. The specific operating mode is then selected by the setting of the MD0 through MD3 pins as listed in the table below.

Operating mode specification						Operating mode
V_{PP}	V_{DD}	MD0	MD1	MD2	MD3	
+12.5 V	+6 V	H	L	H	L	Program memory address clear mode
		L	H	H	H	Write mode
		L	L	H	H	Verify mode
		H	X	H	H	Program inhibit mode

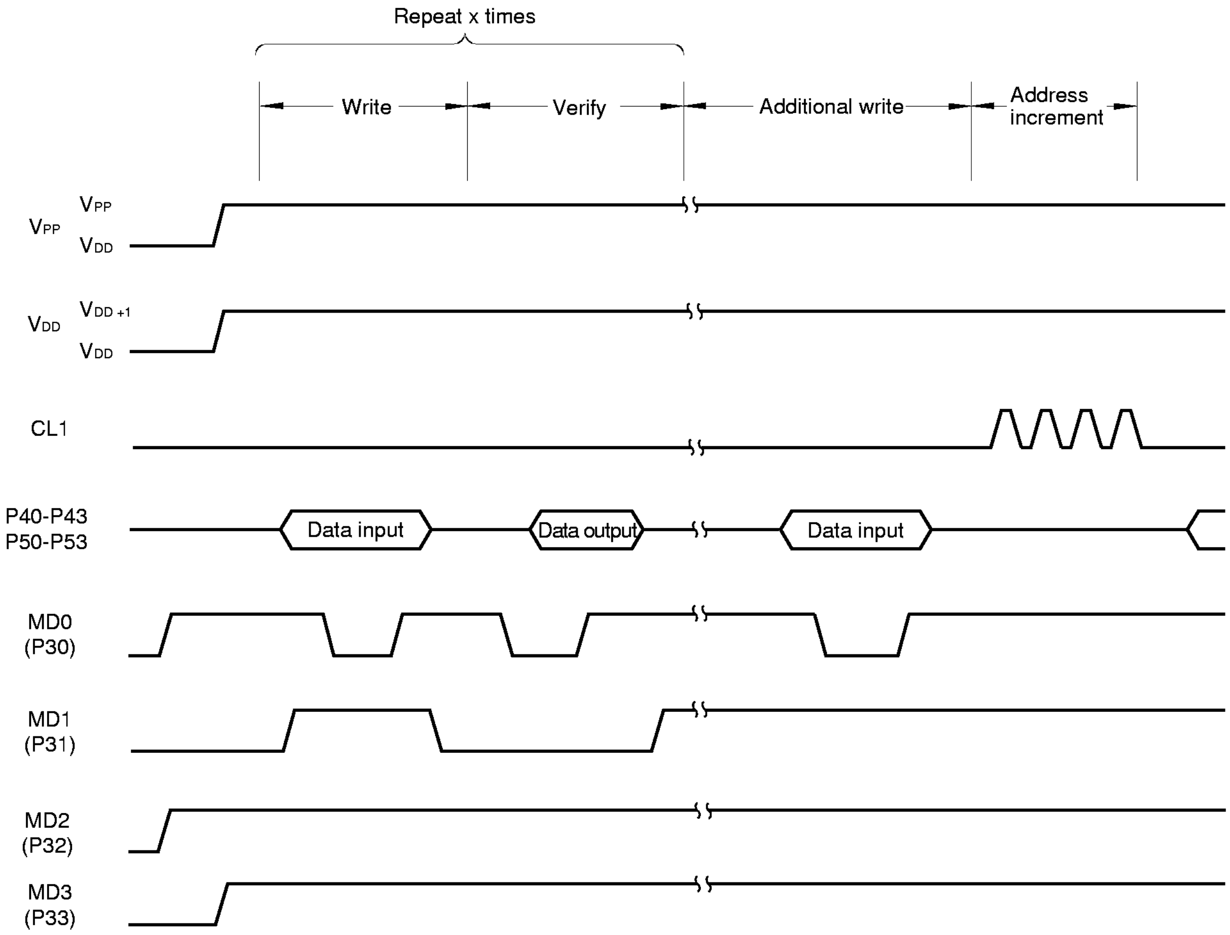
Remark X indicates L or H.

9.2 WRITING TO THE PROGRAM MEMORY

The procedure for writing to program memory is described below; high-speed write is possible.

(1) Pull low all unused pins to V_{SS} by means of resistors.
Bring CL1 to low level.
(2) Apply 5 V to V_{DD} and to V_{PP}.
(3) Wait 10 μs.
(4) Select program memory address clear mode.
(5) Apply 6 V to V_{DD} and 12.5 V to V_{PP}.
(6) Select program inhibit mode.
(7) Select write mode for 1 ms duration and write data.
(8) Select program inhibit mode.
(9) Select verify mode. If write is successful, proceed to step (10). If write fails, repeat steps (7) to (9).
(10) Perform additional write for (Number of repetitions of steps (7) to (9)) x 1 ms duration.
(11) Select program inhibit mode.
(12) Increment the program memory address by inputting four pulses on the CL1 pin.
(13) Repeat steps (7) to (12) until the last address is reached.
(14) Select program memory address clear mode.
(15) Apply 5 V to V_{DD} and to V_{PP}.
(16) Turn the power off.

The timing for steps (2) to (12) is shown below.

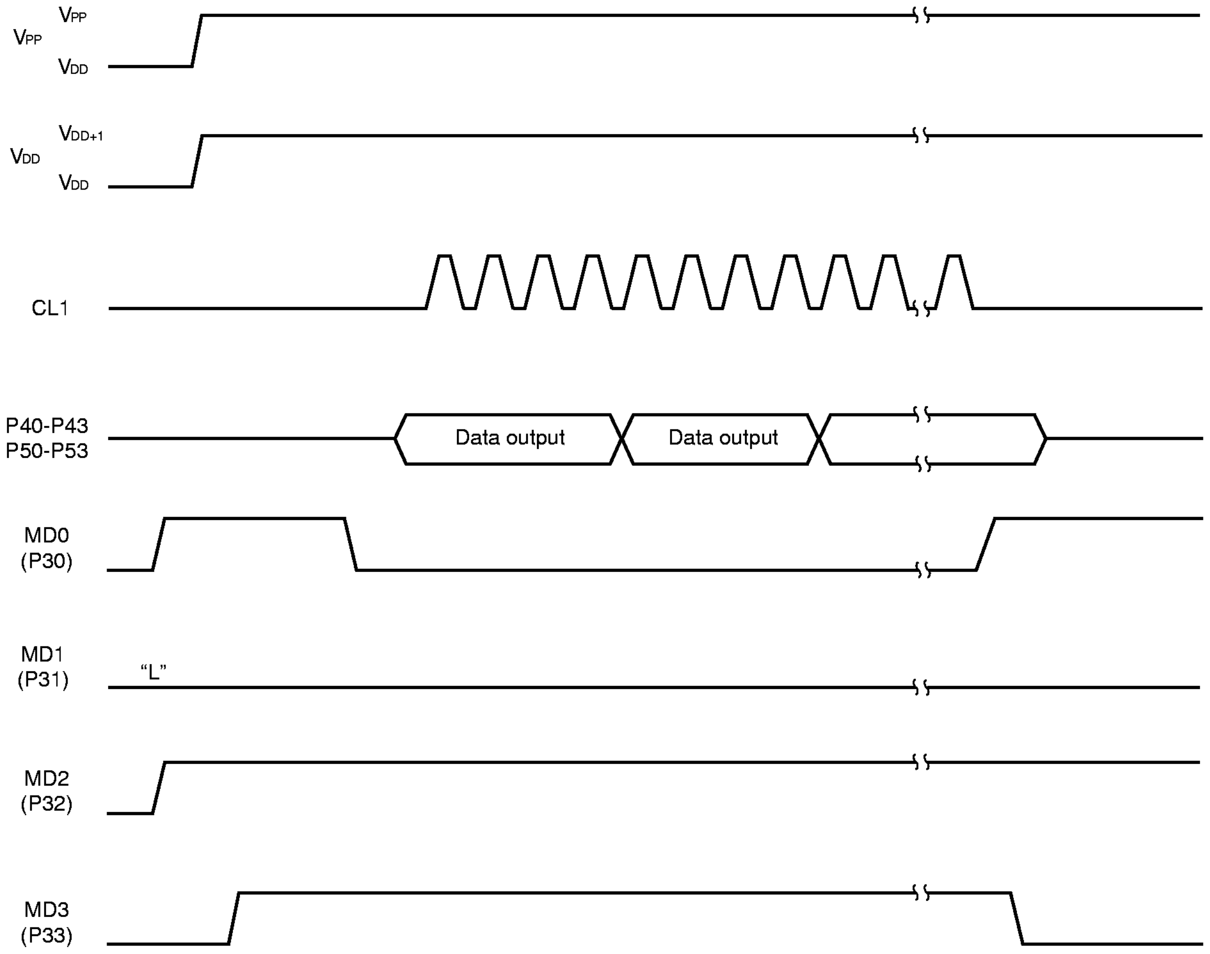


9.3 READING THE PROGRAM MEMORY

The procedure for reading the contents of program memory is described below. The read is performed in the verify mode.

(1) Pull low all unused pins to V_{SS} by means of resistors. Bring CL1 to low level.
(2) Apply 5 V to V_{DD} and V_{PP}.
(3) Wait 10 µs.
(4) Select program memory address clear mode.
(5) Apply 6 V to V_{DD} and 12.5 V to V_{PP}.
(6) Select program inhibit mode.
(7) Select verify mode. Data is output sequentially one address at a time for each cycle of four clock pulses appearing on the CL1 pin.
(8) Select program inhibit mode.
(9) Select program memory address clear mode.
(10) Apply 5 V to V_{DD} and to V_{PP}.
(11) Turn the power off.

The timing for steps (2) to (9) is shown below.



9.4 SCREENING OF ONE-TIME PROM

Because of its structure, it is difficult for NEC to completely test the one-time PROM product before shipment. It is therefore recommended that screening be performed to verify the PROM contents after the necessary data has been written to the PROM and the product has been stored under the following conditions.

Storage Temperature	Storage Time
125°C	24 hours

[MEMO]

CHAPTER 10 MASK OPTION

10.1 PIN

The pins of the µPD750108 have the following mask options:

Table 10-1. Selecting Mask Option of Pin

Pin	Mask Option
P40-P43	Pull-up resistor can be connected in 1-bit units.
P50-P53	

P40 through P43 (port 4) or P50 through P53 (port 5) can be connected with pull-up resistors by mask option. The mask option can be specified in 1-bit units.

If the pull-up resistor is connected by mask option, port 4 or 5 goes high on reset. If the pull-up resistor is not connected, the port goes into a high-impedance state on reset.

The ports of the µPD75P0116 do not have a mask option and is always open.

10.2 MASK OPTION OF STANDBY FUNCTION

The standby function of the µPD750108 allows you to select wait time by using a mask option. The wait time is required for the CPU to return to the normal operation mode after STOP mode has been released by an interrupt (for details, see **Section 7.2**).

The wait time can be set to either of the following:

<1> $2^9/f_{CC}$ (256 µs at 2 MHz, 512 µs at 1 MHz)
<2> No wait

The µPD75P0116 does not have a mask option and its wait time is fixed to $2^9/f_{CC}$.

10.3 MASK OPTION FOR FEEDBACK RESISTOR OF SUBSYSTEM CLOCK

For the subsystem clock of the μPD750108, whether to enable the feedback resistor is selected by the mask option.

<1> Enable the feedback resistor (switches on or off by software).
<2> Disable the feedback resistor (cuts by hardware).

To use the feedback resistor after selecting **<1>**, turn the feedback resistor on by setting SOS.0 to 0 (for details, see **(6)** in **Section 5.2.2**).

Select **<1>** to use the subsystem clock.

For the μPD75P0116, the mask option need not be set; use of the feedback resistor is factory-set.

CHAPTER 11 INSTRUCTION SET

The instruction set of the µPD750108 is an improved and extended version of the 75X series instruction set. This instruction set takes over the instruction set of the 75X series, having the following features:

(1) Bit manipulation instructions allowing a wide variety of applications
(2) Efficient 4-bit manipulation instructions
(3) Eight-bit instructions comparable to 8-bit microcomputers
(4) GETI instruction for reducing program sizes
(5) String-effect instructions and number system conversion instructions for increased program efficiency
(6) Table reference instructions suitable for successive references
(7) 1-byte relative branch instructions
(8) NEC standard mnemonics designed for clarity and readability

See **Section 3.2** for the addressing modes applicable to data memory manipulation and register banks used for instruction execution.

11.1 UNIQUE INSTRUCTIONS

This section outlines the unique instructions among the µPD750108 instruction set.

11.1.1 GETI Instruction

The GETI instruction converts any of the following instructions to a 1-byte instruction:

(a) Subroutine call instruction for the entire space
(b) Branch instruction for the entire space
(c) Arbitrary 2-byte instruction operating with two machine cycles (Except the BRCB and CALLF instructions)
(d) A combination of two 1-byte instructions

The GETI instruction references the table located at addresses 0020H to 007FH in program memory, and executes referenced 2-byte data as an instruction of (a), (b), (c), or (d) above. This means that 48 instructions consisting of (a) to (d) can be converted to 1-byte instructions.

Thus the GETI instruction can be used to convert frequently used instructions of (a) to (d) to 1-byte instructions to reduce the number of program bytes significantly.

11.1.2 Bit Manipulation Instructions

With the μPD750108, a variety of instructions are available for bit manipulation.

(a) Bit setting:	SET1	mem.bit
	SET1	mem.bit*
(b) Bit clearing:	CLR1	mem.bit
	CLR1	mem.bit*
(c) Bit testing:	SKT	mem.bit
	SKT	mem.bit*
(d) Bit testing:	SKF	mem.bit
	SKF	mem.bit*
(e) Bit testing and clearing:	SKTCLR	mem.bit*
(f) Boolean operation:	AND1	CY,mem.bit*
	OR1	CY,mem.bit*
	XOR1	CY,mem.bit*

mem.bit* represents a bit address addressed by using a bit manipulation addressing mode (fmem.bit, pmem.@L, or @H+mem.bit).

Particularly, all of these bit manipulation instructions can be used for the I/O ports, so that I/O port manipulation can be performed in a very efficient manner.

11.1.3 String-Effect Instructions

With the μPD750108, two types of string-effect instructions are available.

(a) MOV A,#n4 or MOV XA,#n8

(b) MOV HL,#n8

"String effect" means the locating of these two types of instructions at contiguous addresses.

Example

```
A0:  MOV A,#0
A1:  MOV A,#1
XA7: MOV XA,#07
```

When string-effect instructions are arranged as in this example, if execution starts at address A0, the following two instructions are replaced with an NOP instruction. If execution starts at address A1, the following one instruction is replaced with an NOP instruction. That is, only the instruction first executed is valid, and any following instructions are processed as an NOP instruction.

By using string-effect instructions, a constant can be set in an accumulator (the A register or the XA register pair) or data pointer (the HL register pair) more efficiently.

11.1.4 Number System Conversion Instructions

An application may need to convert the result of a 4-bit data addition or subtraction (performed in binary) to a decimal number. A time-related application may require sexagesimal conversion.

For this reason, the instruction set of the µPD750108 contains number system conversion instructions for converting the result of a 4-bit data addition or subtraction to a number in an arbitrary number system.

(a) Number system conversion for addition

Let m be a desired number system after conversion. The following combination of instructions adds the contents of an accumulator to data in memory (HL), then converts the result of the addition to number system m.

```
ADDS A,#16 – m
ADDC A,@HL   ; A, CY <– A + (HL) + CY
ADDS A,#m
```

An overflow is set in the carry flag.

If the execution of the instruction ADDC A,@HL generates a carry, the next instruction ADDS A,#n4 is skipped. If no carry is generated, ADDS A,#n4 is executed. In this case, the skip function of this instruction (ADDS A,#n4) is disabled, so that even if this addition generates a carry, the instruction following this instruction is not skipped. Accordingly, programs can be written after ADDS A,#n4.

Example An accumulator is added to memory data in decimal.

```
ADDS A,#6
ADDC A,@HL       ; A,CY <– A + (HL) + CY
ADDS A,#10
      :
```

(b) Number system conversion for subtraction

Let m be a desired number system after conversion. The following combination of instructions subtracts data in memory (HL) from the contents of an accumulator, then converts the result of the subtraction to number system m.

```
SUBC A,@HL
ADDS A,#m
```

An underflow is set in the carry flag.

If the execution of the instruction SUBC A,@HL generates no borrow, the next instruction ADDS A,#n4 is skipped. If a borrow is generated, the instruction ADDS A, #n4 is executed. In this case, the skip function of this instruction (ADDS A,#n4) is disabled, so that even if this addition generates a carry, the instruction following this instruction is not skipped. Accordingly, programs can be written after ADDS A,#n4.

11.1.5 Skip Instructions and the Number of Machine Cycles Required for a Skip

The instruction set of the μPD750108 is designed to organize a program by testing a condition with the skip function.

When a skip instruction satisfies the skip condition, the immediately following instruction is skipped to execute the instruction immediately after the skipped instruction.

A skip requires the following number of machine cycles:

(a) When the instruction (to be skipped) immediately following the skip instruction is a 3-byte instruction (that is, the BR !addr, BRA !addr1, CALL !addr, or CALLA !addr1 instruction): 2 machine cycles

(b) When the instruction (to be skipped) immediately following the skip instruction is an instruction other than the instructions described in (a) above: 1 machine cycle

11.2 INSTRUCTION SET AND OPERATION

(1) Operand identifier and description

The operand field of an instruction must contain an operand coded according to the description rule for the operand identifier of the instruction. (Refer to **RA75X Assembler Package User's Manual: Language (EEU-1343)** for detailed information.) When there are multiple descriptions for an identifier, one item is to be selected. The uppercase letters and + and – signs are keywords, which must be coded as they appear.

For immediate data, a proper numeric value or label must be coded.

