User's Manual

NEC

μPD750068

4-bit Single-Chip Microcontrollers

μPD750064
μPD750066
μPD750068
μPD75P0076

Document No. U10670EJ2V1UM00 (2nd edition)
Date Published November 1999 N CP (K)

[MEMO]

NOTES FOR CMOS DEVICES

① PRECAUTION AGAINST ESD FOR SEMICONDUCTORS

Note:

Strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it once, when it has occurred. Environmental control must be adequate. When it is dry, humidifier should be used. It is recommended to avoid using insulators that easily build static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work bench and floor should be grounded. The operator should be grounded using wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with semiconductor devices on it.

② HANDLING OF UNUSED INPUT PINS FOR CMOS

Note:

No connection for CMOS device inputs can be cause of malfunction. If no connection is provided to the input pins, it is possible that an internal input level may be generated due to noise, etc., hence causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using a pull-up or pull-down circuitry. Each unused pin should be connected to V_{DD} or GND with a resistor, if it is considered to have a possibility of being an output pin. All handling related to the unused pins must be judged device by device and related specifications governing the devices.

③ STATUS BEFORE INITIALIZATION OF MOS DEVICES

Note:

Power-on does not necessarily define initial status of MOS device. Production process of MOS does not define the initial operation status of the device. Immediately after the power source is turned ON, the devices with reset function have not yet been initialized. Hence, power-on does not guarantee out-pin levels, I/O settings or contents of registers. Device is not initialized until the reset signal is received. Reset operation must be executed immediately after power-on for devices having reset function.

The export of this product from Japan is regulated by the Japanese government. To export this product may be prohibited without governmental license, the need for which must be judged by the customer. The export or re-export of this product from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC Electronics (Germany) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 02
Fax: 0211-65 03 490

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Italiana s.r.l.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (Germany) GmbH
Benelux Office
Eindhoven, The Netherlands
Tel: 040-2445845
Fax: 040-2444580

NEC Electronics (France) S.A.
Velizy-Villacoublay, France
Tel: 01-30-67 58 00
Fax: 01-30-67 58 99

NEC Electronics (France) S.A.
Spain Office
Madrid, Spain
Tel: 91-504-2787
Fax: 91-504-2860

NEC Electronics (Germany) GmbH
Scandinavia Office
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
United Square, Singapore 1130
Tel: 65-253-8311
Fax: 65-250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377
Fax: 02-2719-5951

NEC do Brasil S.A.
Electron Devices Division
Rodovia Presidente Dutra, Km 214
07210-902-Guarulhos-SP Brasil
Tel: 55-11-6465-6810
Fax: 55-11-6465-6829

J99.1

Major Revisions in This Version

Page	Contents
Throughout	The μPD750064, 750068 and 75P0076 changed from "under development" to "development completed".
	The μPD750066 added.
	Data bus pins (D0-D7) added.
	The XT2 open changed to the XT1 complement input when an external clock is used.
p.21	**2.4 Processing of Unused Pins** changed.
p.98	Table and caution of **Table 5-5. Maximum Time Required to Select System Clock and CPU Clock** changed.
p.222	**7.4 Selecting Mask Option** added.
p.233	**9.2 Writing Program Memory** changed.
p.234	**9.3 Reading Program Memory** changed.
p.237	**10.3 Subsystem Clock Feedback Resistor Mask Option** added.
p.268	Modification of the instruction list in **11.3 Op Code of Each Instruction**.
p.315	Version of the supported OS updated in **APPENDIX B DEVELOPMENT TOOLS**.
p.321	Procedure for ordering supply media changed in **APPENDIX C ORDERING MASK ROM**.
p.331	**APPENDIX F REVISION HISTORY** added.

The mark ★ shows major revised points.

INTRODUCTION

Readers This manual is intended for engineers who understand the functions of the μPD750064, 750066, 750068 and 75P0076 and wish to design application systems using any of these microcontrollers.

Purpose This manual describes the hardware functions of the μPD750064, 750066, 750068 and 75P0076 organized in the following manner.

Organization This manual contains the following information:

- General
- Pin Functions
- Features of Architecture and Memory Map
- Internal CPU Functions
- Peripheral Hardware Functions
- Interrupt Functions and Test Functions
- Standby Functions
- Reset Function
- Write and Verify PROM
- Mask option
- Instruction Set

How to read this manual It is assumed that the readers of this manual possess general knowledge about electronics, logic circuits, and microcomputers.

- **If you have some experience of using the μPD75068,**
  → Read **APPENDIX A FUNCTIONS OF μPD75068, 750068, AND 75P0076** to check differences between the μPD75316B and the microcontrollers described in this manual.

- **If you intend to use this manual as a manual for the μPD750064, 750066, 750068, or 75P0076,**
  → Unless otherwise specified, the μPD750068 is regarded as the representative model. Descriptions throughout this manual correspond to this model. Refer to **1.3 Differences among μPD750068 Subseries Products** to check the differences among the various models, and substitute the appropriate product name for the μPD750068.

- **To check the functions of an instruction whose mnemonic is known,**
  → Refer to **APPENDIX D INSTRUCTION INDEX**.

- **To check the functions of a specific internal circuit,**
  → Refer to **APPENDIX E HARDWARE INDEX**.

- **To understand the overall functions of the μPD750064, 750066, 750068 and 75P0076,**
  → Read this manual in the order of the Table of Contents.

Legend

Data significance	:	Left: higher, right: lower
Active low	:	$\overline{\times\times\times}$ (top bar over signal or pin name)
Address of memory map	:	Top: low, Bottom: high
Note	:	Points to be noted
Caution	:	Important information
Remark	:	Supplement
Numeric notation	:	Binary ... ×××× or ××××B
		Decimal ... ××××
		Hexadecimal ... ××××H

★ **Related documents** Some documents are preliminary editions but they are not so specified in the following tables.

Documents related to devices

Document Name	Document Number	
	Japanese	English
μPD750064, 750066, 750068, 750064(A), 750066(A), 750068(A) Data Sheet	U10165J	U10165E
μPD75P0076 Data Sheet	U10232J	U10232E
μPD750068 User's Manual	U10670J	U10670E (this manual)
μPD750068 Instruction List	IEM-5606	—
75XL Series Selection Guide	U10453J	U10453E

Documents related to development tools

Document Name			Document Number	
			Japanese	English
Hardware	IE-75000-R/IE-75001-R User's Manual		EEU-846	EEU-1416
	IE-75300-R-EM User's Manual		U11354J	U11354E
	EP-750068CU/GT-R User's Manual		U10950J	U10950E
	PG-1500 User's Manual		EEU-651	EEU-1335
Software	RA75X Assembler Package User's Manual	Operation	EEU-731	EEU-1346
		Language	EEU-730	EEU-1363
	PG-1500 Controller User's Manual	PC-9800 series (MS-DOS) base	EEU-704	EEU-1291
		IBM PC series (PC DOS) base	EEU-5008	U10540E

Other documents

Document Name	Document Number	
	Japanese	English
SEMICONDUCTORS SELECTION GUIDE Products & Packages (CD-ROM)	X13769X	
Semiconductor Device Mounting Technology Manual	C11535J	C11535E
Quality Grades of NEC's Semiconductor Devices	C11531J	C11531E
NEC Semiconductor Device Reliability and Quality Control System	C10983J	C10983E
Guide to Prevent Damage for Semiconductor Devices by Electrostatic Discharge (ESD)	C11892J	C11892E
Microcomputer-Related Products Guide - By Third Parties	U11416J	—

Caution These related documents are subject to change without notice. Be sure to use the latest edition of the documents when you design your system.

[MEMO]

TABLE OF CONTENTS

LIST OF FIGURES (1/3)

LIST OF FIGURES (2/3)

LIST OF FIGURES (3/3)

LIST OF TABLES (1/2)

LIST OF TABLES (2/2)

[MEMO]

CHAPTER 1 GENERAL

The μPD750064, 750066, 750068 and 75P0076 are 4-bit single-chip microcontrollers in the NEC 75XL series, the successor to the 75X series that boasts a wealth of variations. The μPD750068 subseries is a generic name that stands for the μPD750064, 750066, 750068, and 750076.

The μPD750068 is based on the existing μPD75068 but has a higher ROM capacity and more sophisticated CPU functions. It can operate at high speeds at a voltage of as low as 1.8 V.

This model is available in a small plastic shrink SOP (375 mil, 0.8 mm pitch).

The features of the μPD750068 are as follows:

- Low-voltage operation: V_{DD} = 1.8 to 5.5 V
- Variable instruction execution time useful for high-speed operation and power saving
  0.95 μs, 1.91 μs, 3.81 μs, 15.3 μs (at 4.19 MHz)
  0.67 μs, 1.33 μs, 2.67 μs, 10.7 μs (at 6.0 MHz)
  122 μs (at 32.768 kHz)
- Four timer channels
- Low-voltage operatable A/D converter (8-bit resolution × 8 channels, successive approximation type)
- Small package (42-pin plastic shrink (375 mil, 0.8 mm pitch)

The μPD75P0076 is provided with a one-time PROM that can be electrically written and is pin-compatible with the μPD750068. This one-time PROM model is convenient for the trial development of an application system or small-scale production of an application system.

Application Fields

- Cordless phones
- AV equipment
- Home electric appliances
- OA equipments

Remark Unless otherwise specified, the μPD750068 is regarded as the representative model. Descriptions throughout this manual correspond to this model. When using this manual as the user's manual for the μPD750064, 750066, 75P0076, substitute the appropriate product name for the μPD750068.

1.1 Functional Outline

Functional Outline

Item			Function
Instruction execution time			• 0.95, 1.91, 3.81, 15.3 μs (main system clock: 4.19 MHz) • 0.67, 1.33, 2.67, 10.7 μs (main system clock: 6.0 MHz) • 122 μs (subsystem clock: 32.768 kHz)
Internal memory	ROM		4096 × 8 bits (μPD750064)
★			6144 × 8 bits (μPD750066)
			8192 × 8 bits (μPD750068)
			16384 × 8 bits (μPD75P0076)
	RAM		512 × 4 bits
General-purpose register			• When manipulated in 4-bit units: 8 × 4 banks • When manipulated in 8-bit units: 4 × 4 banks
I/O port	CMOS input	12	7 lines can be connected with pull-up resistor via software 4 lines shared with analog input pin
	CMOS I/O	12	12 lines can be connected with pull-up resistor via software 4 lines shared with analog input pin
	N-ch open-drain I/O	8	13V, pull-up resistor can be connected by mask option[Note]
	Total	32	
Timer			4 channels • 8-bit timer/event counter: 2 channels (can be used as 16-bit timer/event counter) • Basic interval timer/watchdog timer: 1 channel • Watch timer: 1 channel
Serial interface			• 3-line serial I/O mode ... MSB/LSB first selectable • 2-line serial I/O mode
A/D converter			8-bit resolution × 8 channels (1.8 V $\leq AV_{REF} \leq V_{DD}$)
Bit sequential buffer (BSB)			16 bits
Clock output (PCL)			• Φ, 1.05 MHz, 262 kHz, 65.5 kHz (main system clock: 4.19 MHz) • Φ, 1.5 MHz, 375 kHz, 93.8 kHz (main system clock: 6.0 MHz)
Buzzer output (BUZ)			• 2, 4, 32 kHz (main system clock: 4.19 MHz or subsystem clock: 32.768 kHz) • 2.93, 5.86, 46.9 kHz (main system clock: 6.0 MHz)
Vector interrupt			External: 3, internal: 4
Test input			External: 1, internal: 1
System clock oscillation circuit			• Ceramic/crystal oscillation circuit for main system clock oscillation • Crystal oscillation circuit for subsystem clock oscillation
Standby function			STOP mode/HALT mode
Supply voltage			V_{DD} = 1.8 to 5.5 V
Package			• 42-pin plastic shrink DIP (600 mil, 1.778 mm pitch) • 42-pin plastic shrink SOP (375 mil, 0.8 mm pitch)

Note The N-ch open-drain I/O port pins of the μPD75P0076 are not connected with pull-up resistors by mask option.

1.2 Ordering Information

Part Number	Package	Internal ROM
μPD750064CU-×××	42-pin plastic shrink DIP (600 mil, 1.778 mm pitch)	Mask ROM
μPD750064GT-×××	42-pin plastic shrink SOP (375 mil, 0.8 mm pitch)	Mask ROM
★ μPD750066CU-×××	42-pin plastic shrink DIP (600 mil, 1.778 mm pitch)	Mask ROM
★ μPD750066GT-×××	42-pin plastic shrink SOP (375 mil, 0.8 mm pitch)	Mask ROM
μPD750068CU-×××	42-pin plastic shrink DIP (600 mil, 1.778 mm pitch)	Mask ROM
μPD750068GT-×××	42-pin plastic shrink SOP (375 mil, 0.8 mm pitch)	Mask ROM
μPD75P0076CU	42-pin plastic shrink DIP (600 mil, 1.778 mm pitch)	One-time PROM
μPD75P0076GT	42-pin plastic shrink SOP (375 mil, 0.8 mm pitch)	One-time PROM

Remark ××× indicates a ROM code number.

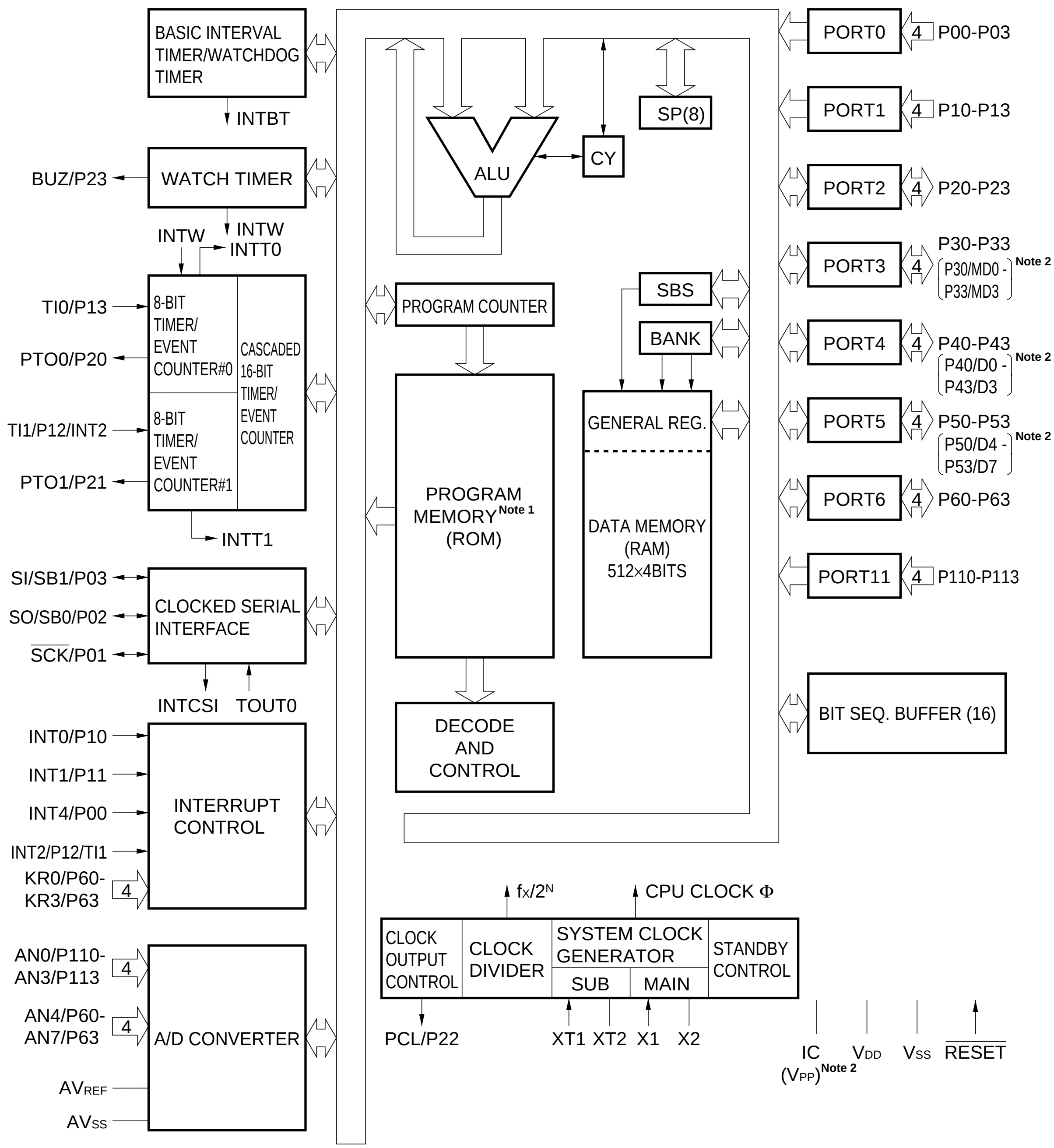
★ 1.3 Differences among μPD750068 Subseries Products

Item		μPD750064	μPD750066	μPD750068	μPD75P0076
Program counter		12 bits	13 bits		14 bits
Program memory (byte)		Mask ROM 4096	Mask ROM 6144	Mask ROM 8192	One-time PROM 16384
Data memory (× 4 bits)		512			
Mask option chip)	Pull-up resistor of port 4, 5	Provided (specifiable)			Not provided (off chip)
	Wait time on $\overline{\text{RESET}}$	Provided (selectable from $2^{17}/f_X$, $2^{15}/f_X$)**Note**			Not provided ($2^{15}/f_X$ fixed)**Note**
	Feedback resistor of subsystem clock	Provided (specifiable)			Not provided (used)
Pin connection	Pins 6-9	P33 to P30			P33/MD3-P30/MD0
	Pin 20	IC			V_{PP}
	Pins 34-37	P53 to P50			P53/D7-P50/D4
	Pins 38-41	P43 to P40			P43/D3-P40/D0
Others		Noise immunity and noise radiation differ because circuit scale and mask layout differ.			

Note $2^{17}/f_X$ is: at 6.0 MHz operation: 21.8 ms, at 4.19 MHz operation: 31.3 ms.
$2^{15}/f_X$ is: at 6.0 MHz operation: 5.46 ms, at 4.19 MHz operation: 7.81 ms.

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 mask ROM model in the course of moving from trial production to mass production, you should perform a through evaluation by using the CS model (not ES model) of the mask ROM model.

1.4 Block Diagram

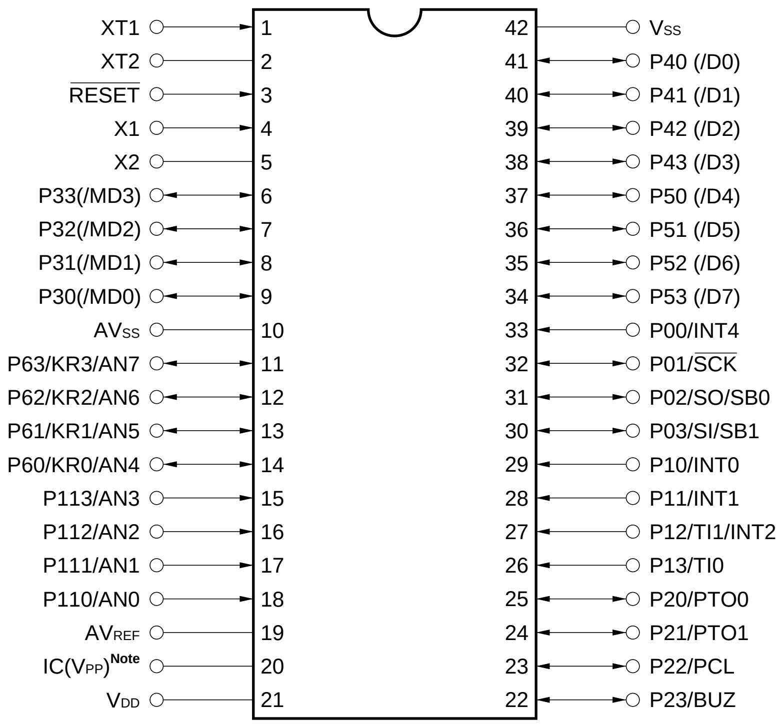


Notes **1.** ROM capacity differs with models.

2. ( ): μPD75P0076

1.5 Pin Connections (Top View)

- 42-pin plastic shrink DIP (600 mil, 1.778 mm pitch)
  μPD750064CU-×××
  ★ μPD750066CU-×××
  μPD750068CU-×××
  μPD75P0076CU
- 42-pin plastic shrink SOP (375 mil, 0.8 mm pitch)
  μPD750064GT-×××
  ★ μPD750066GF-×××
  μPD750068GT-×××
  μPD75P0076GT



Note Directly connect the IC (V_{PP}) pin to V_{DD}.

Remark ( ): μPD75P0076

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
P110-P113	: Port 11
KR0-KR3	: Key Return 0-3
$\overline{SCK}$	: Serial Clock
SI	: Serial Input
SO	: Serial Output
SB0, SB1	: Serial Bus 0, 1
$\overline{RESET}$	: Reset Input
TI0, TI1	: Timer Input 0, 1
PTO0, PTO1	: Programmable Timer Output 0, 1
BUZ	: Buzzer Clock
PCL	: Programmable Clock
INT0, INT1, INT4	: External Vectored Interrupt 0, 1, 4
INT2	: External Test Input 2
X1, X2	: Main System Clock Oscillation 1, 2
XT1, XT2	: Subsystem Clock Oscillation 1, 2
AN0-AN7	: Analog Input 0-7
AV_{REF}	: Analog Reference
AV_{SS}	: Analog Ground
V_{DD}	: Positive Power Supply
V_{SS}	: Ground
IC	: Internally Connected
MD0-MD3	: Mode Selection 0-3
D0-D7	: Data Bus 0-7
V_{PP}	: Programming Power Supply

CHAPTER 2 PIN FUNCTIONS

2.1 Pin Functions of μPD750068

Table 2-1 Pin Functions of Digital I/O Ports (1/2)

Pin Name	I/O	Shared with	Function	8-bit I/O	On Reset	I/O Circuit Type[Note 1]
P00	Input	INT4	4-bit input port (PORT0).	×	Input	Ⓑ
P01	I/O	$\overline{SCK}$	P01-P03 can specify internal pull-up resistor connection			Ⓕ-A
P02	I/O	SO/SB0	in 3-bit units via software.			Ⓕ-B
P03	I/O	SI/SB1				Ⓜ-C
P10	Input	INT0	4-bit input port (PORT1).	×	Input	Ⓑ-C
P11		INT1	Can specify internal pull-up resistor connection in 4-bit			
P12		TI1/INT2	units via software.			
P13		TI0	P10/INT0 can select noise rejection circuit.			
P20	I/O	PTO0	4-bit I/O port (PORT2).	×	Input	E-B
P21		PTO1	Can specify internal pull-up resistor connection in 4-bit			
P22		PCL	units via software.			
P23		BUZ				
P30	I/O	(MD0)[Note 3]	Programmable 4-bit I/O port (PORT3).	×	Input	E-B
P31		(MD1)[Note 3]	Can be set in input or output mode in 1-bit units.			
P32		(MD2)[Note 3]	Can specify internal pull-up resistor connection in 4-bit			
P33		(MD3)[Note 3]	units via software.			
★ P40[Note 2]	I/O	(D0)[Note 3]	N-ch open-drain 4-bit I/O port (PORT4).	○	High level	M-D
P41[Note 2]		(D1)[Note 3]	At open drain: 13 V		(When con-	(M-E)[Note 3]
P42[Note 2]		(D2)[Note 3]	Can be connected with pull-up resistors in 1-bit units		nected with	
P43[Note 2]		(D3)[Note 3]	(mask option).[Note 4]		pull-up resis-	
			Data input/output pins (lower 4 bits) for program		tors) or high	
			memory (PROM) write/verify.		impedance	

Notes 1. ○ indicates Schmitt trigger input.

2. The low-level input leakage current increases when these pins are not connected with pull-up resistors by mask option (when they are used as N-ch open-drain input port pins), or when an input or bit manipulation instruction is executed.

3. ( ): μPD75P0076

4. The μPD75P0076 does not have pull-up resistors by mask option.

Table 2-1 Pin Functions of Digital I/O Ports (2/2)

Pin Name	I/O	Shared with	Function	8-bit I/O	On Reset	I/O Circuit Type[Note 1]
★ P50[Note 2]	I/O	(D4)[Note 3]	N-ch open-drain 4-bit I/O port (PORT5). At open drain: 13 V Can be connected with pull-up resistors in 1-bit units (mask option).[Note 4] Data input/output pins (higher 4 bits) for program memory (PROM) write/verify.	○	High level (when connected with pull-up resistors) or high impedance	M-D (M-E)[Note 3]
P51[Note 2]		(D5)[Note 3]				
P52[Note 2]		(D6)[Note 3]				
P53[Note 2]		(D7)[Note 3]				
P60	I/O	KR0/AN4	Programmable 4-bit I/O port (PORT6). Can be set in input or output mode in 1-bit units. Can specify internal pull-up resistor connection in 4-bit units via software.	×	Input	Ⓨ- D
P61		KR1/AN5				
P62		KR2/AN6				
P63		KR3/AN7				
P110	Input	AN0	4-bit input port (PORT11).	×	Input	Y-A
P111		AN1				
P112		AN2				
P113		AN3				

Notes
1. ○ indicates Schmitt trigger input.
2. The low-level input leakage current increases when these pins are not connected with pull-up resistors by mask option (when they are used as N-ch open-drain input port pins), or when an input or bit manipulation instruction is executed.
3. ( ): μPD75P0076
4. The μPD75P0076 does not have pull-up resistors by mask option.

Table 2-2 Functions of Pins Other Than Port Pins (1/2)

Pin Name	I/O	Shared with	Function		On Reset	I/O Circuit Type[Note]
TI0	Input	P13	External event pulse input to timer/event counter.		Input	Ⓑ- C
TI1		P12/INT2				
PTO0	Output	P20	Timer/event counter output.		Input	E-B
PTO1		P21				
PCL		P22	Clock output.			
BUZ		P23	Outputs any frequency (for buzzer or system clock trimming).			
$\overline{SCK}$	I/O	P01	Serial clock I/O.		Input	Ⓔ- A
SO/SB0		P02	Serial data output. Serial data bus I/O.			Ⓔ- B
SI/SB1		P03	Serial data input. Serial data bus I/O.			Ⓜ- C
INT4	Input	P00	Edge-detected vector interrupt input (both rising and falling edges detection).		Input	Ⓑ
INT0	Input	P10	Edge-detected vector interrupt input	Noise rejection circuit/ asynch selectable	Input	Ⓑ- C
INT1		P11	(edge to be detected is selectable) INT0/P10 have a noise elimination functions.	Asynchronous		
INT2		P12/TI1	Rising edge-detected testable input	Asynchronous	Input	Ⓑ- C
KR0-KR3	Input	P60/AN4-P63/AN7	Falling edge-detected testable input.		Input	Ⓨ- D
AN0-AN3	Input	P110-P113	Analog signal input		Input	Y-A
AN4-AN7		P60/KR0-P63/KR3				Ⓨ-D
AV_{REF}	—	—	AD converter reference voltage		—	Z-N
AV_{SS}	—	—	AD converter reference GND potential		—	Z-N
X1	Input	—	Connect crystal/ceramic oscillator for main system clock oscillation. Input external clock to X1 and its complement to X2.		—	—
X2	—					
XT1	Input	—	Connect crystal oscillator for subsystem clock oscillation. Input external clock to XT1 and its complement to XT2. XT1 can be used as 1-bit input (test) pin.		—	—
★ XT2	—					
$\overline{RESET}$	Input	—	System reset input (low-level active).		—	Ⓑ

Note ◯ indicates Schmitt trigger input.

Table 2-2 Functions of Pins Other Than Port Pins (2/2)

Pin Name	I/O	Shared with	Function	On Reset	I/O Circuit Type
MD0-MD3	Input	P30-P33	Provided to μPD75P0076 only. Select program memory (PROM) write/verify modes.	Input	E-B
★ D0-D3	I/O	P40-P43	Provided to μPD75P0076 only. Data bus pin for writing/verifying program memory (PROM).	Input	M-E
D4-D7		P50-P53			
IC	—	—	Internally connected. Directly connect this pin to V_{DD}.	—	—
V_{PP}	—	—	Provided to μPD75P0076 only. Supplies program voltage for writing/verifying program memory (PROM). In usual operation, directly connect this pin to V_{DD}. Apply +12.5 V to this pin when writing or verifying program memory.	—	—
V_{DD}	—	—	Positive power supply	—	—
V_{SS}	—	—	Ground potential	—	—

2.2 Pin Functions

2.2.1 P00-P03 (PORT0) ... input shared with INT4, SCK, SO/SB0, and SI/SB1
P10-P13 (PORT1) ... input shared with INT0, INT1, TI1/INT2, and TI0
P110-P113 (PORT11) ... input shared with AN0-AN3

4-bit input port.

In addition to the input port function, also have the following functions.

- Port 0 : Vector interrupt input (INT4)
  Serial interface I/Os (SCK, SO/SB0, SI/SB1)
- Port 1 : Vector interrupt inputs (INT0, INT1)
  Edge detection test input (INT2)
  External event pulse input to timer/event counter (TI0, TI1)
- Port 11 : Analog signal input for A/D converter (AN0-AN3)

When the serial interface function is used port 0 can function as an output pin, depending on the operating mode.

Each pin of port 0 and port 1 are Schmitt trigger input pins to prevent malfunctioning due to noise. In addition, the P10 pin can select a noise rejecter circuit (for details, refer to **6.3 (3) Hardware of INT0, INT1, and INT4**).

Port 0 can specify internal pull-up resistors in 3-bit units (P01-P03) via software. Port 1 can specify internal pull-up resistor connection in 4-bit units (P10-P13). Whether the pull-up resistors are connected or not is specified by using pull-up resistor specification register group A (POGA).

When the RESET signal is asserted, pins are set in the input mode.

2.2.2 P20-P23 (PORT2) ... I/O shared with PTO0, PTO1, PCL, and BUZ
P30-P33 (PORT3) ... I/O shared with MD0-MD3[Note]
P40-P43 (PORT4) ... I/O shared with D0-D3[Note]
P50-P53 (PORT5) ... N-ch open-drain, medium-voltage (13 V), I/O shared with D4-D7[Note]
P60-P63 (PORT6) ... I/O shared with KR0-KR3 and AN4-AN7

4-bit I/O ports with output latch. In addition to the I/O port function, port n (n = 2, 3 or 6) has the following functions:

- Port 2 : Timer/event counter outputs (PTO0, PTO1)
  Clock output (PCL)
  Any frequency output (BUZ)
- Port 3 : Mode selection at program memory (PROM) write/verfy (MD0-MD3)[Note]
- ★ Port 4 : Data bus at program memory (PROM) write/verify (D0-D3)[Note]
- ★ Port 5 : Data bus at program memory (PROM) write/verify (D4-D7)[Note]
- Ports 6 : Key interrupt inputs (KR0-KR3)
  Analog signal input for A/D converter (AN4-AN7)

Note Shared only in the μPD75P0076.

Ports 4 and 5 are N-ch open-drain, medium-voltage (13 V) ports.

These ports are set in input or output mode by using a port mode register. Ports 2, 4 and 5 can be set in input or output mode in 4-bit units. Ports 3 and 6 can be set in input or output mode in 1-bit units.

Ports 2, 3 and 6 can specify an internal pull-up resistor connection in 4-bit units via software, by manipulating a pull-up resistor specification register group A (POGA). Ports 4 and 5 of the μPD750068 can be connected with a pull-up resistor in 1-bit units by mask option. However, the corresponding ports of the μPD75P0076 cannot be connected with a pull-up resistor by mask option.

Ports 4 and 5 can be set in input or output mode in pairs in 8-bit units. When the $\overline{\text{RESET}}$ signal is asserted, ports 2, 3 and 6 are set in input mode (high impedance), and ports 4 and 5 are set at high-level (when the pull-up resistor by mask option is connected) or high-impedance state.

2.2.3 TI0, TI1 ... inputs shared with port 1

These are the external pulse event input pins of timers/event counters 0 and 1.

These can be used by selecting external event pulse input to the count pulse (CP) using the timer/event counter mode register (TM0, TM1).

TI0 and TI1 are Schmitt trigger input pins.

2.2.4 PTO0, PTO1 ... outputs shared with port 2

These are the output pins of timers/event counters 0 and 1, and output square wave pulses. To output the signal of a timer/event counter, clear the output latch of the corresponding pin of port 2 to “0”. Then, set the bit corresponding to port 2 of the port mode register to “1” to set the output mode.

The outputs of TOUT F/F are cleared to “0” by the timer start instruction.

For details, refer to **5.5.2(3) Operation in 8-bit timer/event counter mode.**

2.2.5 PCL ... output shared with port 2

This is a programmable clock output pin and is used to supply the clock to a peripheral LSI (such as a slave microcontroller). When the $\overline{RESET}$ signal is asserted, the contents of the clock output mode register (CLOM) are cleared to “0”, disabling the output of the clock. In this case, the PCL pin can be used as an ordinary port pin.

For details, refer to **5.2.4 Clock output circuit.**

2.2.6 BUZ ... output shared with port 2

This is a frequency output pin and is used to issue a buzzer sound or trim the system clock frequency by outputting a specified frequency (2, 4, or 32 kHz @4.19 MHz with main system clock, or @32.768 kHz with subsystem clock). This pin is shared with the P23 pin and is valid only when the bit 7 (WM7) of the watch mode register (WM) is set to “1”.

When the $\overline{RESET}$ signal is asserted, WM7 is cleared to 0, so that the BUZ pin is used as an ordinary port pin.

For details, refer to **5.4.2 Watch mode register.**

2.2.7 $\overline{SCK}$, SO/SB0, and SI/SB1 ... I/Os shared with port 0

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

When the three-wire serial I/O mode is selected, the $\overline{SCK}$, SO, and SI pins function as CMOS I/O, CMOS output, CMOS input, respectively. When the two-wire serial I/O mode is selected, the $\overline{SCK}$ and SB1(SB0) pins function as CMOS I/O and N-ch open-drain I/O, respectively.

When the $\overline{RESET}$ signal is asserted, the serial interface operation is stopped, and these pins served as input port pins.

All these pins are Schmitt trigger input pins.

For details, refer to **5.6 Serial Interface**.

2.2.8 INT4 ... input shared with port 0

This is an external vector interrupt input pin and becomes active at both the rising and falling edges. The interrupt request flag is set whenever there is a positive or negative transition of the signal input to this pin.

INT4 is an asynchronous input pin and the interrupt is acknowledged whenever a high- or low-level signal is input to this pin for a fixed time, regardless of the operating clock of the CPU.

INT4 can also be used to release the STOP and HALT modes. This pin is a Schmitt trigger input pin.

2.2.9 INT0 and INT1 ... inputs shared with port 1

These pins input interrupt signals that are detected by the edge. INT0 can select a noise rejection circuit. The edge to be detected can be specified by using the edge detection mode registers (IM0 and IM1).

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

(a) Active at rising edge
(b) Active at falling edge
(c) Active at both rising and falling edges
(d) External interrupt signal input disabled

(2) INT1 (bit 0 of IM1)

(a) Active at rising edge
(b) Active at falling edge

INT0 and INT1 are asynchronous input pins. The signal input to this pin is acknowledged as long as the signal has a specific high-level width, regardless of the operating clock of the CPU.
When the $\overline{\text{RESET}}$ signal is asserted, IM0 and IM1 are cleared to “0”, and the rising edge is selected as the active edge. INT0 can select a noise rejection circuit by software and the sampling clock that rejects noise can be changed in two steps. The width of the signal that is acknowledged differs depending on the CPU operating clock.
Both INT0 and INT1 can be used to release the STOP and HALT modes. However, when the noise rejection circuit is selected, INT0 cannot be used to release the STOP and HALT modes.
INT0 and INT1 are Schmitt trigger input pins.

2.2.10 INT2 ... input shared with port 1

This pin inputs an external test signal that is active at the rising edges. When INT2 is selected by the edge detection mode register (IM2), and when the signal input to this pin goes high, an internal test flag (IRQ2) is set.

INT2 is an asynchronous input. The signal input to this pin is acknowledged as long as it has a specific high-level width, regardless of the operating clock of the CPU.

When the $\overline{\text{RESET}}$ signal is asserted, the contents of IM2 are cleared to “0”, and the test flag (IRQ2) is set at the rising edge of the INT2 pin.

INT2 can be used to release the STOP and HALT modes. It is a Schmitt trigger input pin.

2.2.11 KR0-KR3 ... inputs shared with port 6

These are key interrupt input pins. KR0 through KR3 are parallel falling edge-detected interrupt input pins.

By using the edge detection mode register (IM2), the interrupt source can be selected from “KR2 and KR3” or “KR0 to KR3”.

When the $\overline{\text{RESET}}$ signal is asserted, these pins serve as port 6 pin and set in input mode.

2.2.12 AN0-AN3 ... inputs shared with port 11
AN4, AN7 ... inputs shared with port 6

These are eight analog signal input pins for the A/D converter.

2.2.13 AV_{REF}

This pin supplies a reference voltage to the A/D converter.

2.2.14 AV_{SS}

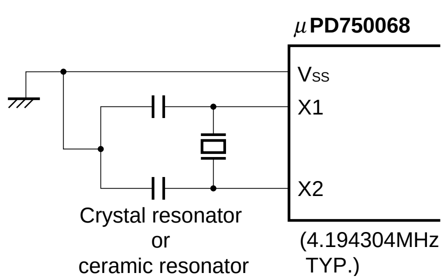
This is a GND pin of the A/D converter. Always keep this pin at the same potential as V_{SS}.

2.2.15 X1 and X2

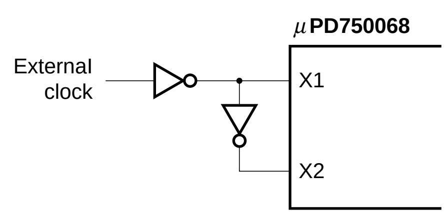
These pins connect a crystal/ceramic oscillator for main system clock oscillation.

An external clock can also be input to these pins.

(a) Crystal/ceramic oscillation



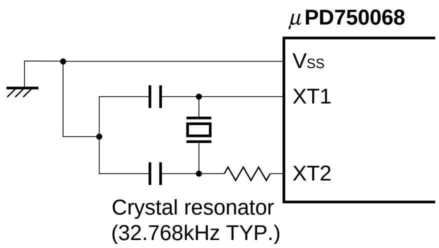
(b) External clock



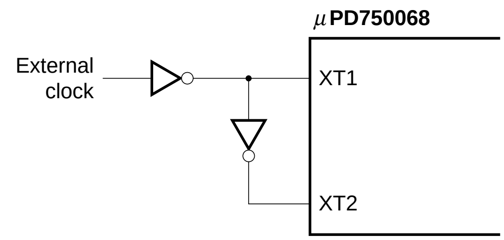
2.2.16 XT1 and XT2

These pins are used to connect a crystal oscillator for subsystem clock oscillation.
An external clock can also be input.

(a) Crystal oscillation



(b) External clock★



Remark Refer to **5.2.2 (6) Suboscillation circuit control register (SOS)** when the subsystem clock is not used.

2.2.17 $\overline{\text{RESET}}$

This pin inputs a low-active reset signal.

The $\overline{\text{RESET}}$ signal is an asynchronous input signal and is asserted when a signal with a specific low-level width is input to this pin regardless of the operating clock. The $\overline{\text{RESET}}$ signal takes precedence over all the other operations.

This pin can not only be used to initialize and start the CPU, but also to release the STOP and HALT modes.

The $\overline{\text{RESET}}$ pin is a Schmitt trigger input pin.

2.2.18 MD0-MD3 (μPD75P0076 only)

These pins are only provided on the μPD75P0076, and are used to select a mode when the program memory (one-time PROM) is written or verified.

★ 2.2.19 D0-D7 (μPD75P0076 only)

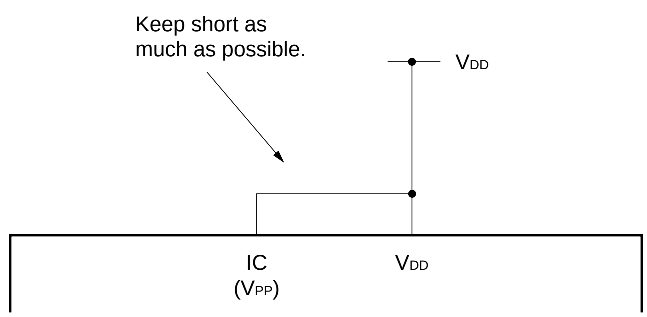
These pins are only provided on the μPD75P0076, and are used as data bus pins when the program memory (one -time PROM) is writter or verified.

2.2.20 IC (μPD750064, 750066, and 750068 only)

The IC (Internally Connected) pin sets a test mode in which the μPD750068 is tested before shipment. Usually, you should directly connect the IC pin to the V_{DD} pin with as short a wiring length as possible.

If a voltage difference is generated between the IC and V_{DD} pins because the wiring length between the IC and V_{DD} pins is too long, or because an external noise is superimposed on the IC pin, your program may not be correctly executed.

- Directly connect the IC pin to the V_{DD} pin.



2.2.21 V_{PP} (μPD75P0076 only)

This pin inputs a program voltage when the program memory (one-time PROM) is written or verified.

Usually, you should directly connect this pin to the V_{DD} (refer to the figure above). Apply 12.5 V to this pin when writing to or verifying the PROM.

2.2.22 V_{DD}

Positive power supply pin.

2.2.23 V_{SS}

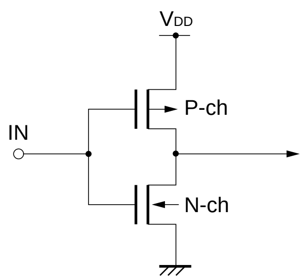
GND potential.

2.3 I/O Circuits of Respective Pins

The following diagrams show the I/O circuits of the respective pins of the μPD750068. Note that in these diagrams the I/O circuits have been slightly simplified.

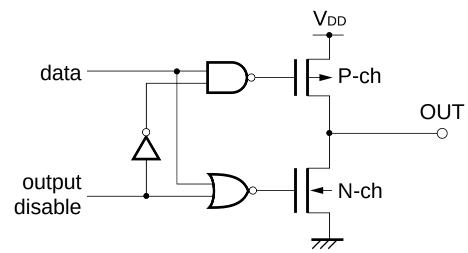
(1/3)

TYPE A



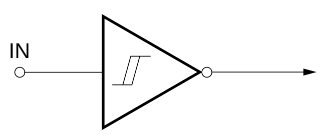
Input buffer of CMOS standard

TYPE D



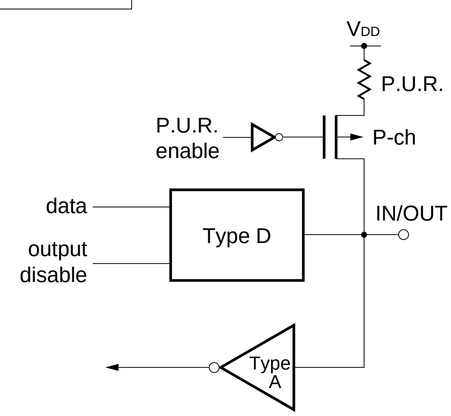
Push-pull output that can go into a high-impedance state (in which both P and N channels are off).

TYPE B



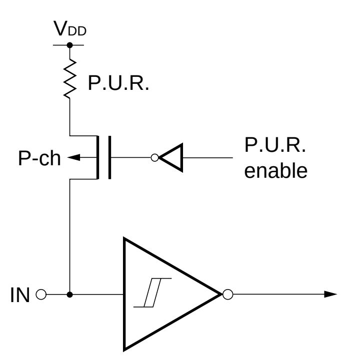
Schmitt trigger input with hysteresis characteristics.

TYPE E-B



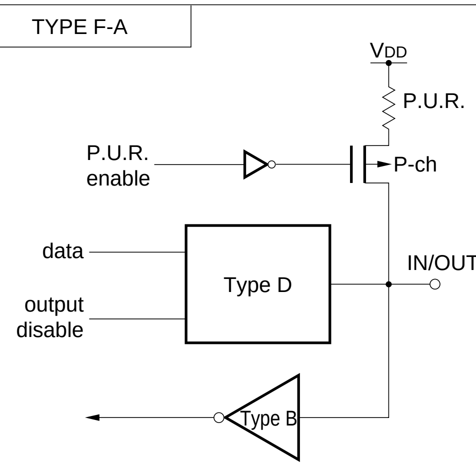
P.U.R. : Pull-Up Resistor

TYPE B-C



P.U.R. : Pull-Up Resistor

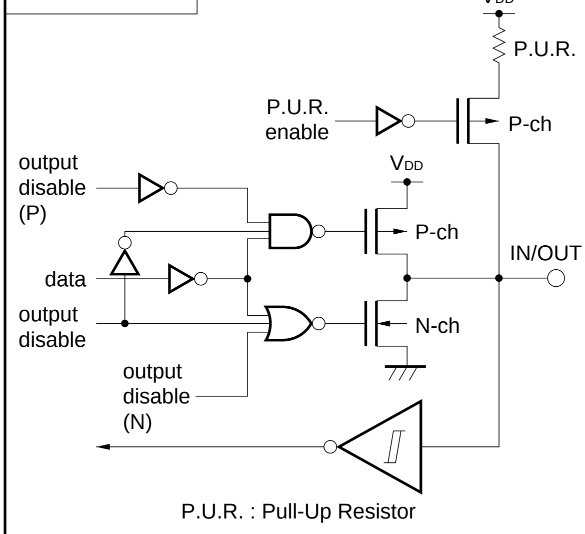
TYPE F-A



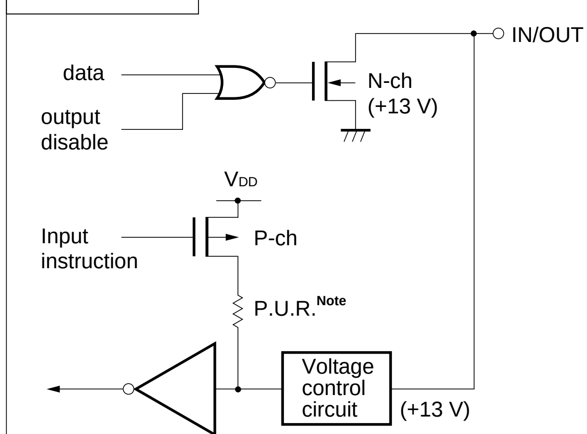
P.U.R. : Pull-Up Resistor

(2/3)

TYPE F-B

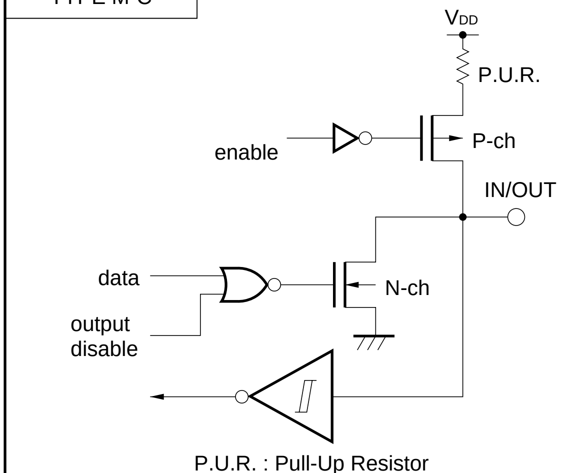


TYPE M-E★

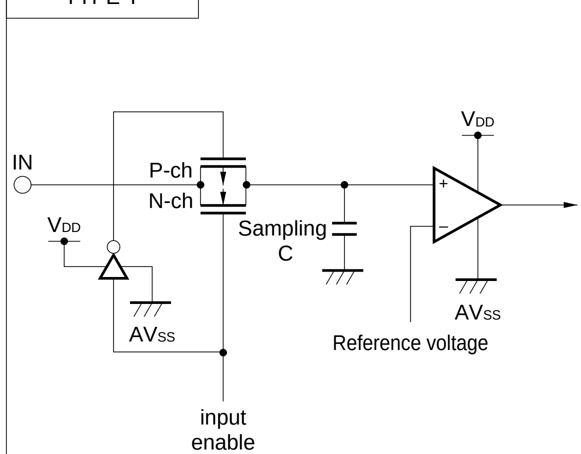


Note This pull-up resistor is effective only when an input instruction is executed (Current flows from V_{DD} to pin when the pin is low).

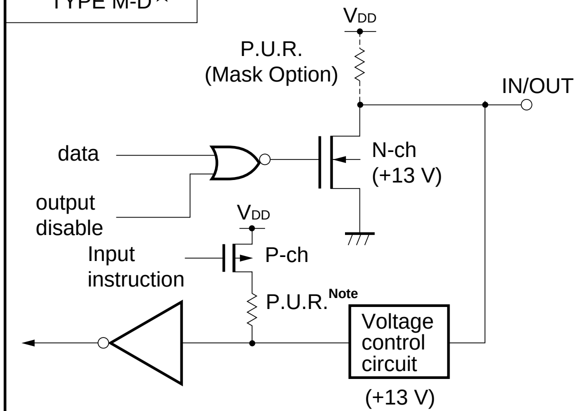
TYPE M-C



TYPE Y

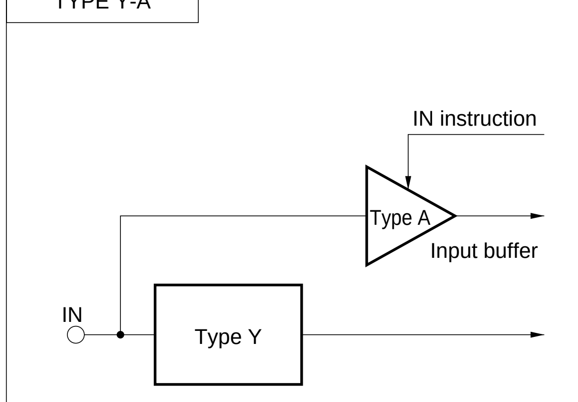


TYPE M-D★

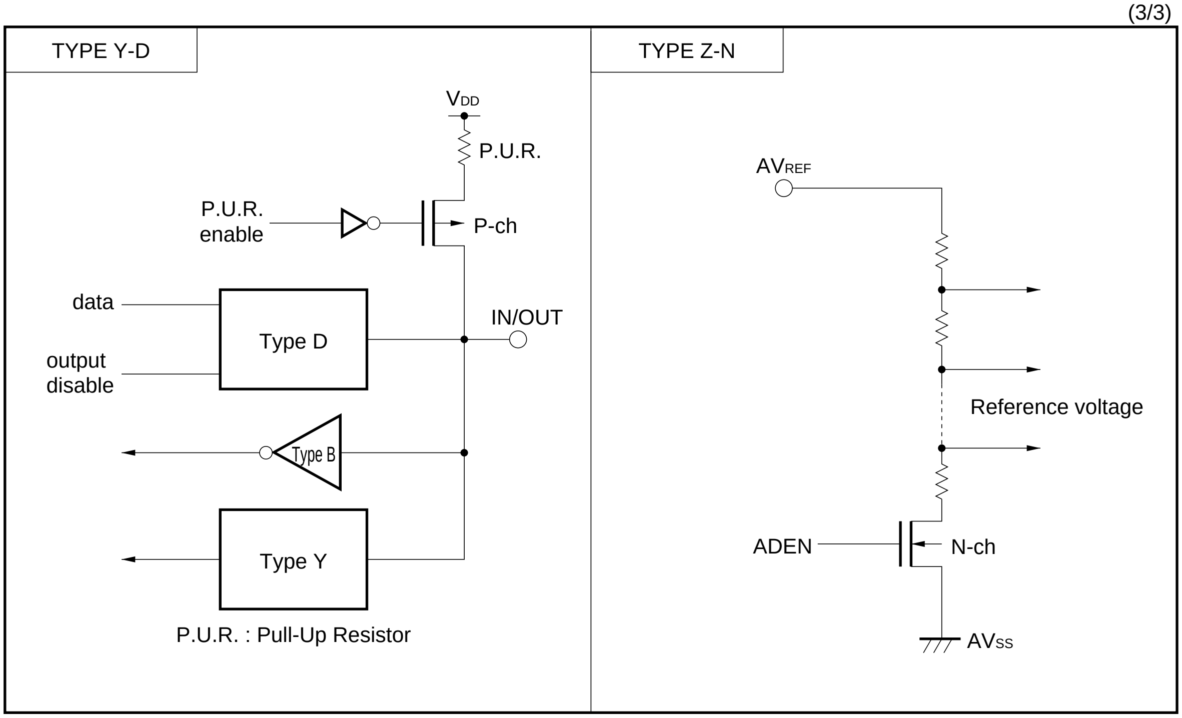


Note This pull-up resistor is effective only when an input instruction is executed without on-chip pull-up resistor by mask option.
(Current flows from V_{DD} to pin when the pin is low.)

TYPE Y-A



(3/3)


TYPE Y-D
VDD
P.U.R.
P.U.R.
enable
P-ch
data
IN/OUT
Type D
output
disable
Type B
Type Y
P.U.R. : Pull-Up Resistor
TYPE Z-N
AVREF
Reference voltage
ADEN
N-ch
AVSS

2.4 Processing of Unused Pins

★ **Table 2-3 Processing of Unused Pins**

Pin	Recommended Connection
P00/INT4	Connected to V_{SS} or V_{DD}
P01/$\overline{\text{SCK}}$	Individually connected to V_{SS} or V_{DD} via resistor
P02/SO/SB0	
P03/SI/SB1	Connected to V_{SS}
P10/INT0, P11/INT1	Connected to V_{SS} or V_{DD}
P12/TI1/INT2	
P13/TI0	
P20/PTO0	Input : Individually connected to V_{SS} or V_{DD} via resistor Output : Open
P21/PTO1	
P22/PCL	
P23/BUZ	
P30 (/MD0)[Note 1]	
P31 (/MD1)[Note 1]	
P32 (/MD2)[Note 1]	
P33 (/MD3)[Note 1]	
P40 (/D0)[Note 1]	Connected to V_{SS} (Do not connect pull-up resistor of mask option)
P41 (/D1)[Note 1]	
P42 (/D2)[Note 1]	
P43 (/D3)[Note 1]	
P50 (/D4)[Note 1]	
P51 (/D5)[Note 1]	
P52 (/D6)[Note 1]	
P53 (/D7)[Note 1]	
P60/KR0/AN4-P63/KR3/AN7	Input : Individually connected to V_{SS} or V_{DD} via resistor Output : Open
P110/AN0-P113/AN3	Directly connected to V_{SS} or V_{DD}
AV_{REF}	Connected to V_{SS}
AV_{SS}	
XT1[Note 2]	Connected to V_{SS} or V_{DD}
XT2[Note 2]	Open
IC (V_{PP})[Note 1]	Directly connect to V_{DD}

Notes 1. ( ): μPD75P0076 only

2. When the subsystem clock is not used, select SOS.0 = 1 (internal feedback resistor is not used).

[MEMO]

CHAPTER 3 FEATURES OF ARCHITECTURE AND MEMORY MAP

The 75XL architecture employed for the μPD750068 has the following features:

- Internal RAM: 4K words × 4 bits MAX. (12-bit address)
- Expansibility of peripheral hardware

To realize these superb features, the following techniques have been employed:

(1) Bank configuration of data memory
(2) Bank configuration of general-purpose registers
(3) Memory mapped I/O

This chapter describes each of these features.