The abbreviations for register flags shown in **Figure 3-7** can be coded as labels in place of mem, fmem, pmem, and bit. (However, not all labels can be coded for the fmem and pmem. For details, see **Table 3-1** and **Figure 3-7**)

Representation format	Description method
reg reg1	X, A, B, C, D, E, H, L X, B, C, D, E, H, L
rp rp1 rp2 rp' rp'1	XA, BC, DE, HL BC, DE, HL BC, DE XA, BC, DE, HL, XA', BC', DE', HL' BC, DE, HL, XA', BC', DE', HL'
rpa rpa1	HL, HL+, HL–, DE, DL DE,DL
n4 n8	4-bit immediate data or label 8-bit immediate data or label
mem bit	8-bit immediate data or label**Note** 2-bit immediate data or label
fmem pmem	FB0H-FBFH and FF0H-FFFH immediate data or label FC0H-FFFH immediate data or label
addr, addr1(for MkII mode only)	0000H-0FFFH immediate data or label (μPD750104) 0000H-17FFH immediate data or label (μPD750106) 0000H-1FFFH immediate data or label (μPD750108) 0000H-3FFFH immediate data or label (μPD75P0116)
caddr	12-bit immediate data or label
faddr	11-bit immediate data or label
taddr	20H-7FH immediate data (bit 0 = 0) or label
PORTn IExxx RBn MBn	PORT0-PORT8 IEBT, IET0, IET1, IE0-IE2, IE4, IECSI, IEW RB0-RB3 MB0, MB1, MB15

Note For mem, only even addresses can be coded for 8-bit data processing.

(2) Legend

A:	A register; 4-bit accumulator
B:	B register
C:	C register
D:	D register
E:	E register
H:	H register
L:	L register
X:	X register
XA:	Register pair (XA), 8-bit accumulator
BC:	Register pair (BC)
DE:	Register pair (DE)
HL:	Register pair (HL)
XA':	Extended register pair (XA')
BC':	Extended register pair (BC')
DE':	Extended register pair (DE')
HL':	Extended register pair (HL')
PC:	Program counter
SP:	Stack pointer
CY:	Carry flag, bit accumulator
PSW:	Program status word
MBE:	Memory bank enable flag
RBE:	Register bank enable flag
PORTn:	Port n (n = 0 to 8)
IME:	Interrupt master enable flag
IPS:	Interrupt priority specification register
IExxx:	Interrupt enable flag
RBS:	Register bank select register
MBS:	Memory bank select register
PCC:	Processor clock control register
.:	Address/bit delimiter
(xx):	Contents addressed by xx
xxH:	Hexadecimal data

(3) Explanation of symbols used for the addressing area column

*1	MB = MBE · MBS (MBS = 0, 1, 15)		Data memory addressing
*2	MB = 0		

*3	MBE = 0 : MB = 0 (00H - 7FH)
	MB = 15 (F80H - FFFH)
	MBE = 1 : MB = MBS (MBS =0, 1, 15)		

*4	MB = 15, fmem = FB0H - FBFH, FF0H - FFFH		
*5	MB = 15, pmem = FC0H - FFFH		
*6	μPD750104	addr, addr1 = 0000H - 0FFFH	Program memory addressing
	μPD750106	addr, addr1 = 0000H - 17FFH	
	μPD750108	addr, addr1 = 0000H - 1FFFH	
	μPD75P0116	addr, addr1 = 0000H - 3FFFH	
*7	addr , addr1 = (Current PC) – 15 to (Current PC) – 1 (Current PC) + 2 to (Current PC) + 16		
*8	μPD750104	caddr = 0000H - 0FFFH	
	μPD750106	caddr = 0000H - 0FFFH (PC_{12} = 0) or 1000H - 17FFH (PC_{12} = 1)	
	μPD750108	caddr = 0000H - 0FFFH (PC_{12} = 0) or 1000H - 1FFFH (PC_{12} = 1)	
	μPD75P0116	caddr = 0000H - 0FFFH (PC_{13}, PC_{12} = 00B) or 1000H - 1FFFH (PC_{13}, PC_{12} = 01B) or 2000H - 2FFFH (PC_{13}, PC_{12} = 10B) or 3000H - 3FFFH (PC_{13}, PC_{12} = 11B)	
*9	faddr = 0000H - 07FFH		
*10	taddr = 0020H - 007FH		
*11	For MkII mode only addr1 = 0000H - 0FFFH (μPD750104) 0000H - 17FFH (μPD750106) 0000H - 1FFFH (μPD750108) 0000H - 3FFFH (μPD75P0116)		

Remarks 1. MB represents an accessible memory bank.

2. For *2, MB = 0 regardless of the setting of MBE and MBS.

3. For *4 and *5, MB = 15 regardless of the setting of MBE and MBS.

4. Each of *6 to *10 indicates an addressable area.

(4) Explanation of the machine cycle column

S represents the number of machine cycles required when a skip instruction with the skip function performs a skip operation. S assumes one of the following values:

- When no skip operation is performed: S = 0
- When a 1-byte instruction or 2-byte instruction is skipped: S = 1
- When a 3-byte instruction[Note] is skipped: S = 2

Note 3-byte instruction: BR !addr, BRA !addr1, CALL !addr, and CALLA !addr1 instructions

Caution The GETI instruction is skipped in one machine cycle.

One machine cycle is equal to one cycle (t_{CY}) of the CPU clock (Φ), and four different machine cycles are available for selection according to the PCC setting. (See **Figure 5-12**.)

Instruction	Mnemonic	Operand	Number of bytes	Machine cycle	Operation	Addressing area	Skip condition
Transfer	**MOV**	A,#n4	1	1	A <– n4		String-effect A
		reg1,#n4	2	2	reg1 <– n4		
		XA,#n8	2	2	XA <– n8		String-effect A
		HL,#n8	2	2	HL <– n8		String-effect B
		rp2,#n8	2	2	rp2 <– n8		
		A,@HL	1	1	A <– (HL)	*1	
		A,@HL+	1	2+S	A <– (HL), then L <– L+1	*1	L=0
		A,@HL–	1	2+S	A <– (HL), then L <– L–1	*1	L=FH
		A,@rpa1	1	1	A <– (rpa1)	*2	
		XA,@HL	2	2	XA <– (HL)	*1	
		@HL,A	1	1	(HL) <– A	*1	
		@HL,XA	2	2	(HL) <– XA	*1	
		A,mem	2	2	A <– (mem)	*3	
		XA,mem	2	2	XA <– (mem)	*3	
		mem,A	2	2	(mem) <– A	*3	
		mem,XA	2	2	(mem) <– XA	*3	
		A,reg	2	2	A <– reg		
		XA,rp’	2	2	XA <– rp’		
		reg1,A	2	2	reg1 <– A		
		rp’1,XA	2	2	rp’1 <– XA		
	XCH	A,@HL	1	1	A <–> (HL)	*1	
		A,@HL+	1	2+S	A <–> (HL), then L <- L+1	*1	L=0
		A,@HL–	1	2+S	A <–> (HL), then L <- L–1	*1	L=FH
		A,@rpa1	1	1	A <–> (rpa1)	*2	
		XA,@HL	2	2	XA <–> (HL)	*1	
		A,mem	2	2	A <–> (mem)	*3	
		XA,mem	2	2	XA <–> (mem)	*3	
		A,reg1	1	1	A <–> reg1		
		XA,rp’	2	2	XA <–> rp’		

Instruction	Mnemonic	Operand	Number of bytes	Machine cycle	Operation	Addressing area	Skip condition
Table reference	**MOVT**	XA,@PCDE	1	3	• **µPD750104** XA <– $(PC_{11-8}+DE)_{ROM}$		
					• **µPD750106, µPD750108** XA <– $(PC_{12-8}+DE)_{ROM}$		
					• **µPD75P0116** XA <– $(PC_{13-8}+DE)_{ROM}$		
		XA,@PCXA	1	3	• **µPD750104** XA <– $(PC_{11-8}+XA)_{ROM}$		
					• **µPD750106, µPD750108** XA <– $(PC_{12-8}+XA)_{ROM}$		
					• **µPD75P0116** XA <– $(PC_{13-8}+XA)_{ROM}$		
		XA,@BCDE	1	3	XA <– $(BCDE)_{ROM}$**Note**	*6	
		XA,@BCXA	1	3	XA <– $(BCXA)_{ROM}$**Note**	*6	
Bit transfer	**MOV1**	CY,fmem.bit	2	2	CY <– (fmem.bit)	*4	
		CY,pmem.@L	2	2	CY <– $(pmem_{7-2}+L_{3-2}.bit(L_{1-0}))$	*5	
		CY,@H+mem.bit	2	2	CY <– $(H+mem_{3-0}.bit)$	*1	
		fmem.bit,CY	2	2	(fmem.bit) <– CY	*4	
		pmem.@L,CY	2	2	$(pmem_{7-2}+L_{3-2}.bit(L_{1-0}))$ <– CY	*5	
		@H+mem.bit,CY	2	2	$(H+mem_{3-0}.bit)$ <– CY	*1	
Arithmetic/logical	**ADDS**	A,#n4	1	1+S	A <– A+n4		carry
		XA,#n8	2	2+S	XA <– XA+n8		carry
		A,@HL	1	1+S	A <– A+(HL)	*1	carry
		XA,rp'	2	2+S	XA <– XA+rp'		carry
		rp'1,XA	2	2+S	rp'1 <– rp'1+XA		carry
	ADDC	A,@HL	1	1	A,CY <– A+(HL)+CY	*1	
		XA,rp'	2	2	XA,CY <– XA+rp'+CY		
		rp'1,XA	2	2	rp'1,CY <– rp'1+XA+CY		
	SUBS	A,@HL	1	1+S	A <– A–(HL)	*1	borrow
		XA,rp'	2	2+S	XA <– XA–rp'		borrow
		rp'1,XA	2	2+S	rp'1 <– rp'1–XA		borrow
	SUBC	A,@HL	1	1	A,CY <– A–(HL)–CY	*1	
		XA,rp'	2	2	XA,CY <– XA–rp'–CY		
		rp'1,XA	2	2	rp'1,CY <– rp'1–XA–CY		

Note Set register B to 0 in the µPD750104. Only the LSB is valid in register B in the µPD750106 and µPD750108. Only the low-order two bits are valid in the µPD75P0116.

Instruction	Mnemonic	Operand	Number of bytes	Machine cycle	Operation	Addressing area	Skip condition
Arithmetic/logical	**AND**	A,#n4	2	2	A <– A∧n4		
		A,@HL	1	1	A <– A∧(HL)	*1	
		XA,rp'	2	2	XA <– XA∧rp'		
		rp'1,XA	2	2	rp'1 <– rp'1∧XA		
	OR	A,#n4	2	2	A <– A∨n4		
		A,@HL	1	1	A <– A∨(HL)	*1	
		XA,rp'	2	2	XA <– XA∨rp'		
		rp'1,XA	2	2	rp'1 <- rp'1∨XA		
	XOR	A,#n4	2	2	A <– A∀n4		
		A,@HL	1	1	A <– A∀(HL)	*1	
		XA,rp'	2	2	XA <– XA∀rp'		
		rp'1,XA	2	2	rp'1 <– rp'1∀XA		
Accumulator manipulation	**RORC**	A	1	1	CY <– A_0, A_3 <– CY, A_{n-1} <– A_n		
	NOT	A	2	2	A <– $\overline{A}$		
Increment/decrement	**INCS**	reg	1	1+S	reg <– reg+1		reg=0
		rp1	1	1+S	rp1 <– rp1+1		rp1=00H
		@HL	2	2+S	(HL) <– (HL)+1	*1	(HL)=0
		mem	2	2+S	(mem) <– (mem)+1	*3	(mem)=0
	DECS	reg	1	1+S	reg <– reg–1		reg=FH
		rp'	2	2+S	rp' <– rp'–1		rp'=FFH
Comparison	**SKE**	reg,#n4	2	2+S	Skip if reg=n4		reg=n4
		@HL,#n4	2	2+S	Skip if (HL)=n4	*1	(HL)=n4
		A,@HL	1	1+S	Skip if A=(HL)	*1	A=(HL)
		XA,@HL	2	2+S	Skip if XA=(HL)	*1	XA=(HL)
		A,reg	2	2+S	Skip if A=reg		A=reg
		XA,rp'	2	2+S	Skip if XA=rp'		XA=rp'
Carry flag manipulation	**SET1**	CY	1	1	CY <– 1		
	CLR1	CY	1	1	CY <– 0		
	SKT	CY	1	1+S	Skip if CY=1		CY=1
	NOT1	CY	1	1	CY <– $\overline{CY}$		

In-struction	Mne-monic	Operand	Number of bytes	Machine cycle	Operation	Address-ing area	Skip condition
Memory bit manipulation	**SET1**	mem.bit	2	2	(mem.bit) <– 1	*3	
		fmem.bit	2	2	(fmem.bit) <– 1	*4	
		pmem.@L	2	2	(pmem$_{7-2}$+L$_{3-2}$.bit(L$_{1-0}$)) <– 1	*5	
		@H+mem.bit	2	2	(H+mem$_{3-0}$.bit) <– 1	*1	
	CLR1	mem.bit	2	2	(mem.bit) <– 0	*3	
		fmem.bit	2	2	(fmem.bit) <– 0	*4	
		pmem.@L	2	2	(pmem$_{7-2}$+L$_{3-2}$.bit(L$_{1-0}$)) <– 0	*5	
		@H+mem.bit	2	2	(H+mem$_{3-0}$.bit) <– 0	*1	
	SKT	mem.bit	2	2+S	Skip if (mem.bit)=1	*3	(mem.bit)=1
		fmem.bit	2	2+S	Skip if (fmem.bit)=1	*4	(fmem.bit)=1
		pmem.@L	2	2+S	Skip if (pmem$_{7-2}$+L$_{3-2}$.bit(L$_{1-0}$))=1	*5	(pmem.@L)=1
		@H+mem.bit	2	2+S	Skip if (H+mem$_{3-0}$.bit)=1	*1	(@H+mem.bit)=1
	SKF	mem.bit	2	2+S	Skip if (mem.bit)=0	*3	(mem.bit)=0
		fmem.bit	2	2+S	Skip if (fmem.bit)=0	*4	(fmem.bit)=0
		pmem.@L	2	2+S	Skip if (pmem$_{7-2}$+L$_{3-2}$.bit(L$_{1-0}$))=0	*5	(pmem.@L)=0
		@H+mem.bit	2	2+S	Skip if (H+mem$_{3-0}$.bit)=0	*1	(@H+mem.bit)=0
	SKTCLR	fmem.bit	2	2+S	Skip if (fmem.bit)=1 and clear	*4	(fmem.bit)=1
		pmem.@L	2	2+S	Skip if (pmem$_{7-2}$+L$_{3-2}$.bit(L$_{1-0}$)) =1 and clear	*5	(pmem.@L)=1
		@H+mem.bit	2	2+S	Skip if (H+mem$_{3-0}$.bit)=1 and clear	*1	(@H+mem.bit)=1
	AND1	CY,fmem.bit	2	2	CY <– CY∧ (fmem.bit)	*4	
		CY,pmem.@L	2	2	CY <– CY∧ (pmem$_{7-2}$+L$_{3-2}$.bit(L$_{1-0}$))	*5	
		CY,@H+mem.bit	2	2	CY <– CY∧ (H+mem$_{3-0}$.bit)	*1	
	OR1	CY,fmem.bit	2	2	CY <– CY∨ (fmem.bit)	*4	
		CY,pmem.@L	2	2	CY <– CY∨ (pmem$_{7-2}$+L$_{3-2}$.bit(L$_{1-0}$))	*5	
		CY,@H+mem.bit	2	2	CY <– CY∨ (H+mem$_{3-0}$.bit)	*1	
	XOR1	CY,fmem.bit	2	2	CY <– CY∀(fmem.bit)	*4	
		CY,pmem.@L	2	2	CY <– CY∀ (pmem$_{7-2}$+L$_{3-2}$.bit(L$_{1-0}$))	*5	
		CY,@H+mem.bit	2	2	CY <– CY∀ (H+mem$_{3-0}$.bit)	*1	

Instruction	Mnemonic	Operand	Number of bytes	Machine cycle	Operation	Addressing area	Skip condition
Branch	**BR**	addr	—	—	• **μPD750104** PC$_{11-0}$ <– addr The assembler selects the most adequate instruction from instructions below. • BR !addr • BR $addr • BRCB !caddr	*6	
					• **μPD750106**, **μPD750108** PC$_{12-0}$ <– addr The assembler selects the most adequate instruction from instructions below. • BR !addr • BRCB !caddr • BR $addr		
					• **μPD75P0116** PC$_{13-0}$ <– addr The assembler selects the most adequate instruction from instructions below. • BR !addr • BRCB !caddr • BR $addr		
		addr1 **Note**	—	—	• **μPD750104** PC$_{11-0}$ <– addr1 The assembler selects the most adequate instruction from instructions below. • BRA !addr1 • BR !addr • BRCB !caddr • BR $addr1	*11	
					• **μPD750106**, **μPD750108** PC$_{12-0}$ <– addr1 The assembler selects the most adequate instruction from instructions below. • BRA !addr1 • BR !addr • BRCB !caddr • BR $addr1		
					• **μPD75P0116** PC$_{13-0}$ <– addr1 The assembler selects the most adequate instruction from instructions below. • BRA !addr1 • BR !addr • BRCB !caddr • BR $addr1		

Note The shaded portion is supported in Mk II mode only.

Instruction	Mnemonic	Operand	Number of bytes	Machine cycle	Operation	Addressing area	Skip condition
Branch	**BR**	!addr	3	3	• **μPD750104** PC_{11-0} <– addr	*6	
					• **μPD750106**, **μPD750108** PC_{12-0} <– addr		
					• **μPD75P0116** PC_{13-0} <– addr		
		$addr	1	2	• **μPD750104** PC_{11-0} <– addr	*7	
					• **μPD750106**, **μPD750108** PC_{12-0} <– addr		
					• **μPD75P0116** PC_{13-0} <– addr		
		$addr1	1	2	• **μPD750104** PC_{11-0} <– addr1	*7	
					• **μPD750106, μPD750108** PC_{12-0} <– addr1		
					• **μPD75P0116** PC_{13-0} <– addr1		
		PCDE	2	3	• **μPD750104** PC_{11-0} <– PC_{11-8}+DE		
					• **μPD750106**, **μPD750108** PC_{12-0} <– PC_{12-8}+DE		
					• **μPD75P0116** PC_{13-0} <– PC_{13-8}+DE		
		PCXA	2	3	• **μPD750104** PC_{11-0} <– PC_{11-8}+XA		
					• **μPD750106**, **μPD750108** PC_{12-0} <– PC_{12-8}+XA		
					• **μPD75P0116** PC_{13-0} <– PC_{13-8}+XA		
		BCDE	2	3	• **μPD750104** PC_{11-0} <– BCDE$^{\textbf{Note 1}}$	*11	
					• **μPD750106**, **μPD750108** PC_{12-0} <– BCDE$^{\textbf{Note 2}}$		
					• **μPD75P0116** PC_{13-0} <– BCDE$^{\textbf{Note 3}}$		

Notes 1. Set register B to 0.