3.1 Bank Configuration of Data Memory and Addressing Mode

3.1.1 Bank configuration of data memory

The μPD750068 is provided with a static RAM at the addresses 000H through 1FFH of the data memory space. Peripheral hardware units (such as I/O ports and timers) are allocated to addresses F80H through FFFH.

The μPD750068 employs a memory bank configuration that directly or indirectly specifies the lower 8 bits of an address by an instruction and the higher 4 bits of the address by a memory bank, to address the data memory space of 12-bit address (4K words × 4 bits).

To specify a memory bank (MB), the following hardware units are provided:

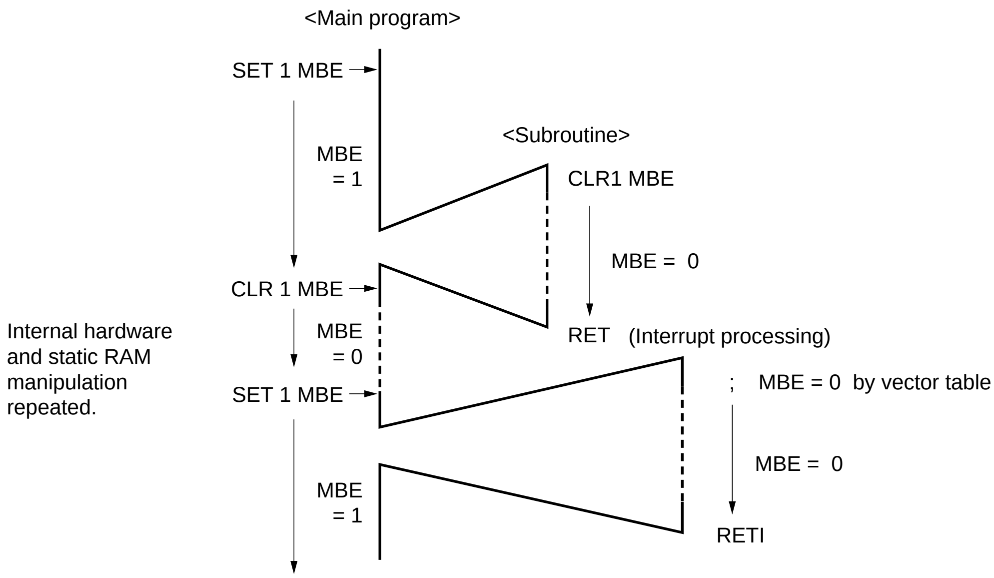
- Memory bank enable flag (MBE)
- Memory bank select register (MBS)

MBS is a register that selects a memory bank. Memory banks 0, 1, and 15 can be set. MBE is a flag that enables or disables the memory bank selected by MBS. When MBE is 0, the specified memory bank (MB) is fixed, regardless of MBS, as shown in Fig. 3-1. When MBE is 1, however, a memory bank is selected according to the setting of MBS, so that the data memory space can be expanded.

To address the data memory space, MBE is usually set to 1 and the data memory of the memory bank specified by MBS is manipulated. By selecting a mode of MBE = 0 or a mode of MBE = 1 for each processing of the program, programming can be efficiently carried out.

	Adapted Program Processing	Effect
MBE = 0 mode	• Interrupt processing	Saving/restoring MBS unnecessary
	• Processing repeating internal hardware manipulation and stack RAM manipulation	Changing MBS unnecessary
	• Subroutine processing	Saving/restoring MBS unnecessary
MBE = 1 mode	• Normal program processing	

Fig. 3-1 Selecting MBE = 0 Mode and MBE = 1 Mode



Remark   Solid line: MBE = 1, dotted line: MBE = 0

Because MBE is automatically saved or restored during subroutine processing, it can be changed even while subroutine processing is being executed. MBE can also be saved or restored automatically during interrupt processing, so that MBE during interrupt processing can be specified as soon as the interrupt processing is started, by setting the interrupt vector table. This feature is useful for high-speed interrupt processing.

To change MBS by using subroutine processing or interrupt processing, save or restore it to stack by using the PUSH or POP instruction.

MBE is set by using the SET1 or CLR1 instruction. Use the SEL instruction to set MBS.

Examples **1.** To clear MBE and fix memory bank

```
CLR1   MBE   ; MBE ← 0
```

2. To select memory bank 1

```
SET1   MBE   ; MBE ← 1
SEL    MB1   ; MBS ← 1
```

3.1.2 Addressing mode of data memory

The 75XL architecture employed for the μPD750068 provides the seven types of addressing modes as shown in Table 3-1. This means that the data memory space can be efficiently addressed by the bit length of the data to be processed and that programming can be carried out efficiently.

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

This mode is used to directly address each bit of the entire data memory space by using the operand of an instruction.

The memory bank (MB) to be specified is fixed to 0 in the mode of MBE = 0 if the address specified by the operand ranges from 00H to 7FH, and to 15 if the address specified by the operand is 80H to FFH. In the mode of MBE = 0, therefore, both the data area of addresses 000H through 07FH and the peripheral hardware area of F80H through FFFH can be addressed.

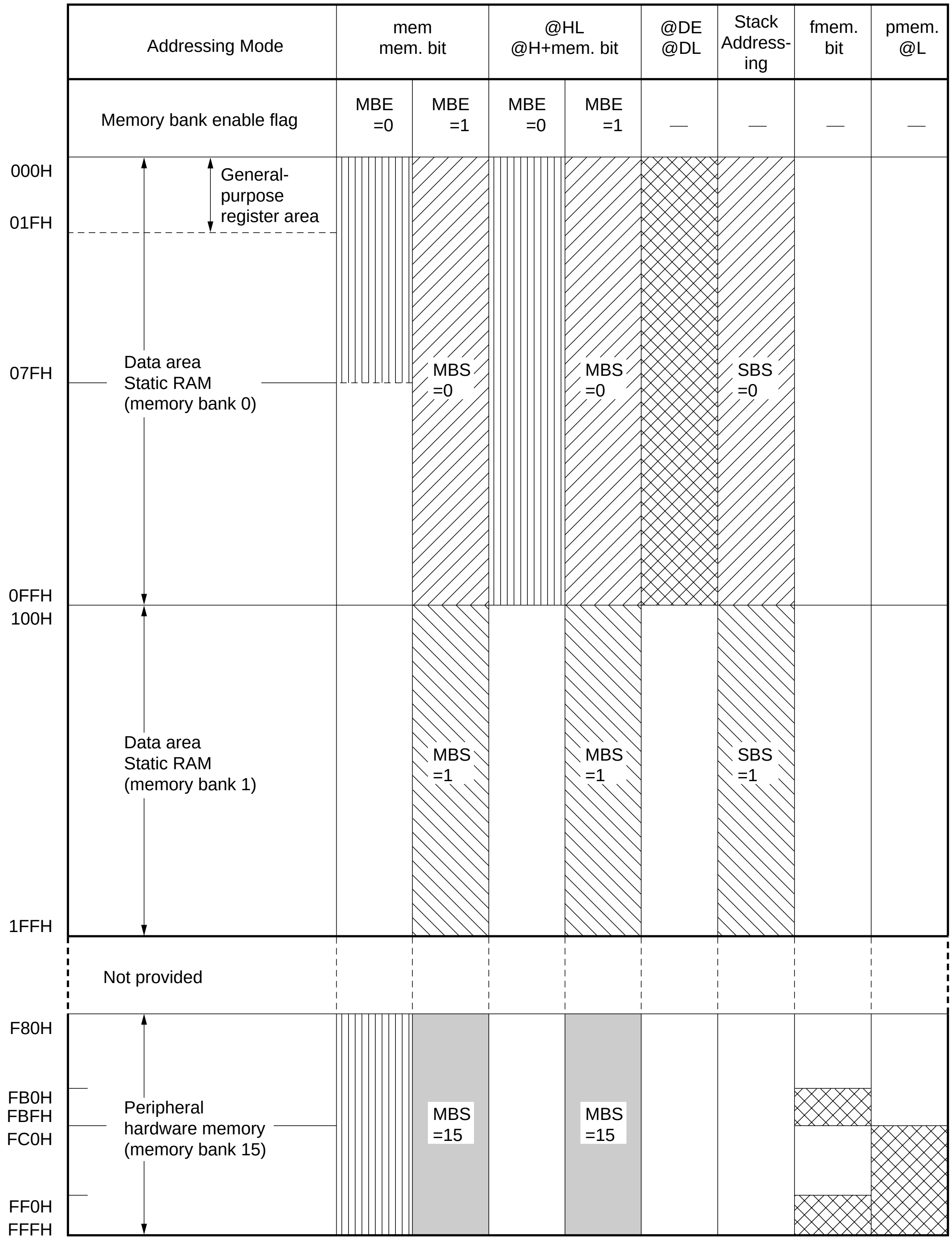
In the mode of MBE = 1, MB = MBS; therefore, the entire data memory space can be addressed.

This addressing mode can be used with four instructions: bit set and the two reset (SET1 and CLR1) instructions, and the two bit test instructions (SKT and SKF).

Example To set FLAG1, reset FLAG2, and test whether FLAG3 is 0

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

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

Fig. 3-2 Configuration of Data Memory and Addressing Ranges of Respective Addressing Modes



Remark – : don't care

Table 3-1 Addressing Modes

Addressing Mode	Representation	Specified Address
1-bit direct addressing	mem.bit	Bit specified by bit of address specified by MB and mem • When MBE = 0 When mem = 00H-7FH : MB = 0 When mem = 80H-FFH : MB = 15 • When MBE = 1 : MB = MBS
4-bit direct addressing	mem	Address specified by MB and mem. • When MBE = 0 When mem = 00H-7FH : MB = 0 When mem = 80H-FFH : MB = 15 • When MBE = 1 : MB = MBS
8-bit direct addressing		Address specified by MB and mem (mem is even address) • When MBE = 0 When mem = 00H-7FH : MB = 0 When mem = 80H-FFH : MB = 15 • When MBE = 1 : MB = MBS
4-bit register indirect addressing	@HL	Address specified by MB and HL. Where, MB = MBE · MBS
	@HL+ @HL–	Address specified by MB and HL. However, MB = MBE MBS. HL+ automatically increments L register after addressing. HL– automatically decrements 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 L register are even number) Where, MB = MBE · MBS
Bit manipulation addressing	fmem.bit	Bit specified by bit at address specified by fmem fmem = FB0H-FBFH (interrupt-related hardware) FF0H-FFFH (I/O port)
	pmem.@L	Bit specified by lower 2 bits of L register at address specified by higher 10 bits of pmem and lower 2 bits of L register. Where, pmem = FC0H-FFFH
	@H+mem.bit	Bit specified by bit at address specified by MB, H, and lower 4 bits of mem. Where, MB = MBE · MBS
Stack addressing	—	Address specified by SP in memory bank 0 and 1 selected by SBS

(2) 4-bit direct addressing (mem)

This addressing mode is used to directly address the entire memory space in 4-bit units by using the operand of an instruction.

Like the 1-bit direct addressing mode, the area that can be addressed is fixed to the data area of addresses 000H through 07FH and the peripheral hardware area of F80H through FFFH in the mode of MBE = 0. In the mode of MBE = 1, MB = MBS, and the entire data memory space can be addressed.

This addressing mode is applicable to the MOV, XCH, INCS, IN, and OUT instructions.

Caution If data related to I/O ports is stored to the static RAM in bank 1 as shown in Example 1 below, the program efficiency is degraded. To program without changing MBS as shown in Example 2, store the data related to I/O ports to the addresses 00H through 7FH of bank 0.

Examples 1. To output data of “BUFF” to port 5

```
BUFF   EQU    11AH       ; “BUFF” is 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. To input data from port 4 and store it to “DATA1”

```
DATA1  EQU    5FH        ; Stores “DATA1” to address 5FH
       CLR1   MBE        ; MBE ← 0
       IN     A, PORT4   ; A ← PORT4
       MOV    DATA1, A   ; (DATA1) ← A
```

(3) 8-bit direct addressing (mem)

This addressing mode is used to directly address the entire data memory space in 8-bit units by using the operand of an instruction.

The address that can be specified by the operand is an even address. The 4-bit data of the address specified by the operand and the 4-bit data of the the address higher than the specified address are used in pairs and processed in 8-bit units by the 8-bit accumulator (XA register pair).

The memory bank that is addressed is the same as that addressed in the 4-bit direct addressing mode.

This addressing mode is applicable to the MOV, XCH, IN, and OUT instructions.

Examples 1. To transfer the 8-bit data of ports 4 and 5 to addresses 20H and 21H

```
DATA   EQU    020H
       CLR1   MBE        ; MBE ← 0
       IN     XA, PORT4; X ← port 5, A ← port 4
       MOV    DATA, XA   ; (21H) ← X, (20H) ← A
```

2. To load the 8-bit data input to the shift register (SIO) of the serial interface and, at the same time, set transfer data to instruct the start of transfer

```
SEL    MB15              ; MBS ← 15
XCH    XA, SIO           ; XA ↔ (SIO)
```

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

This addressing mode is used to indirectly address the data memory space in 4-bit units by using a data pointer (a pair of general-purpose registers) specified by the operand of an instruction.

As the data pointer, three register pairs can be specified: HL that can address the entire data memory space by using MBE and MBS, and DE and DL that always address memory bank 0, regardless of the specification by MBE and MBS. The user selects a register pair depending on the data memory bank to be used in order to carry out programming efficiently.

★ When the HL register pair is specified, auto-increment/auto-decrement mode is used, which increments or decrements the L register by one at the same time the instruction is executed, resulting in reducing the number of program steps.

Example To transfer data 50H through 57H to addresses 110H through 117H

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

The addressing mode that uses register pair HL as the data pointer is widely used to transfer, operate, compare, and input/output data. The addressing mode using register pair DE or DL is used with the MOV and XCH instructions.

By using this addressing mode in combination with the increment/decrement instruction of a general-purpose register or a register pair, the addresses of the data memory can be updated as shown in Fig. 3-3.

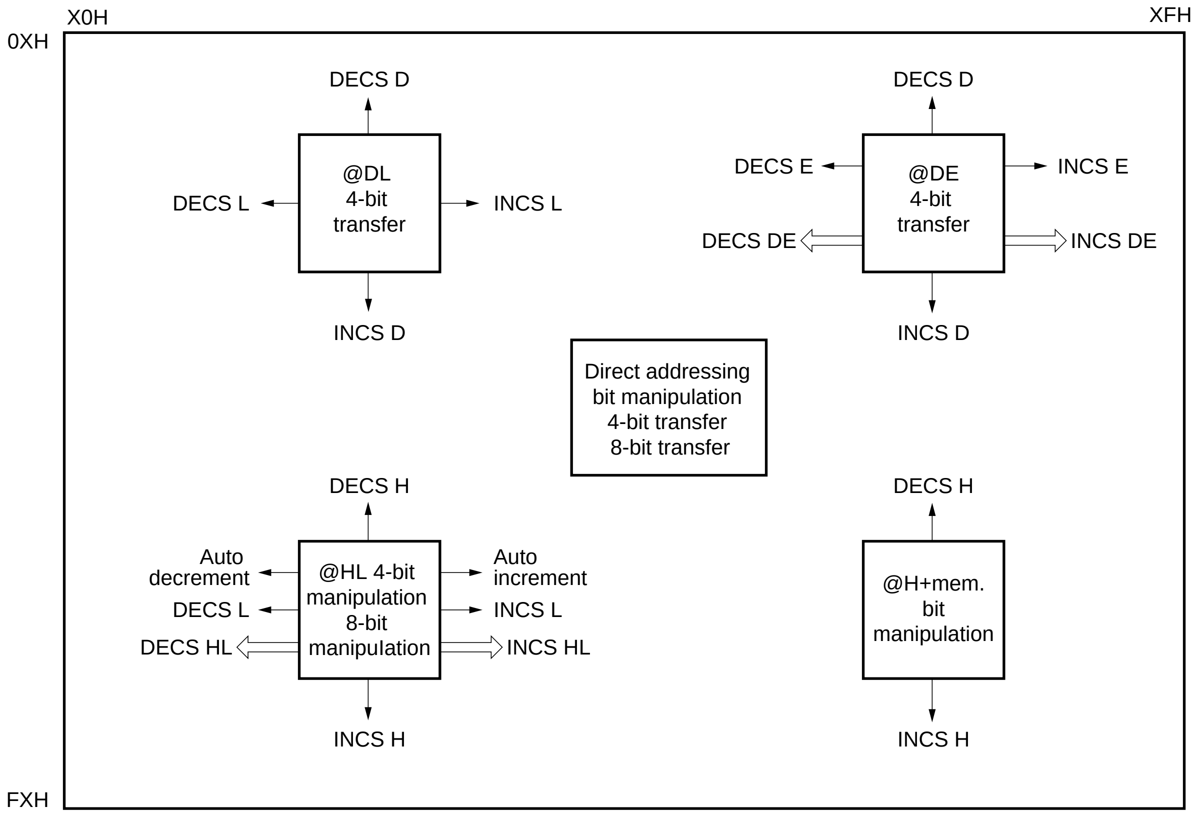
Examples 1. To compare data 50H through 57H with data 110H through 117H

```
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
```

2. To clear data memory of 00H through FFH

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

Fig. 3-3 Updating Address of Static RAM



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

This addressing mode is used to indirectly address the entire data memory space in 8-bit units by using a data pointer (HL register pair).

In this addressing mode, data is processed in 8-bit units, that is, the 4-bit data at an address specified by the data pointer with bit 0 (bit 0 of the L register) cleared to 0 and the 4-bit data at the address higher are used in pairs and processed with the data of the 8-bit accumulator (XA register).

The memory bank is specified in the same manner as when the HL register is specified in the 4-bit register indirect addressing mode, by using MBE and MBS. This addressing mode is applicable to the MOV, XCH, and SKE instructions.

Examples 1. To compare whether the count register (T0) value 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    A, @HL     ;  A = (HL)?
        BR     NO
        INCS   L
        MOV    A, X       ;  A ← X
        SKE    A, @HL     ;  A = (HL)?
```

2. To clear data memory at 00H through FFH

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

(6) Bit manipulation addressing

This addressing mode is used to manipulate the entire memory space in bit units (such as Boolean processing and bit transfer).

While the 1-bit direct addressing mode can be only used with the instructions that set, reset, or test a bit, this addressing mode can be used in various ways such as Boolean processing by the AND1, OR1, and XOR1 instructions, and test and reset by the SKTCLR instruction.

Bit manipulation addressing can be implemented in the following three ways, which can be selected depending on the data memory address to be used.

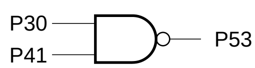
(a) Specific address bit direct addressing (fmem.bit)

This addressing mode is to manipulate the hardware units that use bit manipulation especially often, such as I/O ports and interrupt-related flags, regardless of the setting of the memory bank. Therefore, the data memory addresses to which this addressing mode is applicable are FF0H through FFFH, to which the I/O ports are mapped, and FB0H through FBFH, to which the interrupt-related hardware units are mapped. The hardware units in these two data memory areas can be manipulated in bit units at any time in the direct addressing mode, regardless of the setting of MBS and MBE.

Examples 1. To test timer 0 interrupt request flag (IRQT0) and, if it is set, clear the flag and reset P63

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

2. To reset P53 if both P30 and P41 pins are 1



(i)

```
        SET1  CY             ; CY ← 1
        AND1  CY, PORT3.0    ; CY ∧ P30
        AND1  CY, PORT4.1    ; CY ∧ P41
        SKT   CY             ; CY = 1?
        BR    SETP
        CLR1  PORT5.3        ; P53 ← 0
          :
SETP :  SET1  PORT5.3        ; P53 ← 1
```

```
                    :

(ii)                SKT    PORT3.0    ;  P30 = 1?
                    BR     SETP
                    SKT    PORT4.1    ;  P41 = 1?
                    BR     SETP
                    CLR1   PORT5.3    ;  P53 ← 0
                    :
        SETP:       SET1   PORT5.3    ;  P53 ← 1
```

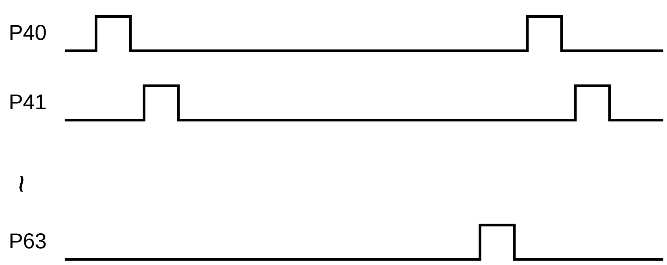
(b) Specific address bit register indirect addressing (pmem, @L)

This addressing mode is to indirectly specify and successively manipulate the bits of the peripheral hardware units such as I/O ports. The data memory addresses to which this addressing mode can be applied are FC0H through FFFH.

This addressing mode specifies the higher 10 bits of a 12-bit data memory address directly by using an operand, and the lower 2 bits by using the L register. Therefore, 16 bits (4 ports) can be successively manipulated depending on the specification of the L register.

This addressing mode can also be used independently of the setting of MBE and MBS.

Example To output pulses to the respective bits of ports 4 to 6



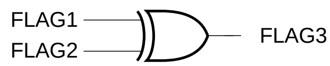

```
LOOP2:  MOV     L, #0
LOOP1:  SET1    PORT4.@L;  Bits of ports 4-6 (L1-0) ← 1
        CLR1    PORT4.@L;  Bits of ports 4-6 (L1-0) ← 0
        INCS    L
        NOP
        SKE     L, #0CH
        BR      LOOP1
        BR      LOOP2
```

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

This addressing mode enables bit manipulation in the entire memory space.

The higher 4 bits of the data memory address of the memory bank specified by MBE and MBS are indirectly specified by the H register, and the lower 4 bits and the bit address are directly specified by the operand. This addressing mode can be used to manipulate the respective bits of the entire data memory area in various ways.

Example To reset bit 2 (FLAG3) at address 32H if both bits 3 (FLAG1) at address 30H and bit 0 (FLAG2) at address 31H are 0 or 1




```
FLAG1   EQU    30H.3
FLAG2   EQU    31H.0
FLAG3   EQU    32H.2
        SEL    MB0
        MOV    H, #FLAG1  SHR  6
        CLR1   CY             ;  CY ← 0
        OR1    CY, @H+FLAG1   ;  CY ← CY ∨ FLAG1
        XOR1   CY, @H+FLAG2   ;  CY ← CY ∀ FLAG2
        SET1   @H+FLAG3       ;  FLAG3 ← 1
        SKT    CY             ;  CY = 1?
        CLR1   @H+FLAG3       ;  FLAG3 ← 0
```

(7) Stack addressing

This addressing mode is used to save or restore data when interrupt processing or subroutine processing is executed.

The address of data memory bank 0 pointed to by the stack pointer (8 bits) is specified in this addressing mode.

In addition to being used during interrupt processing or subroutine processing, this addressing is also used to save or restore register contents by using the PUSH or POP instruction.

Examples 1. To save or restore register contents during subroutine processing

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

2. To transfer contents of register pair HL to register pair DE

```
        PUSH   HL
        POP    DE   ;  DE ← HL
```

3. To branch to address specified by registers [XABC]

```
        PUSH BC
        PUSH XA
        RET         ;  To branch address XABC
```

3.2 Bank Configuration of General-Purpose Registers

The μPD750068 is provided with four register banks with each bank consisting of eight general-purpose registers: X, A, B, C, D, E, H, and L. The general-purpose register area consisting of these registers is mapped to the addresses 00H through 1FH of memory bank 0 (refer to **Fig. 3-5 Configuration of General-Purpose Register (in 4-bit processing)**). To specify a general-purpose register bank, a register bank enable flag (RBE) and a register bank select register (RBS) are provided. RBS selects a register bank, and RBE determines whether the register bank selected by RBS is valid or not. The register bank (RB) that is enabled when an instruction is executed is as follows:

RB = RBE · RBS

Table 3-2 Register Bank Selected by RBE and RBS

RBE	RBS				Register Bank
	3	2	1	0	
0	0	0	×	×	Fixed to bank 0
1	0	0	0	0	Bank 0 selected
			0	1	Bank 1 selected
			1	0	Bank 2 selected
			1	1	Bank 3 selected

(RBS bits 3 and 2: Fixed to 0)

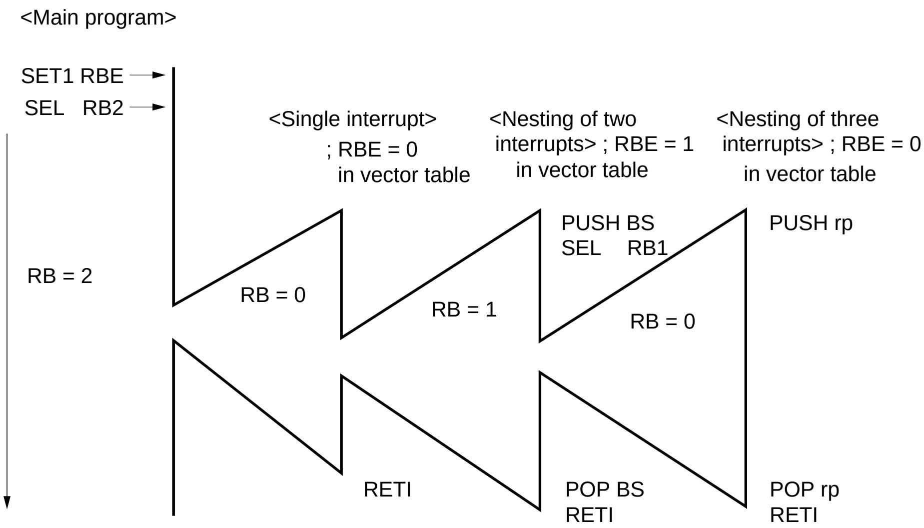
Remark × = don't care

RBE is automatically saved or restored during subroutine processing and therefore can be set while subroutine processing is under execution. When interrupt processing is executed, RBE is automatically saved or restored, and RBE can be set during interrupt processing depending on the setting of the interrupt vector table as soon as the interrupt processing is started. Consequently, if different register banks are used for normal processing and interrupt processing as shown in Table 3-3, it is not necessary to save or restore general-purpose registers when an interrupt is processed, and only RBS needs to be saved or restored if two interrupts are nested. This means that the interrupt processing speed can be increased.

Table 3-3 Example of Using Different Register Banks for Normal Routine and Interrupt Routine

Normal processing	Uses register banks 2 or 3 with RBE = 1
Single interrupt processing	Uses register bank 0 with RBE = 0
Nesting processing of two interrupts	Uses register bank 1 with RBE = 1 (at this time, RBS must be saved or restored)
Nesting processing of three or more interrupts	Registers must be saved or restored by PUSH or POP instructions

Fig. 3-4 Example of Using Register Banks



If RBS is to be changed in the course of subroutine processing or interrupt processing, it must be saved or restored by using the PUSH or POP instruction.

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

Example	SET1	RBE	;	RBE ← 1
	CLR1	RBE	;	RBE ← 0
	SEL	RB0	;	RBS ← 0
	SEL	RB3	;	RBS ← 3

The general-purpose register area provided to the μPD750068 can be used not only as 4-bit registers but also as 8-bit register pairs. This feature allows the μPD750068 to provide transfer, operation, comparison, and increment/decrement instructions comparable to those of 8-bit microcontrollers and allows you to program using mainly only general-purpose registers.

(1) To use as 4-bit registers

When the general-purpose register area is used as a 4-bit register area, a total of eight general-purpose registers, X, A, B, C, D, E, H, and L, specified by RBE and RBS can be used as shown in Fig. 3-5. Of these registers, A plays a central role in transferring, operating, and comparing 4-bit data as a 4-bit accumulator. The other registers can transfer, compare, and increment or decrement data with the accumulator.

(2) To use as 8-bit registers

When the general-purpose register area is used as an 8-bit register area, a total of eight 8-bit register pairs can be used as shown in Fig. 3-6: register pairs XA, BC, DE, and HL of a register bank specified by RBE and RBS, and register pairs XA', BC', DE', and HL' of the register bank whose bit 0 is complemented in respect to the register bank (RB). Of these register pairs, XA serves as an 8-bit accumulator, playing the central role in transferring, operating, and comparing 8-bit data. The other register pairs can transfer, compare, and increment or decrement data with the accumulator. The HL register pair is mainly used as a data pointer. The DE and DL register pairs are also used as auxiliary data pointers.

Examples 1.

```
INCS   HL          ; Skips if HL ← HL+1, HL=00H
ADDS   XA, BC      ; Skips if XA ← XA+BC and carry occurs
SUBC   DE', XA     ; DE' ← DE' – XA – CY
MOV    XA, XA'     ; XA ← XA'
MOVT   XA, @PCDE   ; XA ← (PC12–8+DE) ROM, table reference
SKE    XA, BC      ; Skips if XA = BC
```

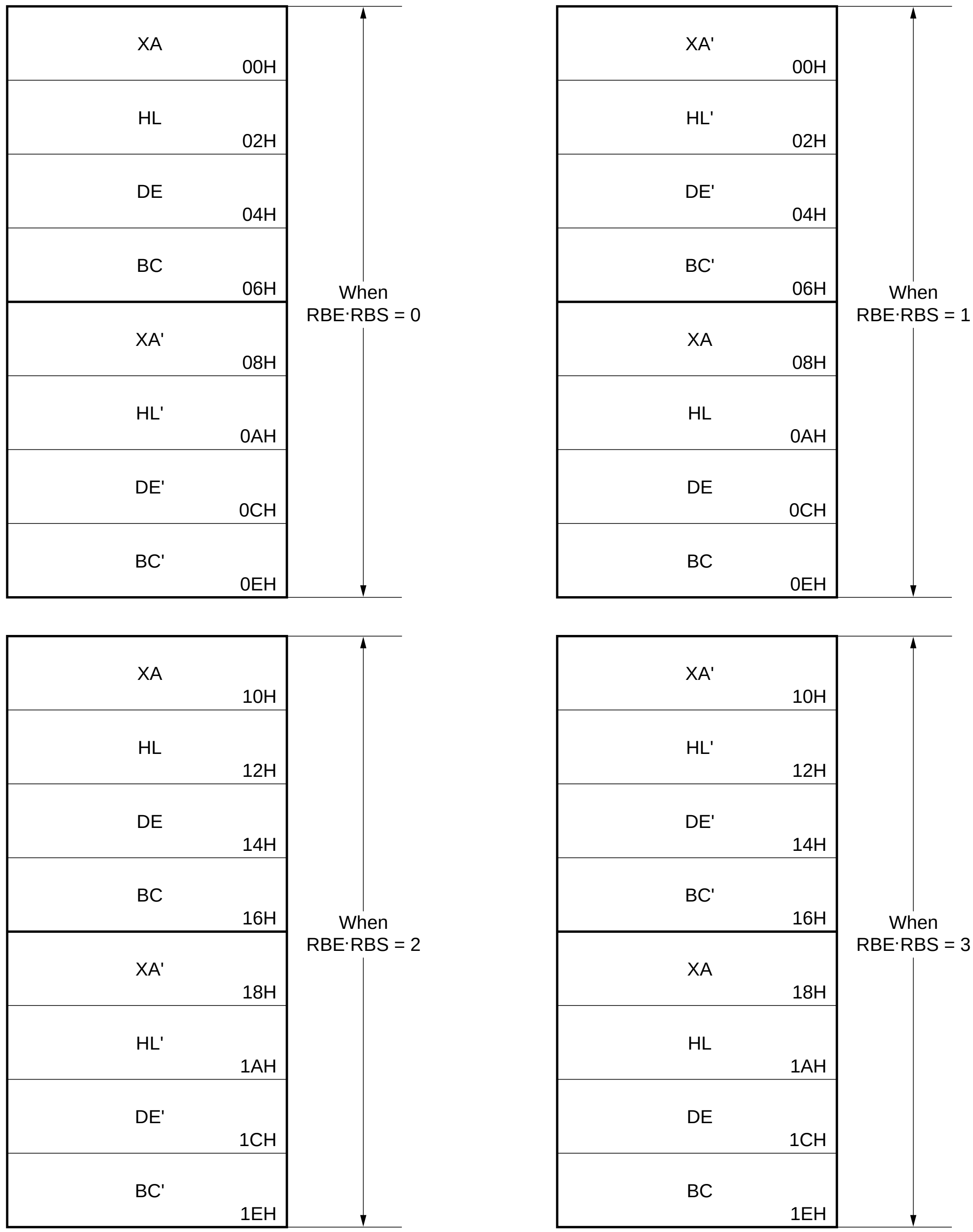
2. To test whether the value of the count register (T0) of timer/event counter is greater than the value of register pair BC' and, if not, wait until it becomes greater

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

Fig. 3-5 Configuration of General-Purpose Registers (in 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	

Fig. 3-6 Configuration of General-Purpose Registers (in 8-bit processing)



3.3 Memory-Mapped I/O

The μPD750068 employs memory-mapped I/O that maps peripheral hardware units such as I/O ports and timers to addresses F80H through FFFH on the data memory space, as shown in Fig. 3-2. Therefore, no special instructions to control the peripheral hardware units are provided, and all the hardware units are controlled by using memory manipulation instructions. (Some mnemonics that make the program easy to read are provided for hardware control.)

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

Table 3-4 Addressing Modes Applicable to Peripheral Hardware Unit Manipulation

	Applicable Addressing Mode	Hardware Units
Bit manipulation	Specified in direct addressing mode mem.bit with MBE = 0 or (MBE = 1, MBS = 15)	All hardware units that can be manipulated in 1-bit units
	Specified in direct addressing mode fmem.bit regardless of setting of MBE and MBS	IST1, IST0, MBE, RBE IE×××, IRQ×××, PORTn.×
	Specified in indirect addressing mode pmem.@L regardless of setting of MBE and MBS	BSBn.× PORTn.×
4-bit manipulation	Specifies in direct addressing mode mem with MBE=0 or (MBE = 1, MBS = 15)	All hardware units that can be manipulated in 4-bit units
	Specified in register indirect addressing @HL with (MBE = 1, MBS = 15)	
8-bit manipulation	Specified in direct addressing mem with MBE = 0 or (MBE = 1, MBS = 15), where mem is even number.	All hardware units that can be manipulated in 8-bit units
	Specified in register indirect addressing @HL with MBE = 1, MBS = 15, where contents of L register are even number	

Example

```
CLR1    MBE        ; MBE = 0
SET1    TM0. 3     ; Starts timer 0
EI      IE0        ; Enables INT0
DI      IE1        ; Disables INT1
SKTCLR  IRQ2       ; Tests and clears INT2 request flag
SET1    PORT4, @L  ; Sets port 4
IN      A, PORT0   ; A ← port 0
OUT     PORT4, XA  ; Port 5, 4 ← XA
```

Fig. 3-7 shows the I/O map of the μPD750068.

The meanings of the symbols shown in this figure are as follows:

- Abbreviation .... Name indicating the address of an internal hardware unit
  It can be written in operands of instructions
- R/W .... Indicates whether a hardware unit in question can be read or written
  R/W : Read/write
  R : Read only
  W : Write only
- Bits for manipulation .... Indicates the bit units in which a hardware unit in question can be manipulated
  ○: Can be manipulated in specified units (1, 4, or 8 bits)
  △: Only some bits can be manipulated. For the bits that can be manipulated, refer to Remark.
  – : Cannot be manipulated in specified units (1, 4, or 8 bits).
- Bit manipulation addressing ... Indicates a bit manipulation addressing mode that can be used to manipulate a hardware unit in question in 1-bit units

Fig. 3-7 μPD750068 I/O Map (1/5)

Address	Hardware Name (abbreviation)				R/W	Bits for Manipulation			Bit Manipulation	Remark
	b3	b2	b1	b0		1 bit	4 bits	8 bits	Addressing	
F80H	Stack pointer (SP)				R/W	–	–	○	–	Bit 0 is fixed to 0
F82H	Register bank select register (RBS) Bank select register (BS)				R	–	○	○	–	**Note 1**
F83H	Memory bank select register (MBS)					–	○			
F84H	Stack bank select register (SBS)				R/W	–	○	–	–	
F85H	Basic interval timer mode register (BTM)				W	△	○	–	mem.bit	Only bit 3 can be manipulated
F86H	Basic interval timer (BT)				R	–	–	○	–	
F8BH	WDTM Note2				W	△	–	–	mem.bit	Only bit 3 can be manipulated

Notes 1. RBS and MBS can be manipulated separately in 4-bit units.
Only BS can be manipulated in 8-bit units.
Write data to MBS and RBS by using the SEL MBn and SEL RBn instructions, respectively.

2. WDTM: watchdog timer enable flag (W): This flag cannot be set by an instruction when it has been once set.

Fig. 3-7 μPD750068 I/O Map (2/5)

Address	Hardware Name (abbreviation)				R/W	Bits for Manipulation			Bit Manipulation	Remark
	b3	b2	b1	b0		1 bit	4 bits	8 bits	Addressing	
F98H	Watch mode register (WM)				R/W	△ (R)	–	○	mem.bit	Only bit 3 can be manipulated
F99H						–	–		–	
FA0H	Timer/event counter 0 mode register (TM0)				R/W	△ (W)	–	○	mem.bit	Only bit 3 can be manipulated
						–	–		–	
FA2H	TOE0[Note1]				W	○	–	–	mem.bit	
FA4H	Timer/event counter 0 count register (T0)				R	–	–	○	–	
FA6H	Timer/event counter 0 modulo register (TMOD0)				R/W	–	–	○	–	
FA8H	Timer/event counter 1 mode register (TM1)				R/W	△ (W)	–	○	mem.bit	Only bit 3 can be manipulated
						–	–		–	
FAAH	TOE1[Note2]				W	○	–	–	mem.bit	
FACH	Timer/event counter 1 count register (T1)				R	–	–	○	–	
FAEH	Timer/event counter 1 modulo register (TMOD1)				R/W	–	–	○	–	

Notes 1. TOE0: timer/event counter 0 output enable flag (W)

2. TOE1: timer/event counter 1 output enable flag (W)

Fig. 3-7 μPD750068 I/O Map (3/5)

Address	Hardware Name (abbreviation) b3	b2	b1	b0	R/W	Bits for Manipulation 1 bit	4 bits	8 bits	Bit Manipulation Addressing	Remark
FB0H	IST1	IST0	MBE	RBE	R/W	○ (R/W)	○ (R/W)	○	fmem.bit	Can only be read in 8-bit units
	Program status word (PSW)									
	CY Note 1	SK2 Note 1	SK1 Note 1	SK0 Note 1		△ Note 2	–	(R)		
FB2H	Interrupt priority select register (IPS)				R/W	–	○	–		**Note 3**
FB3H	Processor clock control register (PCC)				R/W	–	○	–		**Note 4**
FB4H	INT0 edge detection mode register (IM0)				R/W	–	○	–	–	
FB5H	INT1 edge detection mode register (IM1)				R/W	–	○			Only bit 0 can be manipulated
FB6H	INT2 edge detection mode register (IM2)				R/W	–	○			Only bits 0 and 1 can be manipulated
FB7H	System clock control register (SCC)				R/W	△ (R/W)	○ (R)	–	–	Only bits 0 and 3 can be manipulated
FB8H	INTA register (INTA)				R/W	○	○	–	fmem.bit	
	IE4	IRQ4	IEBT	IRQBT						
FBAH	INTC register (INTC)				R/W	○	○			
			IEW	IRQW						
FBCH	INTE register (INTE)				R/W	○	○	–		
	IET1	IRQT1	IET0	IRQT0						
FBDH	INTF register (INTF)				R/W	○	○			
			IECSI	IRQCSI						
FBEH	INTG register (INTG)				R/W	○	○	–		
	IE1	IRQ1	IE0	IRQ0						
FBFH	INTH register (INTH)				R/W	○	○			
			IE2	IRQ2						

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

Remarks 1. IE××× indicates an interrupt enable flag.

2. IEQ××× indicates an interrupt request flag.

Notes 1. Not registered as a reserved word.

2. Use a CY manipulation instruction to write to CY.

3. Only bit 3 can be manipulated by the EI and DI instructions.

4. Bits 3 and 2 can be manipulated when the STOP or HALT instruction is executed.

Fig. 3-7 μPD750068 I/O Map (4/5)

Address	Hardware Name (abbreviation) b3	b2	b1	b0	R/W	Bits for Manipulation 1 bit	4 bits	8 bits	Bit Manipulation Addressing	Remark
FD0H	Clock output mode register (CLOM)				R/W	–	○	–	–	
FD8H	SOC	EOC	–	–	R/W	○	–	○	mem. bit	EOC...R
	A/D conversion mode register (ADM)									SOC, ADEN..W
	ADEN	ADM6[Note]	ADM5[Note]	ADM4[Note]	R/W	△	–			
FDAH	SA register (SA)				R	–	–	○		
FDCH	PO3[Note]	PO2[Note]	PO1[Note]	PO0[Note]	R/W	–	–	○	–	
	Pull-up resistor specification register group A (POGA)									
	–	PO6[Note]	–	–						
FE0H	CSIM3[Note]	CSIM2[Note]	CSIM1[Note]	CSIM0[Note]	W	–	–	○	–	
	Serial operation mode register (CSIM)									
	CSIE			CSIM4[Note]		△	–		mem. bit	
FE2H			CMDT	RELT	W	○	–	–	mem. bit	
	SBI control register (SBC)									
FE4H	Serial I/O shift register (SIO)				R/W	–	–	○	–	
FE8H	PM33[Note]	PM32[Note]	PM31[Note]	PM30[Note]	R/W	–	–	○	–	
	Port mode register group A (PMGA)									
	PM63[Note]	PM62[Note]	PM61[Note]	PM60[Note]						
FECH	–	PM2[Note]	–	–	R/W	–	–	○	–	
	Port mode register group B (PMGB)									
	–	–	PM5[Note]	PM4[Note]						

Note Not registered as a reserved word.

Fig. 3-7 μPD750068 I/O Map (5/5)

Address	Hardware Name (abbreviation) b3	b2	b1	b0	R/W	Bits for Manipulation 1 bit	4 bits	8 bits	Bit Manipulation Addressing	Remark
FF0H	Port 0		SCKP	(PORT0)	R	○	○	–	fmem. bit	**Note 1**
FF1H	Port 1			(PORT1)	R	○	○		pmem. @L	
FF2H	Port 2			(PORT2)	R/W	○	○	–		
FF3H	Port 3			(PORT3)	R/W	○	○			
FF4H	Port 4			(PORT4)	R/W	○	○	○		
FF5H	Port 5			(PORT5)	R/W	○	○			
FE6H **Note 2**	KR3	KR2	KR1	KR0	R/W	○	○	–		
	Port 6			(PORT6)						
FFBH	Port 11			(PORT11)	R	○	○			

Notes **1.** Bit 1 can be used for R/W only in serial operation enable mode.
For operations on bits 0, 2, 3, and 4, only R is possible.

2. KR0-KR3 can only be read in bit units.

4.1 Function to Select MkI and MkII Modes

4.1.1 Difference between MkI and MkII modes

The CPU of the μPD750068 has two modes to be selected: MkI and MkII modes. These modes can be selected by using the bit 3 of the stack bank select register (SBS).

- MkI mode : In this mode, the μPD750068 is upward-compatible with the μPD75068.
  This mode can be used with the CPU in the 75XL series having a ROM capacity of up to 16K bytes.
- MkII mode : In this mode, the μPD750068 is not compatible with the μPD75068.
  This mode can be used with all the CPUs in the 75XL series, including the models having a ROM capacity of 16K bytes or higher.

Table 4-1 Differences between MkI and MkII Modes

	MkI Mode	MkII Mode
Number of stack bytes of subroutine instruction	2 bytes	3 bytes
BRA !addr1 instruction CALLA !addr1 instruction	Not provided	Provided
CALL !addr instruction	3 machine cycles	4 machine cycles
CALLF !faddr instruction	2 machine cycles	3 machine cycles

★ **Caution The Mk II mode supports a program area which exceeds 16K bytes in the 75X and 75XL series. This mode enhances the software compatibility with products which have more than 16K bytes. When the Mk II mode is selected, the number of stack bytes used in execution of a subroutine call instruction increases by 1 per stack for the usable area compared to the Mk I mode. Furthermore, when a CALL !addr, or CALLF !faddr instruction is used, each instruction takes another machine cycle. Therefore, when more importance is attached to RAM utilization or throughput than software compatibility, use the Mk I mode.**

4.1.2 Setting stack bank select register (SBS)

The MkI mode or MkII mode is selected by using the stack bank select register (SBS). Fig. 4-1 shows the format of this register.

The stack bank select register is set by using a 4-bit memory manipulation instruction. To use the MkI mode, be sure to initialize the stack bank select register to 100×B[Note] at the beginning of the program. To use the MkII mode, initialize the register to 000×B[Note].

Note Set the desired value at ×.

Fig. 4-1 Format of Stack Bank Select Register

Address	3	2	1	0	Symbol
F84H	SBS3	SBS2	SBS1	SBS0	SBS

SBS1, SBS0 – Specifies stack area

SBS1	SBS0	
0	0	Memory bank 0
0	1	Memory bank 1
Other than above		Setting prohibited

SBS2

0	Be sure to clear bit 2 to 0.

SBS3 – Selects mode

0	MkII mode
1	MkI mode

Caution The SBS.3 bit is set to "1" after the $\overline{\text{RESET}}$ signal has been asserted. Therefore, the CPU operates in the MkI mode. To use the instructions in the MkII mode, clear SBS.3 to "0" to set the MkII mode.

4.2 Program Counter (PC) ··· 12 bits (μPD750064)
★ ··· 13 bits (μPD750066, 750068)
··· 14 bits (μPD75P0076)

This is a binary counter that holds an address of the program memory.

Fig. 4-2 Configuration of Program Counter

(a) μPD750064

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

★ **(b) μPD750066, 750068**

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

(c) μPD75P0076

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

The value of the program counter (PC) is usually automatically incremented by the number of bytes of an instruction each time that instruction has been executed.

When a branch instruction (BR, BRA, or BRCB) is executed, immediate data indicating the branch destination address or the contents of a register pair are loaded to all or some bits of the PC.

When a subroutine call instruction (CALL, CALLA, or CALLF) is executed or when a vector interrupt occurs, the contents of the PC (a return address already incremented to fetch the next instruction) are saved to the stack memory (data memory specified by the stack pointer). Then, the jump destination address is loaded to the PC.

When the return instruction (RET, RETS, or RETI) instruction is executed, the contents of the stack memory are set to the PC.

Generation of a $\overline{\text{RESET}}$ signal initializes the program counter (PC) content to the content of program memory at addresses 0000H and 0001H, and the program can be started from any address according to that content.

μPD750064 : $PC_{11-8} \leftarrow (0000H)_{3-0}$, $PC_{7-0} \leftarrow (0001H)_{7-0}$
★ μPD750066, 750068 : $PC_{12-8} \leftarrow (0000H)_{4-0}$, $PC_{7-0} \leftarrow (0001H)_{7-0}$
μPD75P0076 : $PC_{13-8} \leftarrow (0000H)_{5-0}$, $PC_{7-0} \leftarrow (0001H)_{7-0}$

4.3 Program Memory (ROM) ··· 4096 × 8 bits (μPD750064)

★ **··· 6144 × 8 bits (μPD750066)**
··· 8192 × 8 bits (μPD750068)
··· 16384 × 8 bits (μPD75P0076)

The program memory stores a program, interrupt vector table, the reference table of the GETI instruction, and table data.

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

Fig. 4-3 shows address ranges in which execution can be branched by a branch or subroutine call instruction. A relative branch instruction (BR $addr instruction) can branch execution to an address of [contents of PC –15 to –1 or +2 to +16], regardless of the block boundary.

The address range of the program memory of each model is as follows:

- 0000H-0FFFH : μPD750064
- ★ 0000H-17FFH : μPD750066
- 0000H-1FFFH : μPD750068
- 0000H-3FFFH : μPD75P0076

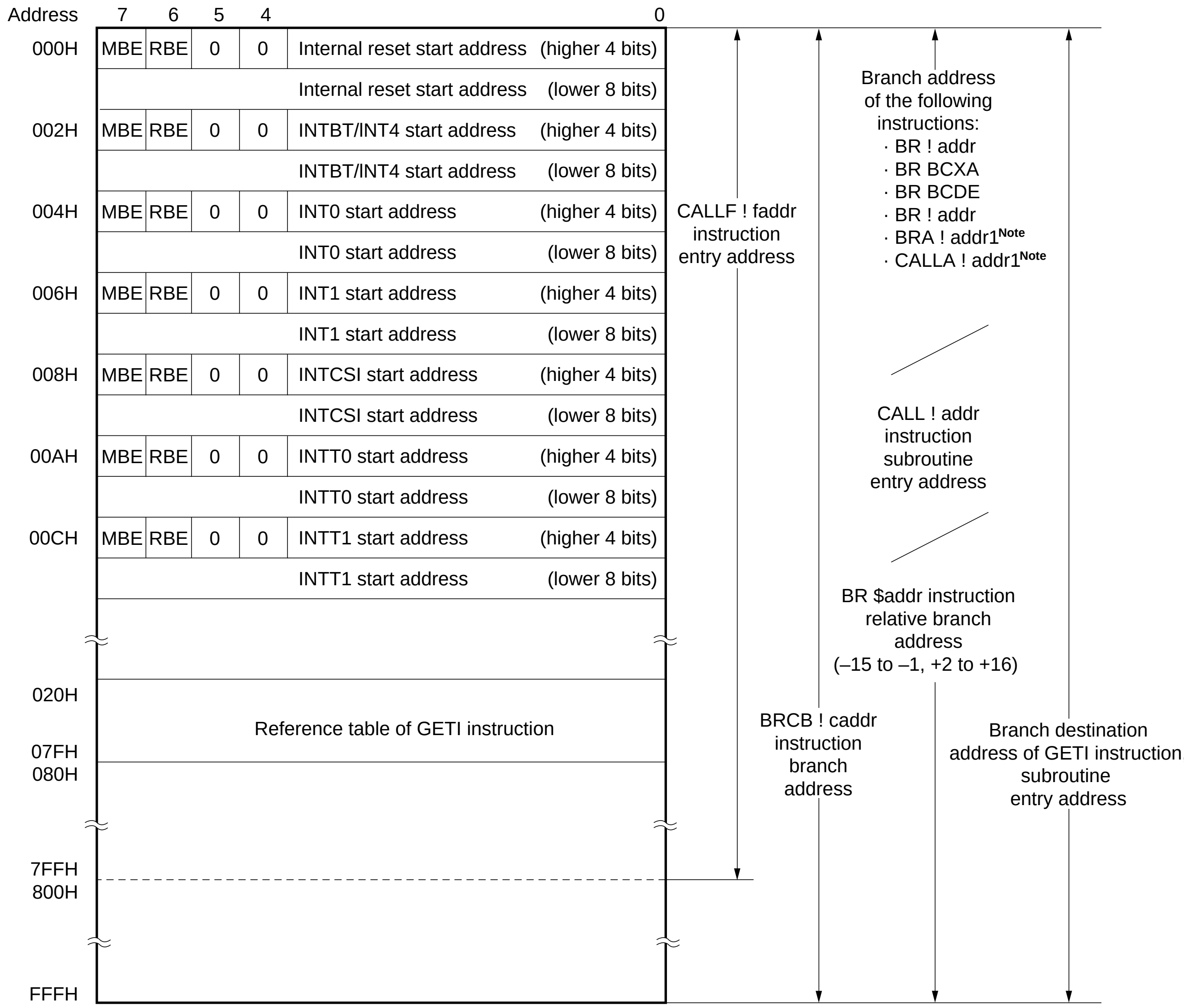
Special functions are assigned to the following addresses. All the addresses other than 0000H and 0001H can be usually used as program memory addresses.

- **Addresses 0000H and 0001H**
  These addresses store a start address from which program execution is to be started when the RESET signal is asserted, and a vector table to which the set values of RBE and MBE are written. Program execution can be reset and started from any address.
- **Addresses 0002H through 000DH**
  These addresses store start addresses from which program execution is to be started when a vector interrupt occurs, and a vector table to which the set values of RBE and MBE are written. Interrupt processing can be started from any address.
- **Addresses 0020H-007FH**
  These addresses constitute a table area that can be referenced by the GETI instruction[Note].

Note The GETI instruction implements any 2- or 3-byte instruction, or two 1-byte instructions with 1 byte. It is used to decrease the number of program steps (refer to **11.1.1 GETI instruction**).

Fig. 4-3 Program Memory Map (1/4)

(a) μPD750064

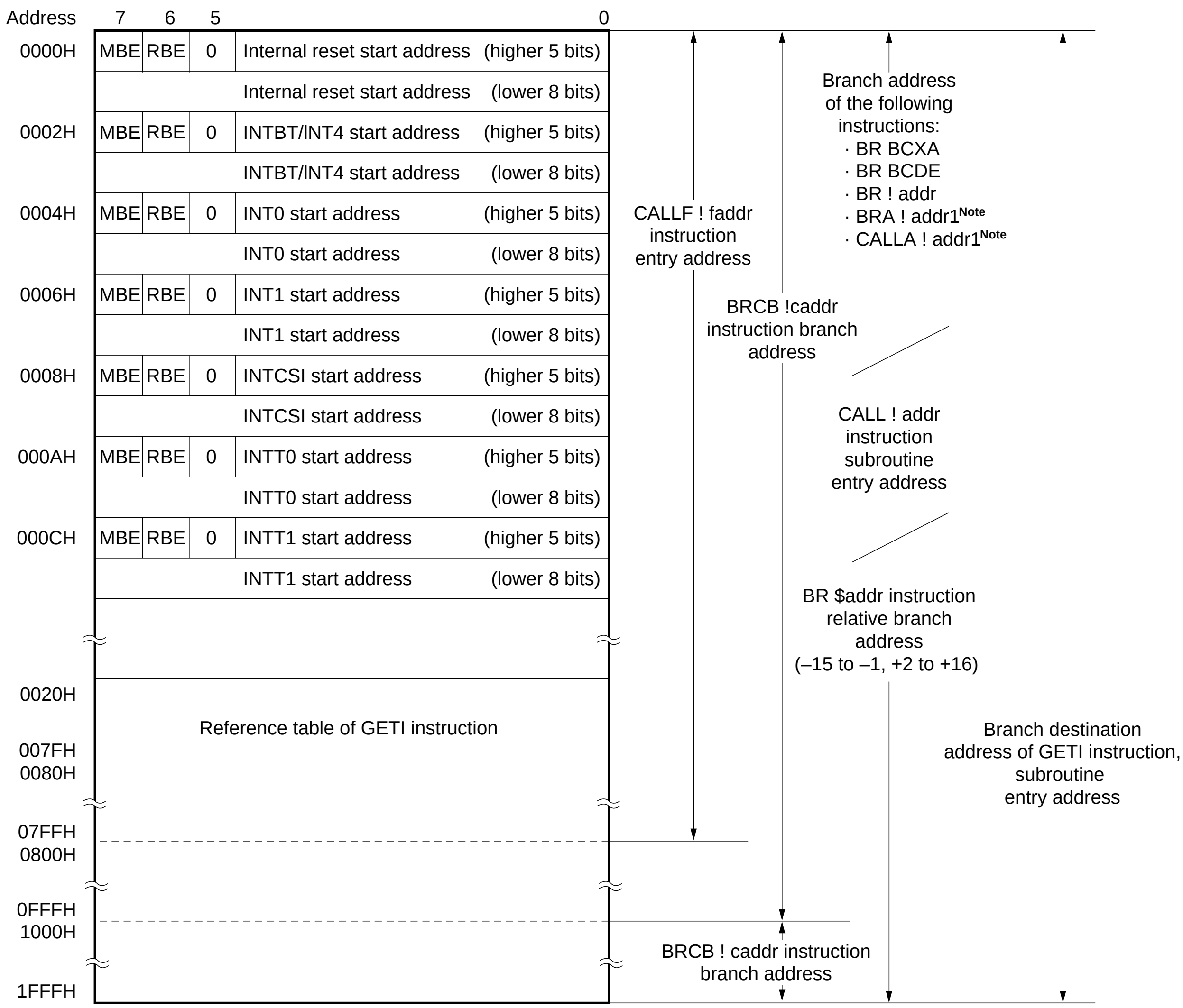


★ **Note** Can be used in the MkII mode only.