2. Only the LSB is valid in register B.

3. Only the low-order two bits are valid in register B.

Instruction	Mnemonic	Operand	Number of bytes	Machine cycle	Operation	Addressing area	Skip condition
Branch	**BR**	BCXA	2	3	• **μPD750104** PC$_{11-0}$ <– BCXA[Note 1]	*11	
					• **μPD750106**, **μPD750108** PC$_{12-0}$ <– BCXA[Note 2]		
					• **μPD75P0116** PC$_{13-0}$ <– BCXA[Note 3]		
	BRA[Note 4]	!addr1	3	3	• **μPD750104** PC$_{11-0}$ <– addr	*11	
					• **μPD750106**, **μPD750108** PC$_{12-0}$ <– addr		
					• **μPD75P0116** PC$_{13-0}$ <– addr1		
	BRCB	!caddr	2	2	• **μPD750104** PC$_{11-0}$ <– caddr$_{11-0}$	*8	
					• **μPD750106**, **μPD750108** PC$_{12-0}$ <– PC$_{12}$+caddr$_{11-0}$		
					• **μPD75P0116** PC$_{13-0}$ <– PC$_{13,\ 12}$+caddr$_{11-0}$		
Subroutine stack control	**CALLA**[Note 5]	!addr1	3	3	• **μPD750104** (SP–2) <– x, x, MBE,RBE (SP–6)(SP–3)(SP–4) <– PC$_{11-0}$ (SP–5) <– 0, 0, 0, 0 PC$_{11-0}$ <– addr, SP <– SP–6	*11	
					• **μPD750106**, **μPD750108** (SP–2) <– x, x, MBE,RBE (SP–6)(SP–3)(SP–4) <– PC$_{11-0}$ (SP–5) <– 0, 0, 0, PC$_{12}$ PC$_{12-0}$ <– addr, SP <– SP–6		
					• **μPD75P0116** (SP–2) <– x, x, MBE,RBE (SP–6)(SP–3)(SP–4) <– PC$_{11-0}$ (SP–5) <– 0, 0, PC$_{13}$, PC$_{12}$ PC$_{13-0}$ <– addr1, SP <– SP–6		

Notes 1. Set register B to 0.

2. Only the LSB is valid in register B.

3. Only the low-order two bits are valid in register B.

4. The shaded portion is supported in Mk II mode only.

5. The shaded portion is supported in Mk II mode only. The other portions are supported in Mk I mode only.

Instruction	Mnemonic	Operand	Number of bytes	Machine cycle	Operation	Addressing area	Skip condition
Subroutine stack control	**CALL**[Note]	!addr	3	3	• **µPD750104** (SP–3) <– MBE,RBE, 0, 0 (SP–4)(SP–1)(SP–2) <– PC$_{11-0}$ PC$_{11-0}$ <– addr, SP <– SP–4	*6	
					• **µPD750106**, **µPD750108** (SP–3) <– MBE,RBE, 0, PC$_{12}$ (SP–4)(SP–1)(SP–2) <– PC$_{11-0}$ PC$_{12-0}$ <– addr, SP <– SP–4		
					• **µPD75P0116** (SP–3) <– MBE,RBE, PC$_{13}$, PC$_{12}$ (SP–4)(SP–1)(SP–2) <– PC$_{11-0}$ PC$_{13-0}$ <– addr1, SP <– SP–4		
				4	• **µPD750104** (SP–2) <– x, x, MBE,RBE (SP–6)(SP–3)(SP–4) <– PC$_{11-0}$ (SP–5) <– 0, 0, 0, 0 PC$_{11-0}$ <– addr, SP <– SP–6		
					• **µPD750106**, **µPD750108** (SP–2) <– x, x, MBE,RBE (SP–6)(SP–3)(SP–4) <– PC$_{11-0}$ (SP–5) <– 0, 0, 0, PC$_{12}$ PC$_{12-0}$ <– addr, SP <– SP–6		
					• **µPD75P0116** (SP–2) <– x, x, MBE,RBE (SP–6)(SP–3)(SP–4) <– PC$_{11-0}$ (SP–5) <– 0, 0, PC$_{13}$, PC$_{12}$ PC$_{13-0}$ <– addr, SP <– SP–6		
	CALLF[Note]	!faddr	2	2	• **µPD750104** (SP–3) <– MBE,RBE, 0, 0 (SP–4)(SP–1)(SP–2) <– PC$_{11-0}$ PC$_{11-0}$ <– 0+faddr, SP <– SP–4	*9	
					• **µPD750106**, **µPD750108** (SP–3) <– MBE,RBE, 0, PC$_{12}$ (SP–4)(SP–1)(SP–2) <– PC$_{11-0}$ PC$_{12-0}$ <– 00+faddr, SP <– SP–4		
					• **µPD75P0116** (SP–3) <– MBE,RBE, PC$_{13}$, PC$_{12}$ (SP–4)(SP–1)(SP–2) <– PC$_{11-0}$ PC$_{13-0}$ <– 000+faddr, SP <– SP–4		

Note The shaded portion is supported in Mk II mode only. The other portions are supported in Mk I mode only.

Instruction	Mnemonic	Operand	Number of bytes	Machine cycle	Operation	Addressing area	Skip condition
Subroutine stack control	CALLF[Note]	!faddr	2	3	• **µPD750104** (SP–2) –> x, x, MBE,RBE (SP–6)(SP–3)(SP–4) <– PC$_{11-0}$ (SP–5) <– 0, 0, 0, 0 PC$_{11-0}$ <– 0+faddr, SP <– SP–6	*9	
					• **µPD750106**, **µPD750108** (SP–2) –> x, x, MBE,RBE (SP–6)(SP–3)(SP–4) <– PC$_{11-0}$ (SP–5) <– 0, 0, 0, PC$_{12}$ PC$_{12-0}$ <– 00+faddr, SP <– SP–6		
					• **µPD75P0116** (SP–2) <– x, x, MBE,RBE (SP–6)(SP–3)(SP–4) <– PC$_{11-0}$ (SP–5) <– 0, 0, PC$_{13}$, PC$_{12}$ PC$_{13-0}$ <– 000+faddr, SP <– SP–6		
	RET[Note]		1	3	• **µPD750104** PC$_{11-0}$ <– (SP)(SP+3)(SP+2) MBE,RBE, 0, 0 <– (SP+1), SP <– SP+4		
					• **µPD750106**, **µPD750108** PC$_{11-0}$ <– (SP)(SP+3)(SP+2) MBE,RBE, 0, PC$_{12}$ <– (SP+1) SP <– SP+4		
					• **µPD75P0116** PC$_{11-0}$ <– (SP)(SP+3)(SP+2) MBE,RBE, PC$_{13}$, PC$_{12}$ <– (SP+1) SP <– SP+4		
				3	• **µPD750104** x, x, MBE, RBE <– (SP+4) 0, 0, 0, 0 <– (SP+1) PC$_{11-0}$ <– (SP)(SP+3)(SP+2) SP <– SP+6		
					• **µPD750106**, **µPD750108** x, x, MBE, RBE <– (SP+4) MBE,0, 0, PC$_{12}$ <– (SP+1) PC$_{11-0}$ <– (SP)(SP+3)(SP+2) SP <– SP+6		
					• **µPD75P0116** x, x, MBE, RBE <– (SP+4) 0,0, PC$_{13}$, PC$_{12}$ <– (SP+1) PC$_{11-0}$ <– (SP)(SP+3)(SP+2) SP <– SP+6		

Note The shaded portion is supported in Mk II mode only. The other portions are supported in Mk I mode only.

Instruction	Mnemonic	Operand	Number of bytes	Machine cycle	Operation	Addressing area	Skip condition
Subroutine stack control	**RETS**[Note]		1	3+S	• **µPD750104** MBE, RBE, 0, 0 <– (SP+1) PC$_{11-0}$ <– (SP)(SP+3)(SP+2) SP <– SP+4 Then skip unconditionally		Unconditionally
					• **µPD750106**, **µPD750108** MBE, 0, 0, PC$_{12}$ <– (SP+1) PC$_{11-0}$ <– (SP)(SP+3)(SP+2) SP <– SP+4 Then skip unconditionally		
					• **µPD75P0116** MBE, RBE, PC$_{13}$, PC$_{12}$ <– (SP+1) PC$_{11-0}$ <– (SP)(SP+3)(SP+2) SP <– SP+4 Then skip unconditionally		
				3+S	• **µPD750104** 0, 0, 0, 0 <– (SP+1) PC$_{11-0}$ <– (SP)(SP+3)(SP+2) x, x, MBE, RBE <– (SP+4) SP <– SP+6 Then skip unconditionally		
					• **µPD750106**, **µPD750108** 0, 0, 0, PC$_{12}$ <– (SP+1) PC$_{11-0}$ <– (SP)(SP+3)(SP+2) x, x, MBE, RBE <– (SP+4) SP <– SP+6 Then skip unconditionally		
					• **µPD75P0116** 0, 0, PC$_{13}$, PC$_{12}$ <– (SP+1) PC$_{11-0}$ <– (SP)(SP+3)(SP+2) x, x, MBE, RBE <– (SP+4) SP <– SP+6 Then skip unconditionally		
	RETI		1	3	• **µPD750104** MBE, RBE, 0, 0 <– (SP+1) PC$_{11-0}$ <– (SP)(SP+3)(SP+2) PSW <– (SP+4)(SP+5), SP <– SP+6		Unconditionally
					• **µPD750106**, **µPD750108** MBE, RBE, 0, PC$_{12}$ <– (SP+1) PC$_{11-0}$ <– (SP)(SP+3)(SP+2) PSW <– (SP+4)(SP+5), SP <– SP+6		
					• **µPD75P0116** MBE, RBE, PC$_{13}$, PC$_{12}$ <– (SP+1) PC$_{11-0}$ <– (SP)(SP+3)(SP+2) PSW <– (SP+4)(SP+5), SP <– SP+6		

Note The shaded portion is supported in Mk II mode only. The other portions are supported in Mk I mode only.

Instruction	Mnemonic	Operand	Number of bytes	Machine cycle	Operation	Addressing area	Skip condition
Subroutine stack control	**RETI**[Note 1]		1	3	• **μPD750104** 0, 0, 0, 0 <– (SP+1) PC$_{11-0}$ <– (SP)(SP+3)(SP+2) PSW <– (SP+4)(SP+5), SP <– SP+6		Unconditionally
					• **μPD750106**, **μPD750108** 0, 0, 0, PC$_{12}$ <– (SP+1) PC$_{11-0}$ <– (SP)(SP+3)(SP+2) PSW <– (SP+4)(SP+5), SP <– SP+6		
					• **μPD75P0116** 0, 0, PC$_{13}$, PC$_{12}$ <– (SP+1) PC$_{11-0}$ <– (SP)(SP+3)(SP+2) PSW <– (SP+4)(SP+5), SP <– SP+6		
	PUSH	rp	1	1	(SP–1)(SP–2) <– rp, SP <– SP–2		
		BS	2	2	(SP–1) <– MBS, (SP–2) <– RBS, SP <– SP–2		
	POP	rp	1	1	rp <– (SP+1)(SP), SP <– SP+2		
		BS	2	2	MBS <– (SP+1), RBS <– (SP), SP <– SP+2		
Interrupt control	**EI**		2	2	IME(IPS.3) <– 1		
		IExxx	2	2	IExxx <– 1		
	DI		2	2	IME(IPS.3) <– 0		
		IExxx	2	2	IExxx <– 0		
I/O	**IN**[Note 2]	A,PORT$_n$	2	2	A <– PORT$_n$ (n=0 - 8)		
		XA,PORT$_n$	2	2	XA <– PORT$_{n+1}$, PORT$_n$ (n=4, 6)		
	OUT[Note 2]	PORT$_n$,A	2	2	PORT$_n$ <- A (n=2 - 8)		
		PORT$_n$,XA	2	2	PORT$_{n+1}$,PORT$_n$ <– XA (n=4, 6)		
CPU control	**HALT**		2	2	Set HALT Mode (PCC.2 <– 1)		
	STOP		2	2	Set STOP Mode (PCC.3 <– 1)		
	NOP		1	1	No Operation		

Notes 1. The shaded portion is supported in Mk II mode only. The other portions are supported in Mk I mode only.

2. MBE = 0, or MBE = 1 and MBS = 15 must be set when an IN/OUT instruction is executed.

Instruction	Mnemonic	Operand	Number of bytes	Machine cycle	Operation	Addressing area	Skip condition
Special	**SEL**	RBn	2	2	RBS <– n (n=0 - 3)		
		MBn	2	2	MBS <– n (n=0, 1, 15)		
	GETI$^{\text{Note}}$	taddr	1	3	• **μPD750104** When the TBR instruction is used PC_{11-0} <– $(taddr)_{3-0}$+(taddr+1)	*10	
					When the TCALL instruction is used (SP–4)(SP–1)(SP–2) <– PC_{11-0} (SP–3) <– MBE, RBE, 0, 0 PC_{11-0} <– $(taddr)_{3-0}$+(taddr+1) SP <– SP–4		
					When an instruction other than the TBR or TCALL instruction is used Execution of (taddr)(taddr+1) instruction		Depends on the referenced instruction
					• **μPD750106**, **μPD750108** When the TBR instruction is used PC_{12-0} <– $(taddr)_{4-0}$+(taddr+1)		
					When the TCALL instruction is used (SP–4)(SP–1)(SP–2) <- PC_{11-0} (SP–3) <– MBE, RBE, 0, PC_{12} PC_{12-0} <– $(taddr)_{4-0}$+(taddr+1) SP <– SP–4		
					When an instruction other than the TBR or TCALL instruction is used Execution of (taddr)(taddr+1) instruction		Depends on the referenced instruction
					• **μPD75P0116** When the TBR instruction is used PC_{13-0} <– $(taddr)_{5-0}$+(taddr+1)		
					When the TCALL instruction is used (SP–4)(SP–1)(SP–2) <- PC_{11-0} (SP–3) <– MBE, RBE, PC_{13}, PC_{12} PC_{13-0} <– $(taddr)_{5-0}$+(taddr+1) SP <– SP–4		
					When an instruction other than the TBR or TCALL instruction is used Execution of (taddr)(taddr+1) instruction		Depends on the referenced instruction

Note The TBR and TCALL instructions are assembler pseudo instructions to define tables used for GETI instructions.

Instruction	Mnemonic	Operand	Number of bytes	Machine cycle	Operation	Addressing area	Skip condition
Special	GETI[Notes1, 2]	taddr	1	3	• **µPD750104** When the TBR instruction is used PC_{11-0} <– $(taddr)_{3-0}$+(taddr+1)	*10	
				4	When the TCALL instruction is used (SP–6)(SP–3)(SP–4) <– PC_{11-0} (SP–5) <– 0, 0, 0, 0 (SP–2) <– x, x, MBE, RBE PC_{11-0} <– $(taddr)_{3-0}$+(taddr+1) SP <– SP–6		
				3	When an instruction other than the TBR or TCALL instruction is used Execution of (taddr)(taddr+1) instruction		Depends on the referenced instruction
				3	• **µPD750106**, **µPD750108** When the TBR instruction is used PC_{12-0} <– $(taddr)_{4-0}$+(taddr+1)		
				4	When the TCALL instruction is used (SP–6)(SP–3)(SP–4) <– PC_{11-0} (SP–5) <– 0, 0, 0, PC_{12} (SP–2) <– x, x, MBE, RBE PC_{12-0} <– $(taddr)_{4-0}$+(taddr+1) SP <– SP–6		
				3	When an instruction other than the TBR or TCALL instruction is used Execution of (taddr)(taddr+1) instruction		Depends on the referenced instruction
				3	• **µPD75P0116** When the TBR instruction is used PC_{13-0} <– $(taddr)_{5-0}$+(taddr+1)		
				4	When the TCALL instruction is used (SP–6)(SP–3)(SP–4) <– PC_{11-0} (SP–5) <– 0, 0, PC_{13}, PC_{12} (SP–2) <– x, x, MBE, RBE PC_{13-0} <– $(taddr)_{5-0}$+(taddr+1) SP <– SP–6		
				3	When an instruction other than the TBR or TCALL instruction is used Execution of (taddr)(taddr+1) instruction		Depends on the referenced instruction

Notes 1. The shaded portion is supported in Mk II mode only. The other portions are supported in Mk I mode only.

2. The TBR and TCALL instructions are assembler pseudo instructions to define tables used for GETI instructions.

11.3 INSTRUCTION CODES OF EACH INSTRUCTION

(1) Explanations of the symbols for the instruction codes

R_2	R_1	R_0	reg		
0	0	0	A	↑	
0	0	1	X		↑
0	1	0	L		
0	1	1	H	reg	
1	0	0	E		reg1
1	0	1	D		
1	1	0	C		
1	1	1	B	↓	↓

P_2	P_1	P_0	reg-pair		
0	0	0	XA	↑	
0	0	1	XA'		↑
0	1	0	HL		
0	1	1	HL'	rp'	
1	0	0	DE		rp'1
1	0	1	DE'		
1	1	0	BC		
1	1	1	BC'	↓	↓

Q_2	Q_1	Q_0	addressing		
0	0	0	@HL	↑	
0	1	0	@HL+		
0	1	1	@HL–	@rpa	
1	0	0	@DE		↑ @rpa1
1	0	1	@DL	↓	↓

P_2	P_1	reg-pair			
0	0	XA			↑
0	1	HL		↑	rp
1	0	DE	↑ rp2	rp1	
1	1	BC	↓	↓	↓

N_5	N_2	N_1	N_0	IExxx
0	0	0	0	IEBT
0	0	1	0	IEW
0	1	0	0	IET0
0	1	0	1	IECSI
0	1	1	0	IE0
0	1	1	1	IE2
1	0	0	0	IE4
1	1	0	0	IET1
1	1	1	0	IE1

I_n : Immediate data for n4 or n8

D_n : Immediate data for mem

B_n : Immediate data for bit

N_n : Immediate data for n or IExxx

T_n : Immediate data for taddr x 1/2

A_n : Immediate data for the address (2 to 16) relative to branch destination address minus one

S_n : Immediate data for the one's complement of the address (15 to 1) relative to the branch destination address

(2) Bit manipulation addressing instruction codes

$\boxed{\ast 1}$ in the operand field indicates that there are three types of bit manipulation addressing, fmem.bit, pmem.@L, and @H+mem.bit.