Remark With instructions other than above, execution can be branched to an address specified by the PC with only the lower 8 bits changed, by using the BR PCDE or BR PCXA instruction.

Fig. 4-3 Program Memory Map (2/4)

★ **(b) μPD750066**

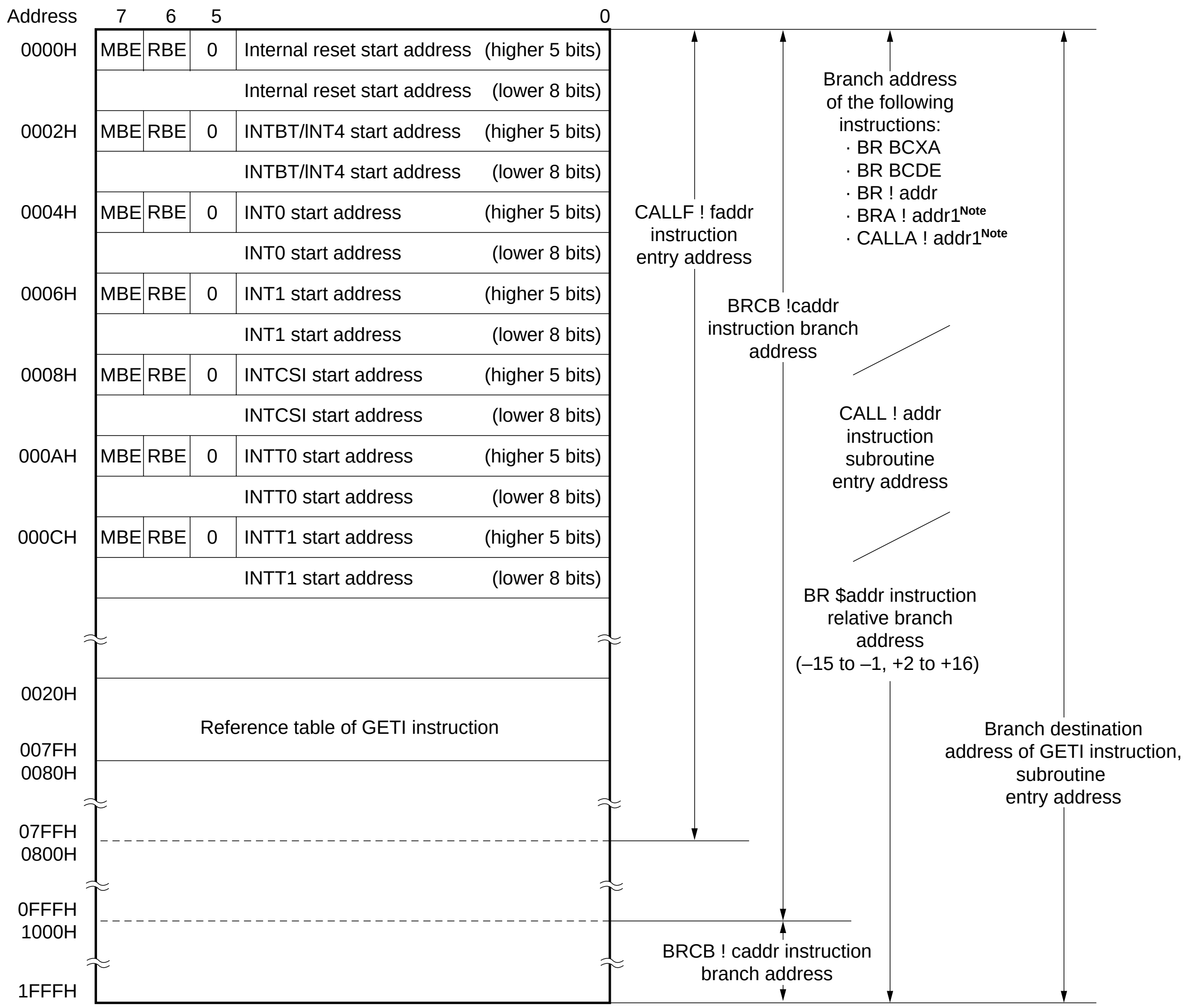


Note Can be used in the MkII mode only.

Remark With instructions other than above, execution can be branched to an address specified by the PC with only the lower 8 bits changed, by using the BR PCDE or BR PCXA instruction.

Fig. 4-3 Program Memory Map (3/4)

(c) μPD750068

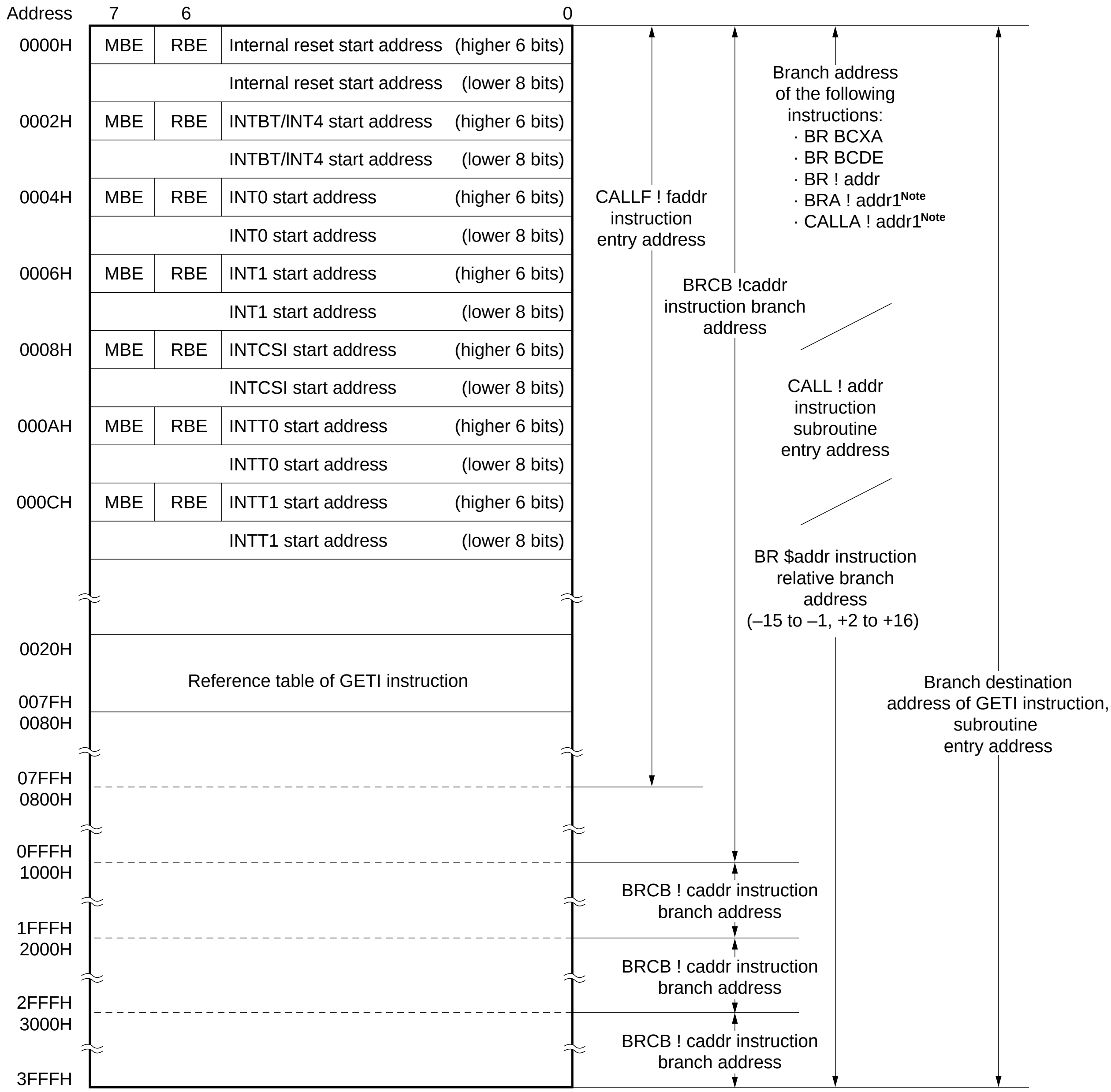


★ **Note** Can be used in the MkII mode only.

Remark With instructions other than above, execution can be branched to an address specified by the PC with only the lower 8 bits changed, by using the BR PCDE or BR PCXA instruction.

Fig. 4-3 Program Memory Map (4/4)

(d) μPD75P0076



★ **Note** Can be used in the MkII mode only.

Remark With instructions other than above, execution can be branched to an address specified by the PC with only the lower 8 bits changed, by using the BR PCDE or BR PCXA instruction.

4.4 Data Memory (RAM) ... 512 words × 4 bits

The data memory consists of data areas and a peripheral hardware area as shown in Fig. 4-4.
The data memory consists the following banks with each bank made up of 256 words × 4 bits:

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

4.4.1 Configuration of data memory

(1) Data area

A data area consists of a static RAM and is used to store data, and as a stack memory when a subroutine or interrupt is executed. The contents of this area can be retained for a long time by battery backup even when the CPU is halted in standby mode. The data area is manipulated by using memory manipulation instructions.

Static RAM is mapped to memory banks 0 and 1 in units of 256 words × 4 bits each. Although banks 0 and 1 are mapped as a data area, it can also be used as a general-purpose register area (000H through 01FH) and as a stack area**Note** (000H through 1FFH).

One address of the static RAM consists of 4 bits. However, it can be manipulated in 8-bit units by using an 8-bit memory manipulation instruction or in 1-bit units by using a bit manipulation instruction). To use an 8-bit manipulation instruction, specify an even address.

Note One stack area can be selected from memory bank 0 and 1.

- **General-purpose register area**

  This area can be manipulated by using a general-purpose register manipulation instruction or memory manipulation instruction. Up to eight 4-bit registers can be used. The registers not used by the program can be used as part of the data area or stack area. (Refer to **4.5 General-Purpose Register**.)

- **Stack area**

  The stack area is set by an instruction and is used as a saving area when a subroutine or interrupt processing is executed. (Refer to **4.7 Stack Pointer (SP) and Stack Bank Select Register (SBS)**.

(2) Peripheral hardware area

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

This area is manipulated by using a memory manipulation instruction, in the same manner as the static RAM. Note, however, that the bit units in which the peripheral hardware units can be manipulated differ depending on the address. The addresses to which no peripheral hardware unit is allocated cannot be accessed because these addresses are not provided to the data memory.

4.4.2 Specifying bank of data memory

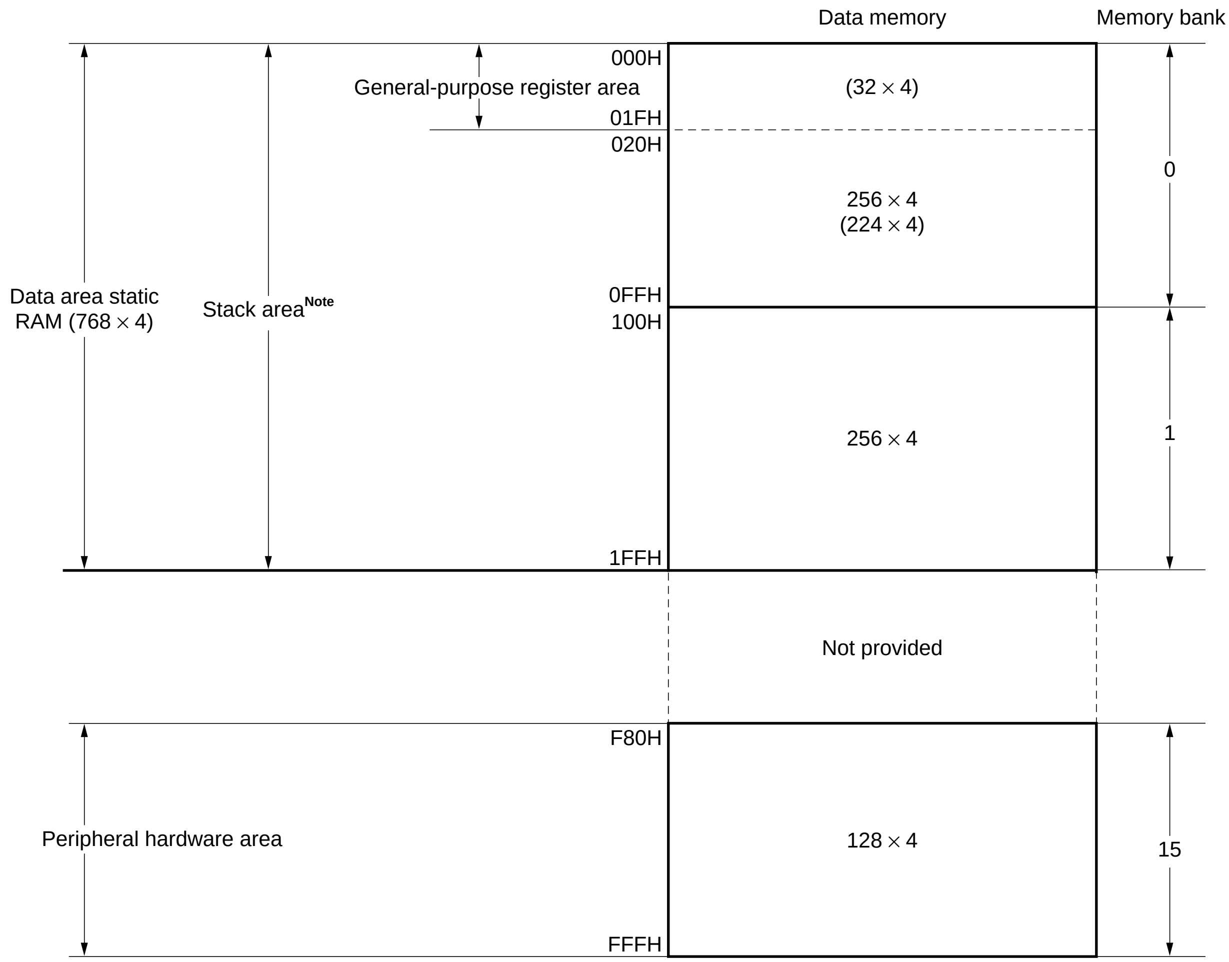
A memory bank is specified by a 4-bit memory bank select register (MBS) when bank specification is enabled by setting a memory bank enable flag (MBE) to 1 (MBS = 0, 1, or 15). When bank specification is disabled (MBE = 0), bank 0 or 15 is automatically specified depending on the addressing mode selected at that time. The addresses in the bank are specified by 8-bit immediate data or a register pair.

For the details of memory bank selection and addressing, refer to **3.1 Bank Configuration of Data Memory and Addressing Mode**.

For how to use a specific area of the data memory, refer to the following:

- General-purpose register area .... **4.5 General-Purpose Register**
- Stack area .... **4.7 Stack Pointer (SP) and Stack Bank Select Register (SBS)**
- Peripheral hardware area .... **CHAPTER 5 PERIPHERAL HARDWARE FUNCTION**

Fig. 4-4 Data Memory Map



Note One of memory banks 0 and 1 can be selected as the stack area.

The contents of the data memory are undefined at reset. Therefore, they must be initialized at the beginning of program execution (RAM clear). Otherwise, unexpected bugs may occur.

Example To clear RAM at addresses 000H through 1FFH

```
          SET1   MBE
          SEL    MB0
          MOV    XA, #00H
          MOV    HL, #04H
RAMC0 :   MOV    @HL, A     ;Clears 04H-FFHNote
          INCS   L          ;L ← L+1
          BR     RAMC0
          INCS   H          ;H ← H+1
          BR     RAMC0
          SEL    MB1
RAMC1 :   MOV    @HL, A     ;Clears 100H-1FFH
          INCS   L          ;L ← L+1
          BR     RAMC1
          INCS   H          ;H ← H+1
          BR     RAMC1
```

Note Data memory addresses 000H through 003H are not cleared because they are used as general-purpose register pairs XA and HL.

4.5 General-Purpose Register ... 8 × 4 bits × 4 banks

General-purpose registers are mapped to the specific addresses of the data memory. Four banks of registers, with each bank consisting of eight 4-bit registers (B, C, D, E, H, L, X, and A), are available.

The register bank (RB) that becomes valid when an instruction is executed is determined by the following expression:

RB = RBE· RBS (RBS = 0-3)

Each general-purpose register is manipulated in 4-bit units. Moreover, two registers can be used in pairs, such as BC, DE, HL, and XA, and manipulated in 8-bit units. Register pairs DE, HL, and DL are also used as data pointers.

When registers are manipulated in 8-bit units, the register pairs of the register bank (RB) with bit 0 inverted (0 ↔ 1, 2 ↔ 3), BC', DE', HL', and XA', can also be used in addition to BC, DE, HL, and XA (refer to **3.2 Bank Configuration of General-Purpose Registers**).

The general-purpose register are can be addressed and accessed as an ordinary RAM area, regardless of whether the registers in this area are used or not.

Fig. 4-5 Configuration of General-Purpose Register

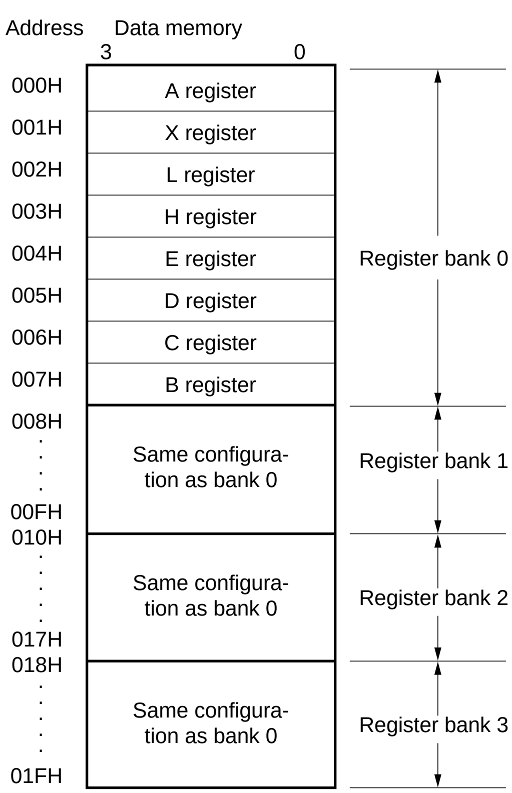
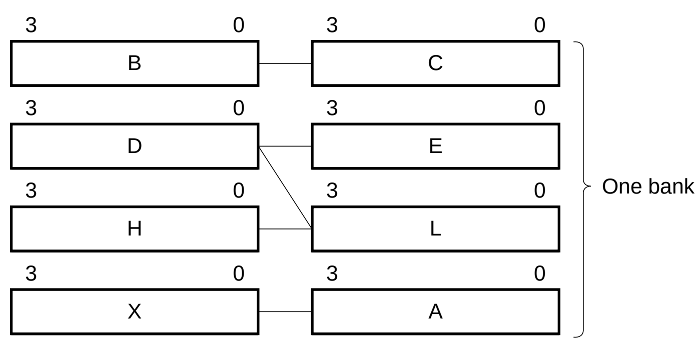


Fig. 4-6 Configuration of Register Pair

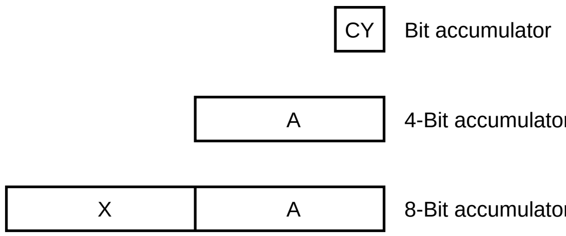


4.6 Accumulator

With the μPD750068, the A register or XA register pair functions as an accumulator. The A register plays a central role in 4-bit data processing, while the XA register pair is used for 8-bit data processing.

When a bit manipulation instruction is used, the carry flag (CY) is used as a bit accumulator.

Fig. 4-7 Accumulator



4.7 Stack Pointer (SP) and Stack Bank Select Register (SBS)

The μPD750068 uses a static RAM as the stack memory (LIFO). The stack pointer (SP) is an 8-bit register that holds information on the first address of the stack area.

The stack area consists of addresses 000H through 1FFH of memory bank 0, or 1. One memory bank is specified by 2-bit SBS (refer to **Table 4-2**).

Table 4-2 Stack Area Selected by SBS

SBS		Stack Area
SBS1	SBS2	
0	0	Memory bank 0
0	1	Memory bank 1
Other than above		Setting prohibited

The value of SP is decremented before data is written (saved) to the stack area, and is incremented after data has been read (restored) from the stack memory.

The data saved or restored to or from the stack are as shown in Figs. 4-9 through 4-12.

The initial values of SP and SBS are respectively set by an 8-bit memory manipulation instruction and 4-bit memory manipulation instruction, to determined the stack area. The values of SP and SBS can also be read.

Remark n = 0, 1

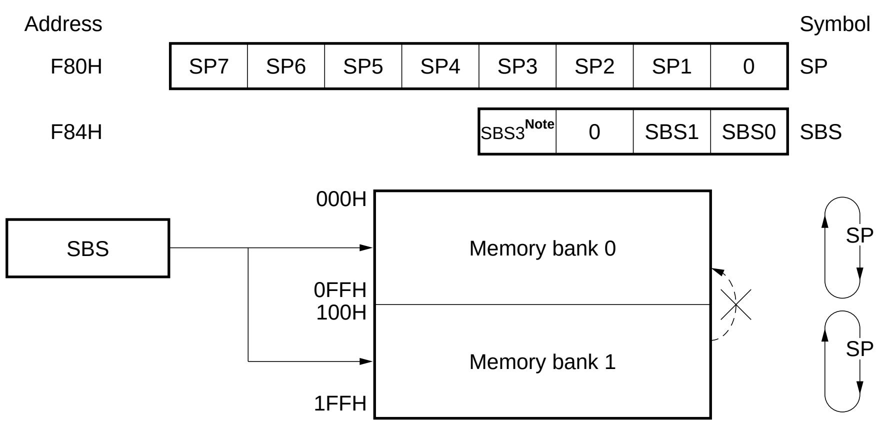
When 00H is set to SP as the initial value, the memory bank (n) specified by SBS is used as the stack area, starting from the highest address (nFFH).

The stack area can be used only in the memory bank specified by SBS. If an attempt is made to use an area exceeding address n00H as the stack area, the address is returned to nFFH in the same bank. This means that an area exceeding the boundary of a memory bank cannot be used as a stack area unless the value of SBS is rewritten.

The contents of SP become undefined, and the contents of SBS become 100B when the $\overline{\text{RESET}}$ signal is asserted. Therefore, be sure to initialize these to the desired values at the beginning of the program.

Remark n = 0, 1

Fig. 4-8 Configuration of Stack Pointer and Stack Bank Select Register



Note SBS3 can select MkI or MkII mode. The stack bank select function can be used in both the MkI and MkII modes (for details, refer to **4.1 Function to Select MkI and MkII Modes**).

Example To initialize SP

To allocate stack area to memory bank 1 and use area starting from address 1FFH as stack

```
SEL    MB15       ; or CLR1 MBE
MOV    A, #1
MOV    SBS, A     ; Specifies memory bank 1 as stack area
MOV    XA, #00H
MOV    SP, XA     ; SP ← 00H
```

Fig. 4-9 Data Saved to Stack Memory (MkI Mode)

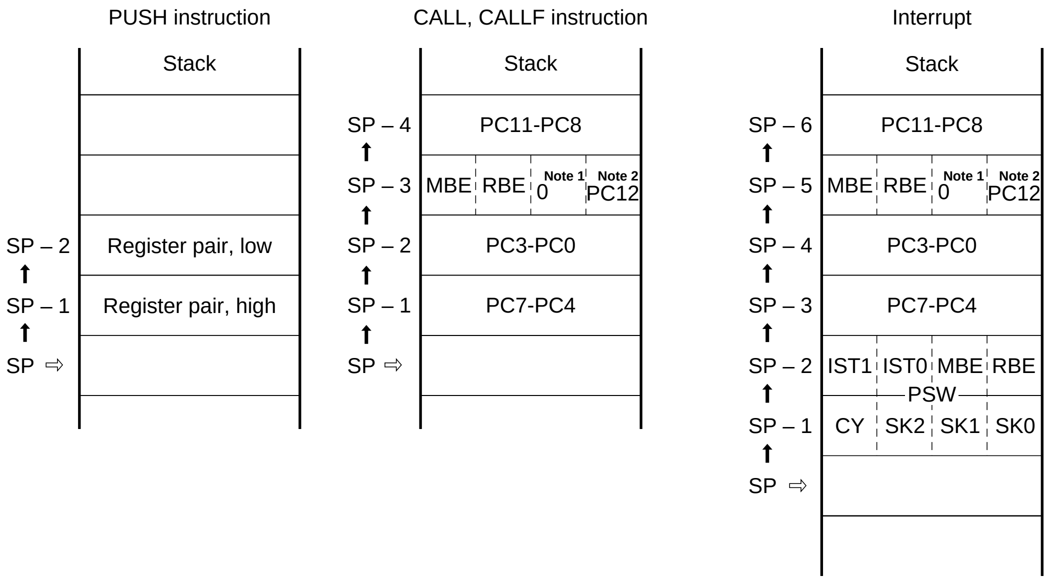
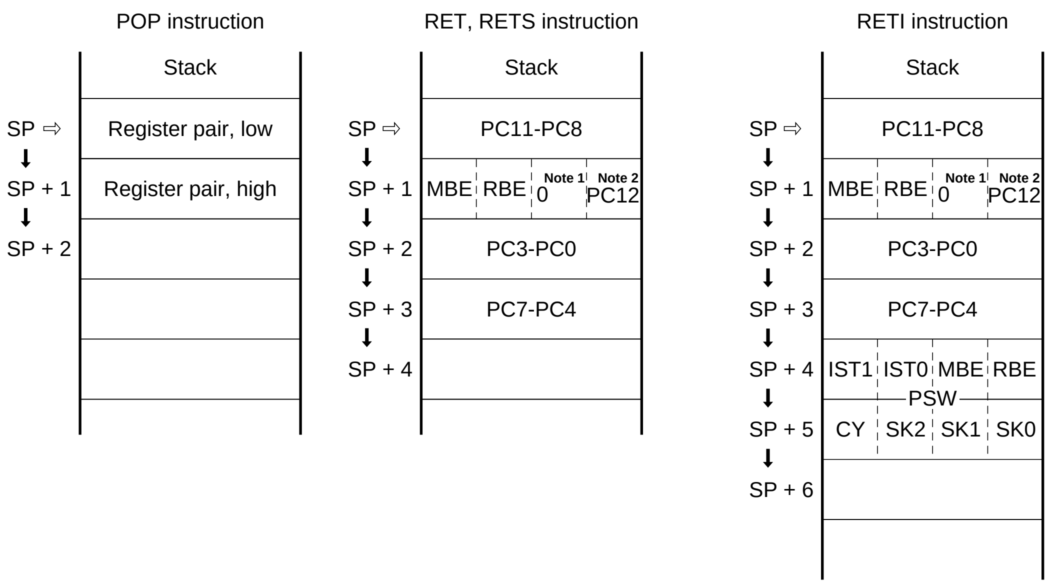


Fig. 4-10 Data Restored from Stack Memory (MkI Mode)



Notes **1.** In the case of the μPD75P0076, PC13 is placed here.

2. In the case of the μPD750064, 0 is placed here.

Fig. 4-11 Data Saved to Stack Memory (MkII Mode)

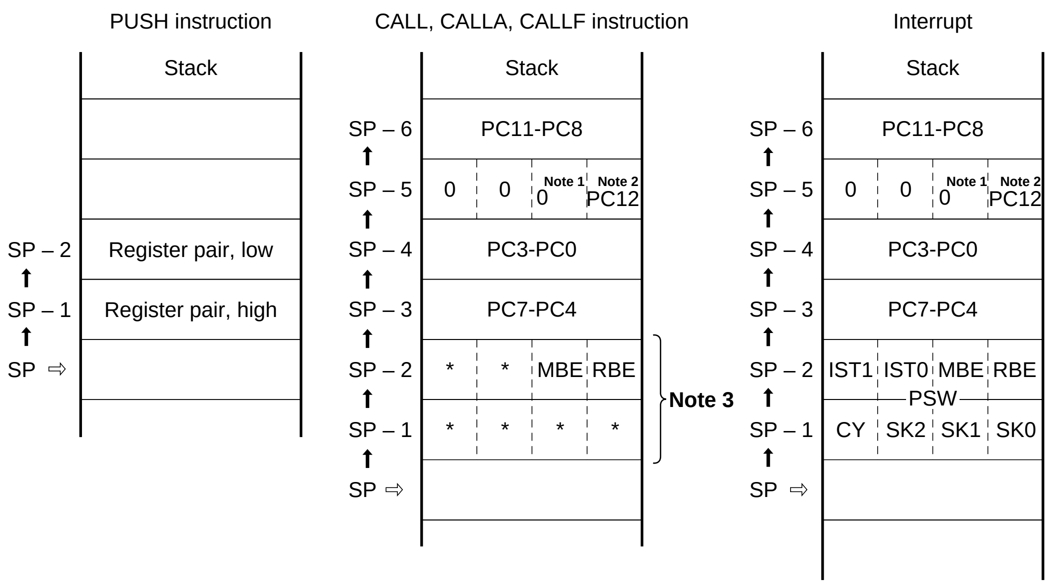
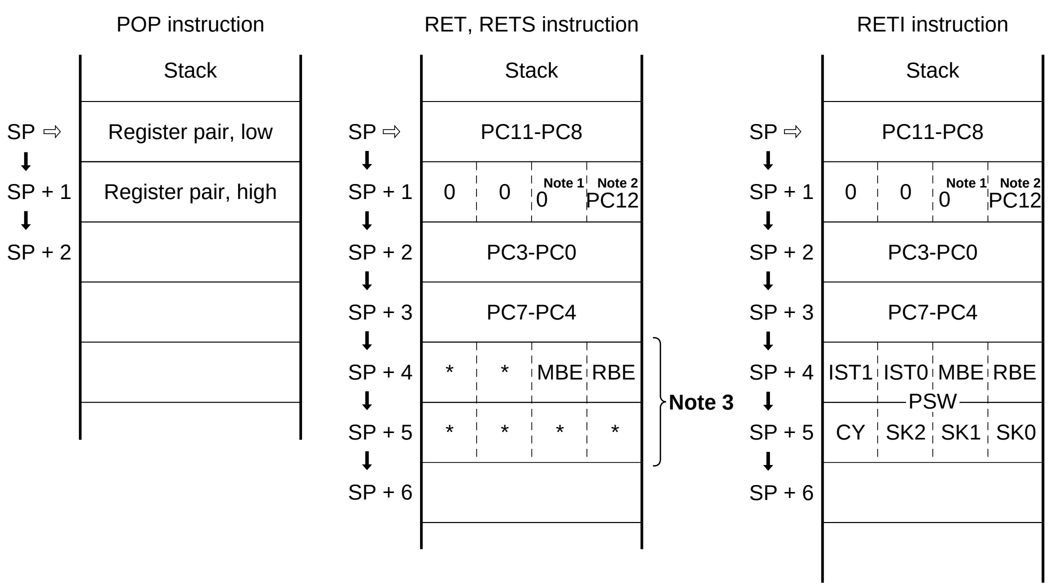


Fig. 4-12 Data Restored from Stack Memory (MkII Mode)



Notes **1.** In the case of the μPD75P0076, PC13 is placed here.
2. In the case of the μPD750064, 0 is placed here.
3. The contents of PSW other than MBE and RBE are not saved or restored.

Remark *: Undefined

4.8 Program Status Word (PSW) ... 8 bits

The program status word (PSW) consists of flags closely related to the operations of the processor.

PSW is mapped to addresses FB0H and FB1H of the data memory space, and the 4 bits of address FB0H can be manipulated by using a memory manipulation instruction.

Fig. 4-13 Configuration of Program Status Word

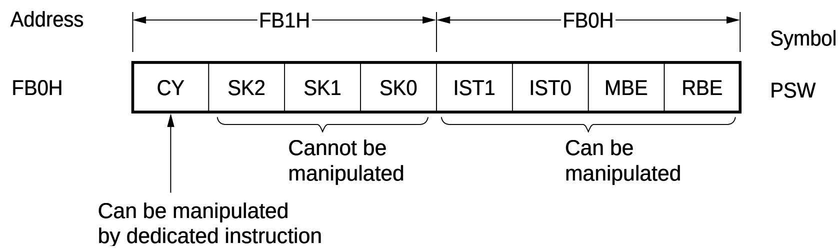


Table 4-3 PSW Flags Saved/Restored to/from Stack

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

(1) Carry flag (CY)

The carry flag records the occurrence of an overflow or underflow when an operation instruction with carry (ADDC or SUBC) is executed.

The carry flag also functions as a bit accumulator and can store the result of a Boolean operation performed between a specified bit address and data memory.

The carry flag is manipulated by using a dedicated instruction and is independent of the other PSW bits.

The carry flag becomes undefined when the $\overline{\text{RESET}}$ signal is asserted.

Table 4-4 Carry Flag Manipulation Instruction

	Instruction (Mnemonic)	Operation and Processing of Carry Flag
Carry flag manipulation instruction	SET1 CY	Sets CY to 1
	CLR1 CY	Clears CY to 0
	NOT1 CY	Inverts content of CY
	SKT CY	Skips if content of CY is 1
Bit transfer instruction	MOV1 mem*.bit, CY	Transfers content of CY to specified bit
	MOV1 CY, mem*.bit	Transfers content of specified bit to CY
Bit Boolean instruction	AND1 CY, mem*.bit	Takes ANDs, ORs, or XORs content of specified bit with content of CY and sets result to CY
	OR1 CY, mem*.bit	
	XOR1 CY, mem*.bit	
Interrupt processing	During interrupt execution	Saved to stack memory in parallel with other PSW bits in 8-bit units
	RETI	Restored from stack memory with other PSW bits

Remark mem*.bit indicates the following three bit manipulation addressing modes:

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

Example To AND bit 3 at address 3FH with P33 and output result to P50

```
MOV     H, #3H          ; Sets higher 4 bits of address to H register
MOV1    CY, @H+0FH.3    ; CY ← bit 3 of 3FH
AND1    CY, PORT3.3     ; CY ← CY∧ P33
MOV1    PORT5.0, CY     ; P50 ← CY
```

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

The skip flags record the skip status, and are automatically set or reset when the CPU executes an instruction. These flags cannot be manipulated directly by the user as operands.

(3) Interrupt status flags (IST1 and IST0)

The interrupt status flags record the status of the processing under execution (for details, refer to **Table 6-3 IST, IST0, and Interrupt Processing**).

Table 4-5 Contents of Interrupt Status Flags

IST1	IST0	Status of Processing being Executed	Processing and Interrupt Control
0	0	Status 0	Normal program is being executed. All interrupts can be acknowledged
0	1	Status 1	Interrupt with lower or higher priority is processed. Only an interrupt with higher priority can be acknowledged
1	0	Status 2	Interrupt with higher priority is processed. All interrupts are disabled from being acknowledged
1	1	—	Setting prohibited

The interrupt priority control circuit (refer to **Fig. 6-1 Block Diagram of Interrupt Control Circuit**) identifies the contents of these flags and controls the nesting of interrupts.

The contents of IST1 and 0 are saved to the stack along with the other bits of PSW when an interrupt is acknowledged, and the status is automatically updated by one. When the RETI instruction is executed, the values before the interrupt was acknowledged are restored to the interrupt status flags.

These flags can be manipulated by using a memory manipulation instruction, and the processing status under execution can be changed by program.

Caution To manipulate these flags, be sure to execute the DI instruction to disable the interrupts before manipulation. After manipulation, execute the EI instruction to enable the interrupts.

(4) Memory bank enable flag (MBE)

This flag specifies the address information generation mode of the higher 4 bits of the 12 bits of a data memory address.

MBE can be set or reset at any time by using a bit manipulation instruction, regardless of the setting of the memory bank.

When this flag is set to “1”, the data memory address space is expanded, and the entire data memory space can be addressed.

When MBE is reset to “0”, the data memory address space is fixed, regardless of MBS (refer to **Fig. 3-2 Configuration of Data Memory and Addressing Ranges of Respective Addressing Modes**).

When the $\overline{\text{RESET}}$ signal is asserted, the content of bit 7 of program memory address 0 is set. Also, MBE is automatically initialized.

When a vector interrupt is processed, the bit 7 of the corresponding vector address table is set. Also, the status of MBE when the interrupt is serviced is automatically set.

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

(5) Register bank enable flag (RBE)

This flag specifies whether the register bank of the general-purpose registers is expanded or not.

RBE can be set or reset at any time by using a bit manipulation instruction, regardless of the setting of the memory bank.

When this flag is set to “1”, one of four general-purpose register banks 0 to 3 can be selected depending on the contents of the register bank select register (RBS).

When RBE is reset to “0”, register bank 0 is always selected, regardless of the contents of the register bank select register (RBS).

When the $\overline{\text{RESET}}$ signal is asserted, the content of bit 6 of program memory address 0 is set to RBE, and RBE is automatically initialized.

When a vector interrupt occurs, the content of bit 6 of the corresponding vector address table is set to RBE. Also, the status of RBE when the interrupt is serviced is automatically set. Usually, RBE is reset to 0 during interrupt processing. Register bank 0 is selected for 4-bit processing, and register banks 0 and 1 are selected for 8-bit processing.

4.9 Bank Select Register (BS)

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

RBS and MBS are set by the SEL RBn and SEL MBn instructions, respectively.

BS can be saved to or restored from the stack area in 8-bit units by the PUSH BS or POP BS instruction.

Fig. 4-14 Configuration of Bank Select Register

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 that records the higher 4 bits of a 12-bit data memory address. This register specifies the memory bank to be accessed. With the μPD750068, however, only banks 0, 1 and 15 can be specified.

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

The address range specified by MBE and MBS is as shown in Fig. 3-2.

When the $\overline{\text{RESET}}$ signal is asserted, MBS is initialized to “0”.

(2) Register bank select register (RBS)

The register bank select register specifies a register bank to be used as general-purpose registers. It can select bank 0 to 3.

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

When the $\overline{\text{RESET}}$ signal is asserted, RBS is initialized to “0”.

Table 4-6 RBE, RBS, and Register Bank Selected

RBE	RBS				Register Bank
	3	2	1	0	
0	0	0	×	×	Fixed to bank 0
1	0	0	0	0	Selects bank 0
			0	1	Selects bank 1
			1	0	Selects bank 2
			1	1	Selects bank 3

(RBS bits 3 and 2:) Fixed to 0

× = don’t care

[MEMO]

5.1 Digital I/O Port

The μPD750068 uses memory mapped I/O, and all the I/O ports are mapped to the data memory space.

Fig. 5-1 Data Memory Address of Digital Port

Address	3	2	1	0	
FF0H	P03	P02	P01	P00	PORT0
FF1H	P13	P12	P11	P10	PORT1
FF2H	P23	P22	P21	P20	PORT2
FF3H	P33	P32	P31	P30	PORT3
FF4H	P43	P42	P41	P40	PORT4
FF5H	P53	P52	P51	P50	PORT5
FF6H	P63	P62	P61	P60	PORT6
FFBH	P113	P112	P111	P110	PORT11

Table 5-2 lists the instructions that manipulate the I/O ports. Ports 4 and 5 can be manipulated in 4-I/O, 8-I/O, and 1-bits. They are used for various control operations.

Examples 1. To test the status of P13 and outputs different values to ports 4 and 5 depending on the result

```
SKT   PORT1.3    ; Skips if bit 3 of port 1 is 1
MOV   XA, #18H   ; XA ← 18H  } String effect
MOV   XA, #14H   ; XA ← 14H  }
SEL   MB15       ; or CLR1 MBE
OUT   PORT4, XA  ; Ports 5, 4 ← XA
```

2.

```
SET1  PORT4.@L   ; Sets the bits of ports 4 and 5 specified by the L register to "1"
```

5.1.1 Types, features, and configurations of digital I/O ports

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

Figs. 5-2 through 5-6 show the configuration of each port.

Table 5-1 Types and Features of Digital Ports

Port (Pin Name)	Function	Operation and Feature	Remark
★ PORT0 (P00-P03)	4-bit input	When the serial interface function is used, shared pins can use output function depending on the operating mode	Shared with INT4, $\overline{\text{SCK}}$, SO/SB0, and SI/SB1
PORT1 (P10-P13)		4-bit input only port	Shared with INT0-INT2/TI1, and TI0
PORT2 (P20-P23)	4-bit I/O	Can be set to input or output mode in 4-bit units	Shared with PTO0, PRO1, PCL, and BUZ
PORT3 (P30-P33)		Can be set to input or output mode bit-wise.	Shared with MD0 to MD3[Note 1]
★ PORT4 (P40-P43)	4-bit I/O (N-ch open-drain withstand voltage 13 V)	Can be set to input or output mode in 4-bit units. Internal pull-up resistor specifiable by mask option[Note 2] bit-wise.	Shared with D0 to D3[Note 1]
PORT5 (P50-P53)		Data input/output in 8-bit units is possible in pairs.	Shared with D4 to D7[Note 1]
PORT6 (P60-P63)	4-bit I/O	Can be set to input or output mode bit-wise.	Shared with KR0 to KR3 and AN4 to AN7
PORT11 (P110-P113)	4-bit input	4-bit input only port	Shared with AN0 to AN3

Notes **1.** Pins only when μPD75P0076 is used.

★ **2.** The μPD75P0076 has no internal pull-up resistor by mask option.

P10 is shared with an external vector interrupt input pin and is provided with a noise rejection circuit (for details, refer to **6.3 Hardware Controlling Interrupt Function**).

Generation of a $\overline{\text{RESET}}$ signal clears all bits of each port register to "0", and therefore, the output buffers are turned off, and the ports are set to the input mode.

Fig. 5-2 Configuration of Ports 0 and 1

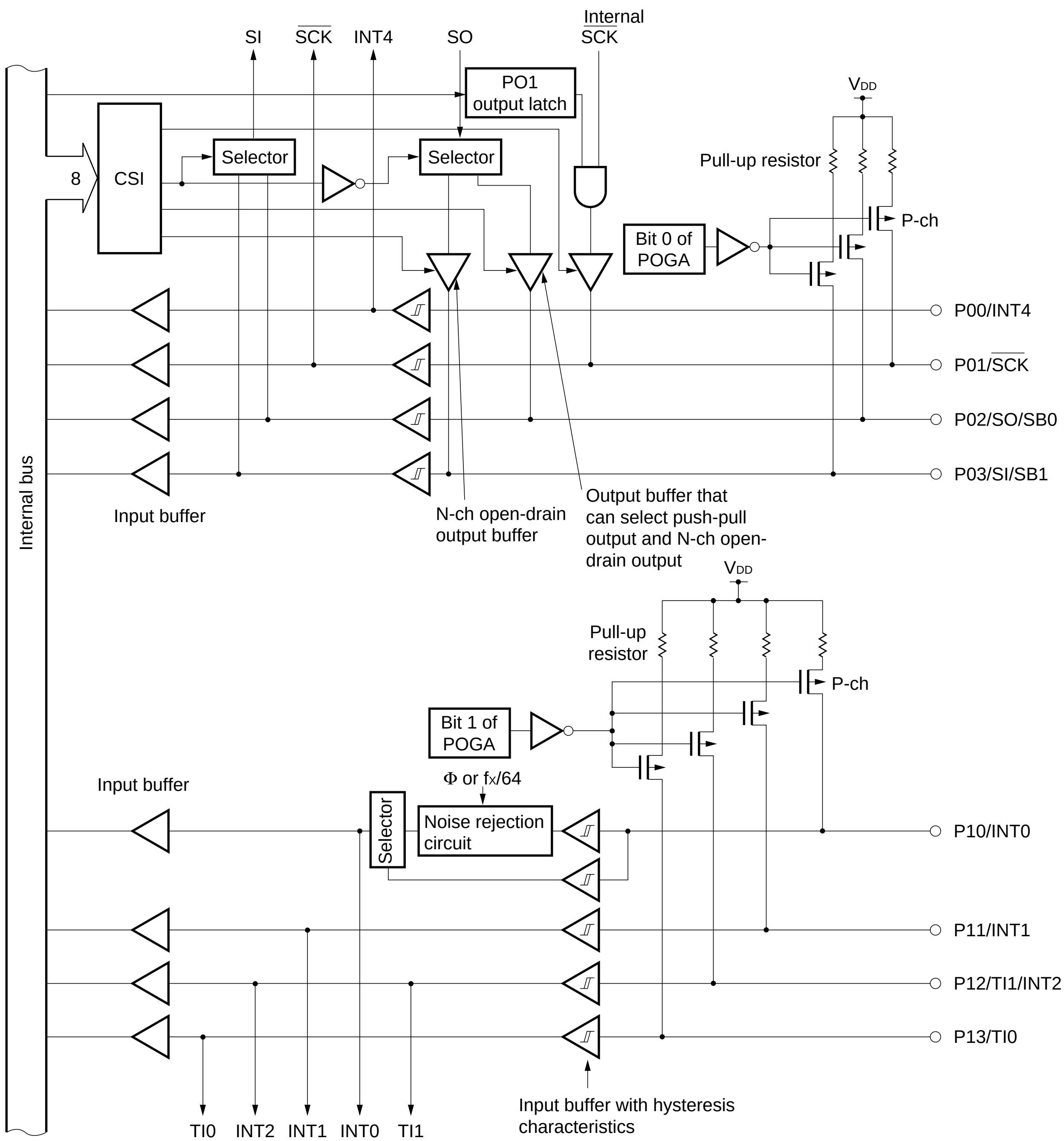


Fig. 5-3 Configuration of Ports 3 and 6

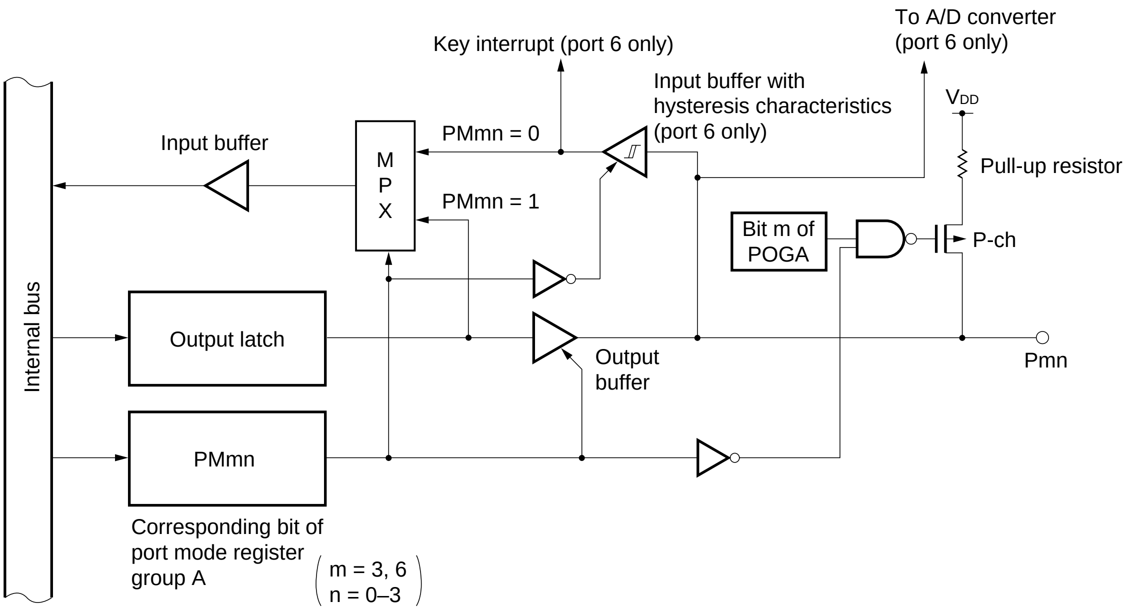


Fig. 5-4 Configuration of Port 2

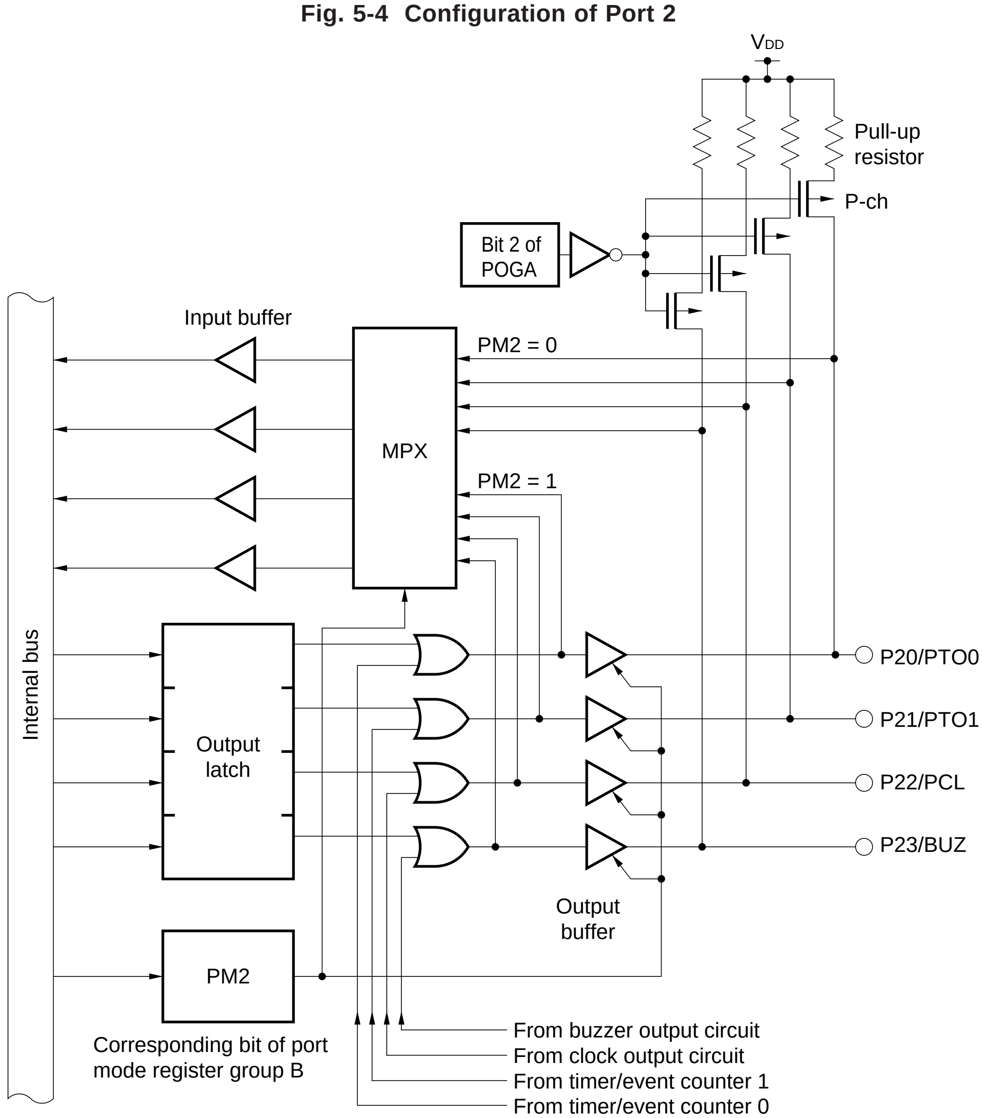


Fig. 5-5 Configuration of Ports 4 and 5

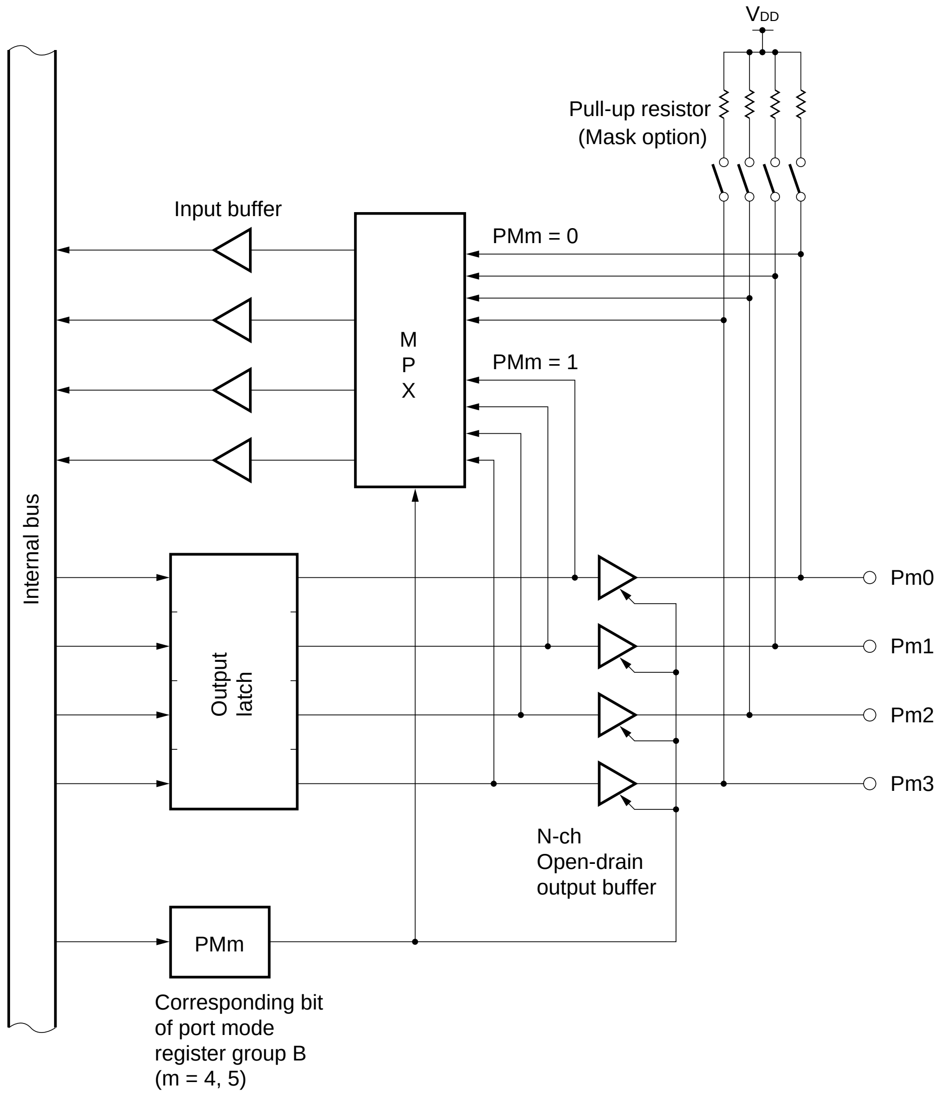
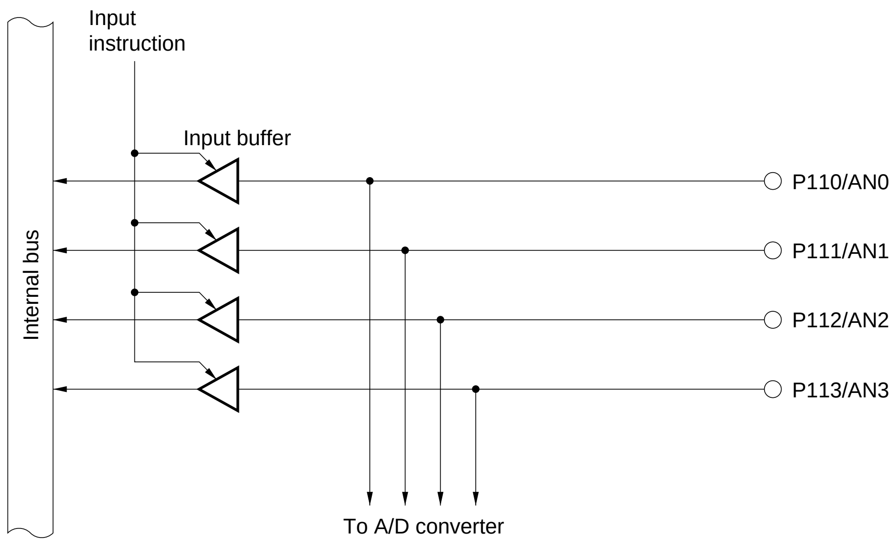


Fig. 5-6 Configuration of Port 11



5.1.2 Setting I/O mode

The input or output mode of each I/O port is set by the corresponding port mode register as shown in Fig. 5-7. Ports 3 and 6 can be set in the input or output mode in 1-bit units by using port mode register group A (PMGA). Ports 2, 4, and 5 are set by using port mode register group B (PMGB) in the input or output mode in 4-bit units.