The table below lists the second byte $\boxed{\ast 2}$ of an instruction code corresponding to the above addressing.

*1	Second byte of instruction code	Accessible bits
fmem.bit	1 0 B_1 B_0 F_3 F_2 F_1 F_0	FB0H-FBFH manipulatable bits
	1 1 B_1 B_0 F_3 F_2 F_1 F_0	FF0H-FFFH manipulatable bits
pmem.@L	0 1 0 0 G_3 G_2 G_1 G_0	FC0H-FFFH manipulatable bits
@H+mem.bit	0 0 B_1 B_0 D_3 D_2 D_1 D_0	Manipulatable bits of accessible memory bank

B_n : Immediate data for bit

F_n : Immediate data for fmem (Low-order four bits of address)

G_n : Immediate data for pmem (Bits 2 to 5 of address)

D_n : Immediate data for mem (Low-order four bits of address)

Instruction	Mne-monic	Operand	Instruction code		
			B_1	B_2	B_3
Transfer	**MOV**	A,#n4	0 1 1 1 I_3 I_2 I_1 I_0		
		reg1,#n4	1 0 0 1 1 0 1 0	I_3 I_2 I_1 I_0 1 R_2 R_1 R_0	
		rp,#n8	1 0 0 0 1 P_2 P_1 1	I_7 I_6 I_5 I_4 I_3 I_2 I_1 I_0	
		A,@rpa1	1 1 1 0 0 Q_2 Q_1 Q_0		
		XA,@HL	1 0 1 0 1 0 1 0	0 0 0 1 1 0 0 0	
		@HL,A	1 1 1 0 1 0 0 0		
		@HL,XA	1 0 1 0 1 0 1 0	0 0 0 1 0 0 0 0	
		A,mem	1 0 1 0 0 0 1 1	D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	
		XA,mem	1 0 1 0 0 0 1 0	D_7 D_6 D_5 D_4 D_3 D_2 D_1 0	
		mem,A	1 0 0 1 0 0 1 1	D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	
		mem,XA	1 0 0 1 0 0 1 0	D_7 D_6 D_5 D_4 D_3 D_2 D_1 0	
		A,reg	1 0 0 1 1 0 0 1	0 1 1 1 1 R_2 R_1 R_0	
		XA,rp'	1 0 1 0 1 0 1 0	0 1 0 1 1 P_2 P_1 P_0	
		reg1,A	1 0 0 1 1 0 0 1	0 1 1 1 0 R_2 R_1 R_0	
		rp'1,XA	1 0 1 0 1 0 1 0	0 1 0 1 0 P_2 P_1 P_0	
	XCH	A,@rpa1	1 1 1 0 1 Q_2 Q_1 Q_0		
		XA,@HL	1 0 1 0 1 0 1 0	0 0 0 1 0 0 0 1	
		A,mem	1 0 1 1 0 0 1 1	D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	
		XA,mem	1 0 1 1 0 0 1 0	D_7 D_6 D_5 D_4 D_3 D_2 D_1 0	
		A,reg1	1 1 0 1 1 R_2 R_1 R_0		
		XA,rp'	1 0 1 0 1 0 1 0	0 1 0 0 0 P_2 P_1 P_0	
Table reference	**MOVT**	XA,@PCDE	1 1 0 1 0 1 0 0		
		XA,@PCXA	1 1 0 1 0 0 0 0		
		XA,@BCXA	1 1 0 1 0 0 0 1		
		XA,@BCDE	1 1 0 1 0 1 0 1		
Bit transfer	**MOV1**	CY, $\boxed{*1}$	1 0 1 1 1 1 0 1	*2	
		$\boxed{*1}$,CY	1 0 0 1 1 0 1 1	*2	

Instruction	Mne-monic	Operand	Instruction code		
			B_1	B_2	B_3
Arithmetic/ logical	**ADDS**	A,#n4	0 1 1 0 I_3 I_2 I_1 I_0		
		XA,#n8	1 0 1 1 1 0 0 1	I_7 I_6 I_5 I_4 I_3 I_2 I_1 I_0	
		A,@HL	1 1 0 1 0 0 1 0		
		XA,rp’	1 0 1 0 1 0 1 0	1 1 0 0 1 P_2 P_1 P_0	
		rp’1,XA	1 0 1 0 1 0 1 0	1 1 0 0 0 P_2 P_1 P_0	
	ADDC	A,@HL	1 0 1 0 1 0 0 1		
		XA,rp’	1 0 1 0 1 0 1 0	1 1 0 1 1 P_2 P_1 P_0	
		rp’1,XA	1 0 1 0 1 0 1 0	1 1 0 1 0 P_2 P_1 P_0	
	SUBS	A,@HL	1 0 1 0 1 0 0 0		
		XA,rp’	1 0 1 0 1 0 1 0	1 1 1 0 1 P_2 P_1 P_0	
		rp’1,XA	1 0 1 0 1 0 1 0	1 1 1 0 0 P_2 P_1 P_0	
	SUBC	A,@HL	1 0 1 1 1 0 0 0		
		XA,rp’	1 0 1 0 1 0 1 0	1 1 1 1 1 P_2 P_1 P_0	
		rp’1,XA	1 0 1 0 1 0 1 0	1 1 1 1 0 P_2 P_1 P_0	
	AND	A,#n4	1 0 0 1 1 0 0 1	0 0 1 1 I_3 I_2 I_1 I_0	
		A,@HL	1 0 0 1 0 0 0 0		
		XA,rp’	1 0 1 0 1 0 1 0	1 0 0 1 1 P_2 P_1 P_0	
		rp’1,XA	1 0 1 0 1 0 1 0	1 0 0 1 0 P_2 P_1 P_0	
	OR	A,#n4	1 0 0 1 1 0 0 1	0 1 0 0 I_3 I_2 I_1 I_0	
		A,@HL	1 0 1 0 0 0 0 0		
		XA,rp’	1 0 1 0 1 0 1 0	1 0 1 0 1 P_2 P_1 P_0	
		rp’1,XA	1 0 1 0 1 0 1 0	1 0 1 0 0 P_2 P_1 P_0	
	XOR	A,#n4	1 0 0 1 1 0 0 1	0 1 0 1 I_3 I_2 I_1 I_0	
		A,@HL	1 0 1 1 0 0 0 0		
		XA,rp’	1 0 1 0 1 0 1 0	1 0 1 1 1 P_2 P_1 P_0	
		rp’1,XA	1 0 1 0 1 0 1 0	1 0 1 1 0 P_2 P_1 P_0	
Accumulator manipulation	**RORC**	A	1 0 0 1 1 0 0 0		
	NOT	A	1 0 0 1 1 0 0 1	0 1 0 1 1 1 1 1	

Instruction	Mne-monic	Operand	Instruction code		
			B_1	B_2	B_3
Increment/ decrement	**INCS**	reg	1 1 0 0 0 R_2 R_1 R_0		
		rp1	1 0 0 0 1 P_2 P_1 0		
		@HL	1 0 0 1 1 0 0 1	0 0 0 0 0 0 1 0	
		mem	1 0 0 0 0 0 1 0	D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	
	DECS	reg	1 1 0 0 1 R_2 R_1 R_0		
		rp'	1 0 1 0 1 0 1 0	0 1 1 0 1 P_2 P_1 P_0	
Comparison	**SKE**	reg,#n4	1 0 0 1 1 0 1 0	I_3 I_2 I_1 I_0 0 R_2 R_1 R_0	
		@HL,#n4	1 0 0 1 1 0 0 1	0 1 1 0 I_3 I_2 I_1 I_0	
		A,@HL	1 0 0 0 0 0 0 0		
		XA,@HL	1 0 1 0 1 0 1 0	0 0 0 1 1 0 0 1	
		A,reg	1 0 0 1 1 0 0 1	0 0 0 0 1 R_2 R_1 R_0	
		XA,rp'	1 0 1 0 1 0 1 0	0 1 0 0 1 P_2 P_1 P_0	
Carry flag manipulation	**SET1**	CY	1 1 1 0 0 1 1 1		
	CLR1	CY	1 1 1 0 0 1 1 0		
	SKT	CY	1 1 0 1 0 1 1 1		
	NOT1	CY	1 1 0 1 0 1 1 0		
Memory bit manipulation	**SET1**	mem.bit	1 0 B_1 B_0 0 1 0 1	D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	
		$\boxed{*1}$	1 0 0 1 1 1 0 1	*2	
	CLR1	mem.bit	1 0 B_1 B_0 0 1 0 0	D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	
		$\boxed{*1}$	1 0 0 1 1 1 0 0	*2	
	SKT	mem.bit	1 0 B_1 B_0 0 1 1 1	D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	
		$\boxed{*1}$	1 0 1 1 1 1 1 1	*2	
	SKF	mem.bit	1 0 B_1 B_0 0 1 1 0	D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	
		$\boxed{*1}$	1 0 1 1 1 1 1 0	*2	
	SKTCLR	$\boxed{*1}$	1 0 0 1 1 1 1 1	*2	
	AND1	CY, $\boxed{*1}$	1 0 1 0 1 1 0 0	*2	
	OR1	CY, $\boxed{*1}$	1 0 1 0 1 1 1 0	*2	
	XOR1	CY, $\boxed{*1}$	1 0 1 1 1 1 0 0	*2	

Instruction	Mne-monic	Operand	Instruction code		
			B_1	B_2	B_3
Branch	**BR**	!addr	1 0 1 0 1 0 1 1	0 0 ←	addr →
		\$addr1 (+16) to (+2)	0 0 0 0 A_3 A_2 A_1 A_0		
		(–1) to (–15)	1 1 1 1 S_3 S_2 S_1 S_0		
		PCDE	1 0 0 1 1 0 0 1	0 0 0 0 0 1 0 0	
		PCXA	1 0 0 1 1 0 0 1	0 0 0 0 0 0 0 0	
		BCDE	0 0 0 0 0 1 0 1		
		BCXA	1 0 0 1 1 0 0 1	0 0 0 0 0 0 0 1	
	BRA	!addr1	1 0 1 1 1 0 1 0	0 ←	addr1 →
	BRCB	!caddr	0 1 0 1 ←	caddr →	
Sub-routine stack control	**CALL**	!addr	1 0 1 0 1 0 1 1	0 1 ←	addr →
	CALLA	!addr1	1 0 1 1 1 0 1 1	0 ←	addr1 →
	CALLF	!faddr	0 1 0 0 0 ←	faddr →	
	RET		1 1 1 0 1 1 1 0		
	RETS		1 1 1 0 0 0 0 0		
	RETI		1 1 1 0 1 1 1 1		
	PUSH	rp	0 1 0 0 1 P_2 P_1 1		
		BS	1 0 0 1 1 0 0 1	0 0 0 0 0 1 1 1	
	POP	rp	0 1 0 0 1 P_2 P_1 0		
		BS	1 0 0 1 1 0 0 1	0 0 0 0 0 1 1 0	
I/O	**IN**	A,PORTn	1 0 1 0 0 0 1 1	1 1 1 1 N_3 N_2 N_1 N_0	
		XA,PORTn	1 0 1 0 0 0 1 0	1 1 1 1 N_3 N_2 N_1 N_0	
	OUT	PORTn,A	1 0 0 1 0 0 1 1	1 1 1 1 N_3 N_2 N_1 N_0	
		PORTn,XA	1 0 0 1 0 0 1 0	1 1 1 1 N_3 N_2 N_1 N_0	
Interrupt control	**EI**		1 0 0 1 1 1 0 1	1 0 1 1 0 0 1 0	
		IExxx	1 0 0 1 1 1 0 1	1 0 N_5 1 1 N_2 N_1 N_0	
	DI		1 0 0 1 1 1 0 0	1 0 1 1 0 0 1 0	
		IExxx	1 0 0 1 1 1 0 0	1 0 N_5 1 1 N_2 N_1 N_0	
CPU control	**HALT**		1 0 0 1 1 1 0 1	1 0 1 0 0 0 1 1	
	STOP		1 0 0 1 1 1 0 1	1 0 1 1 0 0 1 1	
	NOP		0 1 1 0 0 0 0 0		
Special	**SEL**	RBn	1 0 0 1 1 0 0 1	0 0 1 0 0 0 N_1 N_0	
		MBn	1 0 0 1 1 0 0 1	0 0 0 1 N_3 N_2 N_1 N_0	
	GETI	taddr	0 0 T_5 T_4 T_3 T_2 T_1 T_0		

11.4 FUNCTIONS AND APPLICATIONS OF THE INSTRUCTIONS

This section explains functions and applications of the instructions. For the µPD750104, µPD750106, µPD750108, and µPD75P0116, usable instructions and their functions in Mk I mode are different from those in Mk II mode. Read the following explanation.

How to read

○ Can be used in both Mk I mode and Mk II mode for the µPD750104, µPD750106, µPD750108, and µPD75P0116

I Can be used in only Mk I mode for the µPD750104, µPD750106, µPD750108, and µPD75P0116

II Can be used in only Mk II mode for the µPD750104, µPD750106, µPD750108, and µPD75P0116

I/II Can be used in both Mk I mode and Mk II mode for the µPD750104, µPD750106, µPD750108, and µPD75P0116. However, Mk I mode is different from Mk II mode in the functions. Read the explanation of [Mk I mode] for Mk I mode and the explanation of [Mk II mode] for Mk II mode, as required.

Remark "**Function**" in this section is applicable to the µPD750106 and µPD750108 whose program counters consist of 13 bits each. This is also applicable to the µPD750104 whose program counter consists of 12 bits and the µPD75P0116 whose program counter consists of 14 bits, however.

11.4.1 Transfer Instructions

○ **MOV A,#n4**

Function: A <– n4 n4 = $I_{3\text{-}0}$: 0-FH

Transfers the 4-bit immediate data n4 to the A register (4-bit accumulator).

The string effect (group A) can be utilized. When MOV A, #n4 and/or MOV XA, #n8 instructions are located contiguously, the string instructions following an executed instruction are processed as NOP instructions.

Examples 1. The data 0BH is set in the accumulator.

```
MOV A,#0BH
```

2. Data to be output to port 3 is selected from 0 to 2.

```
A0: MOV A,#0
A1: MOV A,#1
A2: MOV A,#2
    OUT PORT3,A
```

○ MOV reg1,#n4

Function: reg1 <– n4 n4 = $I_{3\text{-}0}$: 0-FH

Transfers the 4-bit immediate data n4 to A register reg1 (X, H, L, D, E, B, C).

○ MOV XA,#n8

Function: XA <– n8 n8 = $I_{7\text{-}0}$: 00H-FFH

Transfers the 8-bit immediate data n8 to register pair XA. The string effect can be utilized. When two or more of this instruction are executed in succession or when MOV A,#n4 instruction is located continguously, the string instructions following an executed instruction are processed as NOP instructions.

○ MOV HL,#n8

Function: HL <– n8 n8 = $I_{7\text{-}0}$: 00H-FFH

Transfers the 8-bit immediate data n8 to register pair HL. The string effect can be utilized. When two or more of this instruction are executed in succession, the string instructions following an executed instruction are processed as NOP instructions.

○ MOV rp2,#n8

Function: rp2 <– n8 n8 = $I_{7\text{-}0}$: 00H-FFH

Transfers the 8-bit immediate data n8 to register pair rp2 (BC, DE).

○ MOV A,@HL

○ MOV A,@HL+

○ MOV A,@HL–

○ MOV A,@rpa1

Function: A <– (Register pair specified by the operand)
When HL+ is specified for the register pair: Skip if L = 0
When HL– is specified for the register pair: Skip if L = FH

Transfers the data at the data memory location addressed by the specified register pair (HL, HL+, HL–, DE, DL) to the A register.

When HL+ (automatic increment) is specified for the register pair, automatically increments the contents of the L register by one after the data transfer, and continues the operation until the contents are set to 0.

Then skips the immediately following instruction.

When HL– (automatic decrement) is specified for the register pair, automatically decrements the contents of the L register by one after the data transfer, and continues the operation until the contents are set to FH. Then skips the immediately following instruction.

MOV XA,@HL

Function: A <– (HL), X <– (HL+1)

Transfers the data at the data memory location addressed by the HL register pair to the A register, and transfers the data at the next data memory address to the X register.

However, if the contents of the L register are odd- numbered, an address with the low-order bit ignored is specified.

Example The data at addresses 3EH and 3FH are transferred to the XA register pair.

```
MOV HL, #3EH
MOV XA, @HL
```

MOV @HL,A

Function: (HL) <– A

Transfers the contents of the A register to the data memory location addressed by the HL register pair.

MOV @HL,XA

Function: (HL) <– A, (HL+1) <– X

Transfers the contents of the A register to the data memory location addressed by the HL register pair, and transfers the contents of the X register to the next memory address.

However, if the contents of the L register are odd- numbered, an address with the low-order bit ignored is specified

MOV A,mem

Function: A <– (mem) mem = $D_{7\text{-}0}$: 00H-FFH

Transfers the data at the data memory location addressed by the 8-bit immediate data mem to the A register.

MOV XA,mem

Function: A <– (mem), X <– (mem+1) mem = D_{7-0}: 00H-FEH

Transfers the data at the data memory location addressed by the 8-bit immediate data mem to the A register, and transfers the data at the next address to the X register.

An even address can be specified with mem.

Example The data at addresses 40H and 41H are transferred to the XA register pair.

```
MOV XA,40H
```

MOV mem,A

Function: (mem) <– A mem = D_{7-0}: 00H-FFH

Transfers the contents of the A register to the data memory location addressed by the 8-bit immediate data mem.

MOV mem,XA

Function: (mem) <– A, (mem+1) <– X mem = D_{7-0}: 00H-FEH

Transfers the contents of the A register to the data memory location addressed by the 8-bit immediate data mem, and transfers the contents of the X register to the next memory address.