Each port is set in the input mode when the corresponding port mode register bit is “0” and in the output mode when the corresponding register bit is “1”.

When a port is set in the output mode by the corresponding port mode register, the contents of the output latch are output to the output pin(s). Before setting the output mode, therefore, the necessary value must be written to the output latch.

Port mode register groups A, and B are set by using an 8-bit memory manipulation instruction.

When the $\overline{\text{RESET}}$ signal is asserted, all the bits of each port mode register are cleared to 0, the output buffer is turned off, and the corresponding port is set in the input mode.

Example To use P30, 31, 62, and 63 as input pins and P32, 33, 60, and 61 as output pins

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

Fig. 5-7 Format of Each Port Mode Register

	Specification
0	Input mode (output buffer off)
1	Output mode (output buffer on)

Port mode register group A

Address	7	6	5	4	3	2	1	0	Symbol
FE8H	PM63	PM62	PM61	PM60	PM33	PM32	PM31	PM30	PMGA

- PM30: Sets P30 in input or output mode
- PM31: Sets P31 in input or output mode
- PM32: Sets P32 in input or output mode
- PM33: Sets P33 in input or output mode
- PM60: Sets P60 in input or output mode
- PM61: Sets P61 in input or output mode
- PM62: Sets P62 in input or output mode
- PM63: Sets P63 in input or output mode

Port mode register group B

Address	7	6	5	4	3	2	1	0	Symbol
FECH	–	–	PM5	PM4	–	PM2	–	–	PMGB

- PM2: Sets port 2 (P20-P23) in input or output mode
- PM4: Sets port 4 (P40-P43) in input or output mode
- PM5: Sets port 5 (P50-P53) in input or output mode

5.1.3 Digital I/O port manipulation instruction

Because all the I/O ports of the μPD750068 are mapped to the data memory space, they can be manipulated by using data memory manipulation instructions. Table 5-2 shows these data memory manipulation instructions which are considered to be especially useful for manipulating the I/O pins and their range of applications.

(1) Bit manipulation instruction

Because the specific address bit direct addressing (fmem.bit) and specific address bit register indirect addressing (pmem.@L) are applicable to digital I/O ports 0 through 6, and 11, the bits of these ports can be manipulated regardless of the specifications by MBE and MBS.

Example To OR P50 and P41 and set P61 in output mode

```
MOV1    CY, PORT5.0 ;  CY ← P50
OR1     CY, PORT4.1 ;  CY ← CY∨ P41
MOV1    PORT6.1, CY ;  P61 ← CY
```

(2) 4-bit manipulation instruction

In addition to the IN and OUT instructions, all the 4-bit memory manipulation instructions such as MOV, XCH, ADDS, and INCS can be used to manipulate the ports in 4-bit units. Before executing these instructions, however, memory bank 15 must be selected.

Examples 1. To output the contents of the accumulator to port 3

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

2. To add the value of the accumulator to the data output to port 5

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

3. To test whether the data of port 4 is greater than the value of the accumulator

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

(3) 8-bit manipulation instruction

In addition to the IN and OUT instructions, the MOV, XCH, and SKE instructions can be used to manipulate ports 4 and 5 in 8-bit units. In this case, memory bank 15 must be selected in advance as in the case of manipulating ports in 4-bit units.

Example To output the data of register pair BC to an output specified by the 8-bit data input from 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 List of I/O Pin Manipulation Instructions

<table>
<tr><th colspan="2"></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 11</th></tr>
<tr><td>IN</td><td>A,PORTn Note 1</td><td colspan="8">○</td></tr>
<tr><td>IN</td><td>XA,PORTn Note 1</td><td colspan="2">–</td><td colspan="2">–</td><td colspan="2">○</td><td colspan="2">–</td></tr>
<tr><td>OUT</td><td>PORTn, A Note 1</td><td colspan="2">–</td><td colspan="5">○</td><td>–</td></tr>
<tr><td>OUT</td><td>PORTn, XA Note 1</td><td colspan="2">–</td><td colspan="2">–</td><td colspan="2">○</td><td colspan="2">–</td></tr>
<tr><td>MOV</td><td>A, PORTn Note 1</td><td colspan="8">○</td></tr>
<tr><td>MOV</td><td>XA, PORTn Note 1</td><td colspan="2">–</td><td colspan="2">–</td><td colspan="2">○</td><td colspan="2">–</td></tr>
<tr><td>MOV</td><td>PORTn, A Note 1</td><td colspan="2">–</td><td colspan="5">○</td><td>–</td></tr>
<tr><td>MOV</td><td>PORTn, XA Note 1</td><td colspan="2">–</td><td colspan="2">–</td><td colspan="2">○</td><td colspan="2">–</td></tr>
<tr><td>XCH</td><td>A, PORTn Note 1</td><td colspan="8">○</td></tr>
<tr><td>XCH</td><td>XA, PORTn Note 1</td><td colspan="2">–</td><td colspan="2">–</td><td colspan="2">○</td><td colspan="2">–</td></tr>
<tr><td>MOV1</td><td>CY, PORTn.bit</td><td colspan="8">○</td></tr>
<tr><td>MOV1</td><td>CY, PORTn.@L Note 2</td><td colspan="8">○</td></tr>
<tr><td>MOV1</td><td>PORTn.bit, CY</td><td>–</td><td>–</td><td colspan="6">○</td></tr>
<tr><td>MOV1</td><td>PORTn.@L, CY Note 2</td><td>–</td><td>–</td><td colspan="6">○</td></tr>
<tr><td>INCS</td><td>PORTn Note 1</td><td colspan="8">○</td></tr>
<tr><td>SET1</td><td>PORTn.bit</td><td colspan="2">–</td><td colspan="5">○</td><td>–</td></tr>
<tr><td>SET1</td><td>PORTn.@L Note 2</td><td colspan="2">–</td><td colspan="5">○</td><td>–</td></tr>
<tr><td>CLR1</td><td>PORTn.bit</td><td colspan="2">–</td><td colspan="5">○</td><td>–</td></tr>
<tr><td>CLR1</td><td>PORTn.@L Note 2</td><td colspan="2">–</td><td colspan="5">○</td><td>–</td></tr>
<tr><td>SKT</td><td>PORTn.bit</td><td colspan="8">○</td></tr>
<tr><td>SKT</td><td>PORTn.@L Note 2</td><td colspan="8">○</td></tr>
<tr><td>SKF</td><td>PORTn.bit</td><td colspan="8">○</td></tr>
<tr><td>SKF</td><td>PORTn.@L Note 2</td><td colspan="8">○</td></tr>
<tr><td>SKTCLR</td><td>PORTn.bit</td><td colspan="8">○</td></tr>
<tr><td>SKTCLR</td><td>PORTn.@L Note 2</td><td colspan="8">○</td></tr>
<tr><td>AND1</td><td>CY, PORTn.bit</td><td colspan="8">○</td></tr>
<tr><td>AND1</td><td>CY, PORTn.@L Note 2</td><td colspan="8">○</td></tr>
<tr><td>OR1</td><td>CY, PORTn.bit</td><td colspan="8">○</td></tr>
<tr><td>OR1</td><td>CY, PORTn.@L Note 2</td><td colspan="8">○</td></tr>
<tr><td>XOR1</td><td>CY, PORTn.bit</td><td colspan="8">○</td></tr>
<tr><td>XOR1</td><td>CY, PORTn.@L Note 2</td><td colspan="8">○</td></tr>
</table>

Notes **1.** MBE = 0 or (MBE = 1, MBS = 15) before these instructions are executed.

2. The lower 2 bits of the address and the bit address are indirectly specified by the L register.

5.1.4 Operation of digital I/O port

The operations of each port and port pin when a data memory manipulation instruction is executed to manipulate a digital I/O port differ depending on whether the port is set in the input or output mode (refer to **Table 5-3**). This is because, as can be seen from the configuration of the I/O port, the data of each pin is loaded to the internal bus in the input mode, and the data of the output latch is loaded to the internal bus in the output mode.

(1) Operation in input mode

When a test instruction such as SKT, a bit input instruction such as MOV1, or an instruction that loads port data to the internal bus in 4- or 8-bit units, such as IN, MOV, operation, or comparison instruction, is executed, the data of each pin is manipulated.

When an instruction that transfers the contents of the accumulator in 4- or 8-bit units, such as OUT or MOV, is executed, the data of the accumulator is latched to the output latch. The output buffer remains off.

When the XCH instruction is executed, the data of each pin is input to the accumulator, and the data of the accumulator is latched to the output latch. The output buffer remains off.

When the INCS instruction is executed, the data (4 bits) of each pin incremented by one (+1) is latched to the output latch. The output buffer remains off.

When an instruction that rewrites the data memory contents in 1-bit units, such as SET1, CLR1, MOV1, or SKTCLR, is executed, the contents of the output latch of the specified bit can be rewritten as specified by the instruction, but the contents of the output latches of the other bits are undefined.

(2) Operation in output mode

When a test instruction, bit input instruction, or an instruction in 4- or 8-bit units that loads port data to the internal bus is executed, the contents of the output latch are manipulated.

When an instruction that transfers the contents of the accumulator in 4- or 8-bit units is executed, the data of the output latch is rewritten and at the same time output from the port pins.

When the XCH instruction is executed, the contents of the output latch are transferred to the accumulator. The contents of the accumulator are latched to the output latches of the specified port and output from the port pins.

When the INCS instruction is executed, the contents of the output latches of the specified port are incremented by 1 and output from the port pins.

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

Table 5-3 Operation When I/O Port Is Manipulated

Instruction Executed	Operation of Port and Pin	
	Input mode	Output mode
SKT <1> SKF <1>	Tests pin data	Test output latch data
MOV1 CY, <1>	Transfers pin data to CY	Transfers output latch data to CY
AND1 CY, <1> OR1 CY, <1> XOR1 CY, <1>	Performs operation between pin data and CY	Performs operation between output latch data and CY

IN A, PORTn
IN XA, PORTn
★ MOV A, PORTn
★ MOV XA, PORTn
MOV A, @HL
MOV XA, @HL
	Transfers pin data to accumulator	Transfers output latch data to accumulator

ADDS A, @HL
ADDC A, @HL
SUBS A, @HL
SUBC A, @HL
AND A, @HL
OR A, @HL
XOR A, @HL
	Performs operation between pin data and accumulator	Performs operation between output latch data and accumulator

SKE A, @HL SKE XA, @HL	Compares pin data with accumulator	Compares output latch data with accumulator

OUT PORTn, A
OUT PORTn, XA
★ MOV PORTn, A
★ MOV PORTn, XA
MOV @HL, A
MOV @HL, XA
	Transfers accumulator data to output latch (output buffer remains off)	Transfers accumulator data to output latch and outputs data from pins

XCH A, PORTn XCH XA, PORTn XCH A, @HL XCH XA, @HL	Transfers pin data to accumulator and accumulator data to output latch (output buffer remains off)	Exchanges data between output latch and accumulator
INCS PORT INCS @HL	Increments pin data by 1 and latches it to output latch	Increments output latch contents by 1
SET1 <1> CLR1 <1> MOV1 <1>, CY SKTCLR <1>	Rewrites output latch contents of specified bit as specified by instruction. However, output latch contents of other bits are undefined	Changes status of output pin as specified by instruction

Remark <1> : Indicates two addressing modes: PORTn, bit and PORTn.@L.

5.1.5 Connecting pull-up resistor

Each port pin of the μPD750068 can be connected with a pull-up resistor (except the P00 pin). Some pins can be connected with a pull-up resistor via software and the others can be connected by mask option.

Table 5-4 shows how to specify the connection of the pull-up resistor to each port pin. Connection of the internal pull-up resistor is specified via software in the format shown in Fig. 5-8.

Connection of the internal pull-up resistor can be specified only to the pins of ports 3 and 6 in the input mode. When the pins are set in the output mode, connection of the internal pull-up resistor cannot be specified regardless of the setting of POGA.

Table 5-4 Specifying Connection of Pull-up Resistor

Port (Pin Name)	Specifying Connection of Pull-up Resistor	Specified Bit
Port 0 (P01-P03)[Note]	Specifying connection in 3-bit units via software	POGA.0
Port 1 (P10-P13)	Specifying connection in 4-bit units via software	POGA.1
Port 2 (P20-P23)		POGA.2
Port 3 (P30-P33)		POGA.3
Port 4 (P40-P43)	Connected in 1-bit units by mask option	—
Port 5 (P50-P53)		
Port 6 (P60-P63)	Specifying connection in 4-bit units via software	POGA.6

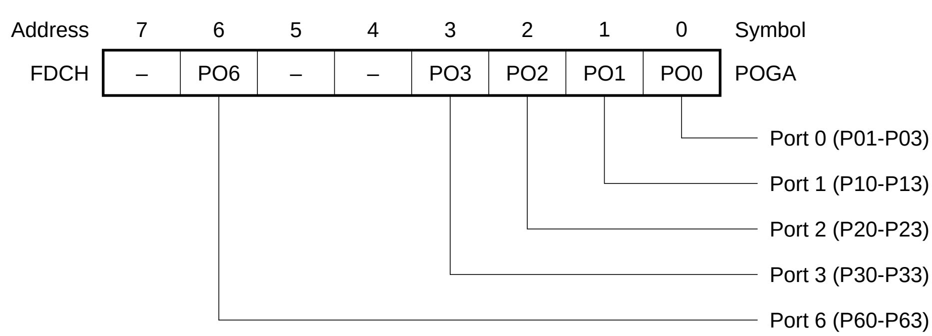
Note Connection of an internal pull-up resistor cannot be specified for the P00 pin.

Remark The port pins of the μPD75P0076 are not connected with the pull-up resistor by the mask option.

Fig. 5-8 Format of Pull-up Resistor Specification Register

	Specification
0	Does not specify connection of internal pull-up resistor
1	Specify connection of internal pull-up resistor

Pull-up resistor specification register group A



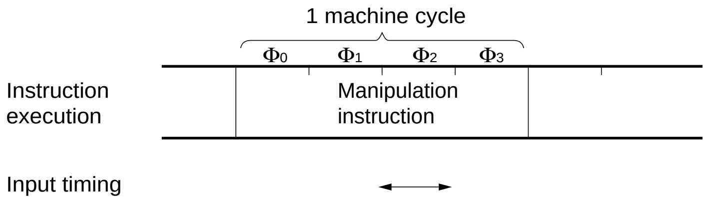
5.1.6 I/O timing of digital I/O port

Fig. 5-9 shows the timing at which data is output to the output latch and the timing at which the pin data or the data of the output latch is loaded to the internal bus.

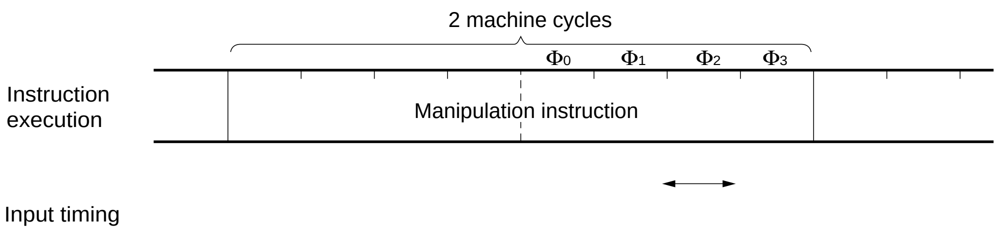
Fig. 5-10 shows the ON timing when connection of an internal pull-up resistor is specified to a port pin via software.

Fig. 5-9 I/O Timing of Digital I/O Port

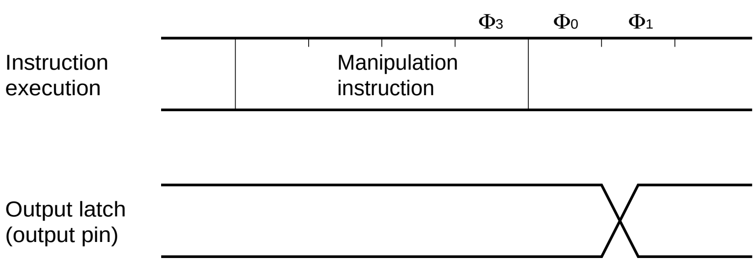
★ **(a) When data is loaded by 1-machine cycle instruction**



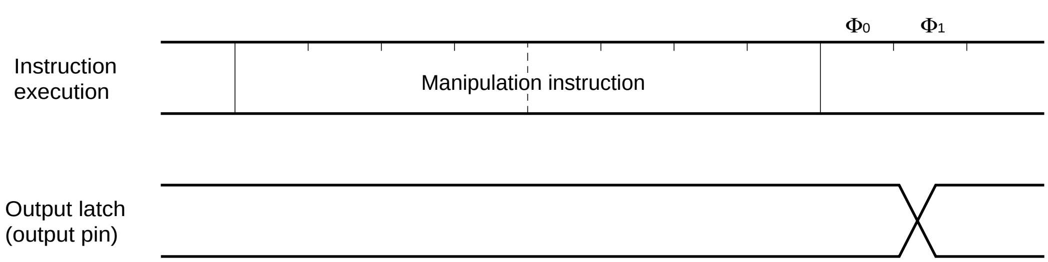
★ **(b) When data is loaded by 2-machine cycle instruction**



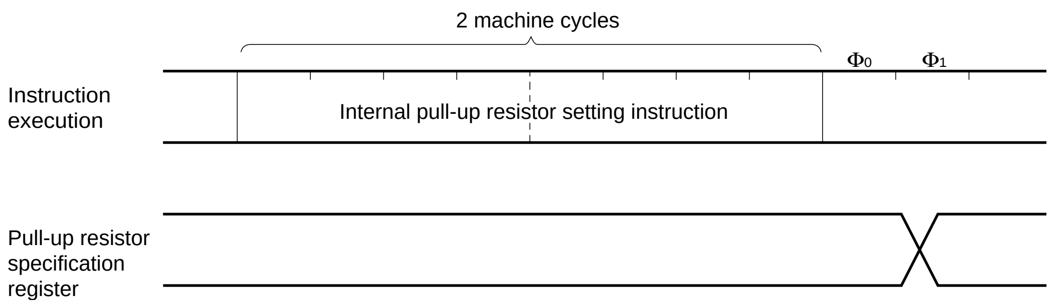
(c) When data is latched by 1-machine cycle instruction



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



★ **Fig. 5-10 ON Timing of Internal Pull-up Resistor Connected via Software**



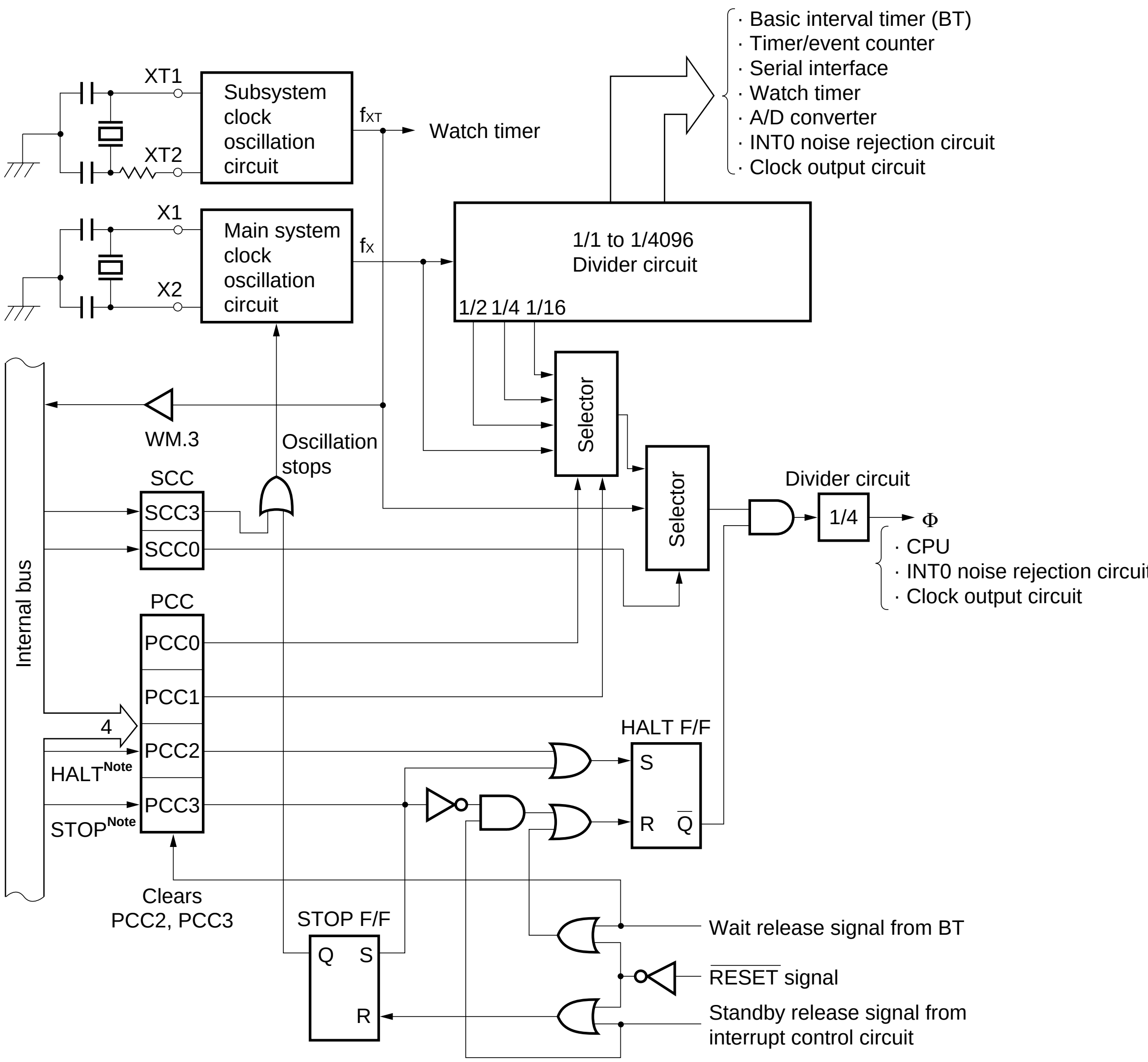
5.2 Clock Generation Circuit

The clock generation circuit supplies various clocks to the CPU and peripheral hardware units and controls the operation mode of the CPU.

5.2.1 Configuration of clock generation circuit

Fig. 5-11 shows the configuration of the clock generation circuit.

Fig. 5-11 Block Diagram of Clock Generation Circuit



Note Instruction execution

Remarks
1. f_X = 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 Φ is one machine cycle of an instruction.

5.2.2 Function and operation of clock generation circuit

The clock generation circuit generates the following types of clocks and controls the operation mode of the CPU in the standby mode:

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

The operation of the clock generation circuit is determined by the processor clock control register (PCC) and system clock control register (SCC), as follows:

(a) When the $\overline{\text{RESET}}$ signal is asserted, the slowest mode of the main system clock (10.7 μs at 6.00 MHz) is selected (PCC = 0, SCC = 0).

(b) The CPU clock can be changed in four steps (0.67, 1.33, 2.67, or 10.7 μs at 6.00 MHz) by PCC with the main system clock selected.

(c) Two standby modes, STOP and HALT, can be used with the main system clock selected.

(d) Ultra low-speed, power-saving (122 μs at 32.768 kHz) can be performed with the subsystem clock selected by SCC. In this case, the value set for PCC has no influence on the CPU clock.

(e) The oscillation of the main system clock can be stopped by SCC when the subsystem clock has been selected. Moreover, the HALT mode can be used. However, the STOP mode cannot be used. (The oscillation of the subsystem clock cannot be stopped.)

(f) The main system clock is divided and supplied to the peripheral hardware units. The subsystem clock can be directly supplied only to the watch timer. Therefore, the watch function, and the buzzer output function that operate on the clock supplied from the watch timer can continue their operations even in the standby mode.

(g) The watch timer and LCD controller can continue their operations when the subsystem clock has been selected. The serial interface and timer/event counter can continue operation when an external clock or watch timer has been used as the clock. The other hardware units, however, operate on the main system clock and therefore, cannot be used when the main system clock is stopped.

(1) Processor clock control register (PCC)

PCC is a 4-bit register that selects the CPU clock Φ with the lower 2 bits and controls the CPU operation mode with the higher 2 bits (refer to **Fig. 5-12**).

When either bit 3 or 2 of this register is set to “1”, the standby mode is set. When the standby mode has been released by the standby release signal, both the bits are automatically cleared and the normal operation mode is set (for details, refer to **CHAPTER 7 STANDBY FUNCTION**).

The lower 2 bits of PCC are set by a 4-bit memory manipulation instruction (clear the higher 2 bits to “0”). Bits 3 and 2 are set to “1” by the STOP and HALT instructions, respectively.

The STOP and HALT instructions can always be executed regardless of the contents of MBE.

The CPU clock can be selected only when the processor operates with the main system clock. When the subsystem clock is used, the lower 2 bits of PCC are invalid, and the clock frequency is fixed to $f_{XT}/4$. The STOP instruction can be executed only when the processor operates with the main system clock.

Examples 1. To set the fastest mode of machine cycle (0.67 μs at 6.00 MHz)

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

2. To set the machine cycle to 1.91 μs (f_X = 4.19 MHz)

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

3. To set STOP mode (be sure to write NOP instruction after STOP and HALT instructions)

```
STOP
NOP
```

PCC is cleared to “0” when the $\overline{\text{RESET}}$ signal is asserted.

Fig. 5-12 Format of Processor Clock Control Register

Address	3	2	1	0	Symbol
FB3H	PCC3	PCC2	PCC1	PCC0	PCC

CPU clock select bit

(f_X = 6.0 MHz)

		SCC3, SCC0 = 00 ( ): f_X = 6.0MHz		SCC3, SCC0 = 01 or 11 ( ): f_{XT} = 32,768 kHz	
		CPU clock frequency	I machine cycle	CPU clock frequency	1 machine cycle
0	0	$\Phi = f_X/64$ (93.8 kHz)	10.7 μs	$\Phi = f_{XT}/4$ (8.192 kHz)	122 μs
0	1	$\Phi = f_X/16$ (375 kHz)	2.67 μs		
1	0	$\Phi = f_X/8$ (750 kHz)	1.33 μs		
1	1	$\Phi = f_X/4$ (1.5 MHz)	0.67 μs		

(f_X = 4.19 MHz)

		SCC3, SCC0 = 00 ( ): f_X = 4.19 MHz		SCC3, SCC0 = 01 or 11 ( ): f_{XT} : 32.768 kHz	
		CPU clock frequency	I machine cycle	CPU clock frequency	1 machine cycle
0	0	$\Phi = f_X/64$ (65.5 kHz)	15.3 μs	$\Phi = f_{XT}/4$ (8.192 kHz)	122 μs
0	1	$\Phi = f_X/16$ (262 kHz)	3.81 μs		
1	0	$\Phi = f_X/8$ (524 kHz)	1.91 μs		
1	1	$\Phi = f_X/4$ (1.05 MHz)	0.95 μs		

Remarks 1. f_X: main system clock oscillation circuit output frequency
2. f_{XT}: subsystem clock oscillation circuit output frequency

CPU operation mode control bit

0	0	Normal operation mode
0	1	HALT mode
1	0	STOP mode
1	1	Setting prohibited

(2) System clock control register (SCC)

SCC is a 4-bit register that selects CPU clock Φ with its least significant bit and controls oscillation of the main system clock with the most significant bit (refer to **Fig. 5-13**).

Although bits 0 and 3 of SCC exist at the same data memory address, both the bits cannot be changed at the same time. To set bits 0 and 3 of SCC, therefore, use a bit manipulation instruction. Bits 0 and 3 of SCC can be always manipulated regardless of the content of MBE.

Oscillation of the main system clock can be stopped by setting bit 3 of SCC only when the processor operates with the subsystem clock. To stop oscillation of the main system clock, use the STOP instruction

SCC is cleared to “0” when the $\overline{\text{RESET}}$ signal is asserted.

Fig. 5-13 Format of System Clock Control Register

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

SCC3	SCC0	CPU clock selection	Main system clock oscillation
0	0	Main system clock	Can oscillate
0	1	Subsystem clock	
1	0	Setting prohibited	
1	1	Subsystem clock	Oscillation stopped

Cautions 1. It takes up to $1/f_{XT}$ to change the system clock. To stop oscillation of the main system clock, therefore, set SCC.3 to 1 after the subsystem clock has been selected and the number of machine cycles shown in Table 5-5 has elapsed.

2. The STOP mode cannot be set even if the oscillation is stopped by setting SCC.3 when the processor operates with the main system clock.

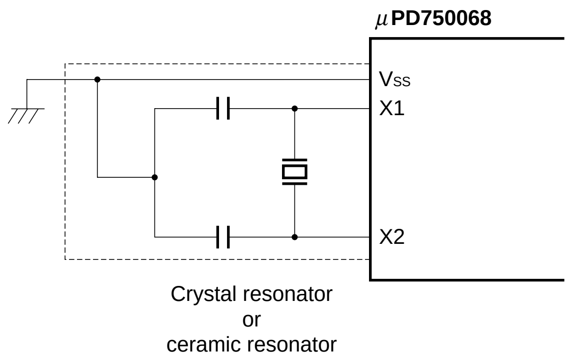
★ **3. When SCC.3 is set to “1”, the X2 pin is internally pulled up to V_{DD} with a resistor of 50 kΩ (TYP.).**

(3) System clock oscillation circuit

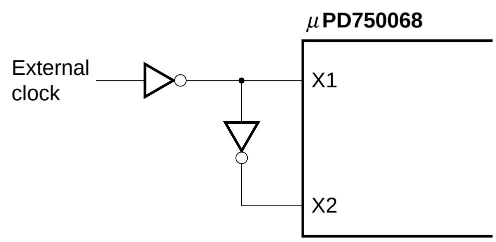
(i) The main system clock oscillation circuit is resonator by the crystal or ceramic resonator connected across the X1 and X2 pins (4.194304 MHz TYP.).
An external clock can also be input.

Fig. 5-14 External Circuit of Main System Clock Oscillation Circuit

(a) Crystal or ceramic oscillation



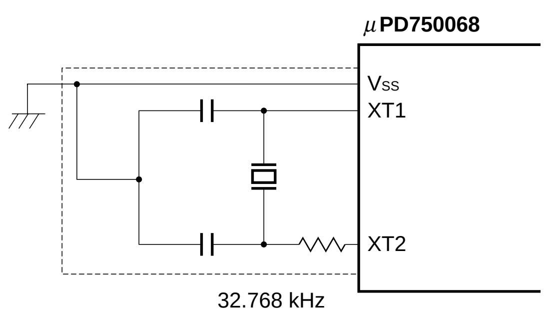
(b) External clock



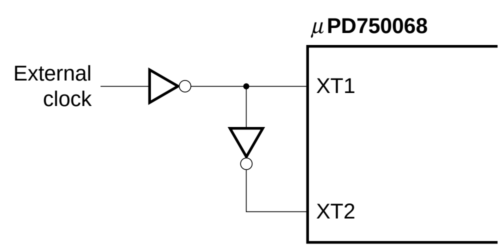
(ii) The subsystem clock oscillation circuit is oscillated by the crystal resonator (32.768 kHz TYP.) connected across the XT1 and XT2 pins.
An external clock can also be input.

Fig. 5-15 External Circuit of Subsystem Clock Oscillation Circuit

(a) Crystal oscillation



(b) External clock ★



★ **Cautions 1. When the STOP mode is set, the X2 pin is internally pulled up to V_{DD} with a resistor of 50 kΩ (TYP.).**

2. Wire the portion enclosed by the dotted line in Figs. 5-14 and 5-15 as follows to prevent adverse influence by wiring capacitance when using the main system clock and subsystem clock oscillation circuits.

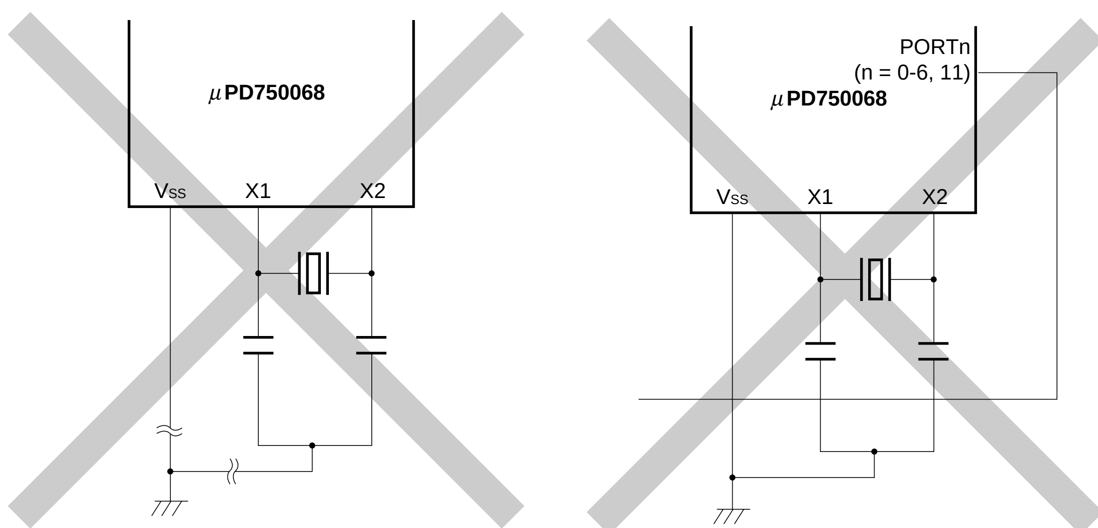
- **Keep the wiring length as short as possible.**
- **Do not cross the wiring with any other signal lines.**
- **Do not route the wiring in the vicinity of any line through which a high alternating current is flowing.**
- **Always keep the potential at the connecting point of the capacitor of the oscillation circuit at the same level as V_{SS}. Do not connect the wiring to a ground pattern through which a high current is flowing.**
- **Do not extract signals from the oscillation circuit.**

The amplification factor of the subsystem clock oscillation circuit is kept low to reduce the power dissipation. Therefore, this is more susceptible to noise than the main system clock oscillation circuit. To use the subsystem clock oscillation circuit, therefore, you should exercise care with the wiring.

Fig. 5-16 shows incorrect examples of connecting the resonator.

Fig. 5-16 Incorrect Example of Connecting Resonator (1/2)

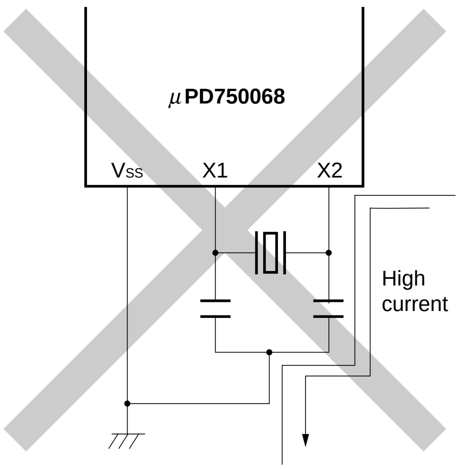
(a) Wiring length too long **(b) Crossed signal line**



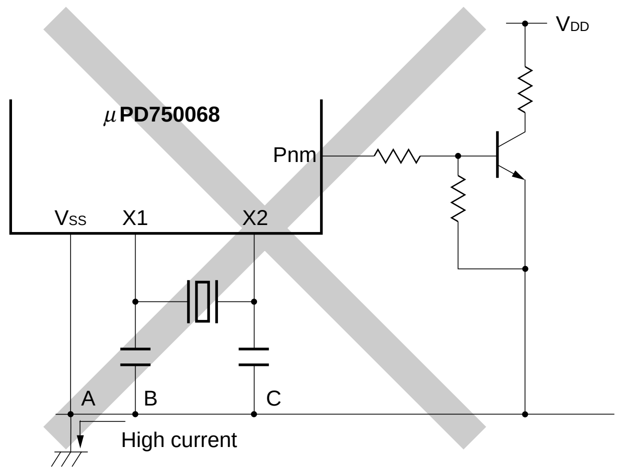
Remark When using the subsystem clock, take X1 and X2 in the above figures as XT1 and XT2. Also, connect a resistor in series with XT2.

Fig. 5-16 Incorrect Example of Connecting Resonator (2/2)

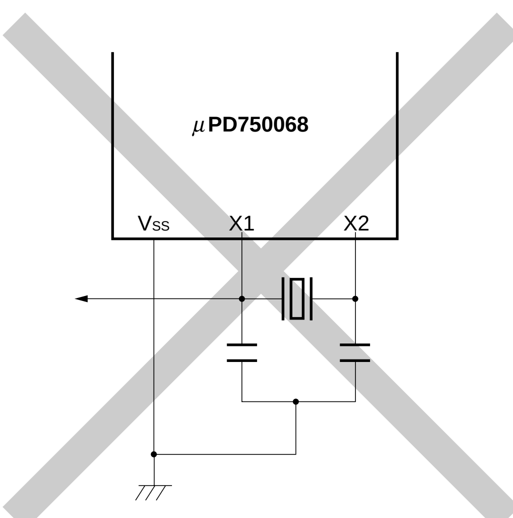
(c) High alternating current close to signal line



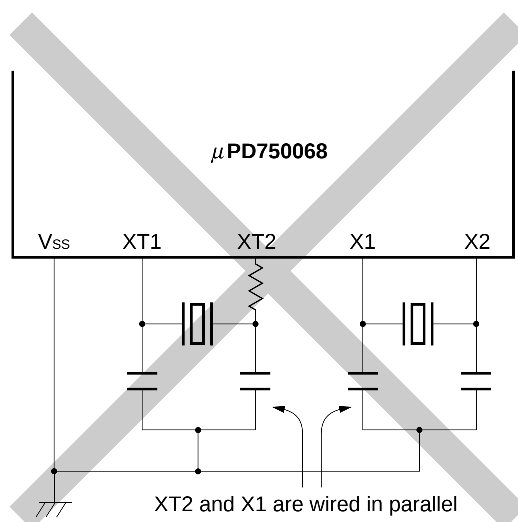
(d) Current flowing through ground line of oscillation circuit (potential at points A, B, and C changes)



(e) Signal extracted

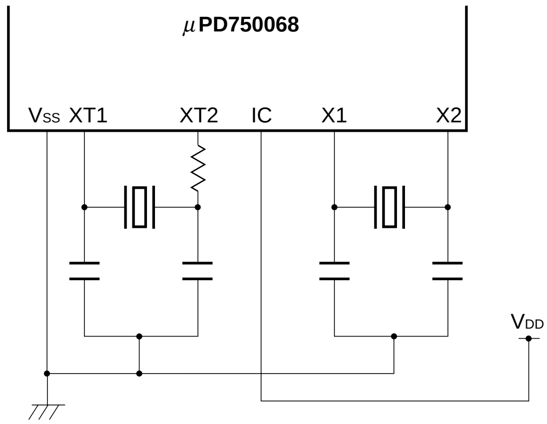


(f) Main system clock and subsystem clock signal lines close and in parallel with each other



Remark When using the subsystem clock, assume X1 and X2 in the above figures as XT1 and XT2. Also, connect a resistor in series with XT2.

Caution 2. In Fig. 5-16(f), XT2 and X1 are wired in parallel. In consequence, the cross-talk noise of X1 may be superimposed on XT2, causing malfunctioning.
To prevent this, connect the IC pin in between the XT2 and X1 pins to V_{DD}.



(4) Divider circuit

The divider circuit divides the output of the main system clock oscillation circuit (f_X) to generate various clocks.

(5) Control function of subsystem clock oscillation circuit

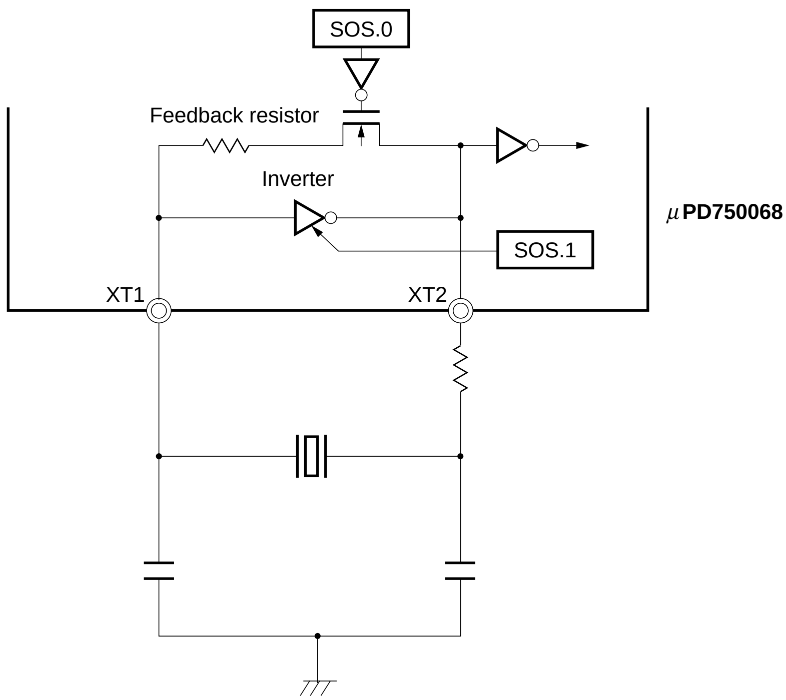
The subsystem clock oscillation circuit of the μPD750068 has the following two control functions:

- Function to select whether the internal feedback resistor is used or not, via software**Note**
- Function to reduce drive current of the internal inverter, and keep consumption current low when operating supply voltage is high ($V_{DD} \geq 2.7$ V)

★ **Note** When not using the subsystem clock, connect XT1 to V_{SS} by setting SOS.0 = 1 (internal feedback resistor not used), and by leaving XT2 open, power supply voltage can be reduced when a STOP instruction is executed.

These functions can be used by setting or resetting the bits 0 and 1 of the suboscillation circuit control register (SOS) (refer to **Fig. 5-17**).

★ **Fig. 5-17 Subsystem Clock Oscillation Circuit**



(6) Suboscillation circuit control register (SOS)

The SOS register selects whether the internal feedback resistor is used or not, and controls drive current of the internal inverter (refer to **Fig. 5-18**).

When the $\overline{RESET}$ signal is asserted, all the bits of this register are cleared to 0. The function of a flag of the SOS register is described below.

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

With the μPD750068, it can be selected via software by changing the status of SOS.0 whether the internal feedback resistor is used.

If SOS.0 is set to “1” when the resonator is not used, the feedback circuit is turned off. Therefore, current consumption can be reduced. When using the resonator, be sure to reset SOS.0 to “0” (to turn on the feedback circuit).

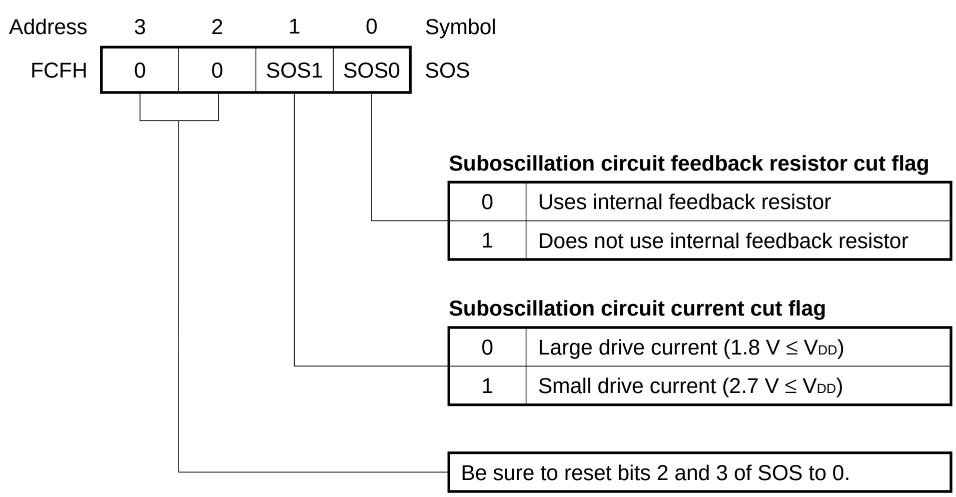
(b) SOS.1 (drive capability select flag)

The drive current of the internal inverter of the μPD750068’s subsystem clock oscillation circuit is high so that the device can be driven on a low voltage (V_{DD} = 1.8 V). Unless the drive current is lowered, the supply current increases if a high supply voltage ($V_{DD} \geq$ 2.7 V) is supplied. The SOS.1 flag is used to lower the drive current of the inverter and thereby to decrease the supply current. To do this, set SOS.1 to 1.

If the flag is set to 1 at V_{DD} of less than 2.7 V, however, oscillation may be stopped because the drive current runs short.

Therefore, be sure to clear the flag to 0 where V_{DD} is less than 2.7 V.

Fig. 5-18 Format of Suboscillation Circuit Control Register (SOS)



Remark When the subsystem clock is not necessary, set the XT1 and XT2 pins and SOS register as follows:

XT1 : Connect to V_{SS} or V_{DD}
XT2 : Open
★ SOS : 00×1B (×: don’t care)

5.2.3 Setting system clock and CPU clock

(1) Time required to select system clock and CPU clock

The system clock and CPU clock can be selected by using the least significant bit of SCC and the lower 2 bits of PCC. The processor does not operate with the selected clock, however, immediately after data has been written to the registers, for the duration of specific machine cycles. To stop oscillation of the main system clock, therefore, execute the STOP instruction or set the bit 3 of SCC after a specific time has elapsed.

★ **Table 5-5 Maximum Time Required to Select System Clock and CPU Clock**

Set Value before Selection			Set Value after Selection				
SCC0	PCC1	PCC0	SCC0 PCC1 PCC0 0 0 0	SCC0 PCC1 PCC0 0 0 1	SCC0 PCC1 PCC0 0 1 0	SCC0 PCC1 PCC0 0 1 1	SCC0 PCC1 PCC0 1 × ×
0	0	0		1 machine cycle	1 machine cycle	1 machine cycle	$\frac{f_X}{64f_{XT}}$ machine cycle (3 machine cycles)
	0	1	4 machine cycles		4 machine cycles	4 machine cycles	$\frac{f_X}{16f_{XT}}$ machine cycle[Note] (12 machine cycles)
	1	0	8 machine cycles	8 machine cycles		8 machine cycles	$\frac{f_X}{8f_{XT}}$ machine cycle (23 machine cycles)
	1	1	16 machine cycles	16 machine cycles	16 machine cycles		$\frac{f_X}{4f_{XT}}$ machine cycle (46 machine cycles)
1	×	×	1 machine cycle	1 machine cycle[Note]	1 machine cycle	1 machine cycle	

★ **Note** Emulation cannot be performed by tools.

Caution The values of f_X and f_{XT} change depending on the ambient temperature of the resonator and variations in the performance of load capacitance. Especially, if f_X is higher than the nominal value, or f_{XT} is lower than the nominal value, the number of machine cycles calculated by ★ expressions $f_X/64f_{XT}$, $f_X/16f_{XT}$, $f_X/8f_{XT}$, and $f_X/4f_{XT}$ in the above table will be greater than the number of machine cycles calculated with the nominal values of f_X and f_{XT}. To set the wait time necessary for selecting the CPU clock, therefore, use the number of machine cycles greater than that calculated with the nominal values of f_X and f_{XT}.

Remarks 1. ( ): f_X = 6.0 MHz, f_{XT} = 32.768 kHz

2. x: don't care

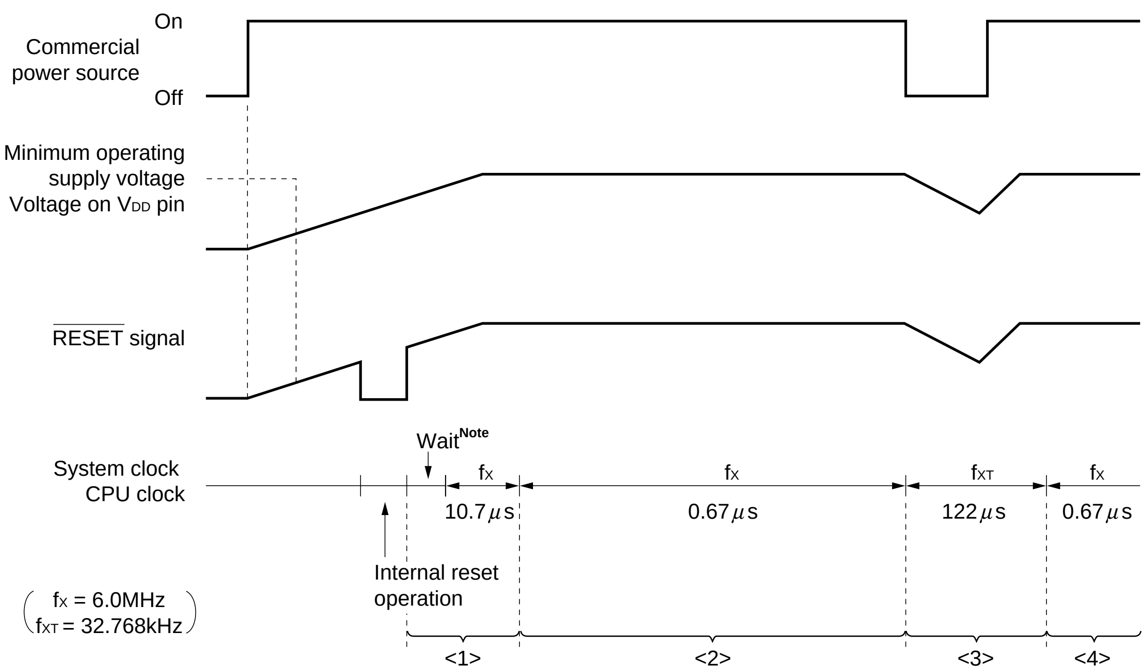
3. The CPU clock Φ is supplied to the internal CPU and its inverse number (defined to be 1 machine cycle in this manual) is the minimum instruction execution time.

(2) Procedure to select system clock and CPU clock

Fig. 5-19 illustrates the procedure to select the system clock and CPU clock.

★

Fig. 5-19 Selecting System Clock and CPU Clock



<1> When the $\overline{\text{RESET}}$ signal is asserted, wait time **Note** during which oscillation is stabilized elapses. The CPU then starts operating at the slowest speed of the system clock (10.7 μs at 6.0 MHz, 15.3 μs at 4.19 MHz).

<2> After the time during which the voltage on the V_{DD} pin rises to the sufficient level at which the CPU can operate at the highest speed has elapsed, the contents of PCC are rewritten, and the CPU operates at the highest speed.

<3> When the commercial power source is turned off, it is detected by an interrupt (use of INT4 is effective). Bit 0 of SCC is set to “1”, and the CPU operates with the subsystem clock (at this time, oscillation of the subsystem clock must be started in advance). After the time required to change the system clock from the main to sub (46 machine cycles) has elapsed, set bit 3 of SCC to “1” to stop oscillation of the main system clock.

<4> When the commercial power source is turned back on again, it is detected by an interrupt. Clear bit 3 of SCC to “0” to start oscillation of the main system clock. After the time necessary for the oscillation to become stabilized has elapsed,clear bit 0 of SCC to “0”. This means that the CPU can operate at the highest speed.

Note The wait time can be selected from the following two by mask option.

$2^{17}/f_X$ (21.8 ms: at 6.0 MHz, 31.3 ms : at 4.19 MHz)

$2^{15}/f_X$ (5.46 ms: at 6.0 MHz, 7.81 ms: at 4.19 MHz)

With the μPD75P0076, however, the wait time is fixed to $2^{15}/f_X$ because no mask option is available.

5.2.4 Clock output circuit

(1) Configuration of clock output circuit

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

(2) Function of clock output circuit

The clock output circuit outputs a clock pulse from the P22/PCL pin and is used to apply to wave form output of a remote control output or to supply a clock pulse to a peripheral LSI.

The clock pulse is output in the following procedure:

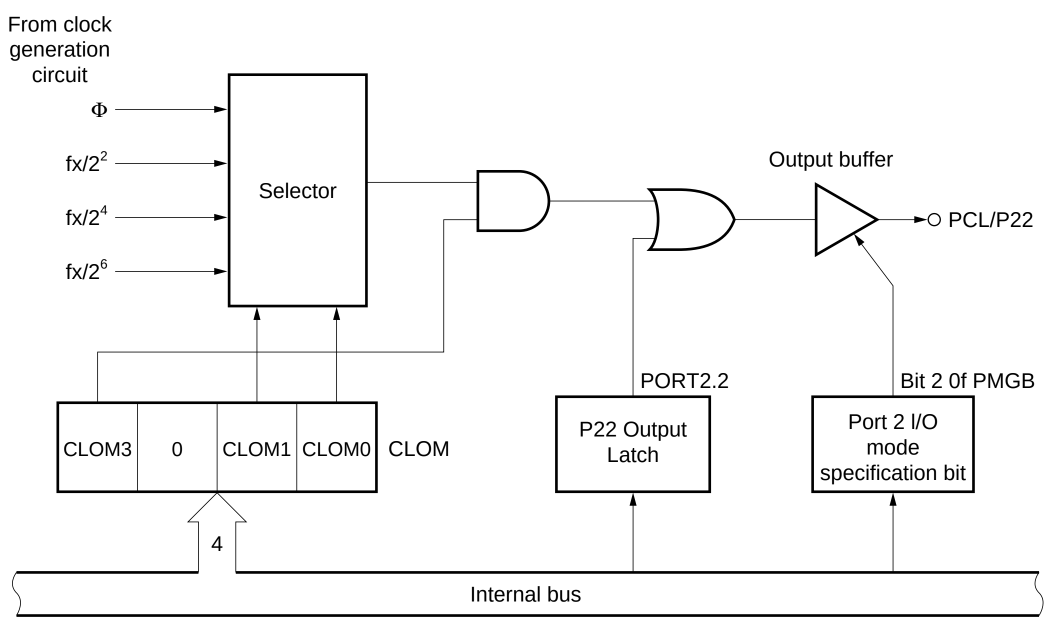
(a) Select the clock output frequency. Disable clock output.