An even address can be specified with mem.

MOV A,reg

Function: A <– reg

Transfers the contents of register reg (X, A, H, L, D, E, B, C) to the A register.

MOV XA,rp'

Function: XA <– rp'

Transfers the contents of register pair rp' (XA, HL, DE, BC, XA', HL', DE', BC') to the XA register pair.

Example The contents of the XA' register pair are transferred to the XA register pair.

```
MOV XA, XA'
```

○ MOV reg1,A

Function: reg1 <– A

Transfers the contents of the A register to register reg1 (X, H, L, D, E, B, C).

○ MOV rp'1,XA

Function: rp'1 <– XA

Transfers the contents of the XA register pair to register pair rp'1 (HL, DE, BC, XA', HL', DE', BC').

○ XCH A,@HL

○ XCH A,@HL+

○ XCH A,@HL–

○ XCH A,@rpa1

Function: A <–> (Register pair specified by the operand)
When HL+ is specified for the register pair: Skip if L = 0
When HL– is specified for the register pair: Skip if L = FH

Exchanges the contents of the A register with the data at the data memory location addressed by the specified register pair (HL, HL+, HL– , DE, DL).

When HL+ (automatic increment) is specified for the register pair, automatically increments the contents of the L register by one after the data exchange, and continues the operation until the contents are set to 0. Then skips the immediately following instruction.

When HL– (automatic decrement) is specified for the register pair, automatically decrements the contents of the L register by one after the data exchange, and continues the operation until the contents are set to FH. Then skips the immediately following instruction.

Example The data at addresses 20H-2FH are exchanged with the data at addresses 30H-3FH.

```
        SEL     MB0
        MOV     D,#2
        MOV     HL,#30H
LOOP:   XCH     A,@HL     ; A <–> (3x)
        XCH     A,@DL     ; A <–> (2x)
        XCH     A,@HL+    ; A <–> (3x)
        BR      LOOP
```

XCH XA,@HL

Function: A <–> (HL), X <–> (HL+1)

Exchanges the contents of the A register with the data at the data memory location addressed by the HL register pair, and exchanges the contents of the X register with the data at the next memory address.

However, if the contents of the L register are odd- numbered, an address with the low-order bit ignored is specified.

XCH A,mem

Function: A <–> (mem)   mem = D_{7-0}: 00H-FEH

Exchanges the contents of the A register with the data at the data memory location addressed by the 8-bit immediate data mem.

XCH XA,mem

Function: A <–> (mem), X <–> (mem+1)   mem = D_{7-0}: 00H-FEH

Exchanges the contents of the A register with the data at the data memory location addressed by the 8-bit immediate data mem, and exchanges the contents of the X register 1 with the data at the next memory address.

An even address can be specified with mem.

XCH A,reg1

Function: A <–> reg1

Exchanges the contents of the A register with register reg1 (X, H, L, D, E, B, C).

XCH XA,rp'

Function: XA <–> rp'

Exchanges the contents of the XA register pair with the contents of register pair rp' (XA, HL, DE, BC, XA', HL', DE', BC').

11.4.2 Table Reference Instructions

○ MOVT XA,@PCDE

Function: For the μPD750106 and μPD750108

XA <– ROM ($PC_{12\text{-}8}$+DE)

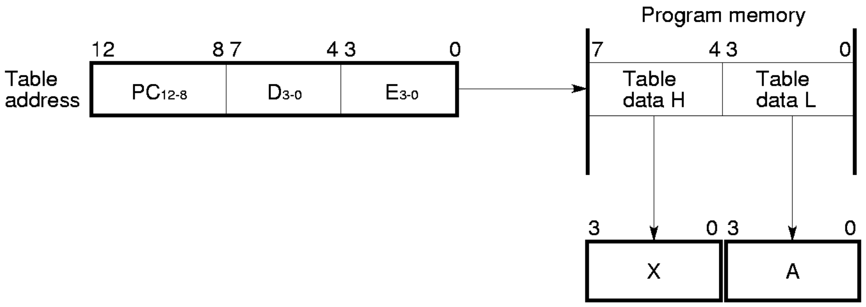
Transfers the low-order four bits of the table data in program memory to the A register, and the high-order four bits to the X register. The table data is addressed by the program counter (PC) with its low-order eight bits ($PC_{7\text{-}0}$) exchanged with the contents of the DE register pair.

The table address is determined by the contents of the program counter (PC) present when this instruction is executed.

The table area must have necessary data loaded by an assembler pseudo instruction (DB instruction). The program counter is not affected by the execution of the pseudo instruction.

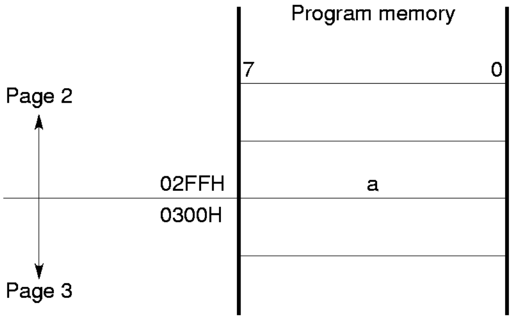
This instruction is useful for consecutive table data references.

Example For the μPD750106 and μPD750108



Remark "**Function**" in this section is applicable to the μPD750106 and μPD750108 whose program counters consist of 13 bits each. This is also applicable to the μPD750104 whose program counter consists of 12 bits and the μPD75P0116 whose program counter consists of 14 bits, however.

Caution The MOVT XA,@PCDE instruction usually references table data in the page containing that instruction. However, when the instruction is located at address xxFFH, table data in the next page is referenced instead of table data in the page containing that instruction.



For example, if MOVT XA,@PCDE is located at a as shown above, the table data in page 3 specified by the contents of the DE register pair is transferred to the XA register pair instead of that in page 2.

Example The 16-byte data at addresses xxF0H-xxFFH in program memory is transferred to addresses 30H-4FH in data memory.

```
SUB:      SEL     MB0
          MOV     HL,#30H       ; HL <– 30H
          MOV     DE,#0F0H      ; DE <– F0H
LOOP:     MOVT    XA,@PCDE      ; XA <– table data
          MOV     @HL, XA       ; (HL) <– XA
          INCS    HL            ; HL <– HL + 2
          INCS    HL
          INCS    E             ; E <– E + 1
          BR      LOOP
          RET
          ORG     xxF0H
          DB      xxH, xxH, ....... ; Table data
```

MOVT XA, @PCXA

Function: For the µPD750106 and µPD750108
XA <– ROM (PC$_{12-8}$+XA)

Transfers the low-order four bits of the table data in program memory to the A register, and the high-order four bits to the X register. The table data is addressed by the program counter (PC) with its low-order eight bits (PC$_{7-0}$) exchanged with the contents of the XA register pair.

The table address is determined by the contents of the program counter present when this instruction is executed.

The table area must have necessary data loaded by an assembler pseudo instruction (DB instruction).

The program counter is not affected by the execution of this instruction.

Caution As with MOVT XA,@PCDE, when the instruction is located at address xxFFH, table data in the next page is transferred.

Remark "**Function**" in this section is applicable to the µPD750106 and µPD750108 whose program counters consist of 13 bits each. This is also applicable to the µPD750104 whose program counter consists of 12 bits and the µPD75P0116 whose program counter consists of 14 bits, however.

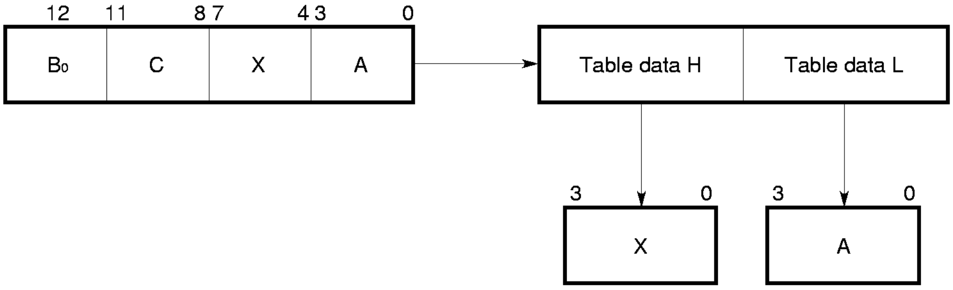
◯ MOVT XA,@BCXA

Function: For the µPD750106 and µPD750108

XA <– (BCXA) $_{\text{ROM}}$

Transfers the low-order four bits of the table data (eight bits) in program memory to the A register, and the high-order four bits to the X register. The table data is addressed by the low-order one bit of the B register and the contents of the C, X, and A registers.

The table area must have necessary data loaded by an assembler pseudo instruction (DB instruction). The program counter is not affected by the execution of this instruction.



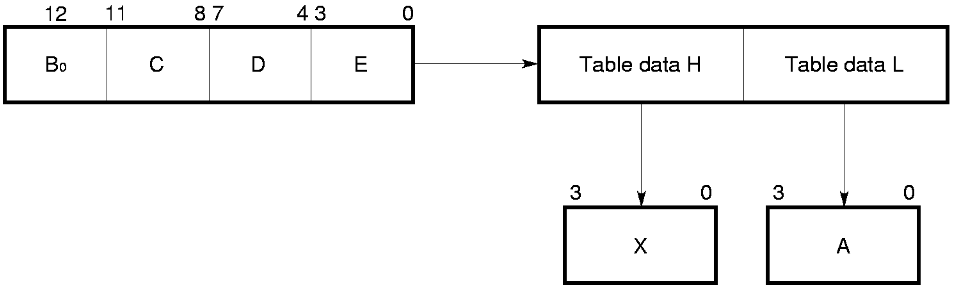
◯ MOVT XA,@BCDE

Function: For the µPD750106 and µPD750108

XA <– (BCDE) $_{\text{ROM}}$

Transfers the low-order four bits of the table data (eight bits) in program memory to the A register, and the high-order four bits to the X register. The table data is addressed by the low-order three bits of the B register and the contents of the C, D, and E registers.

The table area must have necessary data loaded by an assembler pseudo instruction (DB instruction). The program counter is not affected by the execution of this instruction.



Remark "**Function**" in this section is applicable to the µPD750106 and µPD750108 whose program counters consist of 13 bits each. This is also applicable to the µPD750104 whose program counter consists of 12 bits and the µPD75P0116 whose program counter consists of 14 bits, however.

11.4.3 Bit Transfer Instructions

○ **MOV1 CY,fmem.bit**

○ **MOV1 CY,pmem.@L**

○ **MOV1 CY,@H+mem.bit**

Function: CY <– (bit specified in operand)

Transfers the data memory bit specified by bit manipulation addressing (fmem.bit, pmem.@L, @H+mem.bit) to the carry flag (CY).

○ **MOV1 fmem.bit,CY**

○ **MOV1 pmem.@L,CY**

○ **MOV1 @H+mem.bit,CY**

Function: (bit specified in operand) <– CY

Transfers the carry flag (CY) bit to the data memory bit specified by bit manipulation addressing (fmem.bit, pmem.@L,@H+mem.bit)

Example The flag (bit 3 at address 3FH) in data memory is set in bit 2 of port 3.

```
FLAG   EQU 3FH.3
SEL    MB0
MOV    H,#FLAG SHR6    ; H <– high-order 4 bits of FLAG
MOV1   CY,@H+FLAG      ; CY <– FLAG
MOV1   PORT3.2,CY      ; P32 <– CY
```

11.4.4 Arithmetic/Logical Instructions

○ **ADDS A,#n4**

Function: A <– A+n4 ; Skip if carry.   n4 = I_{3-0}: 0-FH

Adds the 4-bit immediate data n4 to the contents of the A register in binary, then skips the next instruction if the addition generates a carry. The carry flag is not affected.

This instruction, when combined with the ADDC A,@HL or SUBC A,@HL instruction, functions as a number system conversion instruction. (See **Section 11.1**.)

ADDS XA,#n8

Function: XA <– XA+n8 ; Skip if carry. n8 = I_{7-0}: 00H-FFH

Adds the 8-bit immediate data n8 to the contents of the XA register pair in binary, then skips the next instruction if the addition generates a carry. The carry flag is not affected.

ADDS A,@HL

Function: A <– A+(HL) ; Skip if carry.

Adds the data at the data memory location addressed by the HL register pair to the contents of the A register in binary, then skips the next instruction if the addition generates a carry. The carry flag is not affected.

ADDS XA,rp'

Function: XA <– XA+rp' ; Skip if carry.

Adds the contents of register pair rp' (XA, HL, DE, BC, XA', HL', DE', BC') to the contents of the XA register pair in binary, then skips the next instruction if the addition generates a carry. The carry flag is not affected.

ADDS rp'1,XA

Function: rp' <– rp'1+XA ; Skip if carry.

Adds the contents of the XA register pair to the contents of register pair rp'1 (HL, DE, BC, XA', HL', DE', BC') in binary, then skips the next instruction if the addition generates a carry. The carry flag is not affected.

Example The register pair is left-shifted.

```
MOV    XA, rp'1
ADDS   rp'1, XA
NOP
```

ADDC A,@HL

Function: A,CY <– A+(HL)+CY

Adds the data at the data memory location addressed by the HL register pair together with the carry flag to the contents of the A register in binary. If the addition generates a carry, the carry flag is set. If no carry is generated, the carry flag is reset.

If the execution of this instruction generates a carry when this instruction is immediately followed by the ADDS A,#n4 instruction, the ADDS A,#n4 instruction is skipped. If no carry is generated, the ADDS A,#n4 instruction is executed, and the skip function of the ADDS A,#n4 instruction is disabled. Accordingly, a combination of these instructions can be used for number system conversion. (See **Section 11.1**.)

ADDC XA,rp'

Function: XA, CY <– XA+rp'+CY

Adds the contents of register pair rp' (XA, HL, DE, BC, XA', HL', DE', BC') together with the carry flag to the contents of the XA register pair in binary. If the addition generates a carry, the carry flag is set. If no carry is generated, the carry flag is reset.

ADDC rp'1,XA

Function: rp'1, CY <– rp'1+XA+CY

Adds the contents of the XA register pair together with the carry flag to the contents of register pair rp'1 (HL, DE, BC, XA', HL', DE', BC') in binary. If the addition generates a carry, the carry flag is set. If no carry is generated, the carry flag is reset.

SUBS A,@HL

Function: A <– A–(HL) ; Skip if borrow

Subtracts the data at the data memory location addressed by the HL register pair from the contents of the A register, then sets the result in the A register. If the subtraction generates a borrow, the immediately following instruction is skipped.

The carry flag is not affected.

SUBS XA,rp'

Function: XA <– XA–rp' ; Skip if borrow

Subtracts the contents of register pair rp' (XA, HL, DE, BC, XA', HL', DE', BC') from the contents of the XA register pair, then sets the result in the XA register pair. If the subtraction generates a borrow, the immediately following instruction is skipped.

The carry flag is not affected.

Example Data memory is compared with register pair rp'.

```
MOV   XA, mem
SUBS  XA, rp'
                  ; (mem) ≥ rp'
                  ; (mem) < rp'
```

○ SUBS rp’1,XA

Function: rp’1 <– rp’1+XA ; Skip if borrow

Subtracts the contents of the XA register pair from the contents of register pair rp’1 (HL, DE, BC, XA’, HL’, DE’, BC’), then sets the result in register pair rp’1. If the subtraction generates a borrow, the immediately following instruction is skipped.

The carry flag is not affected.

○ SUBC A,@HL

Function: A, CY <– A–(HL)–CY

Subtracts the data at the data memory location addressed by the HL register pair together with the carry flag from the contents of the A register, then sets the result in the A register. If the subtraction generates a borrow, the carry flag is set. If no borrow is generated, the carry flag is reset.

If the execution of this instruction generates no borrow when this instruction is followed by the ADDS A, #n4 instruction, the ADDS A, #n4 instruction is skipped. If a borrow is generated, the ADDS A, #n4 instruction is executed, and the skip function of the ADDS A, #n4 instruction is disabled. Accordingly, a combination of these instructions can be used for number system conversion. (See **Section 11.1**.)

○ SUBC XA,rp’

Function: XA, CY <– XA–rp’–CY

Subtracts the contents of register pair rp’ (XA, HL, DE, BC, XA’, HL’, DE’, BC’) together with the carry flag from the contents of the XA register pair, then sets the result in the XA register pair. If the subtraction generates a borrow, the carry flag is set. If no borrow is generated, the carry flag is reset.

○ SUBC rp’1,XA

Function: rp’1, CY <– rp’1–XA–CY

Subtracts the contents of the XA register pair together with the carry flag from the contents of register pair rp’1 (HL, DE, BC, XA’, HL’, DE’, BC’), then sets the result in register pair rp’1. If the subtraction generates a borrow, the carry flag is set. If no borrow is generated, the carry flag is reset.

○ AND A,#n4

Function: A <– A $\wedge$ n4   n4 = I_{3-0}: 0-FH

ANDs the contents of the A register with the 4-bit immediate data n4, then sets the result in the A register.

Example The high-order two bits of an accumulator are set to 0.

```
AND A,#0011B
```

○ AND A,@HL

Function: A <– A ∧ (HL)

ANDs the contents of the A register with the data at the data memory location addressed by the HL register pair, then sets the result in the A register.

○ AND XA,rp'

Function: XA <– XA ∧ rp'

ANDs the contents of the XA register pair with the contents of register pair rp' (XA, HL, DE, BC, XA', HL', DE', BC'), then sets the result in the XA register pair.

○ AND rp'1,XA

Function: rp'1 <– rp'1 ∧ XA

ANDs the contents of register pair rp'1 (HL, DE, BC, XA', HL', DE', BC') with the contents of the XA register pair, then sets the result in the specified register pair.

○ OR A,#n4

Function: A <– A ∨ 4 n4 = $I_{3\text{-}0}$: 0-FH

ORs the contents of the A register with the 4-bit immediate data n4, then sets the result in the A register.

Example The low-order three bits of an accumulator are set to 1.

```
OR A,#0111B
```

○ OR A,@HL

Function: A <– A ∨ (HL)

ORs the contents of the A register with the data at the data memory location addressed by the HL register pair, then sets the result in the A register.

○ OR XA,rp'

Function: XA <– XA ∨ rp'

ORs the contents of the XA register pair with the contents of register pair rp' (XA, HL, DE, BC, XA', HL', DE', BC'), then sets the result in the XA register pair.

OR rp'1,XA

Function: rp'1 <– rp' ∨ XA

ORs the contents of register pair rp'1 (HL, DE, BC, XA', HL', DE', BC') with the contents of the XA register pair, then sets the result in register pair rp'1.

XOR A,#n4

Function: A <– A ⊻ n4 n4 = I_{3-0}: 0-FH

Exclusive-ORs the contents of the A register with the 4-bit immediate data n4, then sets the result in the A register.

Example The high-order four bits of an accumulator is inverted.

```
XOR A,#1000B
```

XOR A,@HL

Function: A <– A ⊻ (HL)

Exclusive-ORs the contents of the A register with the data at the data memory location addressed by the HL register pair, then sets the result in the A register.

XOR XA,rp'

Function: XA <– XA ⊻ rp'

Exclusive-ORs the contents of the XA register pair with the contents of register pair rp' (XA, HL, DE, BC, XA', HL', DE', BC'), then sets the result in the XA register pair.

XOR rp'1,XA

Function: rp'1 <– rp'1 ⊻ XA

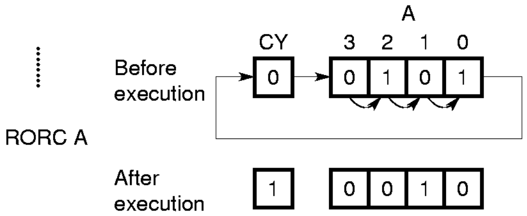
Exclusive-ORs the contents of register pair rp'1 (HL, DE, BC, XA', HL', DE', BC') with the contents of the XA register pair, then sets the result in register pair rp'1.

11.4.5 Accumulator Manipulation Instructions

RORC A

Function: CY <- A_0, A_{n-1} <- A_n, A_3 <- CY (n = 1-3)

Rotates the contents of the A register (4-bit accumulator) through the carry flag one bit position to the right.



NOT A

Function: A <– $\overline{A}$

Obtains the one's complement of the A register (4-bit accumulator), that is, inverts each bit of the A register.

11.4.6 Increment/Decrement Instructions

INCS reg

Function: reg <– reg+1 ; Skip if reg = 0

Increments the contents of register reg (X, A, H, L, D, E, B, C). If the result of increment produces reg = 0, the immediately following instruction is skipped.

INCS rp1

Function: rp1 <– rp1+1 ; Skip if rp1 = 00H

Increments the contents of register pair rp1 (HL, DE, BC). If the result of increment produces rp1 = 00H, the immediately following instruction is skipped.

INCS @HL

Function: (HL) <– (HL)+1 ; Skip if (HL) = 0

Increments the data at the data memory location addressed by the HL register pair. If the result of increment produces data that is 0, the immediately following instruction is skipped.

INCS mem

Function: (mem) <– (mem)+1 ; Skip if (mem) = 0, mem = $D_{7\text{-}0}$: 00H-FFH

Increments the data at the data memory location addressed by the 8-bit immediate data mem. If the result of increment produces data that is 0, the immediately following instruction is skipped.

DECS reg

Function: reg <– reg–1 ; Skip if reg = FH

Decrements the contents of register reg (X, A, H, L, D, E, B, C). If the result of decrement produces reg = FH, the immediately following instruction is skipped.

DECS rp'

Function: rp' <– rp'–1 ; Skip if rp' = FFH

Decrements the contents of register pair rp' (XA, HL, DE, BC, XA', HL', DE', BC'). If the result of decrement produces rp' = FFH, the immediately following instruction is skipped.

11.4.7 Compare Instructions

SKE reg,#n4

Function: Skip if reg = n4 n4 = $I_{3\text{-}0}$: 0-FH

Skips the immediately following instruction if the contents of register reg (X, A, H, L, D, E, B, C) match the 4-bit immediate data n4.

SKE @HL,#n4

Function: Skip if (HL) = n4 n4 = $I_{3\text{-}0}$: 0-FH

Skips the immediately following instruction if the data at the data memory location addressed by the HL register pair match the 4-bit immediate data n4.

SKE A,@HL

Function: Skip if A = (HL)

Skips the immediately following instruction if the contents of the A register match the data at the data memory location addressed by the HL register pair.

○ SKE XA,@HL

Function: Skip if A = (HL) and X = (HL+1)

Skips the immediately following instruction if the contents of the A register match the data at the data memory location addressed by the HL register pair, and the contents of the X register match the data at the next address in data memory.

However, if the contents of the L register are odd- numbered, an address with the lowest-order bit ignored is specified.

○ SKE A,reg

Function: Skip if A = reg

Skips the immediately following instruction if the contents of the A register match the contents of register reg (X, A, H, L, D, E, B, C).

○ SKE XA,rp'

Function: Skip if XA = rp'

Skips the immediately following instruction if the contents of the XA register pair match the contents of register pair rp' (XA, HL, DE, BC, XA', HL', DE', BC').

11.4.8 Carry Flag Manipulation Instructions

○ SET1 CY

Function: CY <– 1

Sets the carry flag.

○ CLR1 CY

Function: CY <– 0

Clears the carry flag.

○ SKT CY

Function: Skip if CY = 1

Skips the immediately following instruction if the carry flag is set to 1.

NOT1 CY

Function: CY <– $\overline{\text{CY}}$

Inverts the carry flag. If it is 0, it is set to 1, or vice versa.

11.4.9 Memory Bit Manipulation Instructions

SET1 mem.bit

Function: (mem.bit) <– 1 mem = $D_{7\text{-}0}$: 00H-FFH, bit = $B_{1\text{-}0}$: 0-3

Sets the bit specified by the 2-bit immediate data bit at the address specified by the 8-bit immediate data mem.

SET1 fmem.bit

SET1 pmem.@L

SET1 @H+mem.bit

Function: (Bit specified in operand) <– 1

Sets the bit in data memory specified by bit manipulation addressing (fmem.bit, pmem.@L, @H+mem.bit).
CLR1 mem.bit

CLR1 mem.bit

Function: (mem.bit) <– 0 mem = $D_{7\text{-}0}$: 00H-FFH, bit = $B_{1\text{-}0}$: 0-3

Clears the bit specified by the 2-bit immediate data bit at the address specified by the 8-bit immediate data mem.

CLR1 fmem.bit

CLR1 pmem.@L

CLR1 @H+mem.bit

Function: (Bit specified in operand) <– 0

Clears the bit in data memory specified by bit manipulation addressing (fmem.bit, pmem.@L, @H+mem.bit).
SKT mem.bit

SKT mem.bit

Function: Skip if (mem.bit) = 1
mem = D_{7-0}: 00H-FFH, bit = B_{1-0}: 0-3

Skips the immediately following instruction if the bit specified by the 2-bit immediate data bit at the address specified by the 8-bit immediate data mem is 1.

SKT fmem.bit

SKT pmem.@L

SKT @H+mem.bit

Function: Skip if (bit specified in operand) = 1

Skips the immediately following instruction if the bit in data memory specified by bit manipulation addressing (fmem.bit, pmem.@L, @H+mem.bit) is set to 1.

SKF mem.bit

Function: Skip if (mem.bit) = 0
mem = D_{7-0}: 00H-FFH, bit = B_{1-0}: 0-3

Skips the immediately following instruction if the bit specified by the 2-bit immediate data bit at the address specified by the 8-bit immediate data mem is 0.

SKF fmem.bit

SKF pmem.@L

SKF @H+mem.bit

Function: Skip if (bit specified in operand) = 0

Skips the immediately following instruction if the bit in data memory specified by bit manipulation addressing (fmem.bit, pmem.@L, @H+mem.bit) is 0.

SKTCLR fmem.bit

SKTCLR pmem.@L

SKTCLR @H+mem.bit

Function: Skip if (bit specified in operand) = 1 then clear

Skips the immediately following instruction if the bit in data memory specified by bit manipulation addressing (fmem.bit, pmem.@L, @H+mem.bit) is 1, then clears the bit to 0.

○ **AND1 CY,fmem.bit**

○ **AND1 CY,pmem.@L**

○ **AND1 CY,@H+mem.bit**

Function: CY <– CY∧(bit specified in operand)

ANDs the content of the carry flag with the bit in data memory specified by bit manipulation addressing (fmem.bit, pmem.@L, @H+mem.bit), then sets the result in the carry flag.

○ **OR1 CY,fmem.bit**

○ **OR1 CY,pmem.@L**

○ **OR1 CY,@H+mem.bit**

Function: CY <– CY∨ (bit specified in operand)

ORs the content of the carry flag with the bit in data memory specified by bit manipulation addressing (fmem.bit, pmem.@L, @H+mem.bit), then sets the result in the carry flag.

○ **XOR1 CY,fmem.bit**

○ **XOR1 CY,pmem.@L**

○ **XOR1 CY,@H+mem.bit**

Function: CY <– CY⊻ (bit specified in operand)

Exclusive-ORs the content of the carry flag with the bit in data memory specified by bit manipulation addressing (fmem.bit, pmem.@L, @H+mem.bit), then sets the result in the carry flag.

11.4.10 Branch Instructions

○ **BR addr**

Function: For the µPD750108 PC_{12-0} <– addr
addr = 0000H-1FFFH

Branches to the address specified by the immediate data addr.

This instruction is an assembler pseudo instruction, and the assembler automatically replaces this instruction with the BR !addr instruction, BRCB !caddr instruction, or BR $addr instruction as required at assembly time.

II BR addr1

Function: For the μPD750108 PC$_{12-0}$ <–addr1
addr1 = 0000H-1FFFH

Branches to the address specified by the immediate data addr1.

This instruction is an assembler pseudo instruction, and the assembler automatically replaces this instruction with the BRA !addr1 instruction, BR !addr instruction, BRCB !caddr instruction, or BR $addr1 instruction as required at assembly time.

Remark "**Function**" in this section is applicable to the μPD750108 whose program counter consists of 13 bits (addr = 0000H to 1FFFH).
However, this is also applicable to the μPD750104 whose program counter consists of 12 bits (addr = 0000H to 0FFFH), the μPD750106 whose program counter consists of 13 bits (addr = 0000H to 17FFH), and the μPD75P0116 whose program counter consists of 14 bits (addr = 0000H to 3FFFH).

II BRA !addr1

Function: For the μPD750108 PC$_{12-0}$ <–addr1

○ BR !addr

Function: For the μPD750108 PC$_{12-0}$ <–addr
addr = 0000H-1FFFH

Transfers the immediate data addr to the program counter (PC), then branches to the location addressed by the program counter.

I BR $addr

Function: For the μPD750108 PC$_{12-0}$ <– addr
addr = (PC–15) to (PC–1), (PC+2) to (PC+16)

Relative branch instruction with branch ranges of (–15 to –1) and (+2 to +16) from the current address. The instruction is not affected by page or block boundaries.

II BR $addr1

Function: For the μPD750108 PC$_{12-0}$ <–addr1
addr = (PC–15) to (PC–1), (PC+2) to (PC+16)

Relative branch instruction with branch ranges of (–15 to –1) and (+2 to +16) from the current address. The instruction is not affected by page or block boundaries.

Remark "**Function**" in this section is applicable to the µPD750108 whose program counter consists of 13 bits (addr = 0000H to 1FFFH).
However, this is also applicable to the µPD750104 whose program counter consists of 12 bits (addr = 0000H to 0FFFH), the µPD750106 whose program counter consists of 13 bits (addr = 0000H to 17FFH), and the µPD75P0116 whose program counter consists of 14 bits (addr = 0000H to 3FFFH).

BRCB !caddr

Function: For the µPD750108 PC_{12-0} <– PC_{12} + $caddr_{11-0}$
caddr = n000H-nFFFH
n = PC_{12} = 0, 1

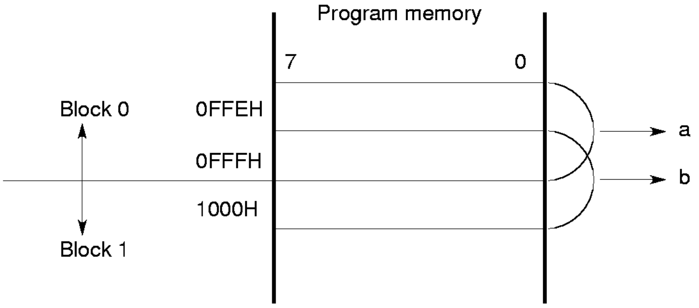
Branches to the address specified by the program counter whose low-order 12 bits (PC_{11-0}) have been replaced with the 12-bit immediate data caddr (A_{11-0}).

Since the program counter of the µPD750104 consists of 11 bits, this instruction enables a branch to any location in the program memory space.

In the µPD750106 and µPD750108, PC_{12} cannot be changed, so no branch occurs beyond the block.

Similarly, in the µPD75P0116, PC_{12} and PC_{13} cannot be changed, so no branch occurs beyond the block.

Caution **The BRCB !caddr instruction usually causes a branch within the block containing the instruction. However, if the first byte is located at address 0FFEH or 0FFFH, a branch to block 1 instead of block 0 occurs.**



If the BRCB !caddr instruction is located at a or b in the figure above, a branch to block 1 instead of block 0 occurs.

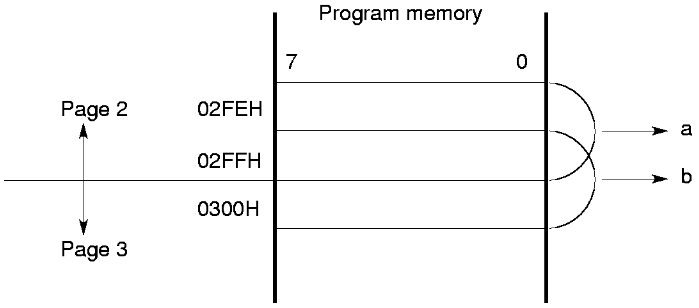
Remark "**Function**" in this section is applicable to the µPD750108 whose program counter consists of 13 bits (addr = 0000H to 1FFFH).
However, this is also applicable to the µPD750104 whose program counter consists of 12 bits (addr = 0000H to 0FFFH), the µPD750106 whose program counter consists of 13 bits (addr = 0000H to 17FFH), and the µPD75P0116 whose program counter consists of 14 bits (addr = 0000H to 3FFFH).

○ BR PCDE

Function: For the μPD750108 $PC_{12\text{-}0}$ <– $PC_{12\text{-}8}$ + DE
$PC_{7\text{-}4}$ <– D, $PC_{3\text{-}0}$ <– E

Branches to the address specified by the program counter whose low-order 8 bits ($PC_{7\text{-}0}$) have been replaced with the contents of the DE register pair. The high-order bits of the program counter are not affected.

Caution The BR PCDE instruction usually causes a branch within the page containing the instruction. However, if the first byte of the instruction code is located at address xxFEH or xxFFH, a branch to the next page instead of that page occurs.



If the BR PCDE instruction is located at a or b in the figure above, a branch to page 3 instead of page 2 occurs, jumping to the low-order 8 bits of the address specified by the contents of the DE register pair.

○ BR PCXA

Function: For the μPD750108 $PC_{12\text{-}0}$ <– $PC_{12\text{-}8}$ + XA
$PC_{7\text{-}4}$ <– X, $PC_{3\text{-}0}$ <– A

Branches to the address specified by the program counter whose low-order 8 bits ($PC_{7\text{-}0}$) have been replaced with the contents of the XA register pair. The high-order bits of the program counter are not affected.

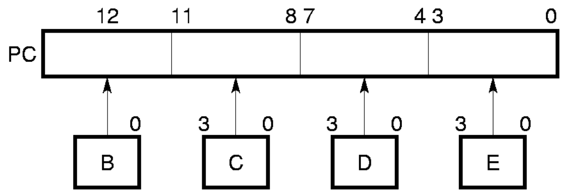
Caution As with the BR PCDE instruction, if the first byte is located at address xxFEH or xxFFH, a branch to the next page instead of the page containing the instruction occurs.

Remark "**Function**" in this section is applicable to the μPD750108 whose program counter consists of 13 bits (addr = 0000H to 1FFFH).
However, this is also applicable to the μPD750104 whose program counter consists of 12 bits (addr = 0000H to 0FFFH), the μPD750106 whose program counter consists of 13 bits (addr = 0000H to 17FFH), and the μPD75P0116 whose program counter consists of 14 bits (addr = 0000H to 3FFFH).

○ BR BCDE

Function: For the μPD750108 $PC_{12\text{-}0}$ <– BCDE

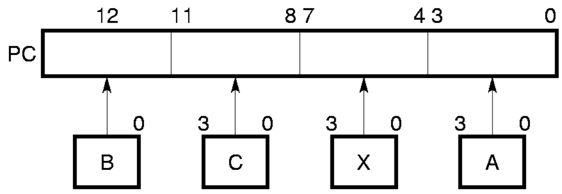
Branches to the address specified by the program counter whose bits have been replaced with the contents of the B_0, C, D, and E registers.



○ BR BCXA

Function: For the μPD750108 $PC_{12\text{-}0}$ <– BCXA

Branches to the address specified by the program counter whose bits have been replaced with the contents of the B_0, C, X, and A registers.



○ TBR addr

Function: Assembler pseudo instruction of the GETI instruction for table definition. This instruction is used to replace a 3-byte BR instruction with a 1-byte GETI instruction. The 12-bit address data must be coded in addr. For detailed information, refer to **RA75X Assembler Package User's Manual: Language (EEU-1363)**.

Remark "**Function**" in this section is applicable to the μPD750108 whose program counter consists of 13 bits (addr = 0000H to 1FFFH).

However, this is also applicable to the μPD750104 whose program counter consists of 12 bits (addr = 0000H to 0FFFH), the μPD750106 whose program counter consists of 13 bits (addr = 0000H to 17FFH), and the μPD75P0116 whose program counter consists of 14 bits (addr = 0000H to 3FFFH).