(b) Write 0 to the output latch of P22.

(c) Set port 2 in the output mode.

(d) Enable clock output.

Fig. 5-20 Block Diagram of Clock Output Circuit



Remark The circuit has been designed so that a pulse with short width is not output when clock output is enabled or disabled.

(3) Clock output mode register (CLOM)

CLOM is a 4-bit register that controls clock output.

This register is set by a 4-bit memory manipulation instruction.

Example To output CPU clock Φ from PCL/P22 pin

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

When the $\overline{\text{RESET}}$ signal is asserted, CLOM is cleared to “0”, and clock output is disabled.

Fig. 5-21 Format of Clock Output Mode Register

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

Clock output frequency select bit

fx = 6.00 MHz

CLOM1	CLOM0	
0	0	Φ output[Note] (1.5 MHz, 750 kHz, 375 kHz, 93.8 kHz)
0	1	$f_X/2^2$ output (1.5 MHz)
1	0	$f_X/2^4$ output (375 kHz)
1	1	$f_X/2^6$ output (93.8 kHz)

fx = 4.19 MHz

CLOM1	CLOM0	
0	0	Φ output[Note] (1.05 MHz, 524 kHz, 262 kHz, 65.5 kHz)
0	1	$f_X/2^2$ output (1.05 MHz)
1	0	$f_X/2^4$ output (262 kHz)
1	1	$f_X/2^6$ output (65.5 kHz)

Note Φ is the CPU clock selected by PCC.

Clock output enable/disable bit

CLOM3	
0	Disables output
1	Enables output

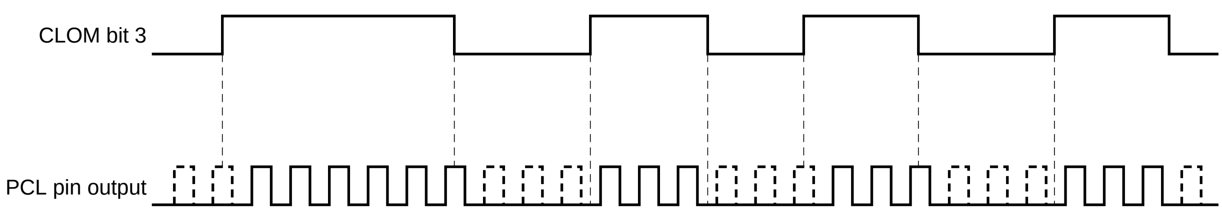
Caution Be sure to set bit 2 of CLOM to 0.

(4) Application example of remote controller waveform output

The clock output function of the μPD750068 can be used for remote controller waveform output. The carrier frequency of the remote controller waveform output is selected by the clock frequency select bit of the clock output mode register. Output of the pulse is enabled or disabled by controlling the clock output enable/disable bit via software.

The circuit has been designed so that a pulse with a narrow width is not output when clock output is enabled or disabled.

Fig. 5-22 Application Example of Remote Controller Waveform Output



5.3 Basic Interval Timer/Watchdog Timer

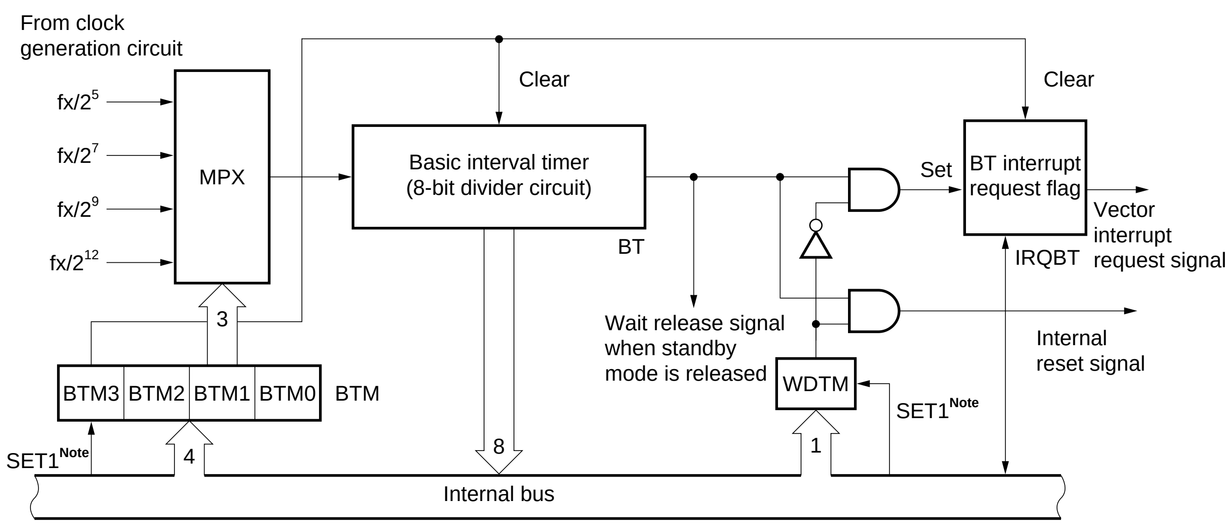
The μPD750068 has an 8-bit basic interval timer/watchdog timer that has the following functions:

(a) Interval timer operation to generate reference time interrupt
(b) Watchdog timer operation to detect program hang-up and reset CPU
(c) To select and count wait time when standby mode is released
(d) To read count value

5.3.1 Configuration of basic interval timer/watchdog timer

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

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



Note Instruction execution

5.3.2 Basic interval timer mode register (BTM)

BTM is a 4-bit register that controls the operation of the basic interval timer (BT).

This register is set by a 4-bit memory manipulation instruction.

Bit 3 of BT can be manipulated by a bit manipulation instruction.

Example To set interrupt generation interval to 1.37 ms (6.00 MHz)

```
SEL     MB15        ; or CLR1 MBE
CLR1    WDTM
MOV     A, #1111B
MOV     BTM,A       ; BTM ← 1111B
```

When bit 3 of this register is set to “1”, the contents of BT are cleared, and at the same time, the basic interval timer/watchdog timer interrupt request flag (IRQBT) is cleared (the basic interval timer/watchdog timer is started).

When the $\overline{\text{RESET}}$ signal is asserted, the contents of this register are cleared to “0”, and the generation interval time of the interrupt request signal is set to the longest value.

Fig. 5-24 Format of Basic Interval Timer Mode Register

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

f_X = 6.00 MHz

BTM2	BTM1	BTM0	Specifies input clock	Interrupt interval time (wait time when standby mode is released)
0	0	0	$f_X/2^{12}$ (1.46 kHz)	$2^{20}/f_X$ (175 ms)
0	1	1	$f_X/2^9$ (11.7 kHz)	$2^{17}/f_X$ (21.8 ms)
1	0	1	$f_X/2^7$ (46.9 kHz)	$2^{15}/f_X$ (5.46 ms)
1	1	1	$f_X/2^5$ (188 kHz)	$2^{13}/f_X$ (1.37 ms)
Others			Setting prohibited	–

f_X = 4.19 MHz

BTM2	BTM1	BTM0	Specifies input clock	Interrupt interval time (wait time when standby mode is released)
0	0	0	$f_X/2^{12}$ (1.02 kHz)	$2^{20}/f_X$ (250 ms)
0	1	1	$f_X/2^9$ (8.19 kHz)	$2^{17}/f_X$ (31.3 ms)
1	0	1	$f_X/2^7$ (32.768 kHz)	$2^{15}/f_X$ (7.81 ms)
1	1	1	$f_X/2^5$ (131 kHz)	$2^{13}/f_X$ (1.95 ms)
Others			Setting prohibited	–

Basic interval timer/watchdog timer start control bit (BTM3)

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

5.3.3 Watchdog timer enable flag (WDTM)

WDTM is a flag that enables assertion of the reset signal when a overflow occurs.
This flag is set by a bit manipulation instruction. Once this flag has been set, it cannot be cleared by an instruction.

Example To set watchdog timer function

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

The content of this flag is cleared to 0 when the $\overline{\text{RESET}}$ signal is asserted.

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

Address
F8BH.3 WDTM

0	BT mode Sets IRQBT when basic interval timer (BT) overflows
1	WT mode Asserts internal reset signal when basic interval timer (BT) overflows

5.3.4 Operation as basic interval timer

When WDTM is reset to “0”, the interrupt request flag (IRQBT) is set by the overflow of the basic interval timer (BT), and the basic interval timer/watchdog timer operates as the basic interval timer. BT is always incremented by the clock supplied by the clock generation circuit and its counting operation cannot be stopped.

Four time intervals at which the interrupt occurs can be selected by BTM (refer to **Fig. 5-24**).

By setting bit 3 of BTM to “1”, BT and IRQBT can be cleared (command to start the interval timer).

The count value of BT can be read by using an 8-bit manipulation instruction. No data can be written to BT.

Start the timer operation as follows (<1> and <2> may be performed simultaneously):

<1> Set interval time to BTM.
<2> Set bit 3 of BTM to “1”.

Example To generate interrupt at intervals of 1.37 ms (at 6.00 MHz)

```
SET1   MBE
SEL    MB15
MOV    A, #1111B
MOV    BTM, A       ; Sets time and starts
EI                  ; Enables interrupt
EI     IEBT         ; Enables BT interrupt
```

5.3.5 Operation as watchdog timer

The basic interval timer/watchdog timer operates as a watchdog timer that asserts the internal reset signal when an overflow occurs in the basic interval timer (BT), if WDTM is set to “1”. However, if the overflow occurs during the oscillation wait time that elapses after the STOP instruction has been released, the reset signal is not asserted. (Once WDTM has been set to “1”, it cannot be cleared by any means other than reset.) BT is always incremented by the clock supplied from the clock generation circuit, and its count operation cannot be stopped.

In the watchdog timer mode, a program hang-up is detected by using the interval time at which BT overflows. As this interval time, four values can be selected by using bits 2 through 0 of BTM (refer to **Fig. 5-24**). Select the interval time best-suited to detecting any hang-up that may occur in you system. Set an interval time, divide the program into several modules that can be executed within the set interval time, and execute an instruction that clears BT at the end of each module. If this instruction that clears BT is not executed within the set interval time (in other words, if a module of the program is not normally executed, i.e., if a hang up occurs), BT overflows, the internal reset signal is asserted, and the program is terminated forcibly. Consequently, asserting of the internal reset signal indicates occurrence and detection of a program hang-up.

Set the watchdog timer as follows (<1> and <2> may be performed simultaneously):

<1> Set interval time to BTM.
<2> Set bit 3 of BTM to “1”.
<3> Set WDTM to “1”.
(<1> through <3>: Initial setting)
<4> After setting <1> through <3> above, set bit 3 of BTM to “1” within the interval time.

Example To use the basic interval timer/watchdog timer as a watchdog timer with a time interval of 5.46 ms (at 6.00 MHz).

Divide the program into several modules, each of which is completed within the set time of BTM (5.46 ms), and clear BT at the end of each module. If a hang-up occurs, BT is not cleared within the set time. As a result, BT overflows, and the internal reset signal is asserted.

Initial setting:

```
SET1    MBE
SEL     MB15
MOV     A, #1101B
MOV     BTM, A      ; Sets time and starts
SET1    WDTM        ; Enables watchdog timer
  :
```

(After that, set bit 3 of BTM to “1” every 5.46 ms.)

Module 1:

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

Processing completed within 5.46 ms

Module 2:

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

Processing completed within 5.46 ms

:

5.3.6 Other functions

The basic interval timer/watchdog timer has the following functions, regardless of the operations as the basic interval timer or watchdog timer:

<1> Selects and counts wait time after standby mode has been released
<2> Reads count value

(1) Selecting and counting wait time after STOP mode has been released

When the STOP mode has been released, a wait time elapses during which the operation of the CPU is stopped until the basic interval timer (BT) overflows, so that oscillation of the system clock becomes stabilized.
The wait time that elapses after the $\overline{\text{RESET}}$ signal has been asserted is fixed by the mask option. When the STOP mode is released by an interrupt, however, the wait time can be selected by BTM. The wait time in this case is the same as the interval time shown in Fig. 5-24. Set BTM before setting the STOP mode (for details, refer to **CHAPTER 7 STANDBY FUNCTION**).

Example To set a wait time of 5.46 ms that elapses when the STOP mode has been released by an interrupt (at 6.00 MHz)

```
SET1   MBE
SEL    MB15
MOV    A, #1101B
MOV    BTM, A       ; Sets time
STOP                ; Sets STOP mode
NOP
```

(2) Reading count value

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

Caution To read the count value of BT, execute the read instruction two times to prevent undefined data from being read while the count value is updated. Compare the two read values. If the values are similar, take the latter value as the result. If the two values are completely different, redo from the beginning.

Examples 1. To read count value of BT

```
        SET1    MBE
        SEL     MB15
        MOV     HL, #BT     ; Sets address of BT to HL
LOOP:   MOV     XA, @HL     ; Reads first time
        MOV     BC, XA
        MOV     XA, @HL     ; Reads second time
        SKE     XA, BC
        BR      LOOP
```

2. To set a high-level width of a pulse input to INT4 interrupt (detected at both the edges) (the pulse width must not exceed the set value of BT, and the set value of BTM is 5.46 ms or longer (at 6.00 MHz))

<INT4 interrupt routine (MBE = 0)>

```
LOOP:   MOV     XA, BT          ;  Reads first time
        MOV     BC, XA          ;  Stores data
        MOV     XA, BT          ;  Reads second time
        SKE     A, C
        BR      LOOP
        MOV     A, X
        SKE     A, B
        BR      LOOP
        SKT     PORT0.0         ;  P00 = 1?
        BR      AA              ;  NO
        MOV     XA, BC          ;  Stores data to data memory
        MOV     BUFF, XA
        CLR1    FLAG            ;  Data found. Clears 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        ;  Stores data
        SET1    FLAG            ;  Data found. Sets flag
        RETI
```

5.4 Watch Timer

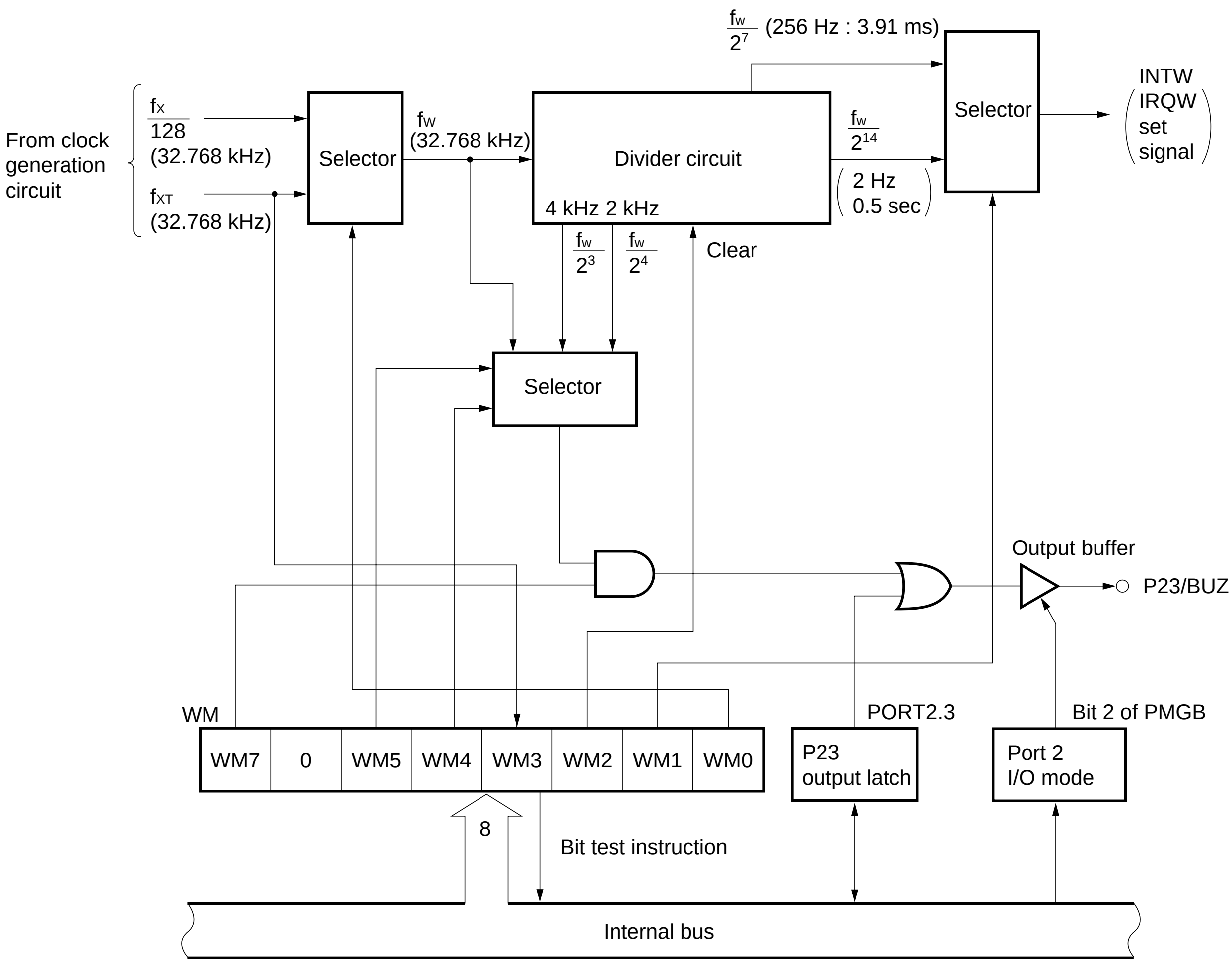
The μPD750068 is provided with one channel of watch timer. This watch timer has the following functions:

(a) Sets a test flag (IRQW) at time intervals of 0.5 second.
IRQW can be used to release the standby mode.
(b) Can generate the time intervals of 0.5 second from both the main system clock (4.19430 MHz) and subsystem clock (32.768 kHz).
(c) Can increase the time interval 128-fold (3.91 ms) in the fast forward mode. This is useful for debugging and testing the program.
(d) Can output any frequency (2.048, 4.096, or 32.768 kHz) to the P23/BUZ pin to active a buzzer or trim the system clock oscillation frequency.
(e) Can start the watch from zero second by clearing the divider circuit.
(f) Use of a 0.5-second clock as the clock source for the timer/event counter can allow the standby mode to continue up to approximately 9 hours (when using timer 0 or 1) and the ultra-low power consumption mode can be set.

5.4.1 Configuration of watch timer

Fig. 5-26 shows the configuration of the watch timer.

Fig. 5-26 Block Diagram of Watch Timer



( ) : f_X = 4.194304 MHz, f_{XT} = 32.768 kHz

5.4.2 Watch mode register

The watch mode register (WM) is an 8-bit register that controls the watch timer. Fig. 5-27 shows the format of this register.

All the bits of WM, except bit 3, are set by an 8-bit manipulation instruction. Bit 3 is used to test the input level of the XT1 pin. No data can be written to this bit.

All the bits, except bit 3, are cleared to “0” when the $\overline{\text{RESET}}$ signal is asserted.

Example To generate time interval from the main system clock (4.19 MHz) with the buzzer output enabled

```
CLR1    MBE
MOV     XA, #84H
MOV     WM, XA      ; Sets WM
```

Fig. 5-27 Format of Watch Mode Register

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 select bit

WM5	WM4	BUZ output frequency
0	0	$\frac{f_W}{2^4}$ (2.048 kHz)
0	1	$\frac{f_W}{2^3}$ (4.096 kHz)
1	0	Setting prohibited
1	1	f_W (32.768 kHz)

Input level of XT1 pin (bit test only can be tested)

WM3	0	Input to XT1 pin is low
	1	Input to XT1 pinis high

Watch operation enable/disable bit

WM2	0	Stops watch operation (clears divider circuit)
	1	Enables watch operation

Operation mode select bit

WM1	0	Normal watch mode ($f_W/2^{14}$: Sets IRQW at 0.5-second intervals)
	1	Fast forward watch mode ($f_W/2^7$: Sets IRQW at 3.91-ms intervals)

Count clock (f_W) select bit

WM0	0	Selects system clock division output: $f_X/128$
	1	Selects subsystem clock: f_{XT}

Remark ( ): f_W = 32.768 kHz

5.5 Timer/Event Counter

The μPD750068 is provided with two channels of timers/event counters. The timers/event counters have the following functions:

(a) Programmable interval timer operation
(b) Outputs square wave of any frequency to PTOn pin
(c) Event counter operation
(d) Divides TIn pin input by N and outputs to PTOn pin (divider circuit operation)
(e) Supplies serial shift clock to interface circuit
(f) Count value read function

Remark n = 0, 1

The timers/event counters can operate in the following two modes selected by the corresponding mode registers.

Table 5-6 Operation Modes

Mode \ Channel	Channel 0	Channel 1
8-bit timer/event counter mode	○	○
16-bit timer/event counter mode	○	

5.5.1 Configuration of timer/event counter

Figs. 5-28 and 5-29 show the configuration of the timers/event counters.

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

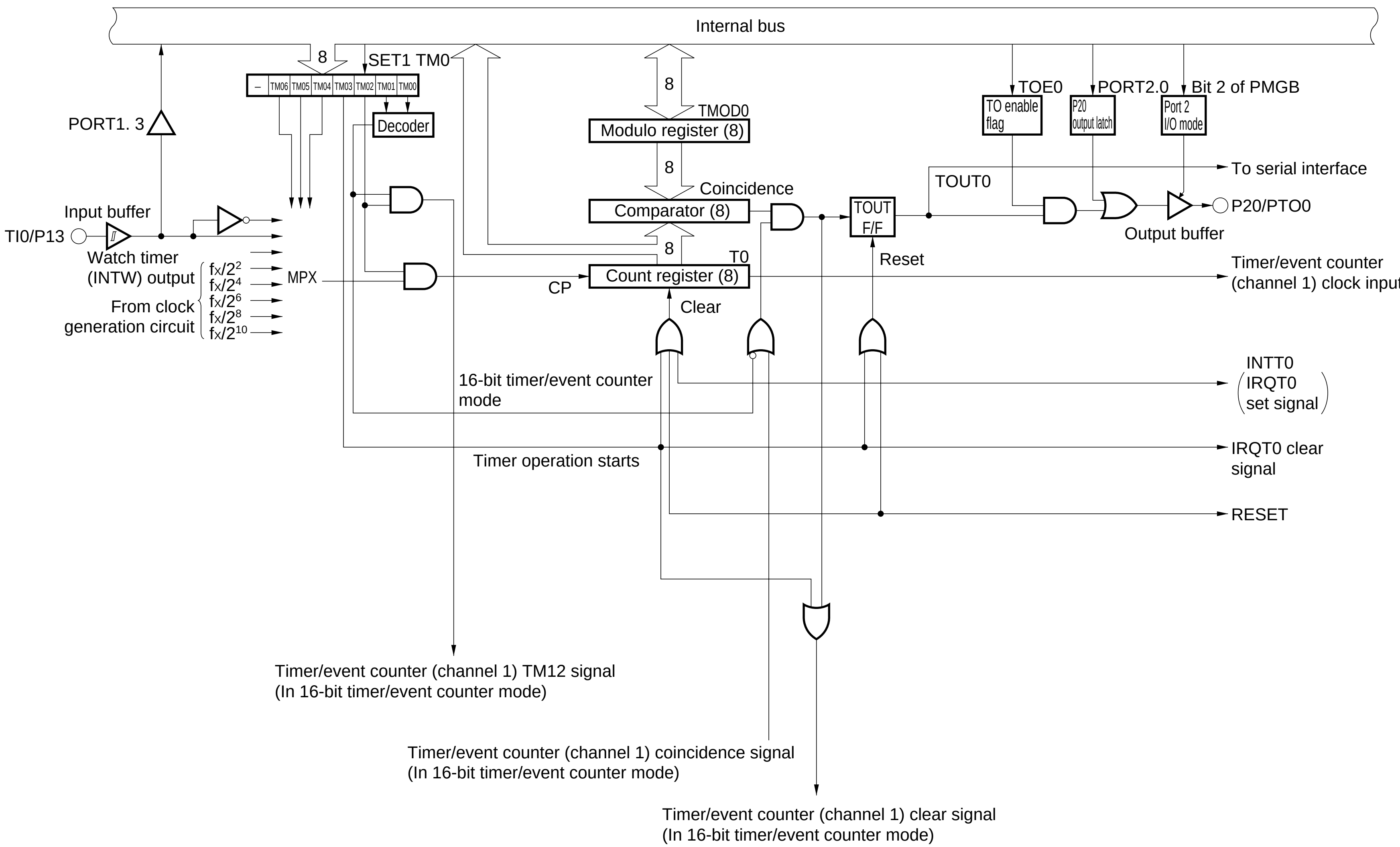
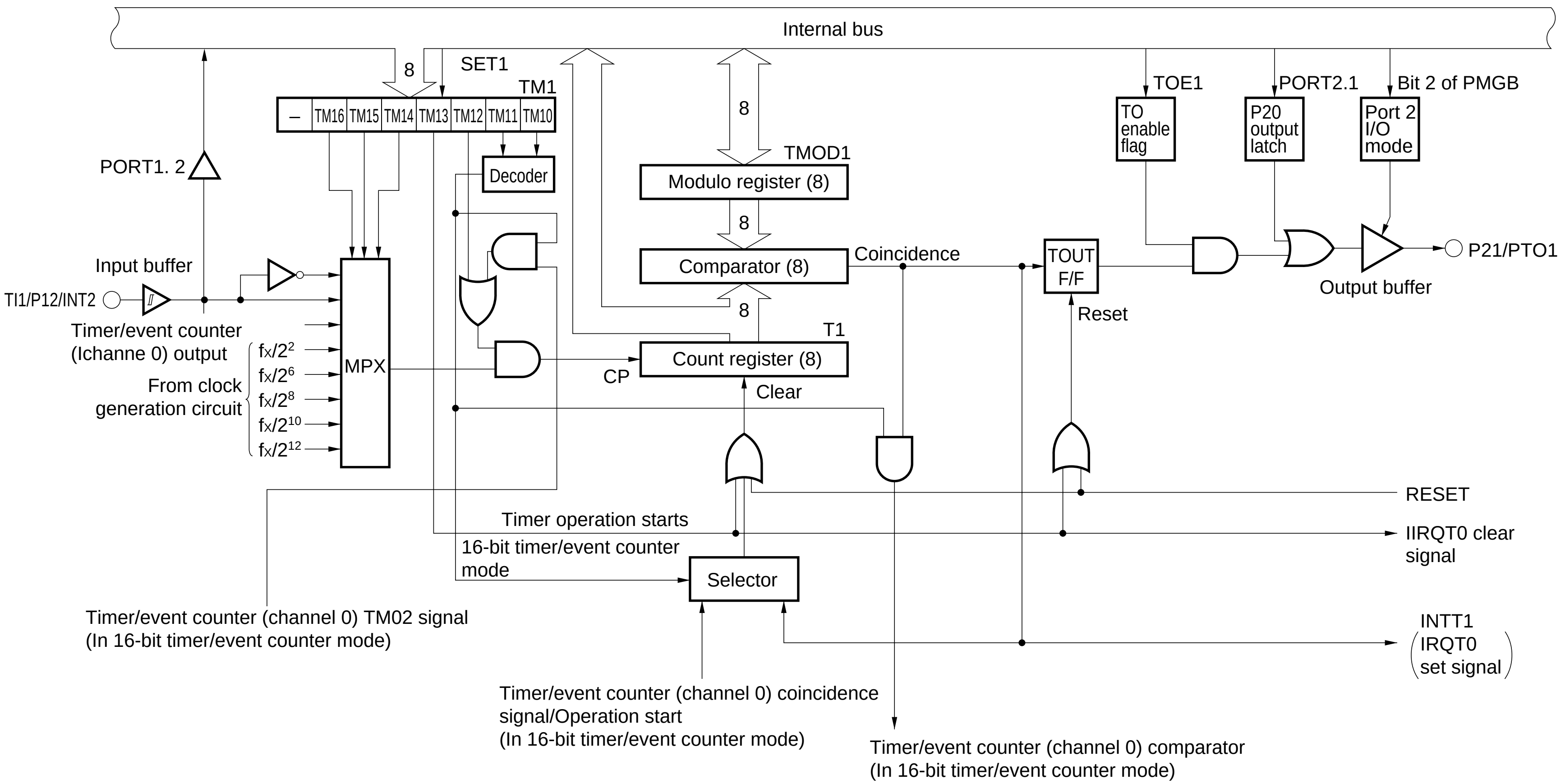


Fig. 5-29 Block Diagram of Timer/Event Counter (Channel 1)



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

Timer/event counter mode registers (TM0, TM1) are 8-bit registers that control the corresponding timer/event counter. Figs. 5-30 and 5-31 show the formats of the various mode registers.

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

Bit 3 of this register is a timer start bit and can be manipulated in 1-bit units independently of the other bits. This bit is automatically reset to "0" when the timer starts operating.

All the bits of the timer/event counter mode register are cleared to "0" when the $\overline{\text{RESET}}$ signal is asserted.

Examples 1. To start timer 0 in interval timer mode of CP = 5.86 kHz (at 6.00 MHz)

```
SEL     MB15            ; or CLR1 MBE
MOV     XA, #01001100B
MOV     TM0, XA         ; TM0 ← 4CH
```

2. To restart timer according to setting of timer/event counter mode register

```
SEL     MB15            ; or CLR1 MBE
SET1    TMn.3           ; TMn.bit3 ← 1 (n = 0, 1)
```

Fig. 5-30 Format of Timer/Event Counter Mode Register (Channel 0) (1/2)

Address	7	6	5	4	3	2	1	0	Symbol
FA0H	–	TM06	TM05	TM04	TM03	TM02	TM01	TM00	TM0

Count pulse (CP) select bit

f_X = 6.00 MHz

TM06	TM05	TM04	Count pulse (CP)
0	0	0	Rising edge of TI0
0	0	1	Falling edge of TI0
0	1	0	$f_W/2^{14}$ or $f_W/2^7$ (watch timer (INTW) output)Note
0	1	1	$f_X/2^2$ (1.5 MHz)
1	0	0	$f_X/2^{10}$ (5.86 kHz)
1	0	1	$f_X/2^8$ (23.4 kHz)
1	1	0	$f_X/2^6$ (93.8 kHz)
1	1	1	$f_X/2^4$ (375 kHz)

Note $f_W = f_{XT}$ or $f_X/2^7$

f_X = 4.19 MHz

TM06	TM05	TM04	Count pulse (CP)
0	0	0	Rising edge of TI0
0	0	1	Falling edge of TI0
0	1	0	$f_W/2^{14}$ or $f_W/2^7$ (watch timer (INTW) output)Note
0	1	1	$f_X/2^2$ (1.05 MHz)
1	0	0	$f_X/2^{10}$ (4.10 kHz)
1	0	1	$f_X/2^8$ (16.4 kHz)
1	1	0	$f_X/2^6$ (65.5 kHz)
1	1	1	$f_X/2^4$ (262 kHz)

Note $f_W = f_{XT}$ or $f_X/2^7$

Timer start command bit

TM03	Clears counter and IRQT0 flag when "1" is written. Starts count operation if bit 2 is set to "1".

Operation mode

TM02	Count operation
0	Stops (count value retained)
1	Count operation

Fig. 5-30 Format of Timer/Event Counter Mode Register (Channel 0) (2/2)

Operation mode select bit

TM01	TM00	Mode
0	0	8-bit timer/event counter mode
1	0	16-bit timer/event counter mode
Others		Setting prohibited

Fig. 5-31 Format of Timer/Event Counter Mode Register (Channel 1) (1/2)

Address	7	6	5	4	3	2	1	0	Symbol
FA8H	–	TM16	TM15	TM14	TM13	TM12	TM11	TM10	TM1

Count pulse (CP) select bit

f_X = 6.00 MHz

TM16	TM15	TM14	Count pulse (CP)
0	0	0	Rising edge of TI1
0	0	1	Falling edge of TI1
0	1	0	Overflow of timer/event counter channel 0
0	1	1	$f_X/2^2$ (1.5 MHz)
1	0	0	$f_X/2^{12}$ (1.46 kHz)
1	0	1	$f_X/2^{10}$ (5.86 kHz)
1	1	0	$f_X/2^8$ (23.4 kHz)
1	1	1	$f_X/2^6$ (93.8 kHz)

f_X = 4.19 MHz

TM16	TM15	TM14	Count pulse (CP)
0	0	0	Rising edge of TI1
0	0	1	Falling edge of TI1
0	1	0	Overflow of timer/event counter channel 0
0	1	1	$f_X/2^2$ (1.05 MHz)
1	0	0	$f_X/2^{12}$ (1.02 kHz)
1	0	1	$f_X/2^{10}$ (4.10 kHz)
1	1	0	$f_X/2^8$ (16.4 kHz)
1	1	1	$f_X/2^6$ (65.5 kHz)

Fig. 5-31 Format of Timer/Event Counter Mode Register (Channel 1) (2/2)

Timer start command bit

TM13	Clears counter and IRQT1 flag when "1" is written. Starts count operation if bit 2 is set to "1".

Operation mode

TM12	Count operation
0	Stops (count value retained)
1	Count operation

Operation mode select bit

TM11	TM10	Mode
0	0	8-bit timer/event counter mode
1	0	16-bit timer/event counter mode
Others		Setting prohibited

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

Timer/event counter output enable flags TOE0 and TOE1 enable or disable output to the PTO0 and PTO1 pins in the timer out F/F (TOUT F/F) status.

The timer out F/F is inverted by a coincidence signal from the comparator. When bit 3 of timer/event counter mode register TM0 or TM1 is set to "1", the timer out F/F is cleared to "0".

TOE0, TOE1, and timer out F/F are cleared to "0" when the $\overline{\text{RESET}}$ signal is asserted.

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

Address		
FA2H	TOE0	Channel 0
FAAH	TOE1	Channel 1

Timer/event counter output enable flag (W)

0	Disabled
1	Enabled

5.5.2 Operation in 8-bit timer/event counter mode

In this mode, a timer/event counter is used as an 8-bit timer/event counter. In this case, the timer/event counter operates as an 8-bit programmable interval timer or event counter.

(1) Register setting

In the 8-bit timer/event counter mode, the following three registers and one flag are used:

- 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)

In the 8-bit timer/event counter mode, set TMn as shown in Fig. 5-33 (for the format of TMn, refer to Figs. 5-30 and 5-31).

TMn is manipulated by an 8-bit manipulation instruction. Bit 3 is a timer start command bit which can be manipulated in 1-bit units. This bit is automatically cleared to 0 when the timer starts operating.

TMn is cleared to 00H when the internal reset signal is asserted.

Remark n = 0, 1

Fig. 5-33 Setting of Timer/Event Counter Mode Register (for 8-bit mode) (1/2)

(a) Timer/event counter (channel 0)

Address	7	6	5	4	3	2	1	0	Symbol
FA0H	–	TM06	TM05	TM04	TM03	TM02	TM01	TM00	TM0

Count pulse (CP) select bit

TM06	TM05	TM04	Count pulse (CP)
0	0	0	Rising edge of TI0
0	0	1	Falling edge of TI0
0	1	0	$f_W/2^{14}$ or $f_W/2^7$ (watch timer (INTW) output)Note
0	1	1	$f_X/2^2$
1	0	0	$f_X/2^{10}$
1	0	1	$f_X/2^8$
1	1	0	$f_X/2^6$
1	1	1	$f_X/2^4$

Note $f_W = f_{XT}$ or $f_X/2^7$

Timer start command bit

TM03	Clears counter and IRQT0 flag when "1" is written. Starts count operation if bit 2 is set to "1".

Operation mode

TM02	Count operation
0	Stops (count value retained)
1	Count operation

Operation mode select bit

TM01	TM00	Mode
0	0	8-bit timer/event counter mode

Fig. 5-33 Setting of Timer/Event Counter Mode Register (for 8-bit mode) (2/2)

(b) Timer/event counter (channel 1)

Address	7	6	5	4	3	2	1	0	Symbol
FA8H	–	TM16	TM15	TM14	TM13	TM12	TM11	TM10	TM1

Count pulse (CP) select bit

TM16	TM15	TM14	Count pulse (CP)
0	0	0	Rising edge of TI1
0	0	1	Falling edge of TI1
0	1	0	Overflow of timer/event counter channel 0
0	1	1	$f_X/2^2$
1	0	0	$f_X/2^{12}$
1	0	1	$f_X/2^{10}$
1	1	0	$f_X/2^8$
1	1	1	$f_X/2^6$

Timer start command bit

TM13	Clears counter and IRQT1 flag when "1" is written. Starts count operation if bit 2 is set to "1".

Operation mode

TM12	Count operation
0	Stops (count value retained)
1	Count operation

Operation mode select bit

TM11	TM10	Mode
0	0	8-bit timer/event counter mode

Fig. 5-34 Setting of Timer/Event Counter Output Enable Flag

Address		
FA2H	TOE0	Channel 0
FAAH	TOE1	Channel 1

Timer/event counter output enable flag (W)

0	Disabled (Outputs low level)
1	Enabled

(2) Time setting of timer/event counter

[Timer set time] (cycle) is calculated by dividing the [contents of modulo register + 1] by the [count pulse (CP) frequency] selected by the mode register.

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

where,

T (sec) : timer set time (seconds)

f_{CP} (Hz) : CP frequency (Hz)

n : contents of modulo register ($n \neq 0$)

Once the timer has been set, interrupt request flag (IRQT0, IRQT1) is set at the set time interval of the timer.

Table 5-7 shows the resolution of each count pulse of the timer/event counter and the longest set time (time when FFH is set to the modulo register).

★ **Table 5-7 Resolution and Longest Set Time (for 8-bit timer)**

(a) Timer/event counter (channel 0)

Mode Register			6.0 MHz Operation		4.19 MHz Operation	
TM06	TM05	TM04	Resolution	Longest set time	Resolution	Longest set time
0	1	0	0.35s (2.73 ms)Note	89.5s (0.7 s)Note	0.5s (3.91 ms)Note	128s (1.00 s)Note
0	1	1	677 ns	171 μs	954 ns	244 μs
1	0	0	171 μs	43.7 ms	244 μs	62.5 ms
1	0	1	42.7 μs	10.9 ms	61.0 μs	15.6 ms
1	1	0	10.7 μs	2.73 ms	15.3 μs	3.91 ms
1	1	1	2.67 μs	683 μs	3.81 μs	977 μs

Note When f_W = 32.768 kHz, WM1 = 0. ( ): WM1 = 1.

(b) Timer/event counter (channel 1)

Mode Register			6.0 MHz Operation		4.19 MHz Operation	
TM16	TM15	TM14	Resolution	Longest set time	Resolution	Longest set time
0	1	1	677 ns	171 μs	954 ns	244 μs
1	0	0	683 μs	175 ms	977 μs	250 ms
1	0	1	171 μs	43.7 ms	244 μs	62.5 ms
1	1	0	42.7 μs	10.9 ms	61.0 μs	15.6 ms
1	1	1	10.7 μs	2.73 ms	15.3 μs	3.91 ms

(3) 8-bit timer/event counter operation

The 8-bit timer/event counter operates as follows.

Fig. 5-35 shows the configuration when the timer/event counter operates.

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

<2> The contents of Tn are compared with those of the timer/event counter modulo register (TMODn). When the contents of these registers coincide, a coincidence signal is generated, and the interrupt request flag (IRQTn) is set. At the same time, the timer out flip/flop (TOUT F/F) is inverted.

Fig. 5-36 shows the timing of the 8-bit timer/event counter operation.

The 8-bit timer/event counter operation is usually started in the following procedure:

<1> Set the number of counts to TMODn.

<2> Sets the operation mode, count pulse, and start command to TMn.

Caution Set a value other than 00H to the timer/event counter modulo register (TMODn).

To use the timer/event counter output pin (PTOn), set the P2n pin as follows:

<1> Clear the output latch of P2n.

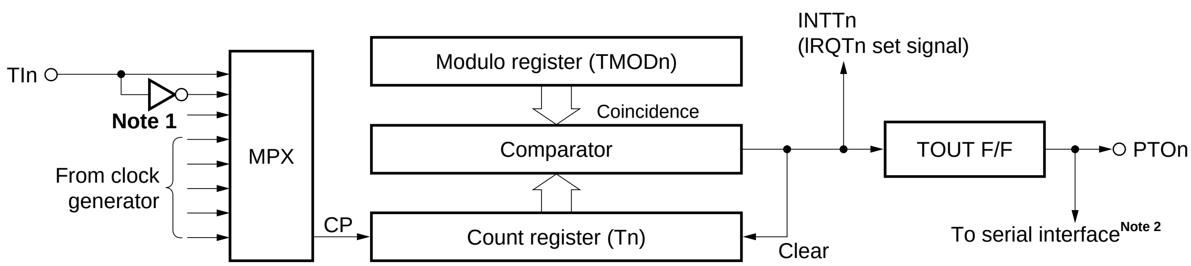
<2> Set port 2 in the output mode.

<3> Disconnect the internal pull-up resistor from port 2.

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

Remark n = 0, 1

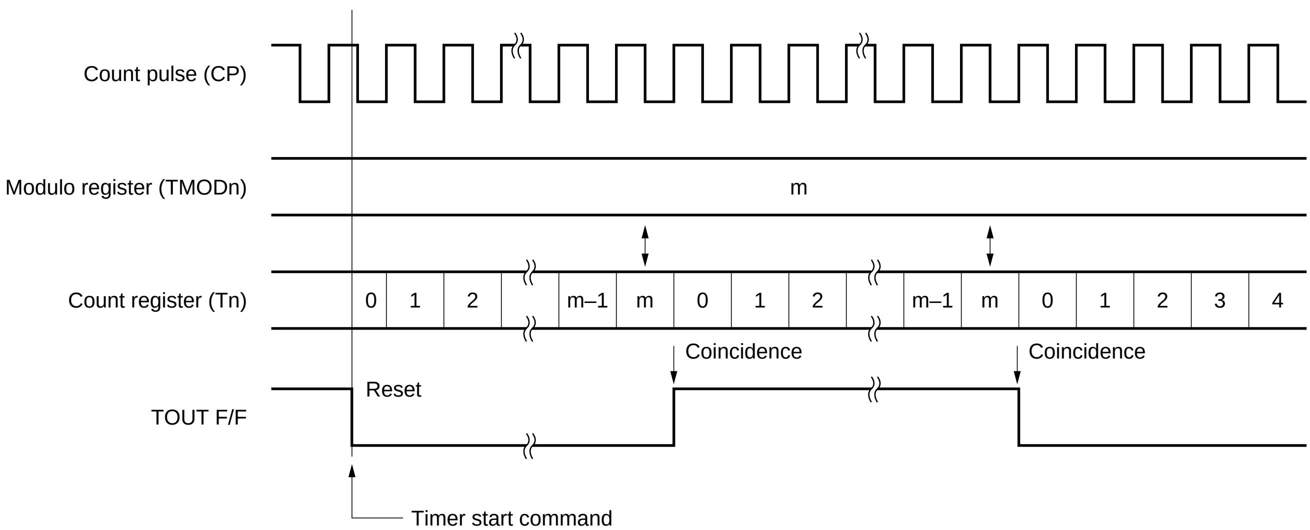
Fig. 5-35 Configuration When Timer/Event Counter Operates



Notes **1.** Watch timer (INTW) output : channel 0
Timer/event counter (channel 0) output : channel 1

2. The signal output to the serial interface can be output only by channel 0 of the timer/event counter.

Fig. 5-36 Count Operation Timing



Remark m: set value of the modulo register
n = 0, 1

(4) Application of 8-bit timer/event counter mode

(a) As an interval timer that generates an interrupt at 50-ms intervals

- Set the higher 4 bits of the timer/event counter mode register (TM0) to 0100B, and select 62.5 ms (f_X = 4.19 MHz) as the longest set time.
- Set the lower 4 bits of TM0 to 1100B.
- The set value of the timer/event counter modulo register (TMOD0) is as follows:

$$\frac{50 \text{ ms}}{244 \ \mu\text{s}} = 205,\ 205 - 1 = \text{CCH}$$

<Program example>

```
SEL   MB15              ; or CLR1 MBE
MOV   XA, #0CCH
MOV   TMOD0, XA         ; Sets modulo
MOV   XA, #01001100B
MOV   TM0, XA           ; Sets mode and starts timer
EI                      ; Enables interrupt
EI    IET0              ; Enables timer interrupt
```

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

(b) To generate interrupt when the number of pulses input from the TIn pin reaches 100 (pulse is high-active)

- Set the higher 4 bits of the timer/event counter mode register (TMn) to 0000B and select the rising edge.
- Set the lower 4 bits of TMn to 1100B.
- Set the timer/event counter modulo register (TMODn) to 100 – 1 = 63H.

<Program example>

```
SEL   MB15              ; or CLR1 MBE
MOV   XA, #100–1
MOV   TMODn, XA         ; Sets modulo
MOV   XA, #00001100B
MOV   TMn, XA           ; Sets mode and starts count
EI
EI    IETn              ; Enables INTTn
```

Remark n = 0, 1

5.5.3 Operation in 16-bit timer/event counter mode

In this mode, two timer/event counter channels, 0 and 1, are used in combination to implement 16-bit programmable interval timer or event timer operation.

(1) Register setting

In the 16-bit timer/event counter mode, the following six registers are used:

- Timer/event counter mode registers TM0 and TM1
- Timer/event counter registers T0 and T1
- Timer/event counter modulo registers TMOD0 and TMO1
- Timer/event counter output enable flag (TOE0)

(a) Timer/event counter mode registers (TM0 and TM1)

In the 16-bit timer/event counter mode, TM0 and TM1 are set as shown in Fig. 5-37 (for the formats of TM0 and TM1, refer to **Fig. 5-30 Format of Timer/Event Counter Mode Register (Channel 0)** and **Fig. 5-31 Format of Timer/Event Counter Mode Register (Channel 1)**).

TM0 and TM1 are manipulated by an 8-bit manipulation instruction. Bit 3 of TM0 (TM03) is a timer start command bit that can be manipulated in 1-bit units and is automatically cleared to 0 when the timer starts operating.

TM0 and TM1 are cleared to 00H when the internal reset signal is asserted.

The flags shown by a solid line are used in the 16-bit timer/event counter mode.

Do not use the flags shown by a dotted line in the 16-bit timer/event counter mode (clear these flags to 0).

Fig. 5-37 Setting of Timer/Event Counter Mode Registers

Address	7	6	5	4	3	2	1	0	Symbol
FA0H	–	TM06	TM05	TM04	TM03	TM02	TM01	TM00	TM0
FA8H	–	TM16	TM15	TM14	TM13	TM12	TM11	TM10	TM1

Count pulse (CP) select bit

TMn6	TMn5	TMn4	TM0	TM1
0	0	0	Rising edge of TI0	Rising edge of TI1
0	0	1	Falling edge of TI0	Falling edge of TI1
0	1	0	$f_W/2^{14}$ or $f_W/2^7$ (INTW output) Note	Overflow of count register (T0)
0	1	1	$f_X 2^2$	$f_X/2^2$
1	0	0	$f_X/2^{12}$	$f_X/2^{12}$
1	0	1	$f_X/2^8$	$f_X/2^{10}$
1	1	0	$f_X/2^6$	$f_X/2^8$
1	1	1	$f_X/2^4$	$f_X/2^6$

Timer start command bit

TM03	Clears counter and IRQT0 flag when "1" is written. Starts count operation if bit 2 is set to "1".

Operation mode

TM02	Count operation
0	Stops (count value retained)
1	Count operation

Operation mode select bit

TM11	TM10	TM01	TM00	Mode
1	0	1	0	16-bit timer/event counter mode

Note $f_W = f_{XT}$ or $f_X/2^7$

Caution When in 16-bit timer/event counter mode, set TM1 to 00100010B.

(b) Timer/event counter output enable flag (TOE0)

When operating 16-bit timer/event counter output, set TOE0 as shown in Fig. 5-38.

Fig. 5-38 Setting of Timer/Event Counter Output Enable Flag

Address	3	2	1	0	
FA2H	TOE0	–	–	–	Channel 0

Timer/event counter output enable flag (W)

TOE0	Timer Output
0	Disabled (low level output)
1	Enabled

(2) 16-bit time setting of timer/event counter

[Timer set time] (cycle) is calculated by dividing the [contents of modulo register + 1] by the [count pulse (CP) frequency] selected by the mode register.

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

where,

T (sec) : timer set time (seconds)

f_{CP} (Hz) : CP frequency (Hz)

n : contents of modulo register ($n \neq 0$)

Once the timer has been set, interrupt request flag IRQT0 is set at the set time interval of the timer.

Table 5-8 shows the resolution of each count pulse of the 16-bit timer/event counter and the longest set time (time when FFH is set to the modulo register 0 and 1).

★ **Table 5-8 Resolution and Longest Set Time (for 16-bit timer)**

Mode Register			6.0 MHz Operation		4.19 MHz Operation	
TM06	TM05	TM04	Resolution	Longest set time	Resolution	Longest set time
0	1	0	0.35 s (2.73 ms)Note	22906 s (179 s)Note	0.5 s (3.91 ms)Note	32768 s (256.2 s)Note
0	1	1	677 μs	43.7 ms	0.95 μs	62.5 ns
1	0	0	171 μs	11.2 s	244 μs	16.0 s
1	0	1	42.7 μs	2.80 s	61.0 μs	4.0 s
1	1	0	10.7 μs	699 ms	15.3 μs	1.0 s
1	1	1	2.67 μs	175 ms	3.82 μs	250 ms

Note When f_W = 32.768 kHz and WM1 = 0. ( ) : when WM1 = 1

Cautions 1. When in 16-bit timer/event counter mode, set TM1 to 00100010B.

2. The resolution is determined by CP of timer channel 0.

(3) 16-bit timer/event counter operation

The timer/event counter operates as follows.

Fig. 5-39 shows the configuration when the timer/event counter operates.

<1> The count pulse (CP) is selected by the timer/event counter mode registers TM0 and TM1 and is input to timer/event counter count register T0. The overflow of T0 is input to timer/event counter count register T1.

<2> The contents of T0 are compared with those of timer/event counter modulo register TMOD0. When the contents of these registers coincide, a coincidence signal is generated.

<3> The contents of T1 are compared with those of timer/event counter modulo register TMOD1. When the contents of these registers coincide, a coincidence signal is generated.

<4> If the coincidence signals in <2> and <3> overlap, interrupt request flag IRQT0 is set. At the same time, timer out flip-flop TOUT F/F is inverted.

Fig. 5-40 shows the operation timing of the 16-bit timer/event counter operation.

The 16-bit timer/event counter operation is usually started by the following procedure:

<1> Set the higher 8 bits of the number of counts 16 bits wide to TMOD1.

<2> Set the lower 8 bits of the number of counts 16 bits wide to TMOD2.

<3> Set 00100010B to TM1.

<4> Set the operation mode, count pulse, and start command to TM0.

Cautions 1. Set a value other than 00H to the timer/event counter modulo register TMOD0.

2. Set timer/event counter interrupt enable flag (IET1) to 0 (disabled).

To use timer/event counter output pin PTO0, set the P20 pin as follows:

<1> Clear the output latch of P20.

<2> Set port 2 in the output mode.

<3> Disconnect the internal pull-up resistor from port 2.

<4> Set timer/event counter output enable flag TOE0 to 1.