11.4.11 Subroutine Stack Control Instructions

II CALLA !addr1

Function: For the µPD750108

(SP–2) <– x, x, MBE, RBE, (SP–3) <– PC_{7-4}
(SP–4) <– $PC_{3-0,}$ (SP–5) <– 0, 0, 0, PC_{12}
(SP–6) <– PC_{11-8}
PC_{12-0} <– addr1, SP <– SP–6

I/II CALL !addr

Function: For the µPD750108

[Mk I mode]
(SP–1) <– PC_{7-4}, (SP–2) <– PC_{3-0}
(SP–3) <– MBE, RBE, 0, PC_{12}
(SP–4) <– PC_{11-8}, PC_{12-0} <– addr, SP <– SP–4

addr = 0000H - 1FFFH

[Mk II mode]
(SP–2) <– x, x, MBE, RBE
(SP–3) <– PC_{7-4}, (SP–4) <– PC_{3-0}
(SP–5) <– 0, 0, 0, PC_{12}, (SP–6) <– PC_{11-8}
PC_{12-0} <– addr, SP <– SP–6

addr = 0000H - 1FFFH

Saves the contents of the program counter (return address), memory bank enable flag (MBE), and register bank enable flag (RBE) to the data memory location (stack) addressed by the stack pointer (SP), then branches to the location addressed by the 14-bit immediate data addr after decrementing SP.

Remark "**Function**" in this section is applicable to the µPD750108 whose program counter consists of 13 bits (addr = 0000H to 1FFFH).
However, this is also applicable to the µPD750104 whose program counter consists of 12 bits (addr = 0000H to 0FFFH), the µPD750106 whose program counter consists of 13 bits (addr = 0000H to 17FFH), and the µPD75P0116 whose program counter consists of 14 bits (addr = 0000H to 3FFFH).

I/II CALLF !faddr

Function: **For the μPD750108**

[Mk I mode]
(SP–1) <– PC$_{7\text{-}4}$, (SP–2) <– PC$_{3\text{-}0}$
(SP–3) <– MBE, RBE, 0, PC$_{12}$
(SP–4) <– PC$_{11\text{-}8}$, SP <– SP – 4
PC$_{12\text{-}0}$ <– 00 + faddr

faddr = 0000H - 07FFH

[Mk II mode]
(SP–2) <– x, x, MBE, RBE
(SP–3) <– PC$_{7\text{-}4}$, (SP–4) <– PC$_{3\text{-}0}$
(SP–5) <– 0, 0, 0, PC$_{12}$, (SP–6) <– PC$_{11\text{-}8}$
SP <– SP–6
PC$_{12\text{-}0}$ <– 00 + faddr

faddr = 0000H - 07FFH

Saves the contents of the program counter (PC; Return address), memory bank enable flag (MBE), and register bank enable flag (RBE) to the data memory location (stack) addressed by the stack pointer (SP), then branches to the location addressed by the 11-bit immediate data faddr after decrementing SP. Only the address range 0000H-07FFH (0-2047) can be called.

TCALL !addr

Function: Assembler pseudo instruction of the GETI instruction for table definition. This instruction is used to replace a 3-byte CALL !addr instruction with a 1-byte GETI instruction. The 12-bit address data must be coded in addr. For detailed information, refer to **RA75X Assembler Package User's Manual: Language (EEU-1363)**.

Remark "**Function**" in this section is applicable to the μPD750108 whose program counter consists of 13 bits (addr = 0000H to 1FFFH).
However, this is also applicable to the μPD750104 whose program counter consists of 12 bits (addr = 0000H to 0FFFH), the μPD750106 whose program counter consists of 13 bits (addr = 0000H to 17FFH), and the μPD75P0116 whose program counter consists of 14 bits (addr = 0000H to 3FFFH).

I/II RET

Function: For the µPD750108

[Mk I mode] PC$_{11-8}$ <– (SP)
MBE, RBE, 0, PC$_{12}$ <– (SP+1)
PC$_{3-0}$ <– (SP+2)
PC$_{7-4}$ <– (SP+3), SP <– SP+4

[Mk II mode] PC$_{11-8}$ <– (SP), x, x, x, PC$_{12}$ <– (SP+1)
PC$_{3-0}$ <– (SP+2), PC$_{7-4}$ <– (SP+3)
x, x, MBE, RBE <– (SP+4)
SP <– SP+6

Restores the program counter (PC), memory bank enable flag (MBE), and register bank enable flag (RBE) with the data at the data memory location (stack) addressed by the stack pointer (SP), then increments the contents of SP.

Caution The program status word (PSW) is not restored except MBE and RBE.

Remark "**Function**" in this section is applicable to the µPD750108 whose program counter consists of 13 bits (addr = 0000H to 1FFFH).
However, this is also applicable to the µPD750104 whose program counter consists of 12 bits (addr = 0000H to 0FFFH), the µPD750106 whose program counter consists of 13 bits (addr = 0000H to 17FFH), and the µPD75P0116 whose program counter consists of 14 bits (addr = 0000H to 3FFFH).

I/II RETS

Function: For the µPD750108

[Mk I mode] PC$_{11-8}$ <– (SP)
MBE, 0, 0, PC$_{12}$ <– (SP+1)
PC$_{3-0}$ <– (SP+2), PC$_{7-4}$ <– (SP+3), SP <– SP+4
Then skip unconditionally

[Mk II mode] PC$_{11-8}$ <– (SP), 0, 0, 0, PC$_{12}$ <– (SP+1)
PC$_{3-0}$ <– (SP+2), PC$_{7-4}$ <– (SP+3)
x, x, MBE, RBE <– (SP+4)
SP <– SP+6
Then skip unconditionally

Restores the program counter (PC), memory bank enable flag (MBE), and register bank enable flag (RBE) with the data at the data memory location (stack) addressed by the stack pointer (SP), then skips unconditionally after incrementing the contents of SP.

Caution The program status word (PSW) is not restored except MBE and RBE.

Remark "**Function**" in this section is applicable to the μPD750108 whose program counter consists of 13 bits (addr = 0000H to 1FFFH).
However, this is also applicable to the μPD750104 whose program counter consists of 12 bits (addr = 0000H to 0FFFH), the μPD750106 whose program counter consists of 13 bits (addr = 0000H to 17FFH), and the μPD75P0116 whose program counter consists of 14 bits (addr = 0000H to 3FFFH).

I/II RETI

Function: For the μPD750108

[Mk I mode] $PC_{11\text{-}8}$ <– (SP), MBE, RBE, 0, PC_{12} <– (SP+1)
$PC_{3\text{-}0}$ <– (SP+2)
$PC_{7\text{-}4}$ <– (SP+3)
PSW_L <– (SP+4), PSW_H <– (SP+5)
SP <– SP+6

[Mk II mode] $PC_{11\text{-}8}$ <– (SP), 0, 0, 0, PC_{12} <– (SP+1)
$PC_{3\text{-}0}$ <– (SP+2)
$PC_{7\text{-}4}$ <– (SP+3),
PSW_L <– (SP+4), PSW_H <– (SP+5)
SP <– SP+6

Restores the program counter (PC) and program status word with the data at the data memory location (stack) addressed by the stack pointer (SP), then increments the contents of SP.

This instruction is used when control is returned from an interrupt service routine.

Remark "**Function**" in this section is applicable to the μPD750108 whose program counter consists of 13 bits (addr = 0000H to 1FFFH).
However, this is also applicable to the μPD750104 whose program counter consists of 12 bits (addr = 0000H to 0FFFH), the μPD750106 whose program counter consists of 13 bits (addr = 0000H to 17FFH), and the μPD75P0116 whose program counter consists of 14 bits (addr = 0000H to 3FFFH).

PUSH rp

Function: (SP–1) <– rp_H, (SP–2) <– rp_L, SP <– SP–2

Saves the contents of register pair rp (XA, HL, DE, BC) to the data memory location (stack) addressed by the stack pointer (SP), then decrements SP.

The high-order part of a register pair (rp_H: X, H, D, B) is saved to the stack location addressed by (SP–1), and the low-order part (rp_L: A, L, E, C) is saved to the stack location addressed by (SP–2).

PUSH BS

Function: (SP–1) <– MBS, (SP–2) <– RBS, SP <– SP–2

Saves the contents of the memory bank select register (MBS) and the register bank select register (RBS) to the data memory location (stack) addressed by the stack pointer (SP), then decrements SP.

POP rp

Function: rp_L <– (SP), rp_H <– (SP+1), SP <– SP+2

Restores register pair rp (XA, HL, DE, BC) with the data at the data memory location (stack) addressed by the stack pointer (SP), then increments SP.

The low-order part of a register pair (rp_L: A, L, E, C) is restored from the contents of (SP), and the high-order part (rp_H: X, H, D, B) is restored with the contents of (SP+).

POP BS

Function: RBS <– (SP), MBS <– (SP+1), SP <– SP+2

Restores the register bank select register (RBS) and the memory bank select register (MBS) with the data at the data memory location (stack) addressed by the stack pointer (SP), then increments SP.

11.4.12 Interrupt Control Instructions

EI

Function: IME (IPS.3) <– 1

Sets the interrupt master enable flag (bit 3 of the interrupt priority specification register) to 1 to enable interrupts. Whether to accept an interrupt is controlled with the corresponding interrupt enable flag.

EI IExxx

Function: IExxx <– 1 xxx = N_5, N_{2-0}

Sets an interrupt enable flag (IExxx) to 1 to enable an interrupt. (xxx = BT, CSI, T0, T1, W, 0, 1, 2, 4)

DI

Function: IME (IPS.3) <– 0

Resets the interrupt master enable flag (bit 3 of the interrupt priority specification register) to 0 to disable all interrupts regardless of the states of the interrupt enable flags.

DI IExxx

Function: IExxx <– 0   xxx = N_5, N_{2-0}

Resets an interrupt enable flag (IExxx) to 0 to disable an interrupt. (xxx = BT, CSI, T0, T1, W, 0, 1, 2, 4)

11.4.13 I/O Instructions

IN A,PORTn

Function: A <– PORTn   n = N_{3-0}: 0-8

Transfers the contents of the port specified by PORTn (n = 0-8) to the A register.

Caution Before this instruction can be executed, MBE = 0 or (MBE = 1, MBS = 15) must be set. A number from 0 to 8 can be specified as n. Depending on I/O mode specification, output latch data (in the output mode) or pin data (in the input mode) are transferred.

IN XA,PORTn

Function: A <– PORTn, X <– $PORT_{n+1}$ n = N_{3-0}: 4, 6

Transfers the contents of the port specified by PORTn (n = 4 or 6) to the A register, then transfers the contents of the next port to the X register.

Caution Only the number 4 or 6 can be specified as n. Before this instruction can be executed, MBE = 0 or (MBE = 1, MBS = 15) must be set. Depending on I/O mode specification, output latch data (in the output mode) or pin data (in the input mode) are transferred.

OUT PORTn, A

Function: PORTn <– A   n = N_{3-0}: 2-8

Transfers the contents of the A register to the output latch of the port specified by PORTn (n = 2–8).

Caution Before this instruction can be executed, MBE = 0 or (MBE = 1, MBS = 15) must be set. A number from 2 to 8 can be specified as n.

OUT PORTn, XA

Function: PORTn <– A, $PORT_{n+1}$ <– X   n = N_{3-0}: 4, 6

Transfers the contents of the A register to the output latch of the port specified by PORTn (n = 4, 6), then transfers the contents of the X register to the output latch of the next port.

Caution Before this instruction can be executed, MBE = 0 or (MBE = 1, MBS = 15) must be set. Only 4 or 6 can be specified as n.

11.4.14 CPU Control Instructions

HALT

Function: PCC.2 <– 1

Sets the HALT mode. (This instruction is used to set bit 2 of the processor clock control register.)

Caution The instruction immediately following a HALT instruction must be a NOP instruction.

STOP

Function: PCC.3 <– 1

Sets the STOP mode. (This instruction is used to set bit 3 of the processor clock control register.)

Caution The instruction immediately following a STOP instruction must be a NOP instruction.

NOP

Function: Uses one machine cycle without performing an action.

11.4.15 Special Instructions

SEL RBn

Function: RBS <– n n = N_{1-0}: 0-3

Sets the 2-bit immediate data n in the register bank select register (RBS).

SEL MBn

Function: MBS <– n n = N_{3-0}: 0, 1, 15

Transfers the 4-bit immediate data n to the memory bank select register (MBS).
Only 0, 1, or 15 can be specified as n.

I/II GETI taddr

Function: taddr = T_{5-0}, 0 : 20H-7FH

For the µPD750108

[Mk I mode]

- **When a table defined by the TBR instruction is referenced**
  PC_{12-0} <– $(taddr)_{4-0}$ + (taddr+1)
- **When a table defined by the TCALL instruction is referenced**
  (SP–1) <– PC_{7-4}, (SP–2) <– PC_{3-0}
  (SP–3) <– MBE, RBE, 0, PC_{12}
  (SP–4) <– PC_{11-8}
  PC_{12-0} <– $(taddr)_{4-0}$ + (taddr+1)
  SP <– SP–4
- **When a table defined by an instruction other than the TBR or TCALL instruction is referenced**
  An instruction using (taddr) (taddr+1) as its operation code is executed.

[Mk II mode]

- **When a table defined by the TBR instruction is referenced**
  PC_{12-0} <– $(taddr)_{4-0}$ + (taddr+1)
- **When a table defined by the TCALL instruction is referenced**
  (SP–2) <– x, x, MBE, RBE
  (SP–3) <– PC_{7-4}, (SP–4) <– PC_{3-0}
  (SP–5) <– 0, 0, 0, PC_{12}, (SP–6) <– PC_{11-8}
  PC_{12-0} <– $(taddr)_{4-0}$ + (taddr+1)
  SP <– SP–6
- **When a table defined by an instruction other than the TBR or TCALL instruction is referenced**
  An instruction using (taddr) (taddr+1) as its operation code is executed.

Remark "**Function**" in this section is applicable to the µPD750108 whose program counter consists of 13 bits (addr = 0000H to 1FFFH).
However, this is also applicable to the µPD750104 whose program counter consists of 12 bits (addr = 0000H to 0FFFH), the µPD750106 whose program counter consists of 13 bits (addr = 0000H to 17FFH), and the µPD75P0116 whose program counter consists of 14 bits (addr = 0000H to 3FFFH).

The 2-byte data at the program memory addresses specified by (taddr) and (taddr+1) is referenced and executed as an instruction.

Addresses 0020H to 007FH are used as a reference table area. Data must be written to this area beforehand. When a 1-byte instruction or 2-byte instruction is written, its mnemonic can be used directly.

For a 3-byte call instruction or 3-byte branch instruction, an assembler pseudo instruction (TCALL, TBR) is used.

Only an even address can be specified as taddr.

Caution All 2-byte instructions (except the BRCB instruction and CALLF instruction) set in the reference table must be 2-machine-cycle instructions. Pairs of 1-byte instructions can be set as indicated in the table below.

First byte instruction	Second byte instruction
MOV A,@HL MOV @HL,A XCH A,@HL	INCS L DECS L INCS H DECS H INCS HL
MOV A,@DE XCH A,@DE	INCS E DECS E INCS D DECS D INCS DE
MOV A,@DL XCH A,@DL	INCS L DECS L INCS D DECS D

The PC is not incremented during execution of a GETI instruction, so that after a reference instruction is executed, execution is resumed starting at the address immediately after the GETI instruction.

If the instruction immediately preceding a GETI instruction has the skip function, the GETI instruction is skipped as with other 1-byte instructions. If an instruction referenced with a GETI instruction has the skip function, the instruction immediately following the GETI instruction is skipped.

If a GETI instruction references an instruction having a string effect, the following processing is performed:

- If the instruction immediately preceding the GETI instruction also has the string effect in the same group, the execution of the GETI instruction cancels the string effect, and the referenced instruction is not skipped.
- If the instruction immediately following the GETI instruction also has the string effect of the same group, the string effect of the referenced instruction remains valid, and the next instruction is skipped.

Example

```
MOV  HL, #00H
MOV  XA, #FFH      are replaced with GETI instructions.
CALL SUB1
BR   SUB2
          ORG       20H
HL00:     MOV       HL, #00H
XAFF:     MOV       XA, #FFH
CSUB1:    TCALL     SUB1
BSUB2:    TBR       SUB2
          :
          :
          :
          :
          GET HL00          ; MOV  HL,#00H
          :
          :
          :
          :
          GETI BSUB2        ; BR  SUB2
          :
          :
          :
          :
          GETI CSUB1        ; CALL  SUB1
          :
          :
          :
          :
          GETI XAFF         ; MOV  XA,#FFH
```

APPENDIX A FUNCTIONS OF THE µPD75008, µPD750108, AND µPD75P0116

(1/2)

Item		µPD75008	µPD750108[Note]	µPD75P0116[Note]
Program memory		Masked ROM 0000H - 1F7FH (8064 x 8 bits)	Masked ROM 0000H - 1FFFH (8192 x 8 bits)	One-time PROM 0000H - 3FFFH (16384 x 8 bits)
Data memory		000H - 1FFH (512 x 4 bits)		
CPU		75X standard CPU	75XL CPU (equivalent to the 75X high-end CPU)	
Main system clock oscillator		Crystal/ceramic oscillator	RC oscillator	
Subsystem clock oscillator		Crystal oscillator		
Time required for start after reset		31.3 ms	56/f_{CC} (28 µs: when operating at 2 MHz, 56 µs: when operating at 1 MHz)	
Wait time applied when STOP mode is released by an interrupt		$2^{20}/f_X$, $2^{17}/f_X$, $2^{15}/f_X$, $2^{13}/f_X$ (selected according to BTM setting)	$2^9/f_{CC}$ or no wait (selected using a mask option)	$2^9/f_{CC}$
Instruction execution time	When selecting the main system clock	0.95, 1.91, 15.3 µs (when operating at 4.19 MHz)	• 4, 8, 16, 64 µs (when operating at 1 MHz) • 2, 4, 8, 32 µs (when operating at 2 MHz)	
	When selecting the subsystem clock	122 µs (when operating at 32.768 kHz)		
Pin connection	20 (CU)	NC	IC	V_{PP}
	38 (GB)			
	24 (CU)	P21	P21/PTO1	
	42 (GB)			
	6 - 9 (CU)	P33 - P30		P33/MD3 - P30/MD0
	23-26 (GB)			

Stack	SBS register	Not provided	Provided	SBS.3 = 1 : Mk I mode selection
				SBS.3 = 0 : Mk II mode selection

	Stack area	000H - 0FFH	n00H - nFFH (n = 0, 1)	
	Stack operation for a subroutine call instruction	2-byte stack	Mk I mode: 2-byte stack Mk II mode: 3-byte stack	

Note Under development

(2/2)

Item		µPD75008	µPD750108[Note]	µPD75P0116[Note]
Instruction	BRA !addr1 CALLA !addr1	Not available	Mk I mode: Not available Mk II mode: Available	
	MOVT XA, @BCDE/@BCXA BR BCDE/BCXA		Available	
	CALL !addr	3 machine cycles	Mk I mode: 3 machine cycles, Mk II mode: 4 machine cycles	
	CALLF !faddr	2 machine cycles	Mk I mode: 2 machine cycles, Mk II mode: 3 machine cycles	
Timer		3 channels • Basic interval timer: 1 • Timer/event counter: 1 • Clock timer: 1	4 channels • Basic interval timer/watchdog timer: 1 • Timer/event counter: 1 • Timer counter: 1 • Clock timer: 1	
Clock output (PCL)		Φ, 524, 262, 65.5 kHz (when the main system clock operates at 4.19 MHz)	• Φ, 125, 62.5, 15.6 kHz (when the main system clock operates at 1 MHz) • Φ, 250, 125, 31.3 kHz (when the main system clock operates at 2 MHz)	
BUZ output (BUZ)		2 kHz	• 2, 4, 32 kHz (when the main system clock operates at 2 MHz or the subsystem clock operates at 32.768 kHz) • 0.488, 0.977, 7.8125 kHz (when the main system clock operates at 1 MHz)	
Serial interface		3 modes supported • Three-wire serial I/O mode: First transferred bit switchable between the LSB and MSB • Two-wire serial I/O mode • SBI mode		
SOS register	Feedback resistor cut flag (SOS.0)	Can incorporate feedback resistors that are specified with the mask option.	Incorporated	
	Sub-oscillator current cut flag (SOS.1)	Not provided	Incorporated	
Register bank selection register (RBS)		Not provided	Provided	
Standby release with INT0		Disable	Enable	
Number of vectored interrupts		External: 3, internal: 3	External: 3, internal: 4	
Processor clock control register		Available when PCC is 0, 2, or 3	Available when PCC is 0 to 3	
Power supply voltage		V_{DD} = 2.7 to 6.0 V	V_{DD} = 1.8 to 5.5 V	
Operating ambient temperature		T_A = –40 to +85 ˚C		
Package		• 42-pin plastic shrink DIP (600 mil) • 44-pin plastic QFP (10 x 10 mm)		

Note Under development

APPENDIX B DEVELOPMENT TOOLS

The following development tools are provided for the development of a system which employs the μPD750108. In the 75XL series, use the common relocatable assembler together with a device file of each model.