Fig. 5-39 Configuration When Timer/Event Counter Operates

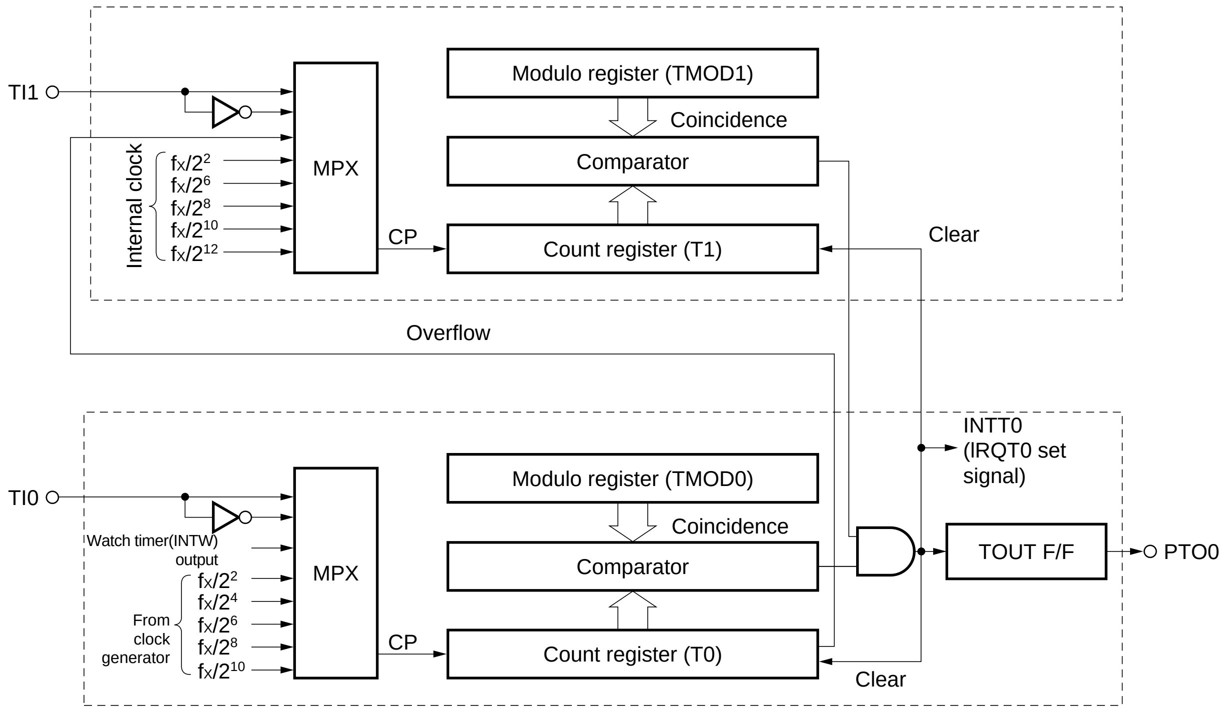
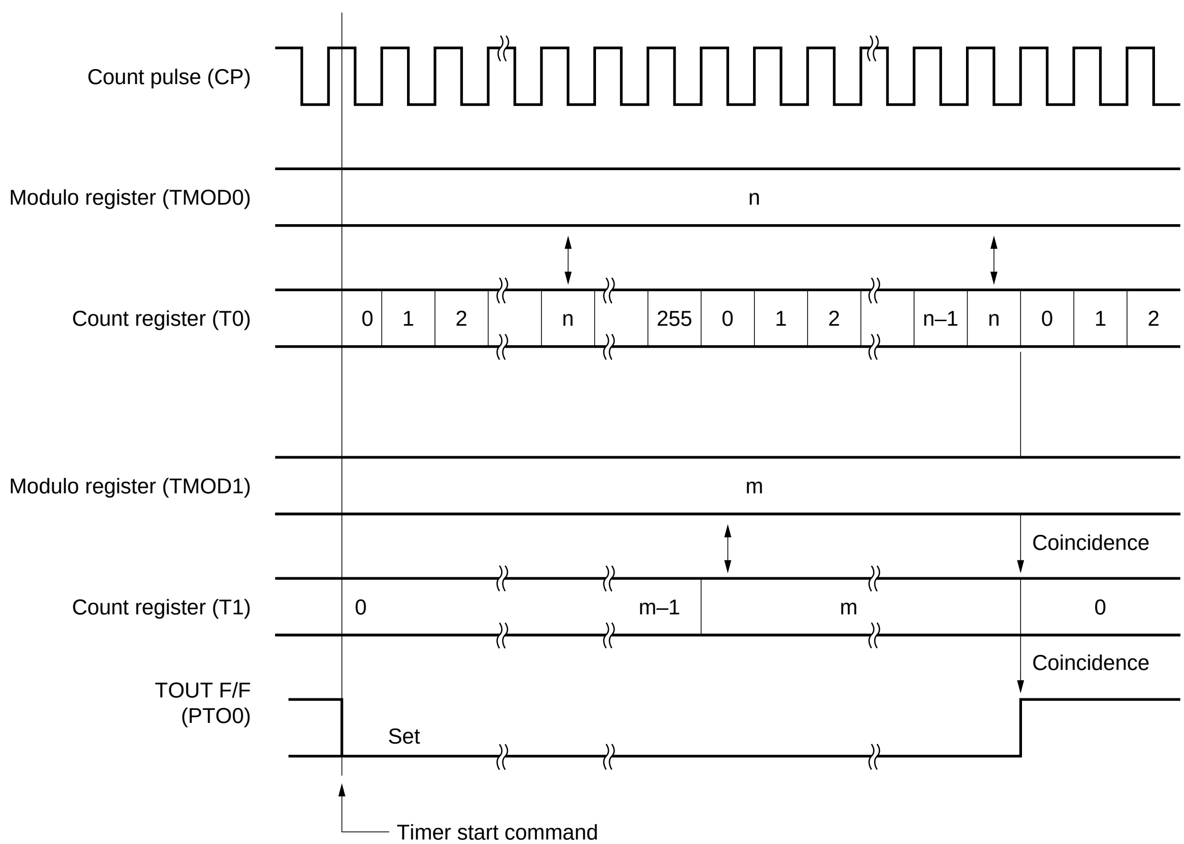


Fig. 5-40 Timing of Count Operation



Remark m: set value of the modulo register (TMOD1)
n: set value of the modulo register (TMOD0)

(4) Application of 16-bit timer/event counter mode

(a) As an interval timer that generates an interrupt at 5-sec intervals

- Set the higher 4 bits of the timer/event counter mode register (TM1) to 0010B, and select the overflow of timer/event counter count register (T0).
- Set the higher 4 bits of TM0 to 0100B and select 16.0 sec as the longest set time.
- Set the lower 4 bits of TM1 to 0010B and select the 16-bit timer/counter mode.
- Set the lower 4 bits of TM0 to 1110B, select the 16-bit timer/counter mode and count operation. Then, issue the timer start command.
- The set values of the timer/event counter modulo registers (TMOD0 and TMOD1) are as follows:

$$\frac{5 \text{ sec}}{244\ \mu s} = 20491.8,\ 20492 - 1 = 500BH$$

<Program example>

```
SEL     MB15            ;   or CLR1 MBE
MOV     XA, #050H
MOV     TMOD1, XA       ;   Sets modulo (higher 8 bits)
MOV     XA, #00B
MOV     TMOD0, XA       ;   Sets modulo (lower 8 bits)
MOV     XA, #00100010B
MOV     TM1, XA         ;   Sets mode
MOV     XA, #01001110B
MOV     TM0, XA         ;   Sets mode and starts timer
DI      IET1            ;   Disables timer (channel 1) interrupt
EI                      ;   Enables interrupts
EI      IET0            ;   Enables timer (channel 0) interrupt
```

Remark n this application, the TI0, and TI1 pins can be used as input pins.

(b) To generate interrupt when the number of pulses input from the TI0 pin reaches 1000 (pulse is high-active)

- Set the higher 4 bits of the timer/event counter mode register (TM1) to 0010B and select the overflow of the timer/event counter count register (T0).
- Set the higher 4 bits of TM0 to 0000B and select the rising edge of the TI0 input.
- Set the lower 4 bits of TM1 to 0010B and select the 16-bit timer/event counter mode.
- Set the lower 4 bits of TM0 to 1110B, select the 16-bit timer/event counter mode and count operation. Then, issue the timer start command.
- The set value of the timer/event counter modulo registers (TMOD0 and TMOD1) is 1000 – 1 = 999 = 03E7H. Set 03H to TMOD0 and E7H to TMOD1.

<Program example>

```
SEL     MB15            ; or CLR1 MBE
MOV     XA, #003
MOV     TMOD1, XA       ; Sets modulo (higher 8 bits)
MOV     XA, #0E7H
MOV     TMOD0, XA       ; Sets modulo (lower 8 bits)
MOV     XA, #00100010B
MOV     TM1, XA         ; Sets mode
MOV     XA, #00001110B
MOV     TM0, XA         ; Sets mode and starts timer
DI      IET1            ; Disables timer (channel 1) interrupt
EI
EI      IET0            ; Enables timer (channel 0) interrupt
```

Remark In this application, TI1 pin can be used as input pin.

(c) This is used as an interval timer to generate an interrupt every 9 hours using the watch timer (INTW) output.

- Set the higher 4 bits of the timer/event counter mode register (TM1) to 0010B to select overflow of the timer/event counter count register (T0).
- Set the higher 4 bits of TM0 to 0010B to select maximum time setting of 32768 sec (set f_W = 32.768 kHz, WM1 = 0).
- Set the lower 4 bits of TM1 to 0010B to select the 16-bit timer/event counter mode.
- Set the lower 4 bits of TM0 to 1110B to select the 16-bit timer/event counter mode and count operation, and specify the timer start.
- The set value for the timer/event counter modulo registers (TMOD0 and TMOD1) is as follows.

$$\frac{\text{9 hours (32400 s)}}{\text{0.5 s}} = 64800,\ 64800–1 = \text{FD1FH}$$

<Program example>

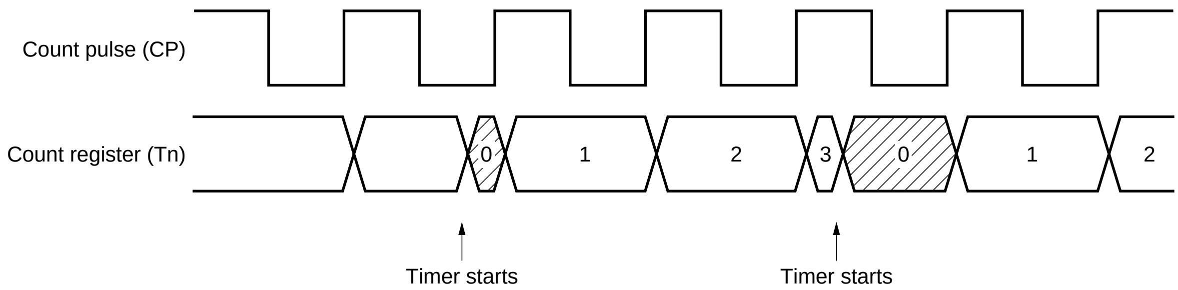
```
SEL   MB15             ; or CLR1 MBE
MOV   XA, #0FDH
MOV   TMOD1, XA        ; Sets modulo (for higher 8 bits).
MOV   XA, #01FH
MOV   TMOD0, XA        ; Sets modulo (for lower 8 bits).
MOV   XA, #00100010B
MOV   TM1, XA          ; Sets mode.
MOV   XA, #00101110B
MOV   TM0, XA          ; Sets mode, timer start
DI    IET1             ; Disables timer (channel 1) interrupt.
EI                     ; Enables interrupt.
EI    IET0             ; Enables timer (channel 0) interrupt.
```

Remark In this application, the TI0 and TI1 pins can be used as input pins.

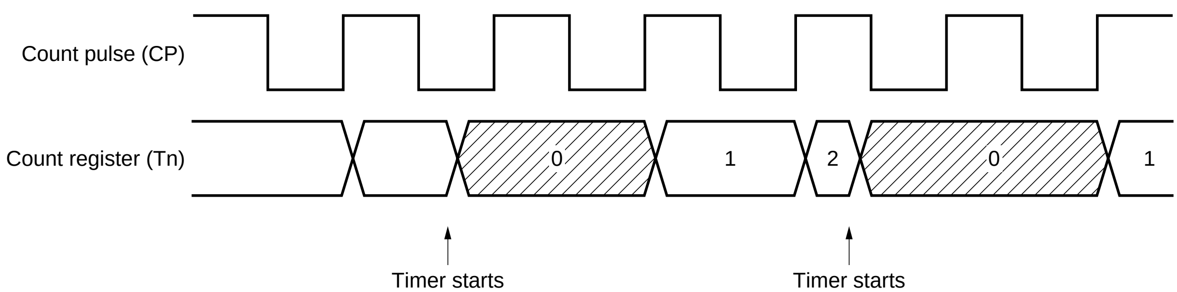
5.5.4 Notes on using timer/event counter

(1) Error when timer starts

After the timer has been started (bit 3 of TMn has been set to “1”), the time required for generation of the coincidence, which is calculated by the expression (contents of modulo register + 1) × resolution, deviates by up to one clock of count pulse (CP). This is because count register Tn is cleared asynchronously with CP, as shown below.



If the frequency of CP is greater than one machine cycle, the time required for generation of the coincidence signal, which is calculated by the expression (modulo register contents + 1) × resolution, deviates by up to CP2 clock after the timer has been started (bit 3 of TMn has been set to “1”). This is because Tn is cleared asynchronously with CP, based on the CPU clock, as shown below.



Remark n = 0, 1

(2) Note on starting timer

Usually, count register Tn and interrupt request flag IRQTn are cleared when the timer is started (bit 3 of TMn is set to “1”). However, if the timer is in an operation mode, and if IRQTn is set as soon as the timer is started, IRQTn may not be cleared. This does not pose any problem when IRQTn is used as a vector interrupt. In an application where IRQTn is being tested, however, IRQTn is not set after the timer has been started and this poses a problem. Therefore, there is a possibility that the timer could be started as soon as IRQTn is set to 1, either stop the timer once (by clearing the bit 2 of TMn to “0”), or start the timer two times.

Example If there is a possibility that timer could be started as soon as IRQTn is set

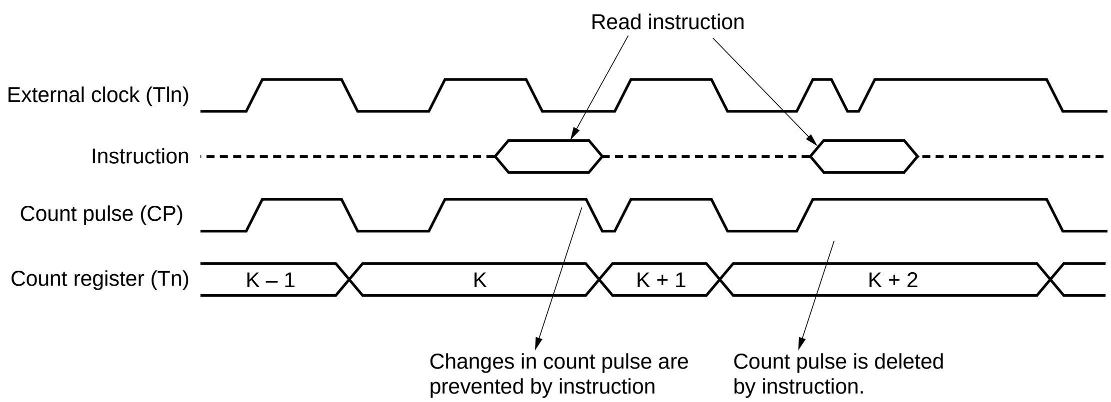
```
SEL     MB15
MOV     XA, #0
MOV     TMn, XA     ; Stops timer
MOV     XA, #4CH
MOV     TMn, XA     ; Restarts
Or,
SEL     MB15
SET1    TMn.3
SET1    TMn.3       ; Restarts
```

Remark n = 0, 1

(3) Error when count register is read

The contents of the count register (Tn) can be read at any time by using an 8-bit data memory manipulation instruction. While this instruction is executed, the count pulse (CP) is prevented from being changed. This means that Tn is not changed. Consequently, if TIn input is used as the signal source of CP, CP is deleted by the instruction execution time. (This phenomenon does not occur if the internal clock is used as CP because it is synchronized with the instruction.)

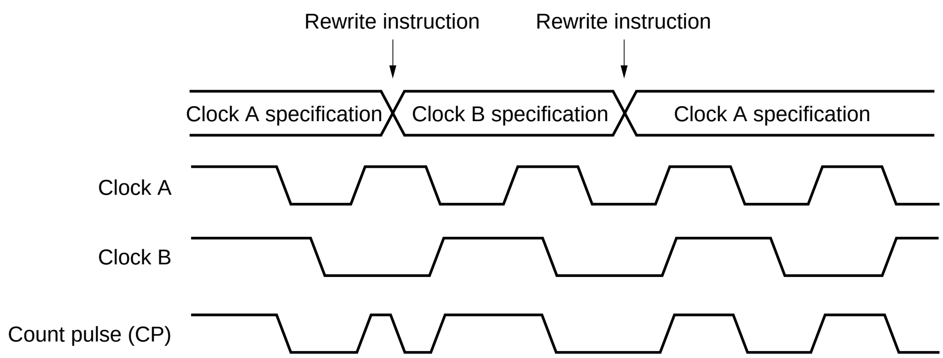
To input TIn as CP and read the contents of Tn, therefore, a signal with a pulse width that does not cause mis-count even if CP is deleted must be input. Because counting is kept pending by a read instruction for the duration of 1 machine cycle, the pulse to be input to TIn must be wider than 1 machine cycle.



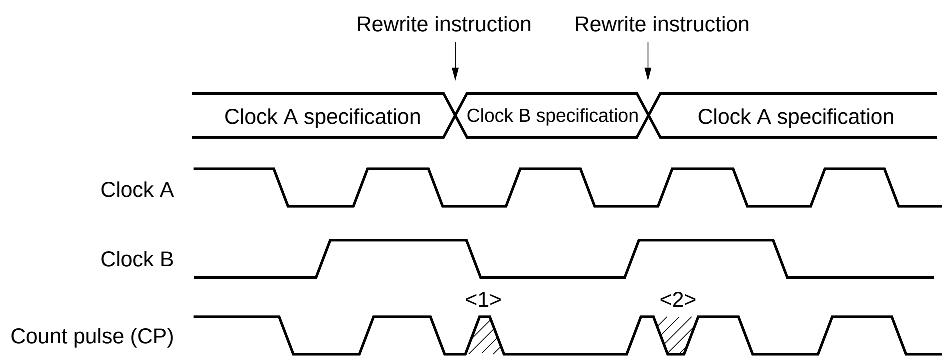
Remark n = 0, 1

(4) Notes on changing count pulse

When it is specified to change the count pulse (CP) by rewriting the contents of the timer/event counter mode register (TMn), the specification becomes valid immediately after execution of the instruction that commands the specification.



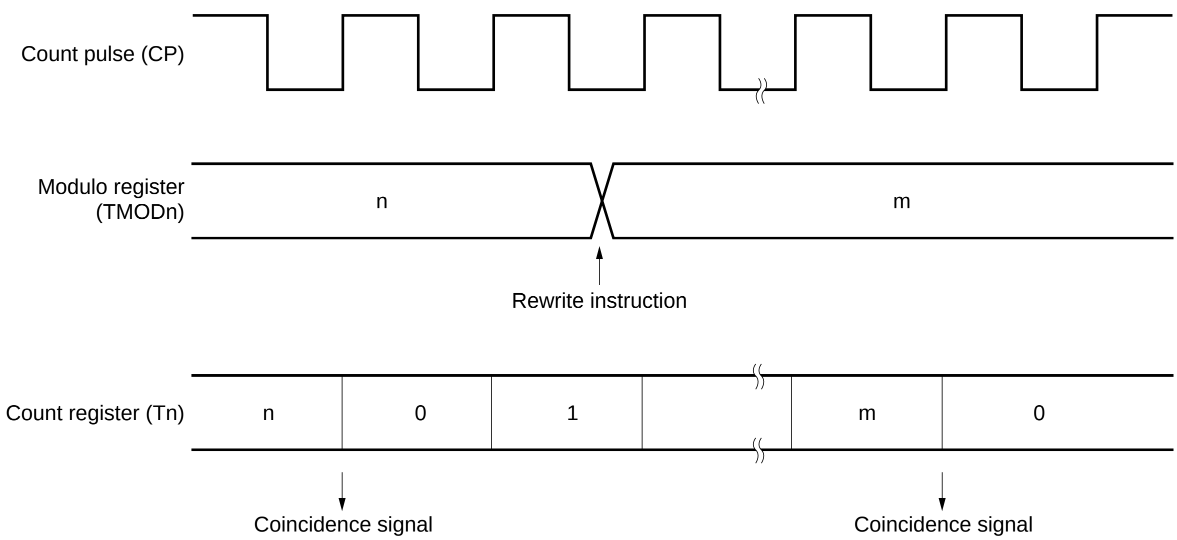
A whisker-like CP (<1> or <2 > in the figure below) may be generated depending on the combination of the clocks for changing CP. In this case, a miscount may occur or the contents of the count register (Tn) may be destroyed. To change CP, be sure to set the bit 3 of TMn bit to “1” and restart the timer at the same time.



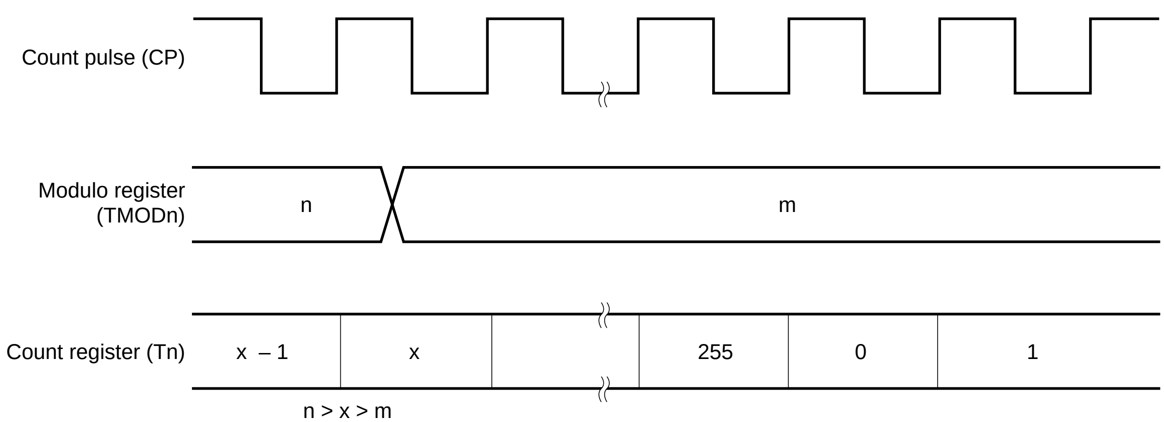
Remark n = 0, 1

(5) Operation after changing modulo register

The content of the modulo register (TMODn) is changed as soon as an 8-bit data memory manipulation instruction has been executed.



If the value of TMODn after change is less than the value of the count register (Tn), Tn continues counting. When an overflow occurs, Tn starts counting again from 0. If the value of TMODn after the change is less than the value before change (n), it is necessary to restart the timer after changing TMODn.



5.6 Serial Interface

5.6.1 Function of serial interface

The μPD750068 has an 8-bit clocked serial interface that can operate in the following three modes:

(1) Operation stop mode

This mode is used when serial transfer is not performed in order to reduce the power dissipation.

(2) 3-line serial I/O mode

In this mode, three lines are used to transfer 8-bit data: serial clock ($\overline{SCK}$), serial output (SO), and serial input (SI).

Because transmission and reception can be simultaneously performed in this mode, the processing time of data transfer is very short.

Moreover, it can be specified whether serial data is transferred starting from the MSB or LSB. This means that the μPD750068 can communicate with any device.

In the three-line serial I/O mode, the devices in the 75XL series, 75X series, and 78K series, and various peripheral I/O devices can be connected.

(3) 2-line serial I/O mode

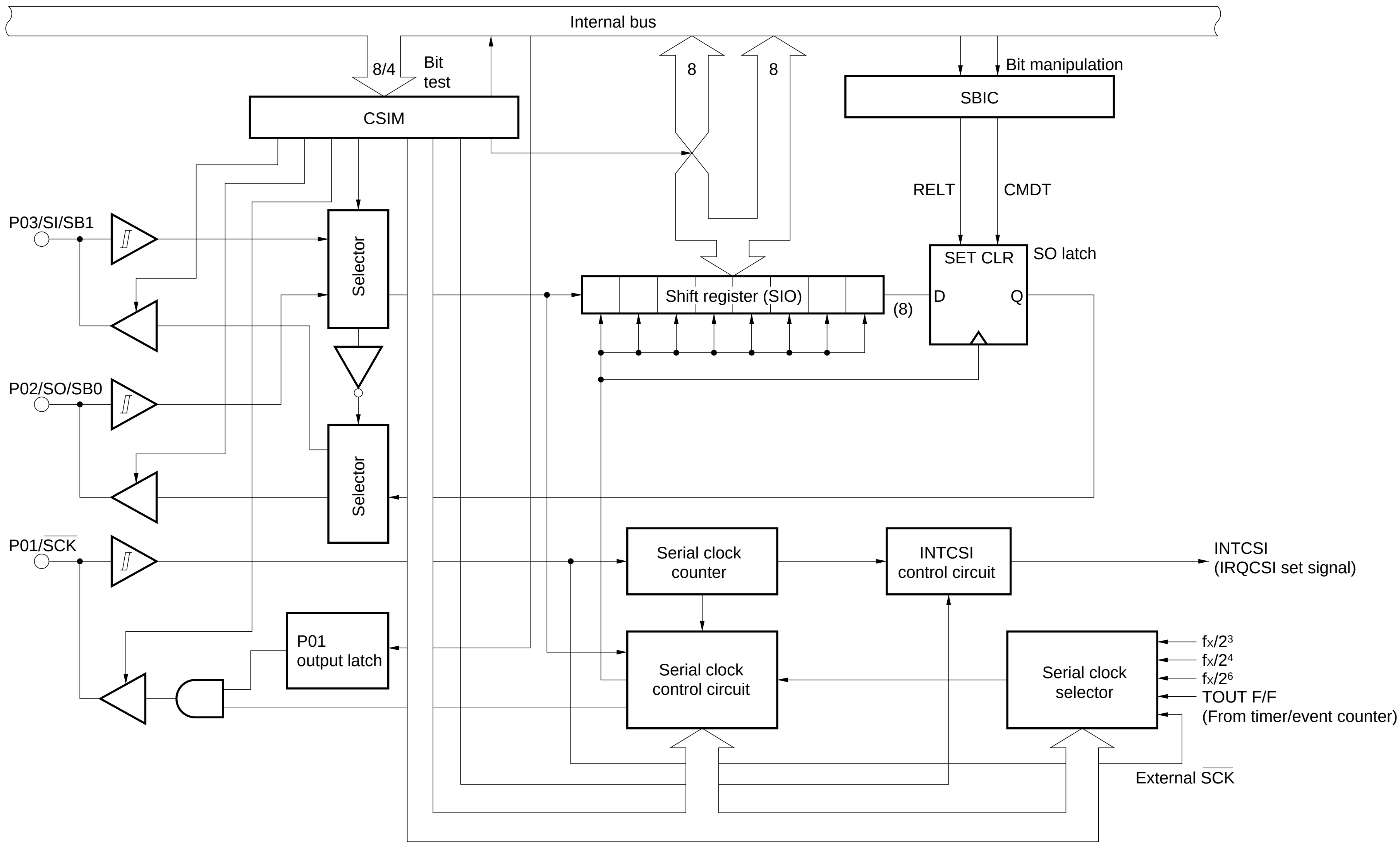
In this mode, two lines, serial clock ($\overline{SCK}$) and serial data bus (SB0 or SB1), are used to transfer 8-bit data. By manipulating the output levels of these lines via software, the μPD750068 can communicate with two or more devices.

Because the output levels of $\overline{SCK}$ and SB0 (or SB1) can be manipulated via software, any transfer format can be used. Therefore, a handshake line which has been conventionally necessary for connecting two or more devices is not necessary, and the I/O ports can be effectively used.

5.6.2 Configuration of serial interface

Fig. 5-41 shows the block diagram of the serial interface.

Fig. 5-41 Block Diagram of Serial Interface



(1) Serial operation mode register (CSIM)

This 8-bit register specifies the operation mode and serial clock wake-up function of the serial interface (for details, refer to **5.6.3 (1) Serial operation mode register (CSIM)**).

(2) Serial bus interface control register (SBIC)

This 8-bit register consists of bits that control the status of the serial bus and flags that indicate the various statuses of the data input from the serial bus. (for details, refer to **5.6.3 (2) Serial bus interface control register (SBIC)**).

(3) Shift register (SIO)

This register converts 8-bit serial data into parallel data or 8-bit parallel data into serial data. It performs transmission or reception (shift operation) in synchronization with the serial clock. The user controls actual transmission or reception by writing data to the SIO (for details, refer to **5.6.3 (3) Shift register (SIO)**).

(4) SO latch

This latch holds the levels of the SO/SB0 and SI/SB1 pins. It can also be controlled directly via software (for details, refer to **5.6.3 (2) Serial bus interface control register (SBIC)**).

(5) Serial clock selector

This selects the serial clock to be used.

(6) Serial clock counter

This counter counts the number of serial clocks output or input when transmission or reception operation is performed in order to check whether 8-bit data has been transmitted or received.

(7) INTCSI control circuit

This circuit controls generation of an interrupt request. The interrupt request (INTCSI) is generated in the following cases. When the interrupt request is generated, an interrupt request flag (IRQCSI) is set (refer to **Fig. 6-1 Block Diagram of Interrupt Control Circuit**).

- **In 3-line and 2-line serial I/O modes**
  The interrupt request is generated each time eight serial clocks have been counted.

(8) Serial clock control circuit

This circuit controls the supply of the serial clock to the shift register. It also controls the clock output to the SCK pin when the internal system clock is used.

(9) P01 output latch

This latch generates the serial clock via software after eight serial clock have been generated.
It is set to “1” when the reset signal is input.
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)

Fig. 5-42 shows the format of the serial operation mode register (CSIM).

CSIM is an 8-bit register that specifies the operation of the serial interface, serial clock function, etc.

This register is manipulated by an 8-bit memory manipulation instruction. The bit 7 of this register can also be manipulated in 1-bit units. To manipulate a bit, use the name of the bit.

All the bits are cleared to 0 when the $\overline{\text{RESET}}$ signal is asserted.

Fig. 5-42 Format of Serial Operation Mode Register (CSIM) (1/3)

Address	7	6	5	4	3	2	1	0	Symbol
FE0H	CSIE	0	0	CSIM4	CSIM3	CSIM2	CSIM1	CSIM0	CSIM

- CSIM1, CSIM0: Serial clock select bits (W)
- CSIM4 to CSIM2: Serial interface operation mode select bits (W)
- CSIE: Serial interface operation enable/disable bit (W)

Caution Be sure to set bits 6 and 5 of CSIM to 0.

Remark (W) : write only

Fig. 5-42 Format of Serial Operation Mode Register (CSIM) (2/3)

Serial interface operation enable/disable bit (W)

		Operation of Shift Register	Serial Clock Counter	IRQCSI Flag	SO/SB0 and SI/SB1 Pins
CSIE	0	Shift operation disabled	Clear	Retained	Port 0 function
	1	Shift operation enabled	Count operation	Can be set	Function in each mode and port 0 function shared

Serial interface operation mode select bit (W)

CSIM4	CSIM3	CSIM2	Operation Mode	Bit Order of Shift Register	SO/SB0/P02 Pin Function	SI/SB1/P03 Pin Function
×	0	0	3-line serial I/O mode	$SIO_{7-0} \leftrightarrow XA$ (MSB first)	SO (CMOS output)	SI (CMOS input)
		1		$SIO_{0-7} \leftrightarrow XA$ (LSB first)		
0	1	1	2-line serial I/O mode	$SIO_{7-0} \leftrightarrow XA$ (MSB first)	SBK0 (N-ch open-drain I/O)	P03 (CMOS input)
1					P02 (CMOS input)	SB1 (N-ch open-drain I/O)
Others			Setting prohibited			

Remark ×: don't care

Serial clock select bit (W)

CSIM1	CSIM0	Serial Clock: 3-line Serial I/O Mode	Serial Clock: 2-line Serial I/O Mode	$\overline{SCK}$ Pin Mode
0	0	External clock input to $\overline{SCK}$ pin		Input
0	1	Timer/event counter output 0 (TO0)		Output
1	0	$f_X/2^4$ (375 kHz at 6.0 MHz, 262 kHz at 4.19 MHz)	$f_X/2^6$ (93.8 kHz at 6.0 MHz, 65.5 kHz at 4.19 MHz)	
1	1	$f_X/2^3$ (750 kHz at 6.0 MHz, 524 kHz at 4.19 MHz)		

Fig. 5-42 Format of Serial Operation Mode Register (CSIM) (3/3)

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

CSIE	CSIM3	CSIM2	Operation Mode
0	×	×	Operation stop mode
1	0	×	3-line serial I/O mode
1	1	1	2-line serial I/O mode

2. P01/$\overline{SCK}$ pin is set in the following status by the setting of CSIE, CSIM1, and CSIM0:

CSIE	CSIM1	CSIM0	Status of P01/$\overline{SCK}$ Pin
0	0	0	Input port (P01)
1	0	0	High-impedance ($\overline{SCK}$ input)
0	0	1	High-level output
0	1	0	
0	1	1	
1	0	1	Serial clock output (high-level output: at serial transfer end)
1	1	0	
1	1	1	

3. Clear CSIE during serial transfer in the following procedure:

<1> Clear the interrupt enable flag (IECSI) to disable the interrupt.

<2> Clear CSIE.

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

Examples 1. To select $f_X/2^4$ as the serial clock, generate serial interrupt IRQCSI each time serial transfer is completed. Then, select a mode in which serial transfer of the MSB-first is performed in 3-line serial I/O mode

```
SEL     MB15            ;  or CLR1 MBE
MOV     XA, #10000010B
MOV     CSIM, XA        ;  CSIM ← 10000010B
```

2. To enable serial transfer according to the contents of CSIM

```
SEL     MB15            ;  or CLR1 MBE
SET1    CSIE
```

(2) Serial bus interface control register (SBIC)

Fig. 5-43 shows the format of the serial bus interface control register (SBIC).

SBIC is an 8-bit register that controls the serial bus.

This register is manipulated by a bit manipulation instruction. It cannot be manipulated by a 4- or 8-bit memory manipulation instruction.

All the bits are cleared to 0 when the $\overline{RESET}$ signal is asserted.

Fig. 5-43 Format of Serial Bus Interface Control Register (SBIC)

Address	7	6	5	4	3	2	1	0	Symbol
FE2H	0	0	0	0	0	0	CMDT	RELT	SBIC

RELT: Bus release trigger bit (W)

CMDT: Command trigger bit (W)

Remark (W): write only

Command trigger bit (W)

CMDT	This bit controls output trigger of command signal (CMD). When this bit is set to 1, SO latch is cleared to 0. Subsequently, the CMDT bit is automatically cleared to 0.

Caution Do not set CMDT bit during serial transfer. Be sure to set it before the start of or after the end of transfer.

Bus release trigger bit (W)

RELT	This bit controls output trigger of bus release signal (REL). When this bit is set to 1, SO latch is set to 1. Subsequently, the RELT bit is automatically cleared to 0.

Caution Do not set RELT bit during serial transfer. Be sure to set it before the start of or after the end of transfer.

(3) Shift register (SIO)

Fig. 5-44 shows the configuration of the peripheral circuits of the shift register (SIO). SIO is an 8-bit register that converts parallel data to serial data or vice versa and performs serial transmission or reception (shift operation) in synchronization with the serial clock.

Serial transfer is started by writing data to SIO.

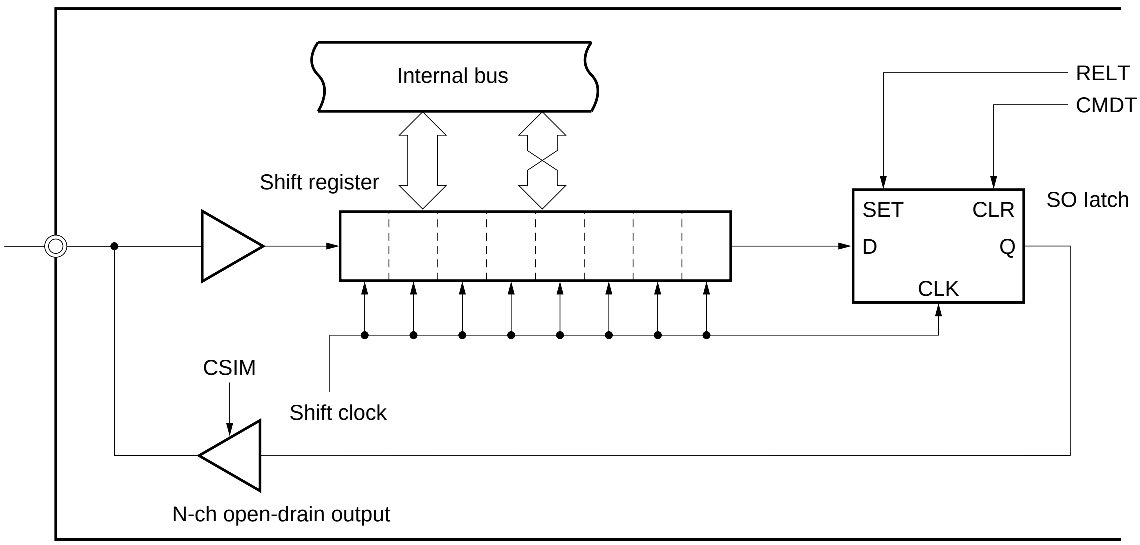
The data written to SIO is output to the serial output (SO) or serial data bus (SB0 or SB1) line during transmission. Data is read from the serial input (SI) or SB0 or SB1 to SIO during reception.

SIO can be read or written by an 8-bit manipulation instruction.

When the $\overline{\text{RESET}}$ signal is asserted during operation of SIO, the value of SIO becomes undefined. When the $\overline{\text{RESET}}$ signal is asserted in the standby mode, the value of SIO is retained.

The shift operation is stopped after 8-bit data has been transmitted or received.

Fig. 5-44 Peripheral Circuits of Shift Register



SIO can be read or serial transfer (write) can be started with the following timing:

- When the serial interface operation enable/disable bit (CSIE) = 1, except when CSIE is set to “1” after data has been written to the shift register
- When the serial clock is masked after 8-bit serial data has been transferred
- When $\overline{\text{SCK}}$ is high

Be sure to write or read data to or from the SIO when $\overline{\text{SCK}}$ is high.

The input pin of the data bus is shared with the output pin in the two-line serial I/O mode. The output pin is of N-ch open-drain configuration. Therefore, set FFH to the SIO of the device that is to receive data.

5.6.4 Operation stop mode

The operation stop mode is used when serial transfer is not performed, to reduce the power consumption.

In this mode, the shift register does not perform its shift operation. Therefore, it can be used as an ordinary 8-bit register.

When the reset signal is input, the operation stop mode is set. The P02/SO/SB0 and P03/SI/SB1 pins are set in the input port mode. The P01/$\overline{SCK}$ pin can be used as an input port pin if so specified by the serial operation mode register.

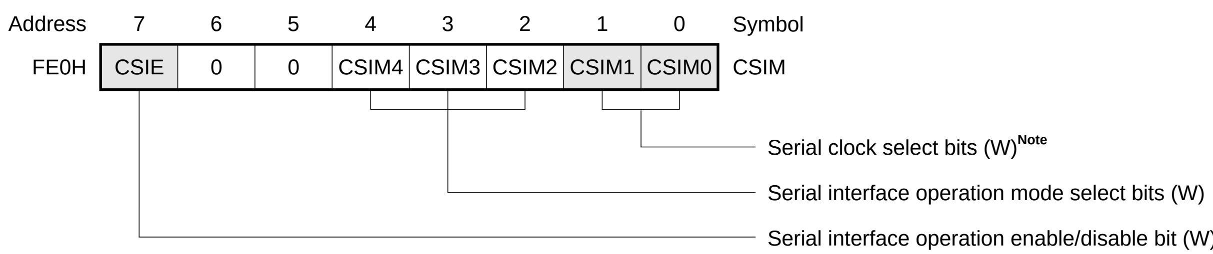
[Register setting]

The operation stop mode is set by using the serial operation mode register (CSIM) (for the format of the CSIM, refer to **5.6.3 (1) Serial operation mode register (CSIM)**).

The CSIM is manipulated in 8-bit units. However, the CSIE bit of this register can be manipulated in 1-bit units. The name of the bit can be used for manipulation.

CSIM is initialized to 00H at reset.

The shaded portions in the figure below indicate the bits used in the operation stop mode.



Note This bit can select the status of the P01/$\overline{SCK}$ pin.

Remark (W) : write only

Serial interface operation enable/disable bit (W)

		Operation of Shift Register	Serial Clock Counter	IRQCSI Flag	SO/SB0 and SI/SB1 Pins
CSIE	0	Shift operation disabled	Cleared	Retained	Dedicated to port 0 function

Serial clock select bit (W)

The P01/$\overline{\text{SCK}}$ pin is set in the following status according to the setting of the CSIM0 and CSIM1 bits.

CSIM1	CSIM0	Status of P01/$\overline{\text{SCK}}$ Pin
0	0	High impedance
0	1	High level
1	0	
1	1	

Clear the CSIE bit in the following procedure during serial transfer:

<1> Clear the interrupt enable flag (IECSI) to disable the interrupt.
<2> Clear CSIE.
<3> Clear the interrupt request flag (IRQCSI).

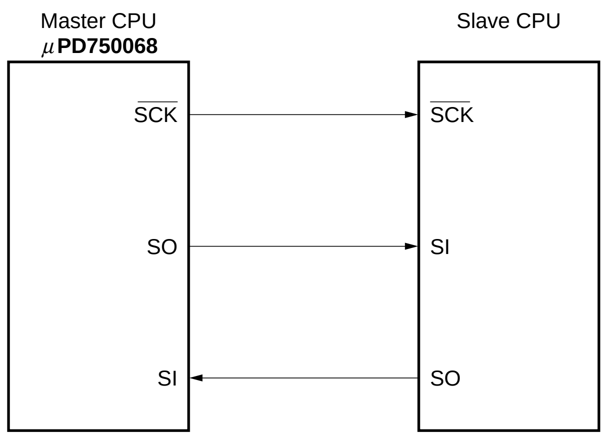
5.6.5 Operation in 3-line serial I/O mode

In the three-line operation mode, the μPD750068 can be connected to microcontrollers in the 75XL series, 75X series, and 78K series, and various peripheral I/O devices.

In this mode, communication is established by using three lines: serial clock ($\overline{SCK}$), serial output (SO), and serial input (SI).

Fig. 5-45 Example of System Configuration in 3-Line Serial I/O Mode

3-line serial I/O ↔ 3-line serial I/O



Remark The μPD750068 can be also used as a slave CPU.

(1) Register setting

When the three-line serial I/O mode is used, the following two registers must be set:

- Serial operation mode register (CSIM)
- Serial bus interface control register (SBIC)

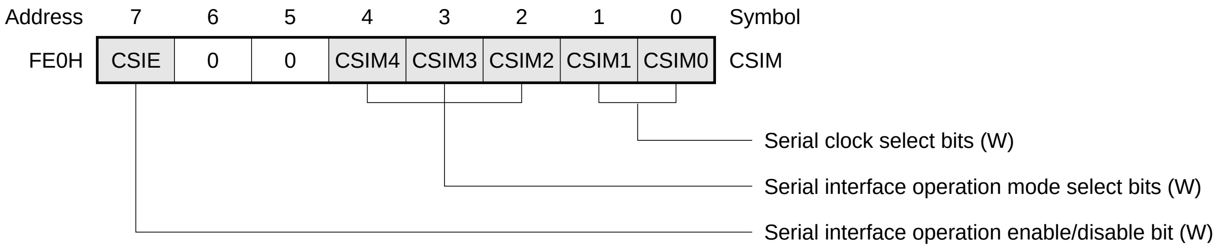
(a) Serial operation mode register (CSIM)

When the three-line serial I/O mode is used, set the CSIM as shown below (for the format of the CSIM, refer to **5.6.3 (1) Serial operation mode register (CSIM)**).

The CSIM is manipulated by using an 8-bit manipulation instruction. Bit 7 can also be manipulated in 1-bit units.

The contents of the CSIM are cleared to 00H at reset.

The shaded portion in the figure indicates the bits used in the three-line serial I/O mode.



Remark (W) : write only

Serial interface operation enable/disable bit (W)

		Operation of Shift Register	Serial Clock Counter	IRQCSI Flag	SO/SB0 and SI/SB1 Pins
CSIE	1	Shift operation enabled	Count operation	Can be set	Function in each mode and port 0 function shared

Serial interface operation mode select bit (W)

CSIM4	CSIM3	CSIM2	Bit Order of Shift Register	SO Pin Function	SI Pin Function
×	0	0	$SIO_{7-0} \leftrightarrow XA$ (MSB first)	SO (CMOS output)	SI (CMOS input)
		1	$SIO_{0-7} \leftrightarrow XA$ (LSB first)		

Remark ×: don't care

Serial clock select bit (W)

CSIM1	CSIM0	Serial Clock	$\overline{SCK}$ Pin Mode
0	0	External clock input to $\overline{SCK}$ pin	Input
0	1	Timer/event counter 0 output (TO0)	Output
1	0	$f_X/2^4$ (375 kHz: at 6.0 MHz operation, 262 kHz: at 4.19 MHz operation)	
1	1	$f_X/2^4$ (750 kHz: at 6.0 MHz operation, 524 kHz: at 4.19 MHz operation)	

(b) Serial bus interface control register (SBIC)

When the three-line serial I/O mode is used, set SBIC as shown below (for the format of SBIC, refer to **5.6.3 (2) Serial bus interface control register (SBIC)**).

This register is manipulated by using a bit manipulation instruction.

The contents of SBIC are cleared to 00H at reset.

The shaded portion in the figure indicate the bits used in the three-line serial I/O mode.

Address	7	6	5	4	3	2	1	0	Symbol
FE2H	0	0	0	0	0	0	CMDT	RELT	SBIC

RELT: Bus release trigger bit (W)

CMDT: Command trigger bit (W)

Remark (W) : write only

Command trigger bit (W)

CMDT	This bit controls the output trigger of a command signal (CMD). When this bit is set to 1, the SO latch is cleared to 0. Subsequently, the CMDT bit is automatically cleared to 0.

Bus release trigger bit (W)

RELT	This bit controls the output trigger of a bus release signal (REL). When this bit is set to 1, the SO latch is set to 1. Subsequently, the RELT bit is automatically cleared to 0.

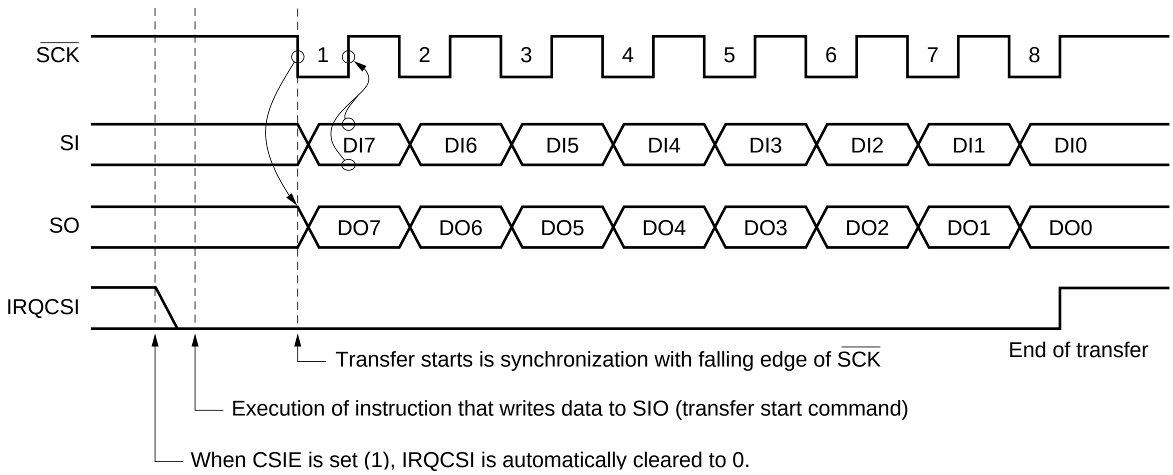
(2) Communication operation

In the three-line serial I/O mode, data is transmitted or received in 8-bit units. Each bit of the data is transmitted or received in synchronization with the serial clock.

The shift register performs its shift operation in synchronization with the falling edge of the serial clock ($\overline{SCK}$). The transmit data is retained by the SO latch and output from the SO pin. The receive data input to the SI pin is latched to the shift register at the rising edge of $\overline{SCK}$.

When 8-bit data has been completely transferred, the shift register automatically stops, and an interrupt request flag (IRQCSI) is set.

Fig. 5-46 Timing in 3-Line Serial I/O mode



Because the SO pin is a CMOS output pin and outputs the status of the SO latch, the output status of the SO pin can be manipulated by setting the RELT and CMDT bits.

However, do not perform this manipulation during serial transfer.

The output status of the SCK pin can be controlled by manipulating the P01 latch in the output mode (mode of the internal system clock)(refer to **5.6.7 Manipulating $\overline{SCK}$ pin output**).

(3) Selecting serial clock

The serial clock is selected by using the bits 0 and 1 of the serial operation mode register (CSIM). The following four types of serial clocks can be selected:

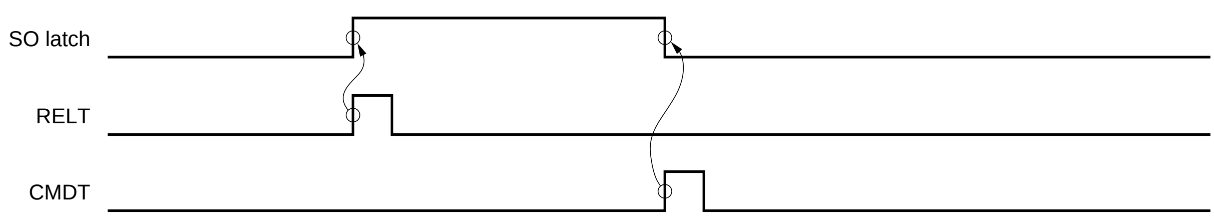
Table 5-9 Selecting Serial Clock and Application (in 3-line serial I/O mode)

Mode Register		Serial Clock		Timing at which shift register can be read/ written and serial transfer can be started	Application
CSIM 1	CSIM 0	Source	Masking Serial Clock		
★ 0	0	External $\overline{SCK}$	Automatically masked at end of transfer of 8-bit data	<1> In operation enable mode (CSIE = 1) <2> If serial clock is masked after 8-bit serial transfer <3> When $\overline{SCK}$ is high	Slave CPU
0	1	TOUT F/F			Half duplex start-stop synchronization transfer (software control)
1	0	$f_X/2^4$			Medium-speed serial transfer
1	1	$f_X/2^3$			High-speed serial transfer

(4) Signals

Fig. 5-47 illustrates the operations of RELT and CMDT.

Fig. 5-47 Operations of RELT and CMDT



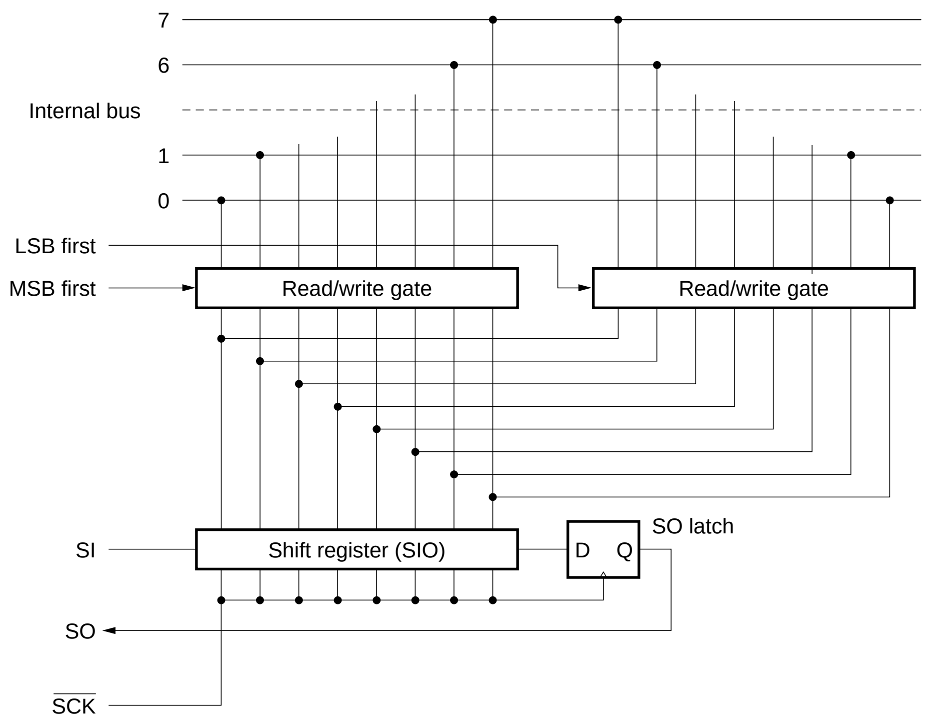
(5) Selecting MSB or LSB

In the three-line serial I/O mode, a function is provided to enable the user to select whether serial data is transferred starting from the MSB or LSB.

Fig. 5-48 shows the configuration of the shift register and internal bus. As shown in this figure, the MSB or LSB can be inverted to read or write data.

Whether transfer is started from the MSB or LSB can be specified by using the bit 2 of the serial operation mode register (CSIM).

Fig. 5-48 Transfer Bit Select Circuit



The bit (MSB or LSB) from which data transfer is started is selected by changing the bit sequence in which the data is written to the shift register (SIO). The shift sequence of SIO is always the same.

Therefore, select the bit from which data transfer is started before writing data to the shift register.

(6) Starting transfer

Serial transfer is started when the transfer data is set to the shift register (SIO), if the following two conditions are satisfied:

- Serial interface operation enable/disable bit (CSIE) = 1
- If the internal serial clock is stopped after 8-bit serial transfer or if $\overline{\text{SCK}}$ is high

Caution Transfer is not started even if CSIE is set to "1" after the data has been written to the shift register.

When 8-bit transfer has been completed, the serial transfer is automatically stopped, and an interrupt request flag (IRQCSI) is set.

Example To transfer the data of an RAM specified by the HL register to SIO and, at the same time, load the data of SIO to the accumulator and start serial transfer

```
MOV   XA, @HL   ; Takes out transfer data from RAM
SEL   MB15      ; or CLR1 MBE
XCH   XA, SIO   ; Exchanges transmit data and receive data, and starts transfer
```

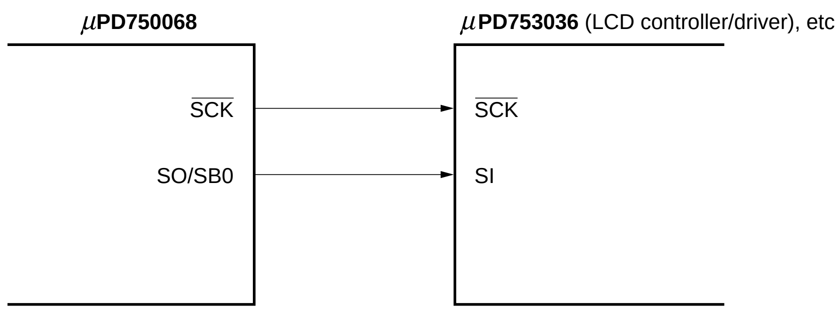
(7) Application of 3-line serial I/O mode

Examples 1. To transfer data with MSB first with 262-kHz transfer clock (at 4.19 MHz) (master operation)

<Program example>

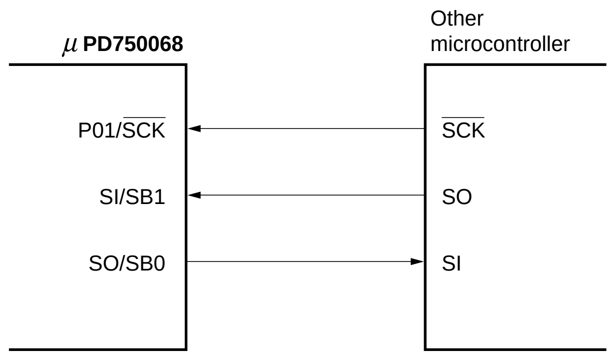
```
CLR1    MBE
MOV     XA, #10000010B
MOV     CSIM, XA        ; Sets transfer mode
MOV     XA, TDATA       ; TDATA is address storing transfer data
MOV     SIO, XA         ; Sets transfer data and starts transfer
```

Caution After transfer has been started for the first time, transfer can be started by setting data to SIO (by using MOV SIO, XA or XCH XA, SIO) the second time and subsequently.



In this example, the SI/SB1 pin of the μPD750068 can be used as an input pin.

Examples 2. To transfer data with LSB first with an external clock (slave operation)
(In this example, a function to invert MSB and LSB is used to read/write the shift register.)



<Program example>

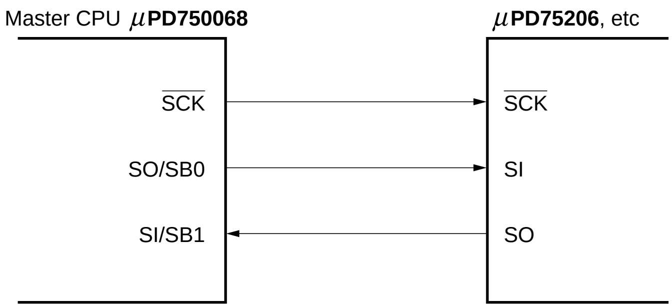
Main routine

```
CLR1  MBE
MOV   XA, #84H
MOV   CSIM, XA    ; Stops serial operation, MSB/LSB inverse mode, external clock
MOV   XA, TDATA
MOV   SIO, XA     ; Sets transfer data and starts transfer
EI    IECSI
EI
```

Interrupt routine (MBE = 0)

```
MOV   XA, TDATA
XCH   XA, SIO     ; Receive data ↔ transfer data, starts transfer
MOV   RDATA, XA   ; Saves receive data
RETI
```

Examples 3. To transmit or receive data at high speeds using a 524-kHz (at 4.19 MHz) transfer clock



<Program example> ... Master

```
        CLR1    MBE
        MOV     XA, #10000011B
        MOV     CSIM, XA        ; Sets transfer mode
        MOV     XA, TDATA
        MOV     SIO, XA         ; Sets transfer data and starts transfer
          :
LOOP :  SKTCLR  IRQCSI          ; Test IRQCSI
        BR      LOOP
        MOV     XA, SIO         ; Receives data
```

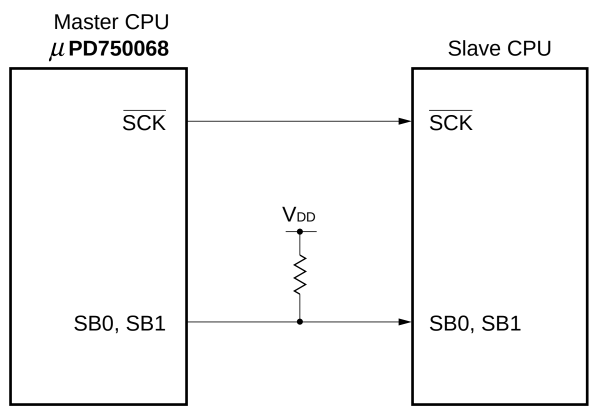
5.6.6 Operation in 2-line serial I/O mode

The two-line serial I/O mode can be used in any communication format if so specified by the program.

Basically, communication is established by using two lines: serial clock ($\overline{SCK}$) and serial data input/output (SB0 or SB1).

Fig. 5-49 Example of System Configuration in 2-Line Serial I/O Mode

2-line serial I/O ↔ 2-line serial I/O



Remark The μPD750068 can be also used as a slave CPU.

(1) Register setting

When the two-line serial I/O mode is used, the following two registers must be set:

- Serial operation mode register (CSIM)
- Serial bus interface control register (SBIC)

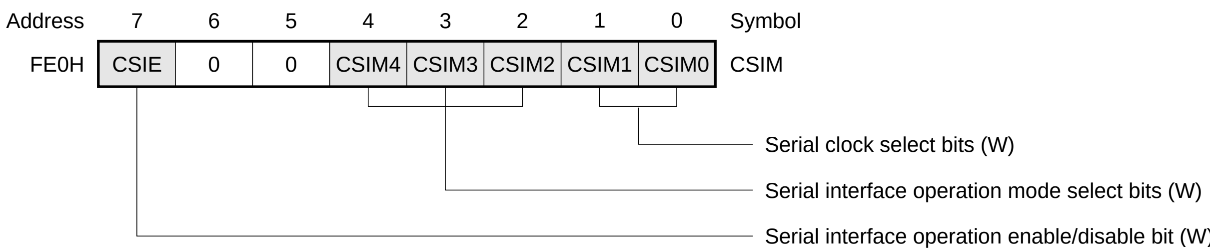
(a) Serial operation mode register (CSIM)

When the two-line serial I/O mode is used, set the CSIM as shown below (for the format of the CSIM, refer to **5.6.3 (1) Serial operation mode register (CSIM)**).

The CSIM is manipulated by using an 8-bit manipulation instruction. Bit 7 can also be manipulated in 1-bit units.

The contents of the CSIM are cleared to 00H at reset.

The shaded portion in the figure indicates the bits used in the two-line serial I/O mode.



Remark (W) : write only

Serial interface operation enable/disable bit (W)

		Operation of Shift Register	Serial Clock Counter	IRQCSI Flag	SO/SB0 and SI/SB1 Pins
CSIE	1	Shift operation enabled	Count operation	Can be set	Function in each mode and port 0 function shared

Serial interface operation mode select bit (W)

CSIM4	CSIM3	CSIM2	Bit Order of Shift Register	SB0/P02 Pin Function	SB1/P03 Pin Function
0	1	1	$SIO_{7-0} \leftrightarrow XA$ (MSB first)	SB0 (N-ch open-drain I/O)	P03 (CMOS input)
1				P02 (CMOS input)	SB1 (N-ch open-drain I/O)

Serial clock select bit (W)

CSIM1	CSIM0	Serial Clock	$\overline{SCK}$ Pin Mode
0	0	External clock input to $\overline{SCK}$ pin	Input
0	1	Timer/event counter 0 output (TO0)	Output
1	0	$f_X/2^6$ (93.8 kHz: at 6.0 MHz operation, 65.5 kHz: at 4.19 MHz operation)	
1	1		

(b) Serial bus interface control register (SBIC)

When the two-line serial I/O mode is used, set SBIC as shown below (for the format of SBIC, refer to **5.6.3 (2) Serial bus interface control register (SBIC)**).

This register is manipulated by using a bit manipulation instruction.

The contents of SBIC are cleared to 00H at reset.

The shaded portion in the figure indicate the bits used in the two-line serial I/O mode.

Address	7	6	5	4	3	2	1	0	Symbol
FE2H	0	0	0	0	0	0	CMDT	RELT	SBIC

RELT: Bus release trigger bit (W)

CMDT: Command trigger bit (W)

Remark (W) : write only

Command trigger bit

CMDT	This bit controls the output trigger of a command signal (CMD). When this bit is set to 1, the SO latch is cleared to 0. After that, the CMDT bit is automatically cleared to 0.

Bus release trigger bit (W)

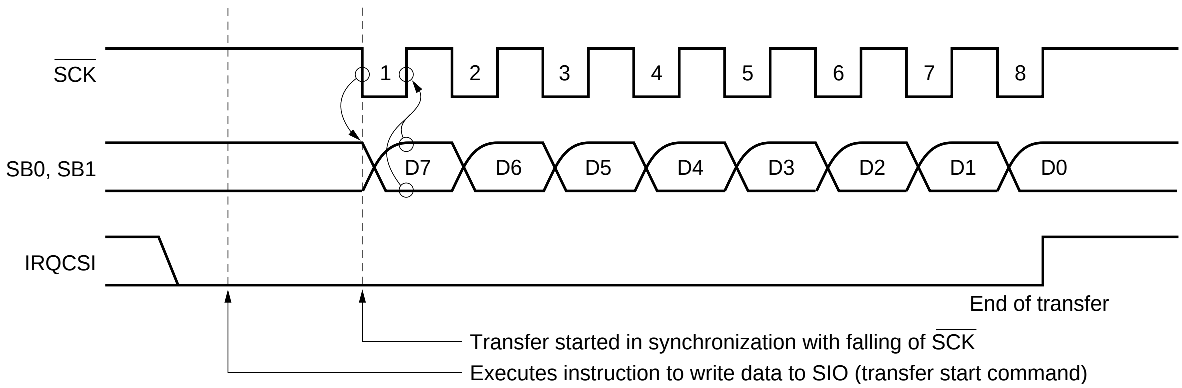
RELT	This bit controls the output trigger of a bus release signal (REL). When this bit is set to 1, the SO latch is set to 1. After that, the RELT bit is automatically cleared to 0.

(2) Communication operation

In the two-line serial I/O mode, data are transmitted or received in 8-bit units. Data are transmitted or received in synchronization with the serial clock, on a bit-by-bit basis.

The shift register performs its shift operation in synchronization with the falling edge of the serial clock ($\overline{SCK}$). The transmit data is retained by the SO latch and output from the SB0/P02 (or SB1/P03) pin with the MSB first. The receive data input from the SB0 pin (or SB1) is latched to the shift register at the rising edge of $\overline{SCK}$. When the 8-bit data has been completely transferred, the shift register is automatically stopped, and an interrupt request flag (IRQCSI) is set.

Fig. 5-50 Timing in 2-Line Serial I/O Mode



The SB0 (or SB1) pin specified as the serial data bus is an N-ch open-drain I/O pin, and must be externally pulled up. Because it is necessary to turn off the N-ch transistor when data is received, write FFH to SIO in advance.

Because the SB0 (or SB1) pin outputs the status of the SO latch, the output status of the SB0 (or SB1) pin can be manipulated by setting the RELT and CMDT bits.

However, do not perform this manipulation during serial transfer.

The output status of the $\overline{SCK}$ pin can be controlled by manipulating the P01 output latch in the output mode (mode of the internal system clock) (refer to **5.6.7 Manipulating $\overline{SCK}$ pin output**).

(3) Selecting serial clock

The serial clock is selected by using the bits 0 and 1 of the serial operation mode register (CSIM). The following three types of serial clocks can be selected:

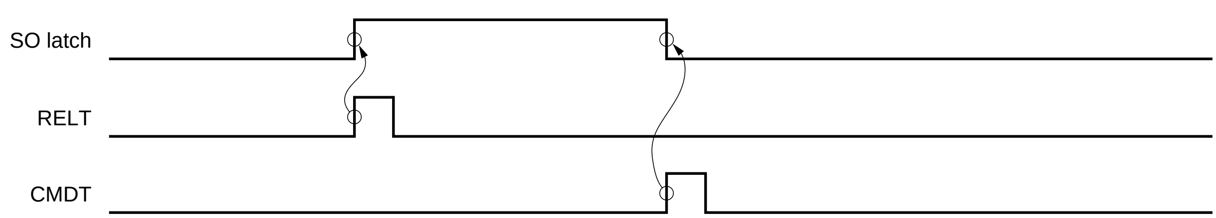
Table 5-10 Selecting Serial Clock and Application (in 2-line serial I/O mode)

Mode Register		Serial Clock		Timing at which shift register can be read/ written and serial transfer can be started	Application
CSIM 1	CSIM 0	Source	Masking Serial Clock		
★ 0	0	External $\overline{SCK}$	Automatically masked at end of transfer of 8-bit data	<1> In operation enable mode (CSIE = 1) <2> If serial clock is masked after 8-bit serial transfer <3> When $\overline{SCK}$ is high	Slave CPU
0	1	TOUT F/F			Serial transfer at any speed
1	0	$f_X/2^6$			Low-speed serial transfer
1	1				

(4) Signals

Fig. 5-51 illustrates the operations of RELT and CMDT.

Fig. 5-51 Operations of RELT and CMDT



(5) Starting transfer

Serial transfer is started when the transfer data is set to the shift register (SIO), if the following two conditions are satisfied:

- Serial interface operation enable/disable bit (CSIE) = 1
- If the internal serial clock is stopped after 8-bit serial transfer or if $\overline{SCK}$ is high

Cautions 1. Transfer is not started even if CSIE is set to “1” after the data has been written to the shift register.

2. Because it is necessary to turn off the N-ch transistor when data is received, write FFH to SIO in advance.

When 8-bit transfer has been completed, the serial transfer is automatically stopped, and an interrupt request flag (IRQCSI) is set.

(6) Error detection

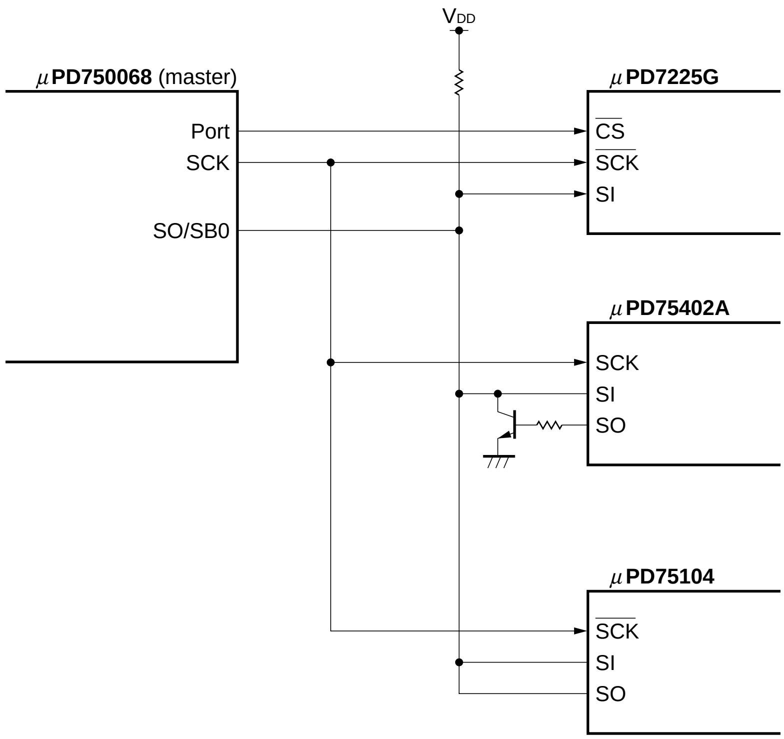
In the two-line serial I/O mode, because the status of the serial bus SB0 or SB1 during transmission is also loaded to the shift register SIO of the device transmitting data, an error can be detected by the following methods:

- **By comparing SIO data before and after transmission**
  If the two data differ from each other, it may be assumed that a transmission error has occurred.

(7) Application of two-line serial I/O mode

The two-line serial I/O mode can be used to connect plural devices by configuring a serial bus.

Example To configure a system by connecting the μPD750068 as the master and μPD75104, μPD75402A, and μPD7225G as slaves



The SI and SO pins of the μPD75104 are connected together. When serial data is not output, the serial operation mode register is manipulated and the output buffer is turned off to release the bus.

Because the SO pin of the μPD75402A cannot go into a high-impedance state, a transistor is connected to the SO pin as shown in the figure, so that the SO pin can be used as an open-collector output pin. When data is input to the μPD75402A, the transistor is turned off by writing 00H to the shift register in advance.

When each microcontroller outputs data is determined in advance.

The serial clock is output by the μPD750068, which is the master. All the slave microcontrollers operate on an external clock.

5.6.7 Manipulating $\overline{SCK}$ pin output

Because the $\overline{SCK}$/P01 pin is provided with an output latch, it can perform static output through software manipulation, in addition to normal clock output.

By manipulating the P01 output latch, a chosen number of $\overline{SCK}$s can be set via software. (The SO/SB0 and SI/SB1 pins are controlled by the RELT and CMDT bits of SBIC.)

The $\overline{SCK}$/P01 pin output is manipulated as follows:

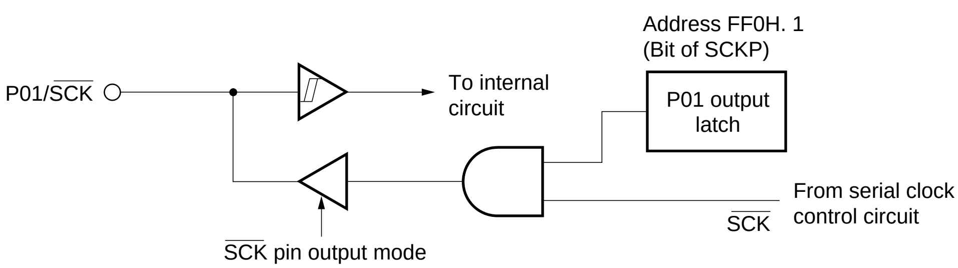
<1> Set the serial operation mode register (CSIM) ($\overline{SCK}$ pin: output mode). While serial transfer is stopped, $\overline{SCK}$ from the serial clock control circuit is 1.

<2> Manipulate the P01 output latch by using a bit manipulation instruction.

Example To output 1 clock to $\overline{SCK}$/P01 via software

```
SEL    MB15             ; or CLR1 MBE
MOV    XA, #10000011B   ; SCK (fx/2^3), output mode
MOV    CSIM, XA
CLR1   0FF0H.1          ; SCK/P01 ← 0
SET1   0FF0H.1          ; SCK/P01 ← 1
```

Fig. 5-52 Configuration of $\overline{SCK}$/P01 Pin



The P01 output latch is mapped to bit 1 of address FF0H. It is set to "1" when the $\overline{RESET}$ signal is asserted.

Cautions 1. Set the P01 output latch to 1 during normal serial transfer.

2. The address of the P01 output latch cannot be specified as "PORT0.1", as shown in the
★ **example below. Whether or not to describe the address (0FF0H.1) directly as the operand of an instruction by SCKP. When the instruction is executed, however, it is necessary that MBE = 0 or (MBE = 1, MBS = 15) has been set in advance.**

Must not be used		**Can be used**	
CLR	**PORT0.1**	**CLR1**	**0FF0H.1**
SET1	**PORT0.1**	**SET1**	**0FF0H.1**
		CLR1	**SCKP**
		SET1	**SCKP**

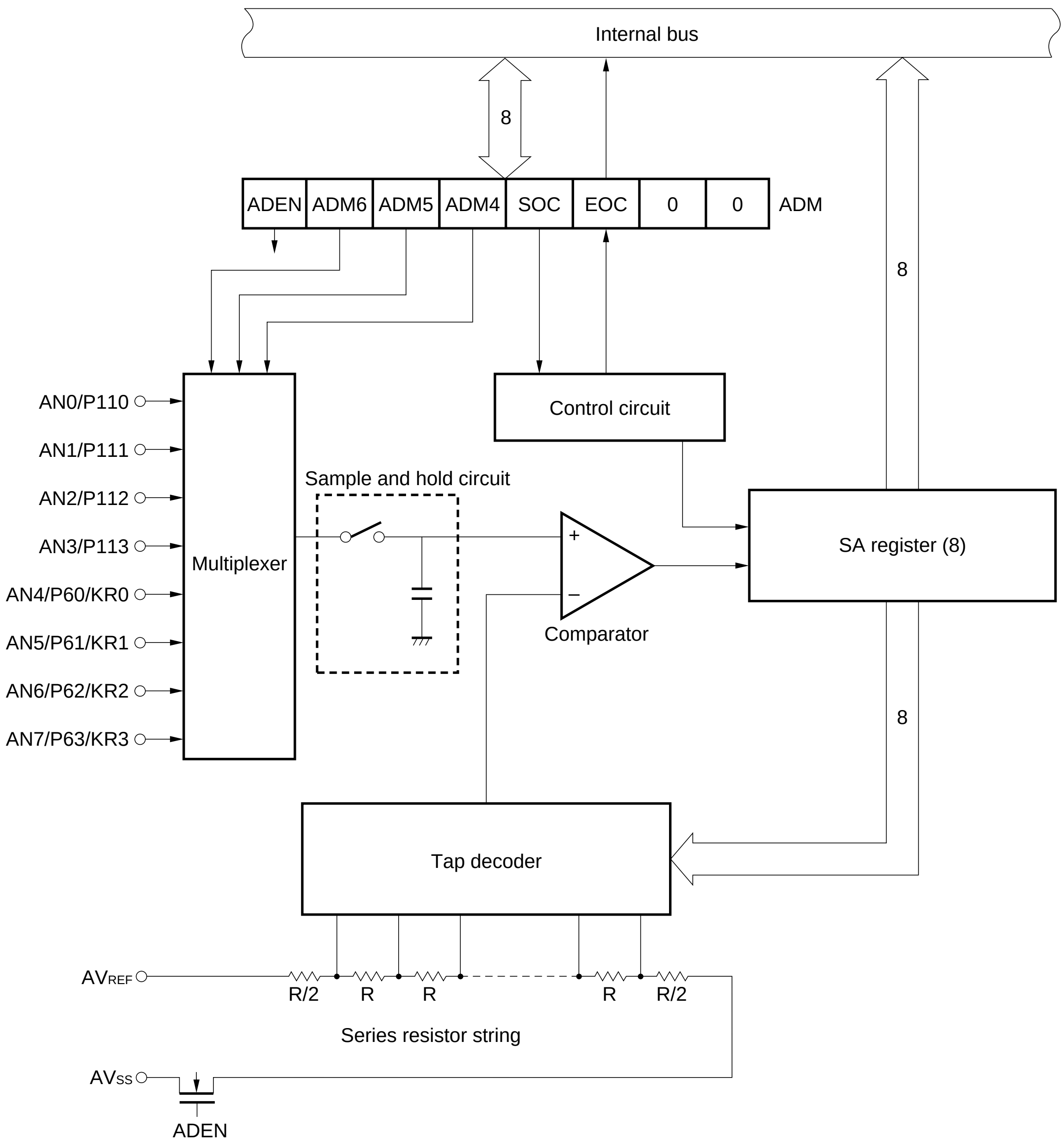
5.7 A/D Converter

The μPD750068 has an analog-to-digital (A/D) converter with eight analog input channels (AN0 through AN7) and 8-bit accuracy. This A/D converter is of the successive approximation type.

5.7.1 Configuration of the A/D converter

Fig. 5-53 shows the configuration of the A/D converter.

★ **Fig. 5-53 Block Diagram of A/D Converter**



(1) Pins of A/D converter

(a) AN0-AN7

These pins are eight channels of analog signal inputs to the A/D converter; they input analog signals to be converted into digital signals.

AN0 through AN3 are multiplexed with P110 through P113, and AN4 through AN7, with P60/KR0 through P63/KR3**Note**.

The A/D converter is provided with a sample and hold circuit. During A/D conversion, the analog input voltage is internally retained.

Note When using AN4 to AN7, the following setting is necessary before starting A/D conversion.

<1> Set port 6 to input mode.

<2> Disconnect the internal pull-up resistor from port 6.

(For details, refer to **5.1 Digital I/O Port**.)

Caution Be sure to keep the input voltages AN0 through AN7 within the rated range. If a voltage higher than V_{DD} or lower than V_{SS} (even within the range of the absolute maximum ratings) is input, the converted value of that channel becomes undefined, and the converted values of the other channels may be adversely affected.

(b) AV_{REF}

This input inputs a reference voltage ot the A/D converter. The signal input to AN0 through AN7 is converted into a digital signal based on the voltage applied across AV_{REF} and AV_{SS}.

(c) AV_{SS}

This is the GND pin of the A/D converter. Always keep this pin at the same potential as V_{SS}.

(2) A/D conversion mode register (ADM)

ADM is an 8-bit register that enables conversion, selects analog input channels, starts conversion, and detects end of conversion. This register is set by an 8-bit manipulation instruction. Bits 2 (EOC), 3 (SOC), and 7 (ADEN) can be manipulated in 1-bit units.

The contents of ADM are initialized to 04H when the $\overline{\text{RESET}}$ signal is asserted (only EOC is set to "1" and the other bits are cleared to "0".)

Fig. 5-54 Format of A/D Conversion Mode Register

Address	7	6	5	4	3	2	1	0	Symbol
FD8H	ADEN	ADM6	ADM5	ADM4	SOC	EOC	0	0	ADM

A/D conversion enable flag

ADEN	0	Does not use A/D converter
	1	Uses A/D converter

Analog channel select bit

ADM6	ADM5	ADM4	Analog channel
0	0	0	AN0
0	0	1	AN1
0	1	0	AN2
0	1	1	AN3
1	0	0	AN4
1	0	1	AN5
1	1	0	AN6
1	1	1	AN7

Conversion start bit

SOC	A/D conversion is started when this bit is set. This bit is automatically cleared after conversion has ended.

End of conversion detection flag

EOC	0	Conversion in progress
	1	End of conversion

Caution A/D conversion is started $2^4/f_X$ seconds (2.67 μs: f_X = 6.0 MHz) after SOC has been set[Note] (refer to 5.7.2 Operation of A/D converter).

Note 3.81 μs at f_X = 4.19 MHz

(3) SA register (SA)

The SA (Successive Approximation) register is an 8-bit register that stores the result of A/D conversion. This register can be read by an 8-bit manipulation instruction. This register is a read-only register and therefore, data cannot be written to it nor can its bits be manipulated.

The contents of this register are initialized to 7FH when the $\overline{\text{RESET}}$ signal is asserted.

Cautions 1. When A/D conversion is started with bit 3 (SOC) of the ADM register set to "1", the results of conversion stored in SA are lost, and the contents of SA are undefined, until a new conversion result is stored to the register.

2. If GND level is input to the AV_{REF} pin or an electric potential between AV_{REF} and V_{DD} is input to an analog input pin, or if A/D conversion is started with ADEN cleared to 0, FFH is stored to SA.

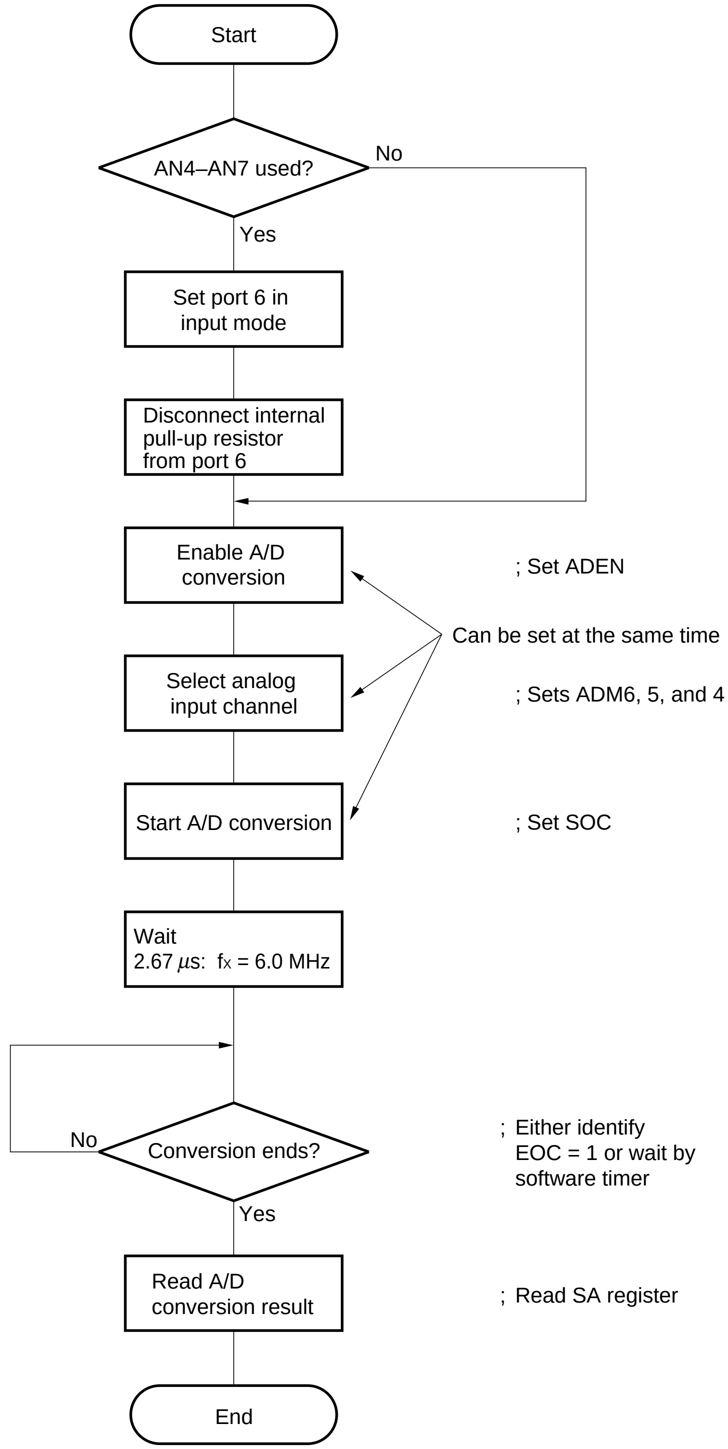
5.7.2 Operation of A/D converter

The input analog signal to be converted to a digital signal is specified by the bits 6, 5, and 4 (ADM6, 5, and 4) of the A/D conversion mode register.

A/D conversion is started when bits 7 (ADEN) and 3 (SOC) of ADM are set to "1" (setting ADEN is necessary only after the $\overline{\text{RESET}}$ signal has been asserted). SOC is automatically cleared to 0 after it has been set. A/D conversion is executed by hardware by means of successive approximations, and the resulting 8-bit data is stored to the SA register. Bit 2 (EOC) of ADM is set to "1" when conversion has ended.

Fig. 5-55 shows the timing chart for A/D conversion.

Operate the A/D converter in the following procedure:



Caution **After SOC has been set, up to $2^4/f_X$ (2.67 μs when f_X = 6.0 MHz)[Note] of delay is generated from the start of A/D conversion until EOC is cleared. Therefore, test EOC after SOC has been set and the time shown in Table 5-11 has elapsed. Table 5-11 also shows the A/D conversion time.**

Note 3.81 μs when f_X = 4.19 MHz

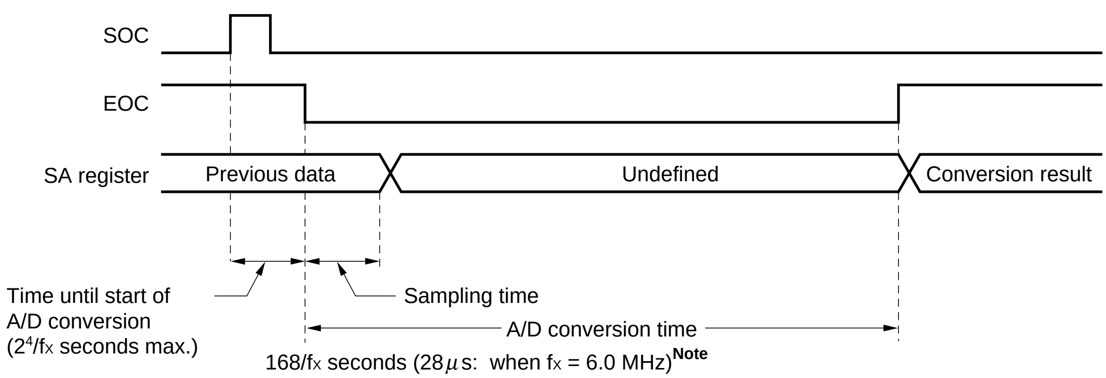
Table 5-11 Setting of SCC and PCC

Setting of SCC, PCC				A/D Conversion Time	Wait Time Until EOC Is Tested after Setting of SOC	Wait Time Until A/D Conversion Ends after Setting of SOC
SCC3	SCC0	PCC1	PCC0			
0	0	0	0	168/fx seconds (28 μs: at fx = 6.0 MHz)Note	No wait	3 machine cycles
		0	1		1 machine cycle	11 machine cycles
		1	0		2 machine cycles	21 machine cycles
		1	1		4 machine cycles	42 machine cycles
0	1	×	×		No wait	No wait
1	×	×	×	Conversion operation stops	–	–

Note 40.1 μs when fx = 4.19 MHz

Remark ×: don't care

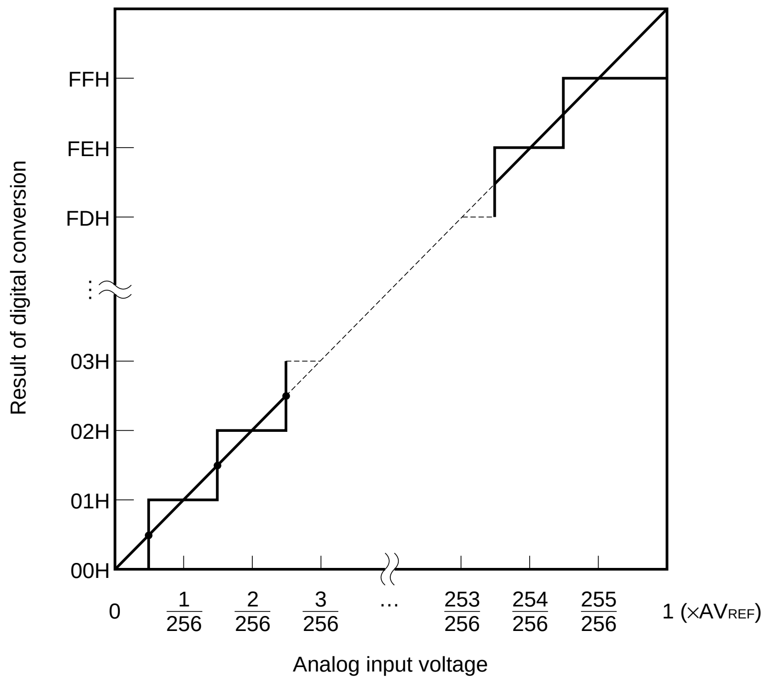
Fig. 5-55 Timing Chart of A/D Conversion



Note 40.1 μs when fx = 4.19 MHz

Fig. 5-56 shows the correspondence between the analog input voltages and the converted 8-bit digital data.

Fig. 5-56 Relation between Analog Input Voltage and Result of A/D Conversion (ideal case)



5.7.3 Notes on standby mode

The A/D converter operates on the main system clock. Therefore, it stops in STOP mode or HALT mode, in which the device operates on the subsystem clock. At this time, however, a current flows into the AV_{REF} pin. To reduce the overall power consumption of the system, this current must be cut off. To do so, disable A/D conversion (ADEN = 0).

5.7.4 Use notes

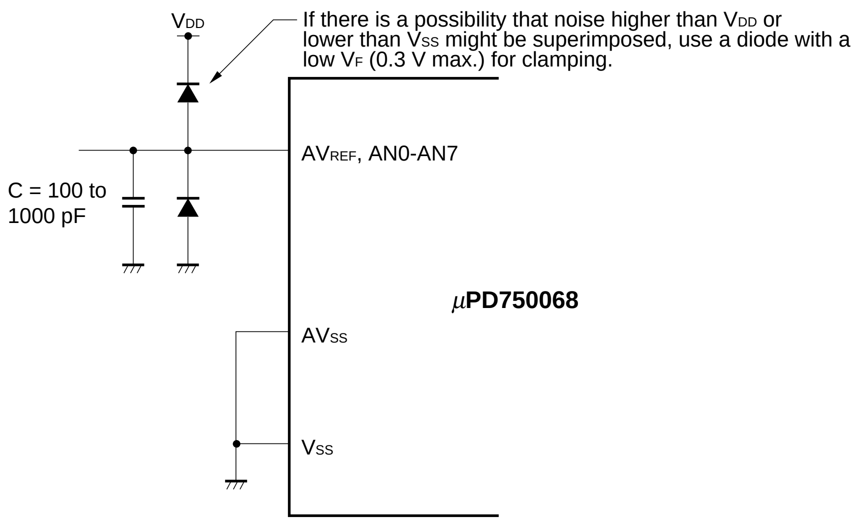
(1) AN0-AN7 input range

Be sure to keep the input voltages AN0 through AN7 within the rated range. If a voltage higher than V_{DD} or lower than V_{SS} (even within the range of the absolute maximum ratings) is input, the converted value of that channel is undefined, and the converted values of the other channels may be adversely affected.

(2) Measures against noise

To maintain 8-bit accuracy, care must be exercised so that noise is not superimposed on the AV_{REF} and AN0 through AN7 pins. The higher the output impedance of the analog signal input source, the heavier the influence of noise. To reduce noise, therefore, it is recommended that C be externally connected as shown in Fig. 5-57.

Fig. 5-57 Handling of Analog Input Pins



(3) AN0-AN3 pins

Analog input pins AN0 to AN3 also function as input port (PORT 11) pins. When performing A/D conversion with one of AN0 to AN3 selected, do not execute a PORT11 input instruction during conversion. Otherwise the conversion accuracy may decrease.

If a digital pulse is applied to a pin adjacent to the pin whose input signal is being converted, the expected result may not be obtained due to coupling noise. Therefore, do not apply a digital pulse to such a pin.

(4) AN4-AN7 pins

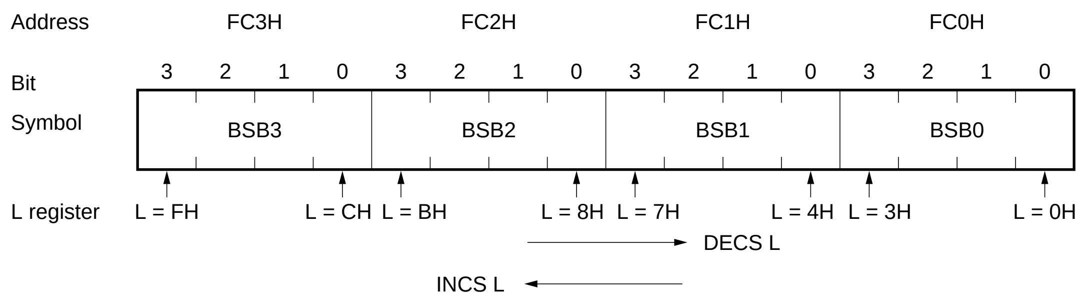
Analog input pins AN4 to AN7 also function as input/output (PORT 6) pins and pins KR0 to KR3. When performing A/D conversion with one of AN4 to AN7 selected, set PORT 6 to the input mode, first. In this case, do not execute PORT 6 input/output instructions during conversion. Furthermore, do not specify connection of internal pull-up resistors. Otherwise the conversion accuracy may decrease.

5.8 Bit Sequential Buffer ... 16 bits

The bit sequential buffer (BSB) is a special data memory used for bit manipulation. It can manipulate bits by sequentially changing the address and bit specification. Therefore, this buffer is useful for processing data with a long bit length in bit units.

This data memory is configured of 16 bits and can be addressed by a bit manipulation instruction in the pmem.@L addressing mode. Its bits can be indirectly specified by the L register. The processing can be executed by only incrementing or decrementing the L register in a program loop and by moving the specified bit sequentially.

Fig. 5-58 Format of Bit Sequential Buffer



Remarks 1. The specified bit is moved according to the L register in the pmem.@L addressing mode.

2. BSB can be manipulated at any time in the pmem.@L addressing mode, regardless of the specification by MBE and MBS.

The data in this buffer can also be manipulated even in direct addressing mode. By using 1-, 4-, or 8-bit direct addressing mode and pmem.@L addressing mode in combination, 1-bit data can be successively input or output. To manipulate BSB in 8-bit units, the higher and lower 8 bits are manipulated by specifying BSB0 and BSB2.

Example For serial output of the 16-bit data of BUFF1, 2 from bit 0 of port 3

```
        CLR1    MBE
        MOV     XA, BUFF1
        MOV     BSB0, XA      ; Sets BSB0, 1
        MOV     XA, BUFF2
        MOV     BSB2, XA      ; Sets BSB2, 3
        MOV     L, #0
LOOP0:  SKT     BSB0, @L      ; Tests specified bit of BSB
        BR      LOOP1
        NOP                   ; Dummy (to adjust timing)
        SET1    PORT3.0       ; Sets bit 0 of port 3
        BR      LOOP2
LOOP1:  CLR1    PORT3.0       ; Clears bit 0 of port 3
        NOP                   ; Dummy (to adjust timing)
        NOP
LOOP2:  INCS    L             ; L ← L + 1
        BR      LOOP0
        RET
```

[MEMO]

CHAPTER 6 INTERRUPT AND TEST FUNCTIONS

The μPD750068 has seven vector interrupt sources and two test inputs that can be used for various applications.

The interrupt control circuit of the μPD750068 has unique features and can process interrupts at extremely high speed.

(1) Interrupt function

(a) Hardware-controlled vector interrupt functions that can control acknowledgment of an interrupt by using an interrupt enable flag (IE×××) and interrupt master enable flag (IME)
(b) Any interrupt start address can be set.
(c) Interrupt nesting function that can specify priority by using an interrupt priority select register (IPS)
(d) Test function of interrupt request flag (IRQ×××) (Occurrence of an interrupt can be checked by software.)
(e) Releases standby mode (The interrupt that is used to release the standby mode can be selected by the interrupt enable flag.)

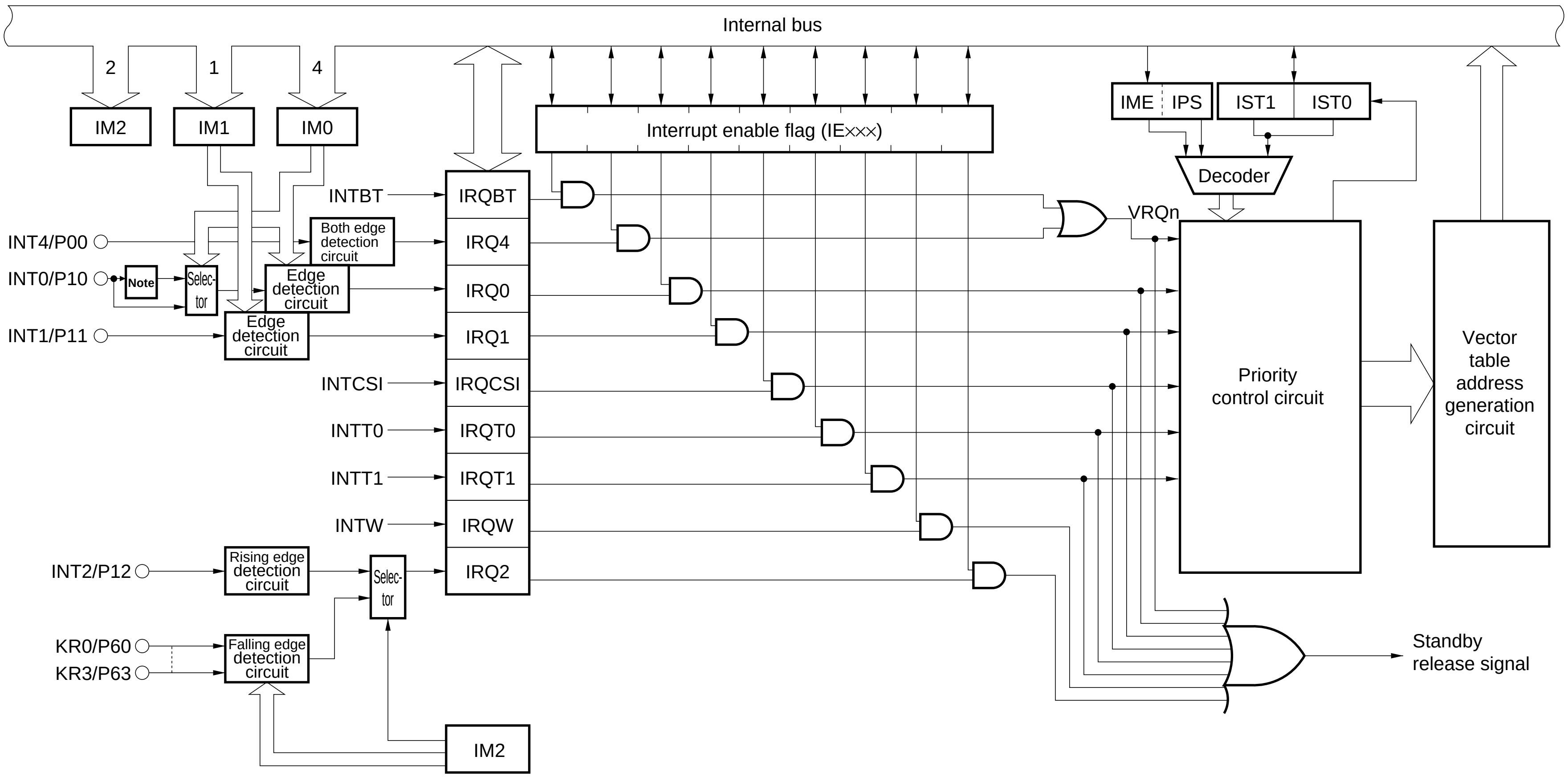
(2) Test function

(a) Checks setting of a test request flag (IRQ×××) via software
(b) Releases standby mode (The test source that releases the standby mode can be selected by the test enable flag.)

6.1 Configuration of Interrupt Control Circuit

The interrupt control circuit is configured as shown in Fig. 6-1, and each hardware unit is mapped to the data memory space.

Fig. 6-1 Block Diagram of Interrupt Control Circuit



Note Noise rejection circuit (The standby mode cannot be released when the noise rejection circuit is selected.)

6.2 Types of Interrupt Sources and Vector Table

The μPD750068 has the following seven interrupt sources and nesting of interrupts can be controlled by software.

Table 6-1 Types of Interrupt Sources

Interrupt Source		Internal/External	Interrupt Priority[Note]	Vector Interrupt Request Signal (vector table address)
INBT	(reference time interval signal from basic interval timer/watchdog timer)	Internal	1	VRQ1 (0002H)
INT4	(detection of both rising and falling edges)	External		
INT0	(rising edge or falling edge is selected)	External	2	VRQ2 (0004H)
INT1		External	3	VRQ3 (0006H)
INTCSI	(serial data transfer end signal)	Internal	4	VRQ4 (0008H)
INTT0	(signal indicating coincidence between count register of timer/event counter 0 and modulo register)	Internal	5	VRQ5 (000AH)
INTT1	(signal indicating coincidence between count register of timer/event counter 1 and modulo register)	Internal	6	VRQ6 (000CH)

Note If two or more interrupts occur at the same time, the interrupts are processed according to this priority.

Fig. 6-2 Interrupt Vector Table

Address			
0002H	MBE	RBE	INTBT/INT4 start address (higher 6 bits)
			INTBT/INT4 start address (lower 8 bits)
0004H	MBE	RBE	INT0 start address (higher 6 bits)
			INT0 start address (lower 8 bits)
0006H	MBE	RBE	INT1 start address (higher 6 bits)
			INT1 start address (lower 8 bits)
0008H	MBE	RBE	INTCSI start address (higher 6 bits)
			INTCSI start address (lower 8 bits)
000AH	MBE	RBE	INTT0 start address (higher 6 bits)
			INTT0 start address (lower 8 bits)
000CH	MBE	RBE	INTT1 start address (higher 6 bits)
			INTT1 start address (lower 8 bits)

The priority column in Table 6-1 indicates the priority according to which interrupts are executed if two or more interrupts occur at the same time, or if two or more interrupt requests are kept pending.

Write the start address of interrupt processing to the vector table , and the set values of MBE and RBE during interrupt processing. The vector table is set by using an assembler directive (VENTn: n=1-6).

Example Setting of vector table of INTBT/INT4

```
VENT1     MBE=0, RBE=0,GOTOBT
  ↑         ↑      ↑     ↑
 <1>       <2>    <3>   <4>
```

<1> Vector table of address 0002

<2> Setting of MBE in interrupt processing routine

<3> Setting of RBE in interrupt processing routine

<4> Symbol indicating start address of interrupt processing routine

★ **Caution The contents described in the operand of the VENTn (n = 1-6) instruction (MBE, RBE, start address) are stored in the vector table address 2n.**

Example Setting of vector tables of INTBT/INT4 and INTT0

```
VENT1    MBE=0, RBE=0, GOTOBT; INTBT/INT4 start address
VENT5    MBE=0, RBE=1, GOTOT0; INTT0 start address
```

6.3 Hardware Controlling Interrupt Function

(1) Interrupt request flag and interrupt enable flag

The μPD750068 has the following seven interrupt request flags (IRQ×××) corresponding to the respective interrupt sources:

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 0 interrupt request flag (IRQT0)
Timer/event counter 1 interrupt request flag (IRQT1)

Each interrupt request flag is set to “1” when the corresponding interrupt request is generated, and is automatically cleared to “0” when the interrupt processing is executed. However, because IRQBT and IRQ4 share the vector address, these flags are cleared differently from the other flags (refer to **6.6 Processing of Interrupts Sharing Vector Address**).

The μPD750068 also has seven interrupt enable flags (IE×××) corresponding to the respective interrupt request flags.

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 0 interrupt enable flag (IET0)
Timer/event counter 1 interrupt enable flag (IET1)

The interrupt enable flag enables the corresponding interrupt when it is “1”, and disables the interrupt when it is “0”.

If an interrupt request flag is set and the corresponding interrupt enable flag enables the interrupt, a vector interrupt (VRQn: n=1-6) occurs. This signal is also used to release the standby mode.

The interrupt request flags and interrupt enable flags are manipulated by a bit manipulation or 4-bit manipulation instruction. When a bit manipulation instruction is used, the flags can be directly manipulated, regardless of the setting of MBE. The interrupt enable flags are manipulated by the EI IE××× and DI IE××× instructions. To test an interrupt request flag, the SKTCLR instruction is usually used.

Example

```
EI       IE0       ; Enables INT0
DI       IE1       ; Disables INT1
SKTCLR   IRQCSI    ; Skips and clears if IRQCSI is 1
```

When an interrupt request flag is set by an instruction, a vector interrupt is executed even if an interrupt does not occur, in the same manner as when the interrupt occurs.

The interrupt request flags and interrupt enable flags are cleared to “0” when the $\overline{\text{RESET}}$ signal is asserted, disabling all the interrupts.

Table 6-2 Signals Setting Interrupt Request Flags

Interrupt Request Flag	Signal Setting Interrupt Request Flag	Interrupt Enable Flag
IRQBT	Set by reference time interval signal from basic interval/watchdog timer	IEBT
IRQ4	Also set by detection of both rising and falling edges of INT4/P00 pin input signal	IE4
IRQ0	Set by detection of edge of INT0/P10 pin input signal. Edge to be detected is selected by INT0 edge detection mode register (IM0)	IE0
IRQ1	Set by detection of edge of INT1/P11 pin input signal. Edge to be detected is selected by INT1 edge detection mode register (IM1)	IE1
IRQCSI	Set by serial data transfer end signal from serial interface	IECSI
IRQT0	Set by coincidence signal from timer/event counter 0	IET0
IRQT1	Set by coincidence signal from timer/event counter 1	IET1

(2) Interrupt priority select register (IPS)

The interrupt priority select register selects an interrupt with the higher priority that can be nested. The lower 3 bits of this register are used for this purpose.

Bit 3 is an interrupt master enable flag (IME) that enables or disables all the interrupts.

IPS is set by a 4-bit memory manipulation instruction, but bit 3 is set or reset by the EI or DI instruction.

To change the contents of the lower 3 bits of IPS, the interrupt must be disabled (IME = 0).

Example

```
DI                    ; Disables interrupt
CLR1    MBE
MOV     A, #1011B
MOV     IPS, A        ; Gives higher priority to INT1 and enables interrupt
```

When the $\overline{\text{RESET}}$ signal is asserted, all the bits of this register are cleared to "0".

★ **Caution When setting the IPS, be sure to disable interrupt before start setting.**

Fig. 6-3 Interrupt Priority Select Register

Address	3	2	1	0	Symbol
FB2H	IPS3	IPS2	IPS1	IPS0	IPS

Selects interrupt with higher priority

IPS2	IPS1	IPS0		
0	0	0	Does not give high priority to any interrupt	
0	0	1	VRQ1 (INTBT/INT4)	Gives high priority to interrupts shown on left
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	Setting prohibited	

Interrupt master enable flag (IME)

IPS3	
0	Disables all interrupts and vector interrupt is not started
1	Interrupt is enabled or disabled by corresponding interrupt enable flag

(3) Hardware of INT0, INT1, and INT4

(a) Fig. 6-4 (a) shows the configuration of INT0, which is an external interrupt input that can be detected at the rising or falling edge depending on specification.
INT0 also has a noise rejection function which uses a sampling clock (refer to **Fig. 6-5 I/O Timing of Noise Rejection Circuit**). The noise rejection circuit rejects a pulse having a width narrower than 2 cyclesNote of the sampling clock as a noise. However, a pulse having a width wider than one cycle of the sampling clock may be accepted as the interrupt signal depending on the timing of sampling (refer to **Fig. 6-5 <2> (a)**). A pulse having a width wider than two cycles of the sampling clock is always accepted as the interrupt without fail.
INT0 has two sampling clocks for selection: Φ and $f_X/64$. These sampling clocks are selected by using bit 3 (IM03) of the INT0 edge detection mode register (IM0) (refer to **Fig. 6-6 (a)**).
The edge of INT0 to be detected is selected by using bits 0 and 1 of IM0.
Fig. 6-6 (a) shows the format of IM0. This register is manipulated by a 4-bit manipulation instruction. All the bits of this register are cleared to "0" when the RESET signal is asserted, and the rising edge of INT0 is specified to be detected.

Note When sampling clock is Φ : $2t_{CY}$
When sampling clock is $f_X/64$: $128/f_X$

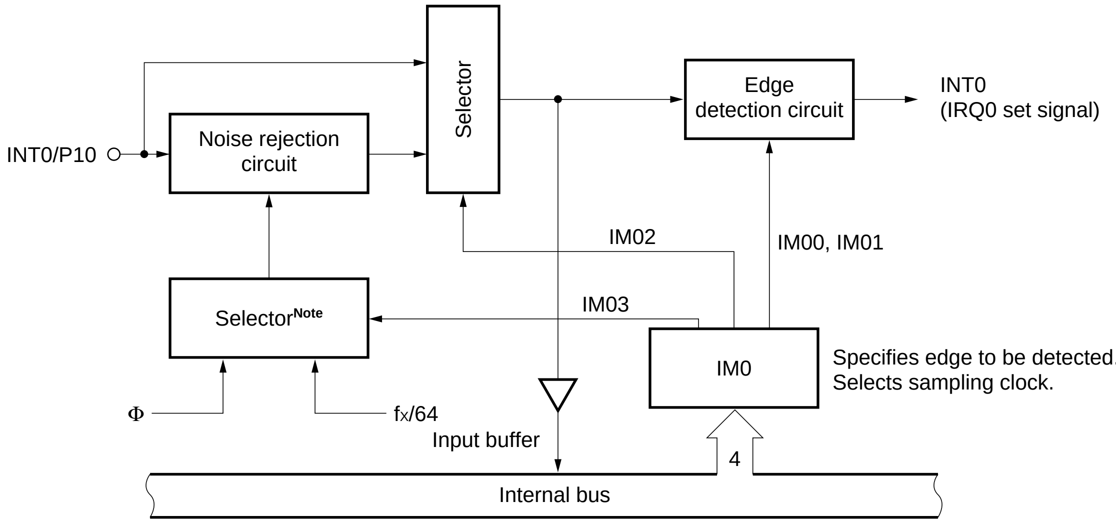
Cautions 1. Even when a signal is input to the INT0/P10 pin in the port mode, it is input through the noise rejection circuit. Therefore, input a signal having a width wider than two cycles of the sampling clock.
2. When the noise rejection circuit is selected (by clearing IM02 to 0), INT0 does not operate in the standby mode because it performs sampling by using the clock (the noise rejection circuit does not operate when CPU clock Φ is not supplied) . Therefore, do not select the noise rejection circuit if it is necessary to release the standby mode by INT0 (set IM02 to 1). ★

(b) Fig. 6-4 (b) shows the configuration of INT1, which is an external interrupt input that can be specified for detection at the rising or falling edge.
The edge to be detected is selected by using the INT1 edge detection mode register (IM1).
Fig. 6-6 (b) shows the format of IM1. This register is manipulated by a 4-bit manipulation instruction. All the bits of this register are cleared to 0 when the RESET signal is asserted, and the rising edge is specified for detection.

(c) Fig. 6-4 (c) shows the configuration of INT4, which is an external interrupt input that can be specified for detection at both the rising and falling edges.

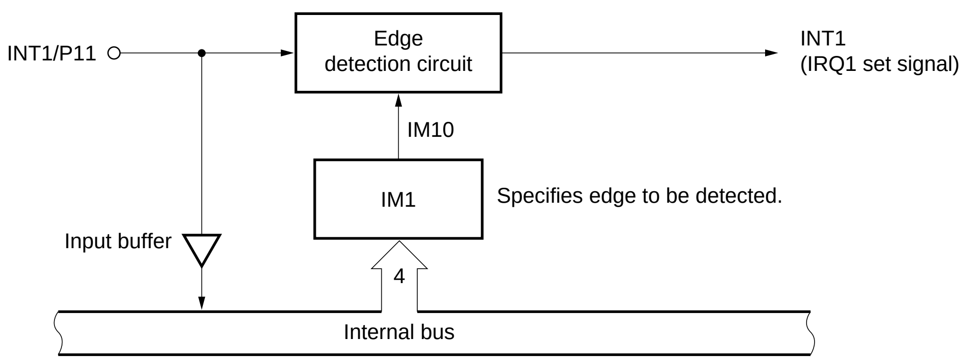
Fig. 6-4 Configuration of INT0, INT1, and INT4

(a) Hardware of INT0



★ **Note** HALT mode by INT0 cannot be released even if fX/64 is selected.