RA75X relocatable assembler	Host machine			Part number
		OS	Distribution media	
	PC-9800 series	MS-DOS (Ver. 3.30 to Ver. 6.2[Note])	3.5-inch 2HD	μS5A13RA75X
			5.25-inch 2HD	μS5A10RA75X
	IBM PC/AT™ and compatibles	See "**OS for IBM PC**."	3.5-inch 2HC	μS7B13RA75X
			5.25-inch 2HC	μS7B10RA75X

Device file	Host machine			Part number
		OS	Distribution media	
	PC-9800 series	MS-DOS (Ver. 3.30 to Ver. 6.2[Note])	3.5-inch 2HD	μS5A13DF750008
			5.25-inch 2HD	μS5A10DF750008
	IBM PC/AT and compatibles	See "**OS for IBM PC**."	3.5-inch 2HC	μS7B13DF750008
			5.25-inch 2HC	μS7B10DF750008

Note These software products cannot use the task swap function, which is available in MS-DOS Ver. 5.00 or later.

Remark The operations of the assembler and device file are guaranteed only on the above host machines and OSs.

PROM programming tools

Hardware	PG-1500	The PG-1500 PROM programmer is used together with an accessory board and optional program adapter. It allows the user to program a single chip microcomputer containing PROM from a standalone terminal or a host machine. The PG-1500 can be used to program typical 256K-bit to 4M-bit PROMs.			
	PA-75P008CU	The PA-75P008CU is a PROM programmer adapter provided for the μPD75P008CU/GB, μPD75P0016CU/GB, and μPD75P0116CU/GB. It is used in conjunction with the PG-1500.			
Software	PG-1500 controller	This program enables the host machine to control the PG-1500 through the serial and parallel interfaces.			
		Host machine			Part number
			OS	Distribution media	
		PC-9800 series	MS-DOS (Ver. 3.30 to Ver. 6.2[Note])	3.5-inch 2HD	μS5A13PG1500
				5.25-inch 2HD	μS5A10PG1500
		IBM PC/AT and compatibles	See "**OS for IBM PC**."	3.5-inch 2HD	μS7B13PG1500
				5.25-inch 2HC	μS7B10PG1500

Note These software products cannot use the task swap function, which is available in MS-DOS Ver. 5.00 or later.

Remark Operation of the PG-1500 controller is guaranteed only on the above host machines and OSs.

Debugging tools

The in-circuit emulators (IE-75000-R and IE-75001-R) are provided to debug programs used for the µPD750108.

The following system is shown below.

Hardware	IE-75000-R[Note 1]	The IE-75000-R is an in-circuit emulator used to debug hardware and software when developing an application system using the 75X series and 75XL series. Use this emulator together with optional emulation board IE-75300-R-EM and emulation probe to develop application systems of the µPD750108 subseries. For efficient debugging, connect the emulator to the host machine and a PROM programmer. The IE-75000-R contains emulation board IE-75000-R-EM. The board is connected to the IE-75000-R.
	IE-75001-R	The IE-75001-R is an in-circuit emulator used to debug hardware and software when developing an application system using the 75X series and 75XL series. Use this emulator together with optional emulation board IE-75300-R-EM and emulation probe. For efficient debugging, connect the emulator to the host machine and a PROM programmer.
	IE-75300-R-EM[Note 2]	The IE-75300-R-EM is an emulation board used to evaluate an application system using the µPD750108 subseries. Use this board together with the IE-75000-R or IE-75001-R.
	EP-75008GB-R	The EP-75008GB-R is an emulation probe for the µPD75008GB, µPD750008GB, and µPD750108GB. Connect this emulation probe to the IE-75000-R or IE-75001-R, and the IE-75300-R-EM.
	EV-9200G-44	A 44-pin conversion socket, the EV-9200G-44, supplied with this probe facilitates the connection of the probe to the target system.
	EP-75008CU-R	The EP-75008CU-R is an emulation probe for the µPD75008CU, µPD750008CU, and µPD750108CU. Connect this emulation probe to the IE-75000-R or IE-75001-R, and the IE-75300-R-EM.
Software	IE control program	This program enables the host machine to control the IE-75000-R or IE-75001-R through the RS-232-C and Centronics interface.

Host machine			Part number
	OS	Distribution media	
PC-9800 series	MS-DOS Ver. 3.30 to Ver. 6.2[Note 3]	3.5-inch 2HD	µS5A13IE75X
		5.25-inch 2HD	µS5A10IE75X
IBM PC/AT and compatibles	See "**OS for IBM PC**."	3.5-inch 2HC	µS7B13IE75X
		5.25-inch 2HC	µS7B10IE75X

Notes 1. Maintenance service only

2. To be ordered.

3. These software products cannot use the task swap function, which is available in MS DOS Ver. 5.00 or later.

Remark Operation of the IE control program is guaranteed only on the above host machines and OSs.

OS for IBM PC

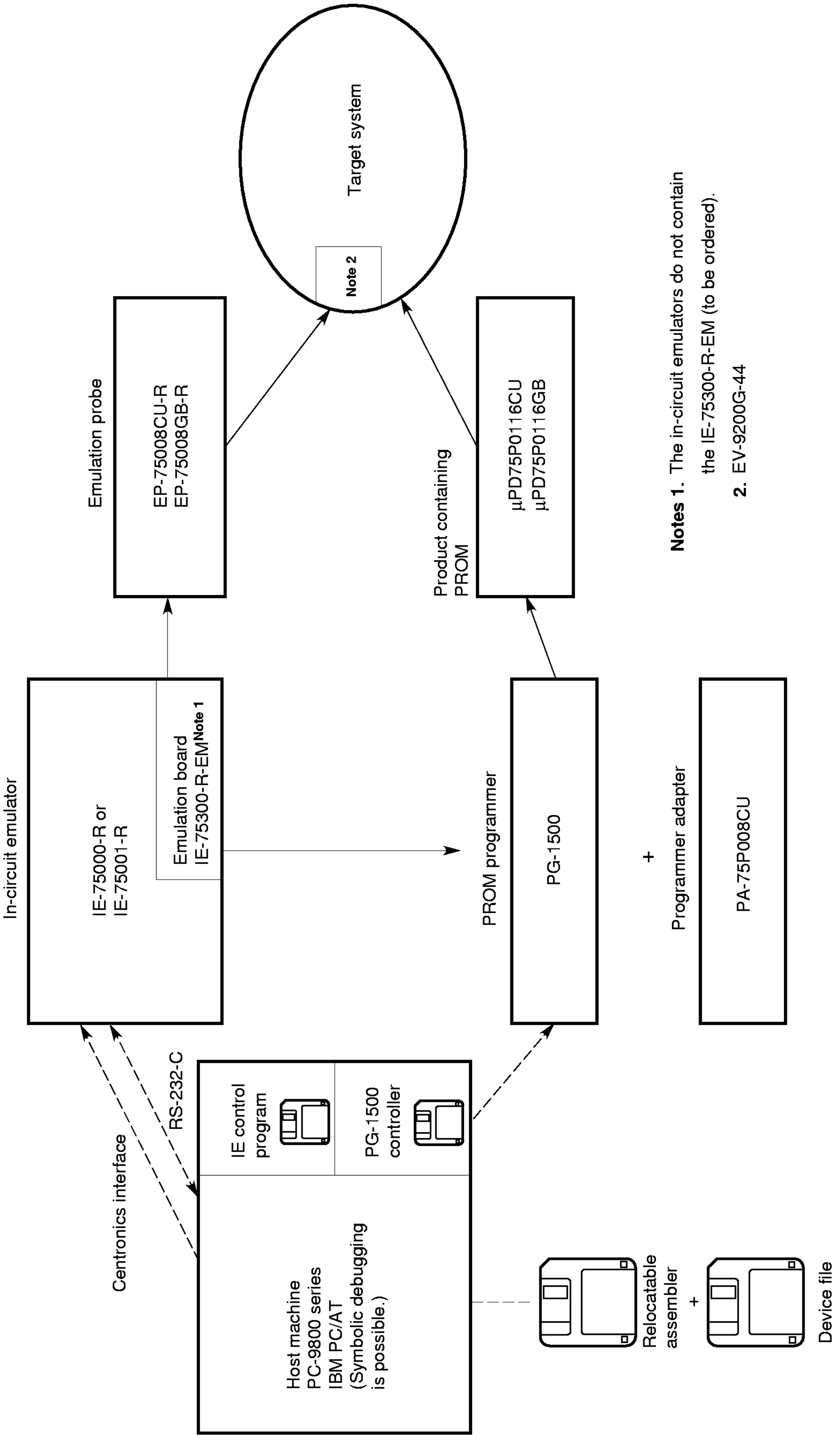
The following IBM PC OSs are supported.

OS	Version
PC DOS	Ver. 5.02 to Ver. 6.3 J6.1/V[Note] to J6.3/V[Note]
MS-DOS	Ver. 5.0 to Ver. 6.22 5.0/V[Note] to 6.2/V[Note]
IBM DOS™	J5.02/V[Note]

Note Only English version is supported.

Caution These software products cannot use the task swap function, which is available in MS-DOS Ver. 5.00 or later.

Development Tool Configuration



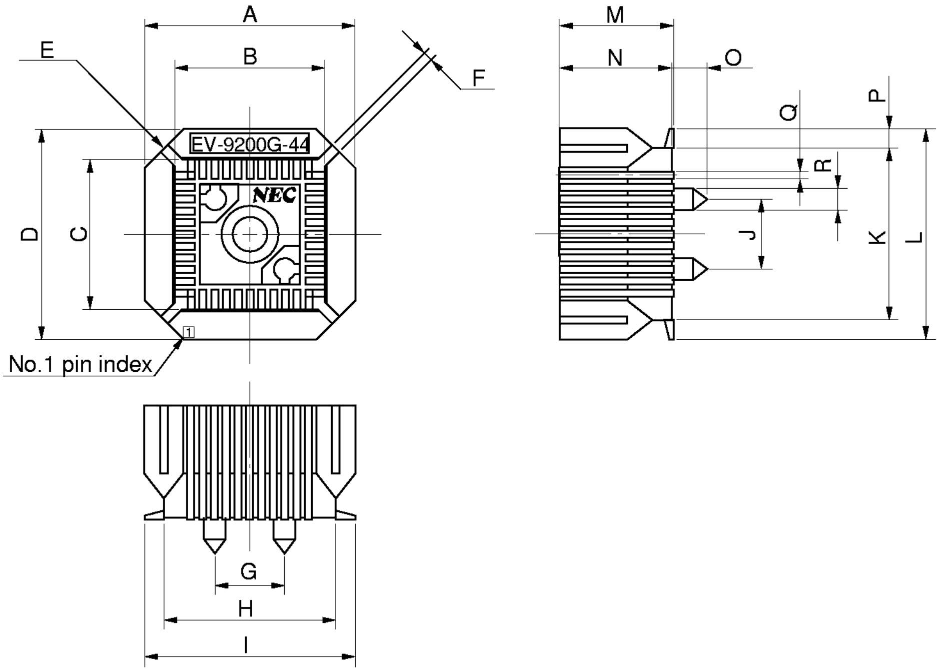
Notes 1. The in-circuit emulators do not contain the IE-75300-R-EM (to be ordered).

2. EV-9200G-44

Drawings of the Conversion Socket (EV-9200G-44) and Recommended Pattern on Boards

Figure B-1. Drawings of the EV-9200G-44 (Reference)

Based on EV-9200G-44
(1) Package drawing (in mm)

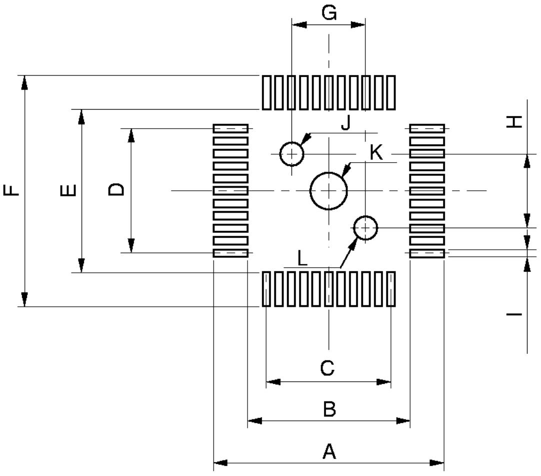


EV-9200G-44-G0

ITEM	MILLIMETERS	INCHES
A	15.0	0.591
B	10.3	0.406
C	10.3	0.406
D	15.0	0.591
E	4-C 3.0	4-C 0.118
F	0.8	0.031
G	5.0	0.197
H	12.0	0.472
I	14.7	0.579
J	5.0	0.197
K	12.0	0.472
L	14.7	0.579
M	8.0	0.315
O	7.8	0.307
N	2.0	0.079
P	1.35	0.053
Q	0.35±0.1	$0.014^{+0.004}_{-0.005}$
R	φ1.5	φ0.059

Figure B-2. Recommended Pattern on Boards for the EV-9200G-44 (Reference)

Based on EV-9200G-44
(2) Pad drawing (in mm)



EV-9200G-44-P0

ITEM	MILLIMETERS	INCHES
A	15.7	0.618
B	11.0	0.433
C	0.8±0.02 × 10=8.0±0.05	$0.031^{+0.002}_{-0.001} \times 0.394 = 0.315^{+0.002}_{-0.002}$
D	0.8±0.02 × 10=8.0±0.05	$0.031^{+0.002}_{-0.001} \times 0.394 = 0.315^{+0.002}_{-0.002}$
E	11.0	0.433
F	15.7	0.618
G	5.00±0.08	$0.197^{+0.003}_{-0.004}$
H	5.00±0.08	$0.197^{+0.003}_{-0.004}$
I	0.5±0.02	$0.02^{+0.001}_{-0.002}$
J	ϕ1.57±0.03	$\phi 0.062^{+0.001}_{-0.002}$
K	ϕ2.2±0.1	$\phi 0.087^{+0.004}_{-0.005}$
L	ϕ1.57±0.03	$\phi 0.062^{+0.001}_{-0.002}$

Caution Dimensions of mount pad for EV-9200 and that for target device (QFP) may be different in some parts. For the recommended mount pad dimensions for QFP, refer to "SEMICONDUCTOR DEVICE MOUNTING TECHNOLOGY MANUAL" (IEI-1207).

Caution Dimensions of mount pad for EV-9200 and that for target device (QFP) may be different in some parts. For the recommended mount pad dimensions for QFP, refer to "SEMICONDUCTOR DEVICE MOUNTING TECHNOLOGY MANUAL" (C10535E).

[MEMO]

APPENDIX C MASKED ROM ORDERING PROCEDURE

After program development is completed, the masked ROM is ordered by the following procedure:

<1> **Advance notice of an order for masked ROM**

Give advance notice of masked ROM ordering to a special agent or NEC's Sales Department, otherwise the ordered products may be delivered with delay.

<2> **Preparation of media for ordering**

Use three UV-EPROMs having the same contents, or 3.5- or 5.25-inch IBM format floppy disk in ordering a masked ROM. (Prepare a mask option information sheet describing the mask option data.)

<3> **Preparation of the required documents**

Prepare the following documents when ordering a masked ROM:

- Masked ROM order sheet
- Masked ROM order check sheet
- Mask option information sheet

<4> **Ordering**

Send a set of the media created in **<2>** and the documents created in **<3>** to a special agent or NEC's Sales Department by the date indicated in the advance notice.

[MEMO]

APPENDIX D INSTRUCTION INDEX

D.1 INSTRUCTION INDEX (BY FUNCTION)

D.2 INSTRUCTION INDEX (ALPHABETICAL ORDER)

APPENDIX E HARDWARE INDEX

E.1 HARDWARE INDEX (ALPHABETICAL ORDER WITH RESPECT TO THE HARDWARE NAME)

E.2 HARDWARE INDEX (ALPHABETICAL ORDER WITH RESPECT TO THE HARDWARE SYMBOL)