(b) Hardware of INT1



(c) Hardware of INT4

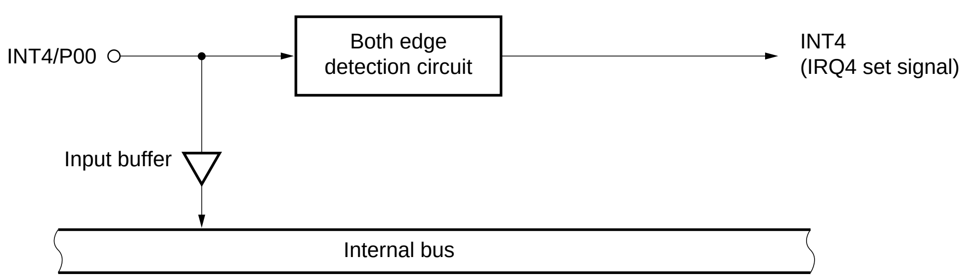
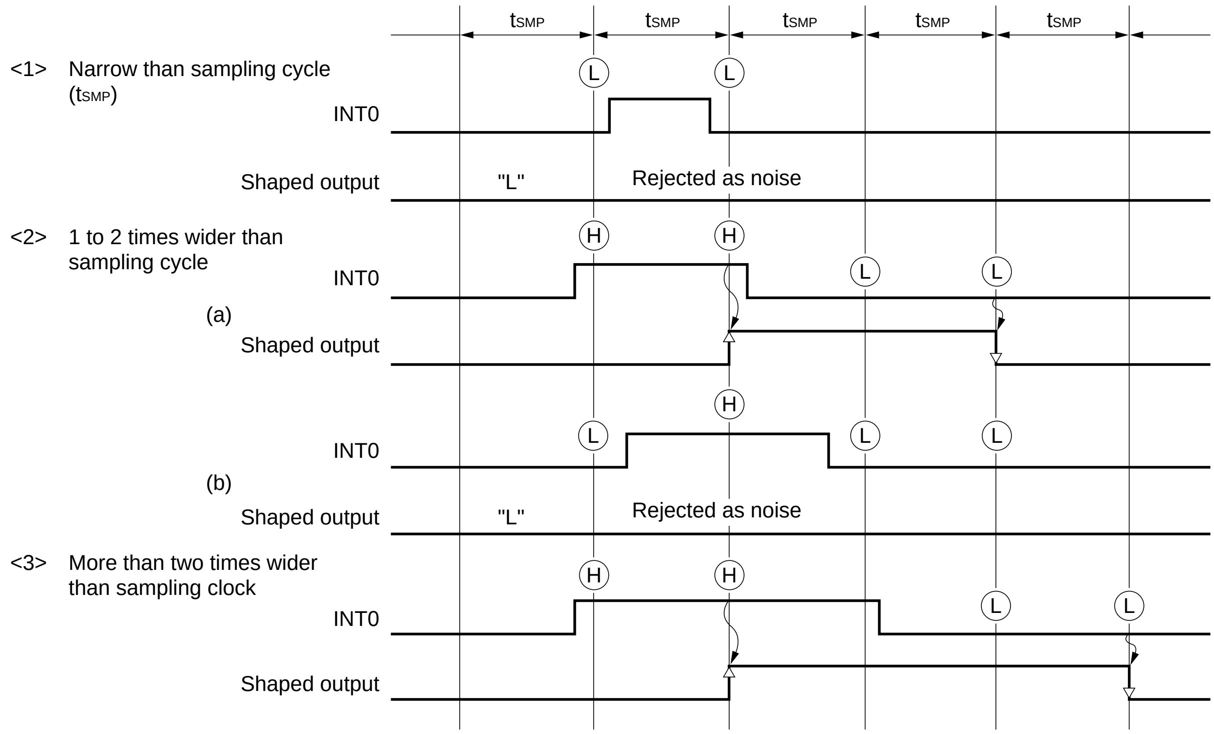


Fig. 6-5 I/O Timing of Noise Rejection Circuit



Remark $t_{SMP} = t_{CY}$ or $64/f_X$

Fig. 6-6 Format of Edge Detection Mode Register

(a) INT0 edge detection mode register (IM0)

Address	3	2	1	0	Symbol
FB4H	IM03	IM02	IM01	IM00	IM0

IM01	IM00	Specifies edge to be detected
0	0	Rising edge
0	1	Falling edge
1	0	Both rising and falling edges
1	1	Ignored (interrupt request flag is not set)

IM02	Noise rejection circuit select bit	Samp]ing	Standby re]ease
0	Selects noise rejection circuit	Enabled	Disabled
1	Does not select noise rejection circuit	Disabled	Enabled

IM03	Sampling clock
0	Φ (0.67 μs, 1.33 μs, 2.67 μs, 10.7 μs at 6.00 MHz)
1	f_X/64 (10.7 μs at 6.00 MHz)

(b) INT1 edge detection mode register (IM1)

Address	3	2	1	0	Symbol
FB5H	0	0	0	IM10	IM1

IM10	Specifies edge to be detected
0	Rising edge
1	Falling edge

Caution When the contents of the edge detection mode register are changed, the interrupt request flag may be set. Therefore, you should disable interrupts before changing the contents of the mode register. Then, clear the interrupt request flag by using the CLR1 instruction to enable the interrupts. If the contents of IM0 are changed and the sampling clock of f_X/64 is selected, clear the interrupt request flag after 16 machine cycles after the contents of the mode register have been changed.

(4) Interrupt status flag

The interrupt status flags (IST0 and IST1) indicate the status of the processing currently executed by the CPU and are included in PSW.

The interrupt priority control circuit controls nesting of interrupts according to the contents of these flags as shown in Table 6-3.

Because IST0 and IST1 can be changed by using a 4-bit or bit manipulation instruction, interrupts can be nested with the status under execution changed. IST0 and IST1 can be manipulated in 1-bit units regardless of the setting of MBE.

Before manipulating IST0 and IST1, be sure to execute the DI instruction to disable the interrupt. Execute the EI instruction after manipulating the flags to enable the interrupt.

IST1 and IST0 are saved to the stack memory along with the other flags of PSW when an interrupt is acknowledged, and their statuses are automatically changed one higher. When the RETI instruction is executed, the original values of IST1 and IST0 are restored.

The contents of these flags are cleared to "0" when the $\overline{\text{RESET}}$ signal is asserted.

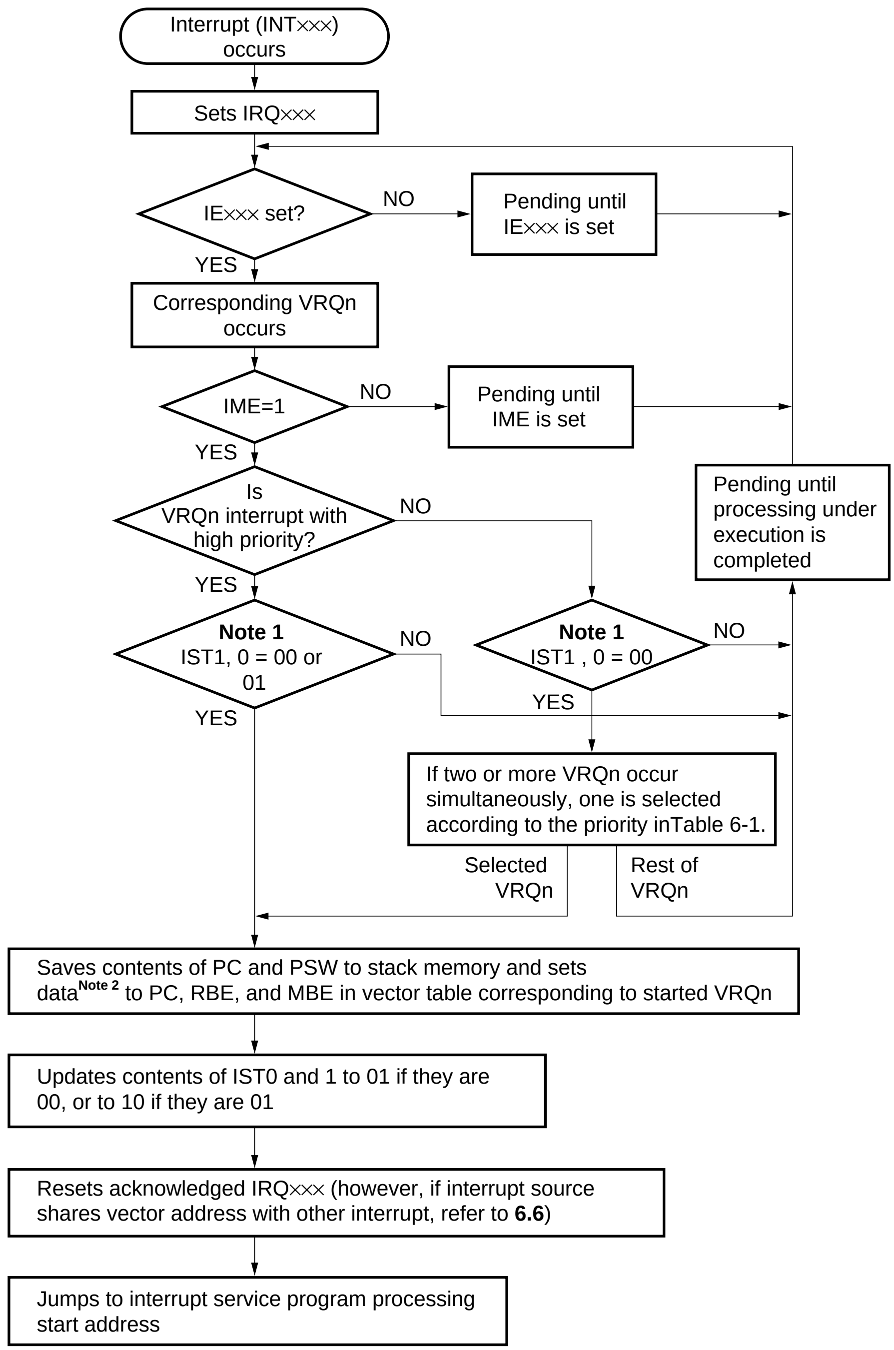
Table 6-3 IST1 and IST0 and Interrupt Processing Status

IST1	IST0	Status of Processing under Execution	Processing by CPU	Interrupt Request That Can Be Acknowledged	After Interrupt Acknowledged	
					IST1	IST0
0	0	Status 0	Executes normal Program	All interrupts can be acknowledged	0	1
0	1	Status 1	Processes interrupt with low or high	Interrupt with high priority can be acknowledged	1	0
1	0	Status 2	Processes interrupt with high priority	Acknowledging all interrupts is disabled	–	–
1	1	Setting prohibited				

6.4 Interrupt Sequence

When an interrupt occurs, it is processed in the procedure illustrated below.

Fig. 6-7 Interrupt Processing Sequence



Notes **1.** IST1 and 0: interrupt status flags (bits 3 and 2 of PSW; Refer to **Table 6-3**.)

2. Each vector table stores the start address of an interrupt service program and the preset values of MBE and RBE when the interrupt is started.

6.5 Nesting Control of Interrupts

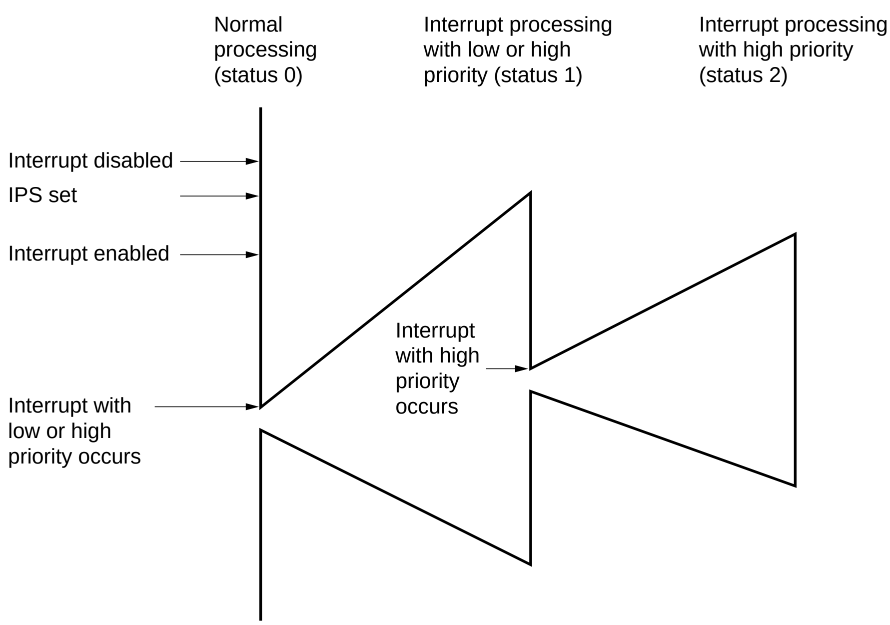
The μPD750068 can nest interrupts by the following two methods:

(1) Nesting with interrupt having high priority specified

This method is the standard nesting method of the μPD750068. One interrupt source is selected and nested. An interrupt with the higher priority specified by the interrupt priority select register (IPS) can occur when the status of the processing under execution is 0 or 1, and the other interrupts (interrupts with the lower priority) can occur when the status is 0 (refer to **Fig. 6-8** and **Table 6-3**).

Therefore, if you use this method when you wish to nest only one interrupt, operations such as enabling and disabling interrupts while the interrupt is processed need not to be performed, and the nesting level can be kept to 2.

Fig. 6-8 Nesting of Interrupt with High Priority

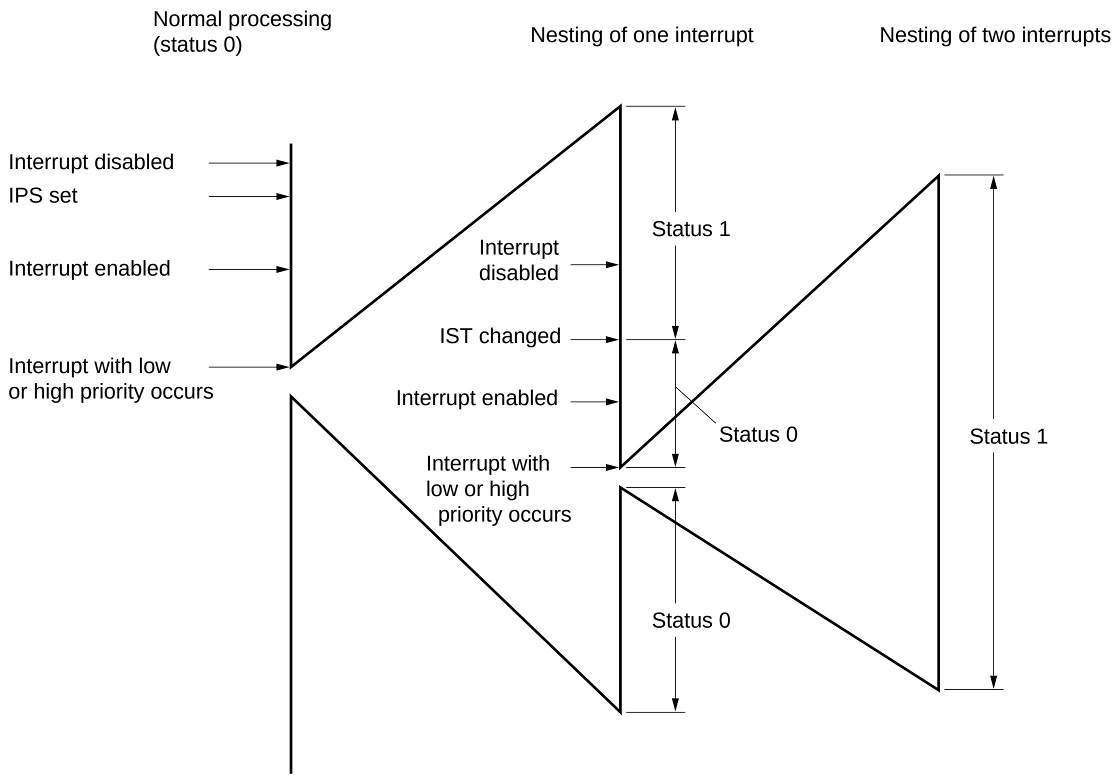


(2) Nesting by changing interrupt status flags

Nesting can be implemented if the interrupt status flags are changed by program. In other words, nesting is enabled when IST1 and IST0 are cleared to "0, 0" by an interrupt processing program, and status 0 is set. This method is used to nest two or more interrupts, or to implement nesting level 3 or higher.

Before changing IST1 and IST0, disable interrupts by using the DI instruction.

Fig. 6-9 Interrupt Nesting by Changing Interrupt Status Flag



6.6 Processing of Interrupts Sharing Vector Address

Because interrupt sources INTBT and INT4 share vector tables, you should select one or both of the interrupt sources in the following way:

(1) To use one interrupt

Of the two interrupt sources sharing a vector table, set the interrupt enable flag of the necessary interrupt source to “1”, and clear the interrupt enable flag of the other interrupt source to “0”. In this case, an interrupt request is generated by the interrupt source that is enabled (IE××× = 1). When the interrupt is acknowledged, the interrupt request flag is reset.

(2) To use both interrupts

Set the interrupt enable flags of both the interrupt sources to “1”. In this case, the interrupt request flags of the two interrupt sources are ORed.
In this case, if an interrupt request is acknowledged when one or both the interrupt flags are set, the interrupt request flags of both the interrupt sources are not reset.
Therefore, it is necessary to identify which interrupt source has generated the interrupt by using an interrupt service routine. This can be done by checking the interrupt request flags by executing the SKTCLR instruction at the beginning of the interrupt service routine.
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 the other interrupt is identified differently, depending whether it has the higher priority, as shown in Table 6-4.

Table 6-4 Identifying Interrupt Sharing Vector 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 of INT4
         EI                  ⎪
         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

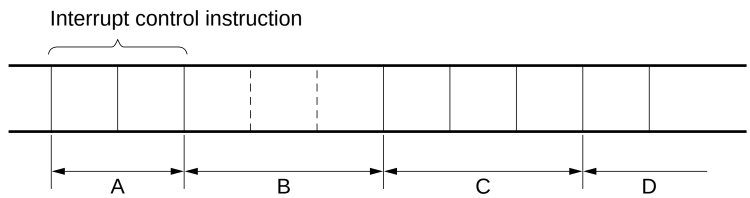
```
         SKTCLR    IRQ4       ;  IRQ4=1?
         BR        VSUBBT    ⎫
            :                ⎪
                             ⎬ Processing routine of INT4
         RETI                ⎪
            :                ⎭

VSUBBT:     CLR1   IRQBT     ⎫
            :                ⎬ Processing routine of INTBT
         RETI                ⎭
```

6.7 Machine Cycles until Interrupt Processing

The number of machine cycles required from when an interrupt request flag (IRQ×××) has been set until the interrupt routine is executed is as follows:

(1) If IRQ××× is set while interrupt control instruction is executed

If IRQ××× is set while an interrupt control instruction is executed, the next one instruction is executed. Then three machine cycles of interrupt processing is performed and the interrupt routine is executed.



A: Sets IRQ×××
B: Executes next one instruction (1 to 3 machine cycles; differs depending on instruction)
C: Interrupt processing (3 machine cycles)
D: Executes interrupt routine

Cautions 1. If two or more interrupt control instructions are successively executed, the one instruction following the interrupt control instruction executed last is executed, three machine cycles of interrupt processing is performed, and then the interrupt routine is executed.

2. If the DI instruction is executed when or after IRQ××× is set (A in the above figure), the interrupt request corresponding to IRQ××× that has been set is kept pending until the EI instruction is executed next time.

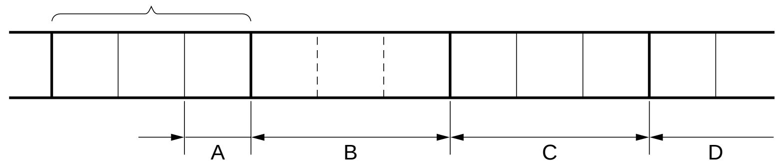
Remarks 1. An interrupt control instruction manipulates the hardware units related to interrupt (address FB×H of the data memory). The EI and DI instructions are interrupt control instructions.

2. The three machine cycles of interrupt processing is the time required to manipulate the stack which will be manipulated when an interrupt is acknowledged.

(2) If IRQ××× is set while instruction other than (1) is executed

(a) If IRQ××× is set at the last machine cycle of the instruction under execution

In this case, the one instruction following the instruction under execution is executed, three machine cycles of interrupt processing is performed, and then the interrupt routine is executed.



A: Sets IRQ×××

B: Executes next one instruction (1 to 3 machine cycles; differs depending on instruction)

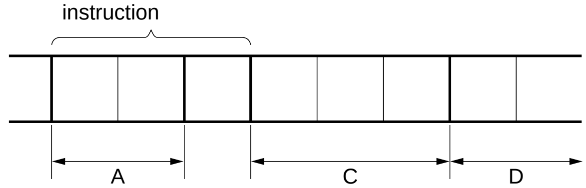
C: Interrupt processing (3 machine cycles)

D: Executes interrupt routine

Caution If the next instruction is an interrupt control instruction, the one instruction following the interrupt control instruction executed last is executed, three machine cycles of interrupt processing is performed, and then the interrupt routine is executed. If the DI instruction is executed after IRQ××× has been set, the interrupt request corresponding to the set IRQ××× is kept pending.

(b) If IRQ××× is set before the last machine cycle of the instruction under execution

In this case, three machine cycles of processing is performed after execution of the current instruction, and then the interrupt routine is executed.



A: Sets IRQn

B: Interrupt processing (3 machine cycles)

C: Executes interrupt routine

6.8 Effective Usage of Interrupts

Use the interrupt function effectively as follows:

(1) Clear MBE to 0 in interrupt processing routine.

If the memory used in the interrupt routine is allocated to addresses 00H through 7FH, and MBE is cleared to 0 by the interrupt vector table, you can program without having to be aware of the memory bank.

If it is necessary to use memory bank 1, save the memory bank select register by using the PUSH BS instruction, and then select 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 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 BR instruction, and set RBS to 1 to select register bank 1.

(3) Use the software interrupt for debugging.

Even if an interrupt request flag is set by an instruction, the same operation as when an interrupt occurs is performed. For debugging of an irregular interrupt or debugging when two or more interrupts occur at the same time, the efficiency can be increased by using an instruction to set the interrupt flag.

6.9 Application of Interrupt

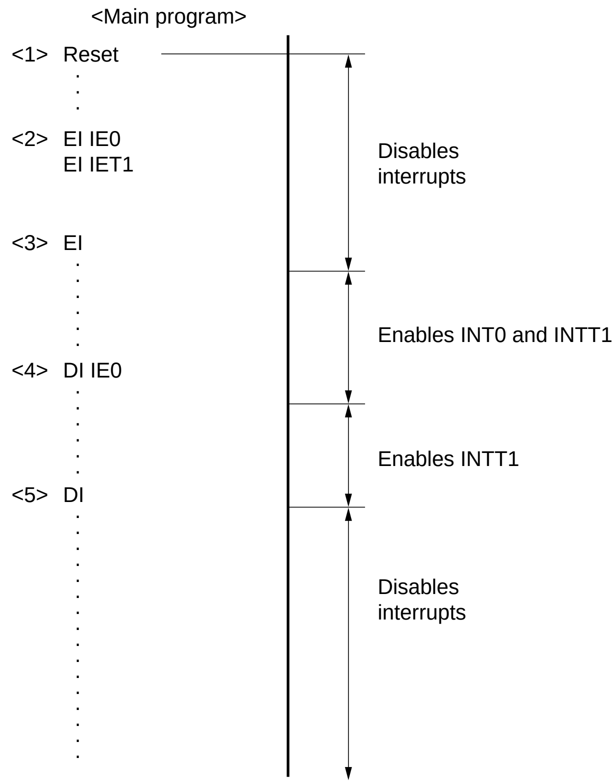
To use the interrupt function, first set as follows by the main program:

(a) Set the interrupt enable flag of the interrupt used (by using the EI IE××× instruction).
(b) To use INT0 or INT1, select the active edge (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 (by using the EI instruction).

In the interrupt program, 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) Enabling or disabling interrupt



<1> All the interrupts are disabled by the $\overline{\text{RESET}}$ signal.

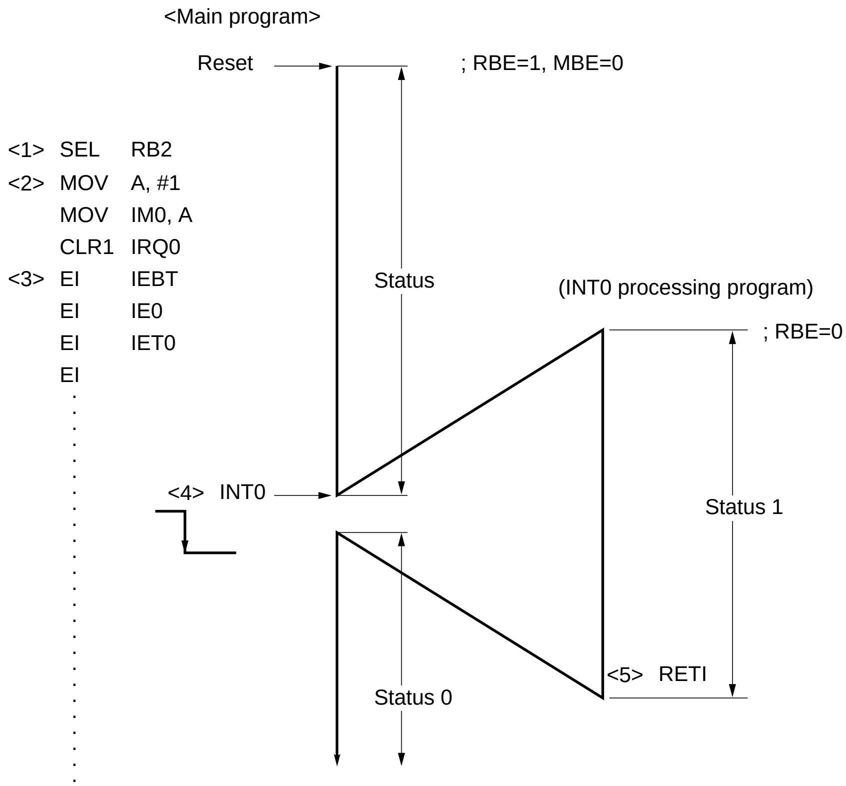
<2> An interrupt enable flag is set by the EI IE××× instruction. At this stage, the interrupts are still disabled.

<3> The interrupt master enable flag is set by the EI instruction. INT0 and INTT1 are enabled at this time.

<4> The interrupt enable flag is cleared by the DI IE××× instruction, and INT0 is disabled.

<5> All the interrupts are disabled by the DI instruction.

(2) Example of using INTBT and INT0 (falling edge active). Not nested (all interrupts have higher priority)



<1> All the interrupts are disabled by the $\overline{\text{RESET}}$ signal and status 0 is set.
RBE = 1 is specified by the reset vector table. The SEL SB2 instruction uses register banks 2 and 3.

<2> INT0 is specified to be active at the falling edge.

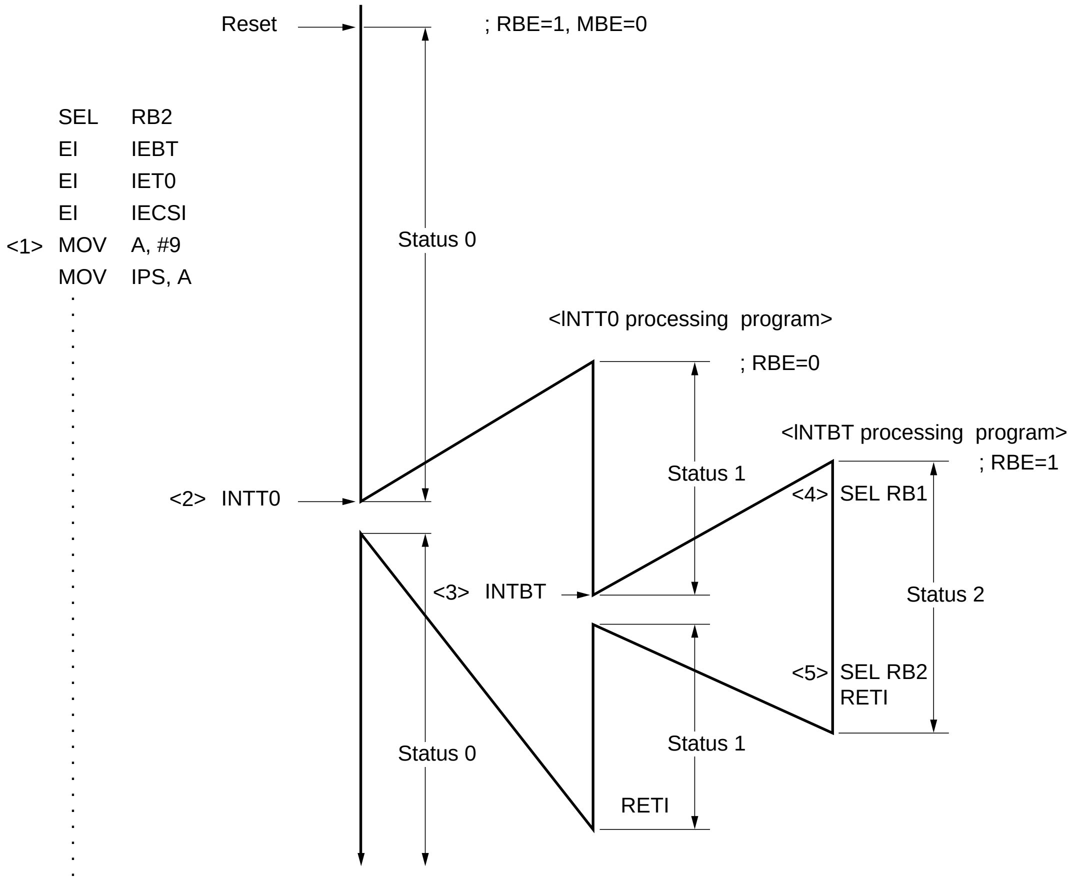
<3> The interrupt is enabled by the EI, EI IE××× instruction.

<4> The INT0 interrupt processing program is started at the falling edge of INT0. The status is changed to 1, and all the interrupts are disabled.
RBE = 0, and register banks 0 and 1 are used.

<5> Execution returns from the interrupt routine when the RETI instruction is executed. The status is returned to 0 and the interrupt is enabled.

Remark If all the interrupts are used with 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 routine 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.

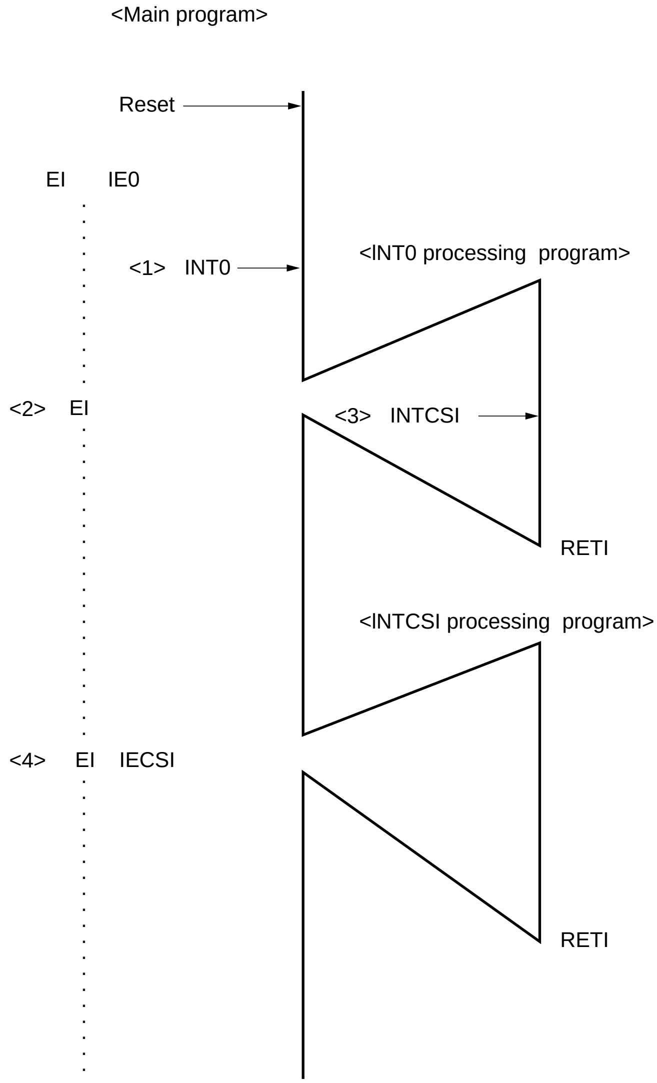
<2> INTT0 processing program is started when INTT0 with the lower priority occurs. Status 1 is set and the other interrupts with the lower priority are disabled. RBE = 0 to select register bank 0.

<3> INTBT with the higher priority occurs. The interrupts are nested. 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).

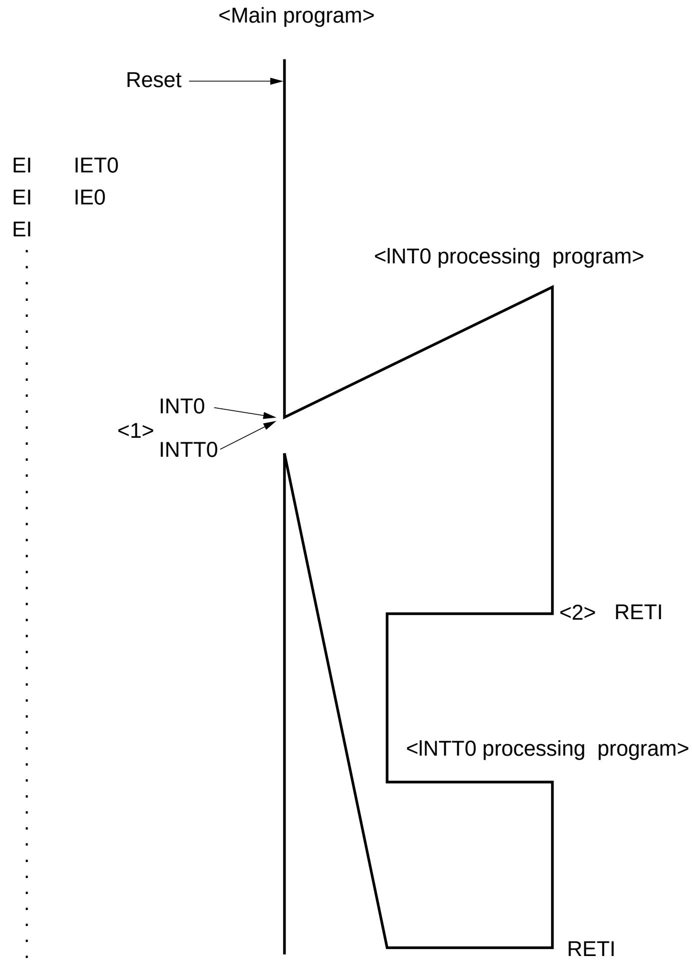
<5> RBS is returned to 2, and execution returns to the main routine. The status is returned to 1.

(4) Executing pending interrupt - interrupt input while interrupts are disabled -



<1> The request flag is kept pending even if INT0 is set while the interrupts are disabled.
<2> INT0 processing program is started when the interrupts are enabled by the EI instruction.
<3> Same as <1>.
<4> INTCSI processing program is started when the pending INTCSI is enabled.

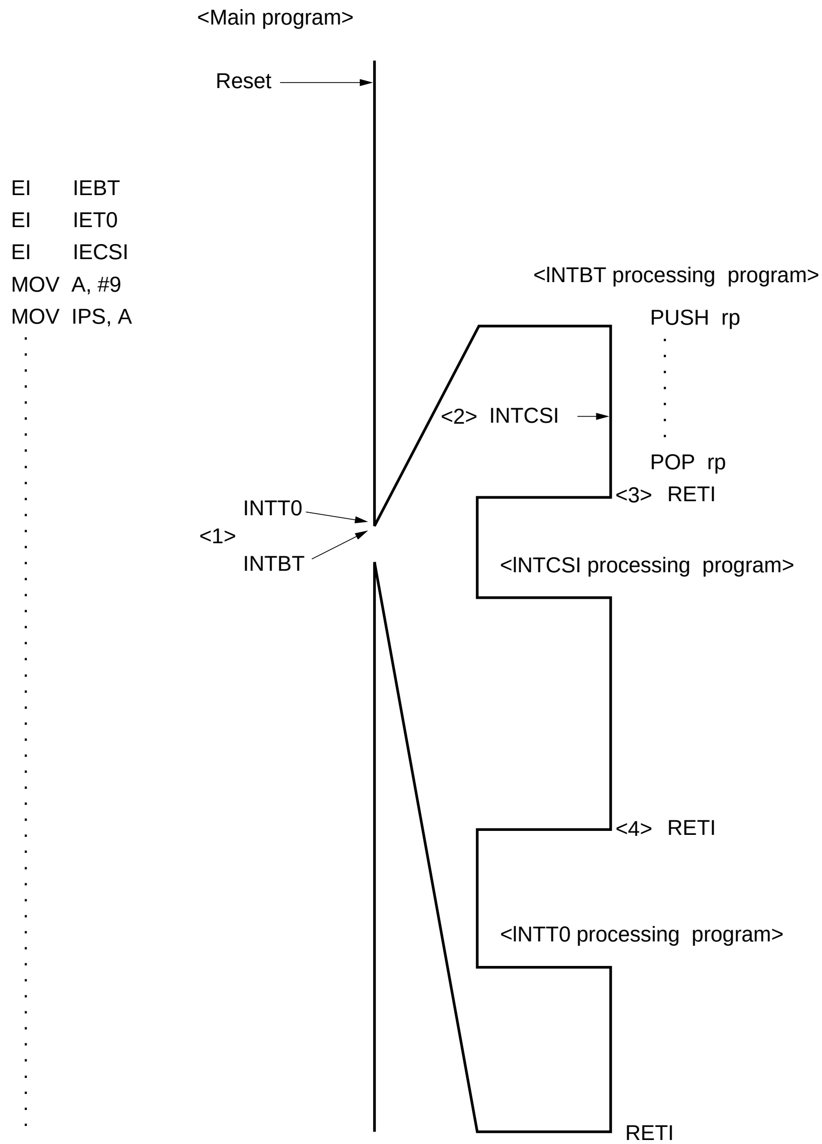
(5) Executing pending interrupt - two interrupts with lower priority occur simultaneously -



<1> If INT0 and INTT0 with the lower priority occur at the same time (while the same instruction is executed), INT0 with the higher priority is executed first (INTT0 is kept pending).

<2> When the INT0 processing routine is terminated by the RETI instruction, the pending INTT0 processing program is started.

(6) Executing pending interrupt - interrupt occurs during interrupt processing (INTBT has higher priority and INTT0 and INTCSI have lower priority) -



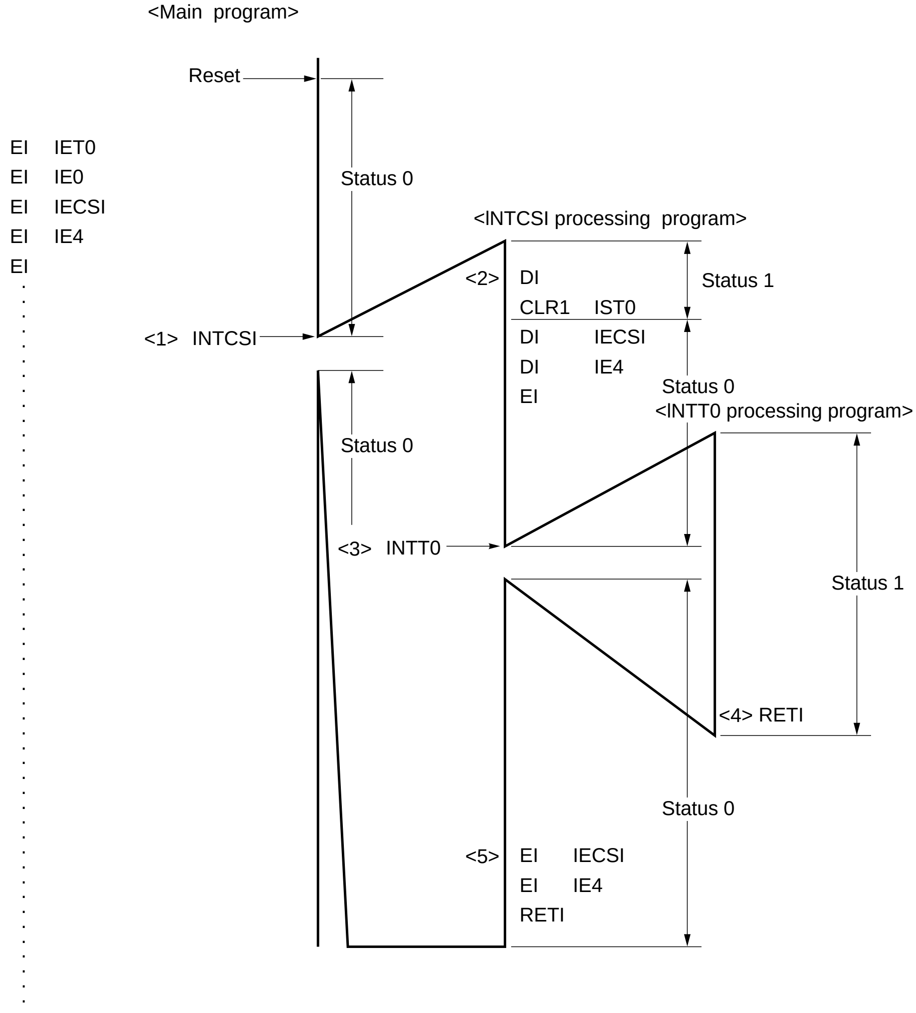
<1> If INTBT with the higher priority and INTT0 with the lower priority occur at the same time, the processing of the interrupt with the higher priority is started. (If there is no possibility that an interrupt with the higher priority will occur while another interrupt with the higher priority is being processed, DI IE×× is not necessary.)

<2> If 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 two nesting of interrupts - INTT0 and INT0 are nested doubly and INTCSI and INT4 are nested singly -



<1> When an INTCSI that does not enable nesting occurs, the INTCSI processing routine is started. The status is 1.

<2> The status is changed to 0 by clearing IST0. INTCSI and INT4 that do not enable nesting are disabled.

<3> When an INTT0 that enables nesting occurs, nesting is executed. The status is changed to 1, and all the interrupts are disabled.

<4> The status is returned to 1 when INTT0 processing is completed.

<5> The disabled INTCSI and INT4 are enabled, and execution returns to the main routine.

6.10 Test Function

6.10.1 Types of test sources

The μPD750068 has two types of test sources. Of these, INT2 is provided with two types of edge-detection testable inputs.

Table 6-5 Types of Test Sources

Test Source	Internal/External
INT2 (detects rising edge input to INT3 or falling edge of input to KR0-KR7)	External
INTW (signal from watch timer)	Internal

6.10.2 Hardware controlling test function

(1) Test request and test enable flags

A test request flag (IRQ×××) is set to "1" when a test request is generated. Clear this flat to "0" by software after the test processing has been executed.

A test enable flag (IE×××) is provided to each test enable flag. When this flag is "1", the standby release signal is enabled; when it is "0", the signal is disabled.

If both the test request flag and test enable flag are set to "1", the standby release signal is generated.

Table 6-6 shows the signals that set the test request flags.

Table 6-6 Test Request Flag Setting Signals

Test Request Flag	Test Request Flag Setting Signal	Test Enable Flag
IRQW	Signal from watch timer	IEW
IRQ2	Detection of rising edge of INT2/P12 pin input signal or detection of falling edge of any input to KR0/P60-KR3/P63 pins. Edge to be detected is selected by INT2 edge detection mode register (IM2)	IE2

(2) Hardware of INT2 and key interrupts (KR0-KR3)

Fig. 6-10 shows the configuration of INT2 and KR0 through KR3.

The IRQ2 setting signal is output when a specified edge is detected on either of the following two types of pins. Which pin is selected is specified by using the INT2 edge detection mode register (IM2).

(a) Detection of rising edge of INT2 pin input

When the rising edge of INT2 pin input is detected, IRQ2 is set.

(b) Detection of rising edge of any of KR0 through KR3 pin inputs (key interrupt)

Of KR0 through KR3, select the pin used for interrupt input by using the INT2 edge detection mode register (IM2). When the rising edge of input to the selected pin is detected, IRQ2 is set.

Fig. 6-11 shows the format of IM2. IM2 is set by a 4-bit manipulation instruction. When the reset signal is asserted, all the bits of this register are cleared to “0” and the rising edge of INT2 is specified.

Fig. 6-10 Block Diagram of INT2 and KR0-KR3

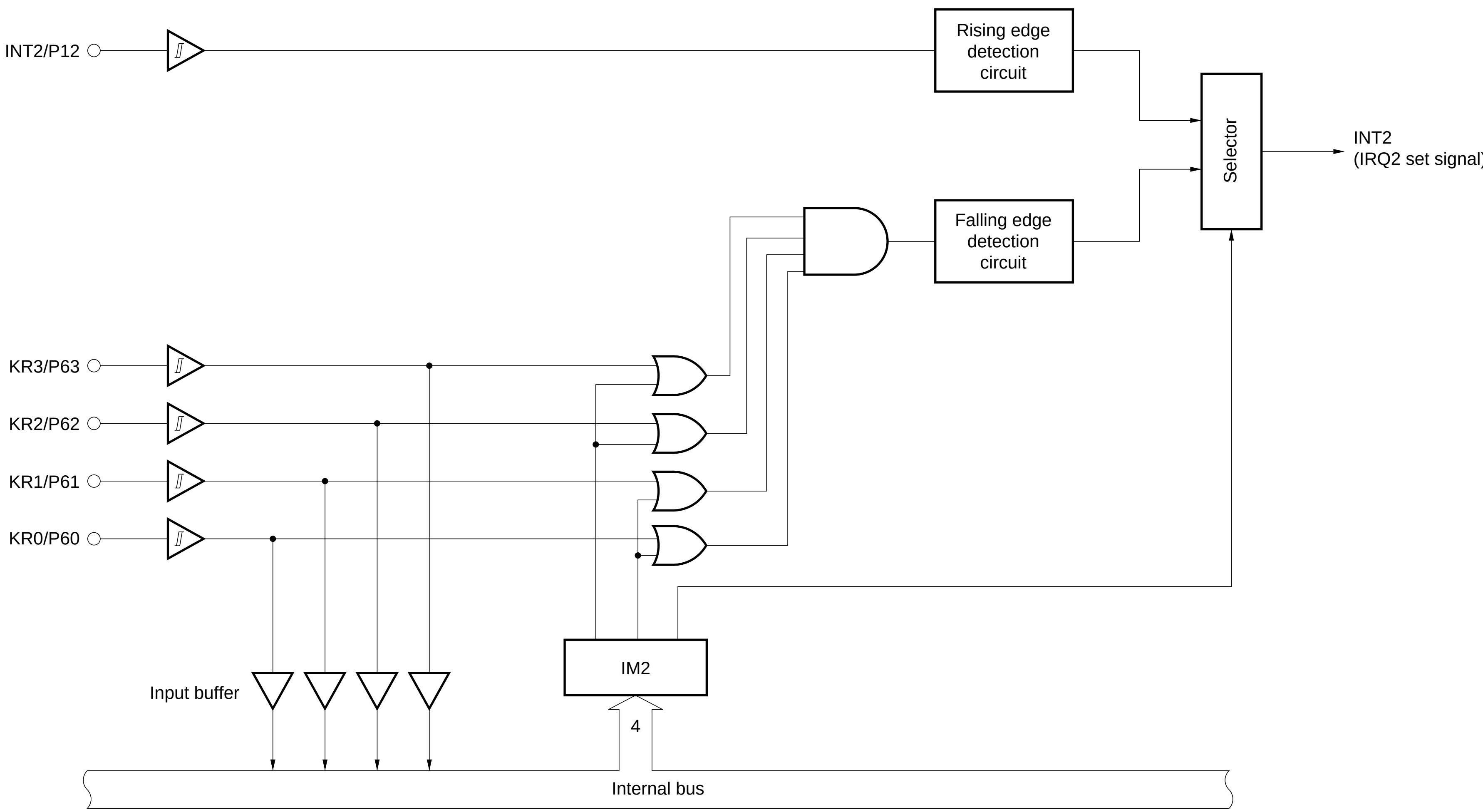


Fig. 6-11 Format of INT2 Edge Detection Mode Register (IM2)

Address	3	2	1	0	Symbol
FB6H	0	0	IM21	IM20	IM2

IM21	IM20	INT2 test source	Test input pin
0	0	Specifies rising edge of INT2 pin input	INT2 (1 pin)
0	1	Setting prohibited	–
1	0	Specifies falling edge of any of KRX pin input	KR2, KR3 (2 pins)
1	1		KR0-KR3 (4 pins)

Cautions

1. **If the contents of the edge detection mode register are changed, the test request flag may be set. Disable the test input before changing the contents of the mode register. Then, clear the test request flag by the CLR1 instruction and enable the test input.**
2. **If a low level is input to even one of the pins selected for falling edge detection, IRQ2 is not set even if the falling edge is input to the other pins.**

[MEMO]

CHAPTER 7 STANDBY FUNCTION

The μPD750068 possesses a standby function that reduces the power consumption of the system. This standby function can be implemented in the following two modes:

- STOP mode
- HALT mode

The functions of the STOP and HALT modes are as follows:

(1) STOP mode

In this mode, the main system clock oscillation circuit is stopped and therefore, the entire system is stopped. The power consumption of the CPU is substantially reduced.

Moreover, the contents of the data memory can be retained at a low voltage (V_{DD} = 1.8 V MIN.). This mode is therefore useful for retaining the data memory contents with an extremely low current consumption.

The STOP mode of the μPD750068 can be released by an interrupt request; therefore, the microcontroller can operate intermittently. However, because a certain wait time is required for stabilizing the oscillation of the clock oscillation circuit when the STOP mode has been released, use the HALT mode if processing must be started immediately after the standby mode has been released by an interrupt request.

(2) HALT mode

In this mode, the operating clock of the CPU is stopped. Oscillation of the system clock oscillation circuit continues. This mode does not reduce the power consumption as much as the STOP mode, but it is useful when processing must be resumed immediately when an interrupt request is issued, or for an intermittent operation such as a watch operation.

In either mode, all the contents of the registers, flags, and data memory immediately before the standby mode is set are retained. Moreover, the contents of the output latches and output buffers of the I/O ports are also retained; therefore, the statuses of the I/O ports are processed in advance so that the current consumption of the overall system can be minimized.

The following page describes the points to be noted in using the standby mode.

Cautions 1. The STOP mode can be used only when the system operates with the main system clock (oscillation of the subsystem clock cannot be stopped). The HALT mode can be used regardless of whether the system operates with the main system clock or subsystem clock.

2. If the STOP mode is set when the watch timer operates with main system clock f_X, the operation of the watch timer is stopped.
To continue the operations of these, therefore, you should change the operating clock to subsystem clock f_{XT} before setting the STOP mode.

3. You can operate the μPD750068 efficiently with a low current consumption at a low voltage by selecting the standby mode, CPU clock, and system clock. In any case, however, the time described in 5.2.3 Setting of system clock and CPU clock is required until the operation is started with the new clock when the clock has been changed by manipulating the control register. To use the clock selecting function and standby mode in combination, therefore, set the standby mode after the time required for selection has elapsed.

4. To use the standby mode, process so that the current consumption of the I/O ports is minimized.
Especially, do not open the input port, and be sure to input either low or high level to it.

7.1 Setting of and Operating Status in Standby Mode

Table 7-1 Operating Status in Standby Mode

		STOP Mode	HALT Mode
Setting instruction		STOP instruction	HALT instruction
System clock on setting		Can be set only when processor operates with main system clock	Can be set regardless of whether processor operates with main system clock or subsystem clock
Operating status	Clock generation circuit	Oscillation of main system clock is stopped	Only CPU clock Φ is stopped (oscillation continues)
	Basic interval timer/ watchdog timer	Stops	Operates only during oscillation of main system clock (sets IRQBT at reference time intervals)
	Serial interface	Can operate only when external $\overline{SCK}$ input is selected as serial clock	Can operate only when external $\overline{SCK}$ input is selected as serial clock
	Timer/event counter	Can operates only when watch timer for which TI0 and TI1 pin inputs or f_{XT} are selected is specified as count clock	Can operate only when watch timer for TI0 and TI1 pin inputs or f_{XT} are selected is specified as count clock or during oscillation of main system clock
	Watch timer	Can operate when f_{XT} is selected as count clock	Can operate
	A/D converter	Stops	Can operate only during oscillation of main system clock
	External interrupt	INT1, 2, and 4 can operate. Only INT0 cannot operate[Note]	
	CPU	Stops	
Releasing signal		Interrupt request signal enabled by interrupt enable flag from hardware units that can operate, or $\overline{RESET}$ signal generation	

Note Can operate only when the noise rejection circuits not selected by bit 2 of the edge detection mode register (IM02 = 1).

The STOP mode is set by the STOP instruction, and the HALT mode is set by the HALT instruction (the STOP and HALT instructions respectively set bits 3 and 2 of PCC).

Be sure to write the NOP instruction after the STOP and HALT instructions.

When changing the CPU operating clock by using the lower 2 bits of PCC, a certain time elapses after the bits of PCC have been rewritten until the CPU clock is actually changed, as indicated in **Table 5-5 Maximum Time Required for Changing System Clock and CPU Clock**. To change the operating clock before the standby mode is set and the CPU clock after the standby mode has been released, set the standby mode after the lapse of the machine cycles necessary for changing the CPU clock, after rewriting the contents of PCC.

In the standby mode, the data is retained for all the registers and data memory that stop in the standby mode, such as general-purpose registers, flags, mode registers, and output latches.

★ **Cautions 1. When the STOP mode is set, the XT2 pin is internally pulled up to V_{DD} with a resistor of 50 kΩ (TYP.).**

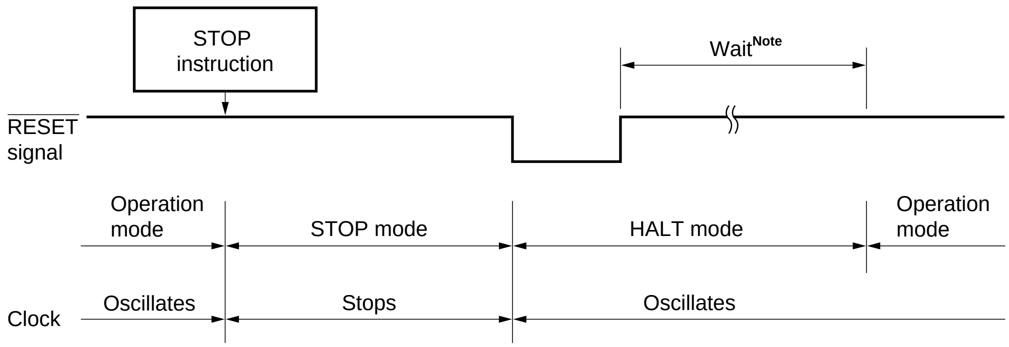
2. Reset all the interrupt request flags before setting the standby mode.
If there is an interrupt source whose interrupt request flag and interrupt enable flag are both set, the standby mode is released immediately after it has been set (refer to Fig. 6-1 Block Diagram of Interrupt Control Circuit). If the STOP mode is set, however, the HALT mode is set immediately after the STOP instruction has been executed, and the time set by the BTM register elapses. Then, the normal operation mode is restored.

7.2 Releasing Standby Mode

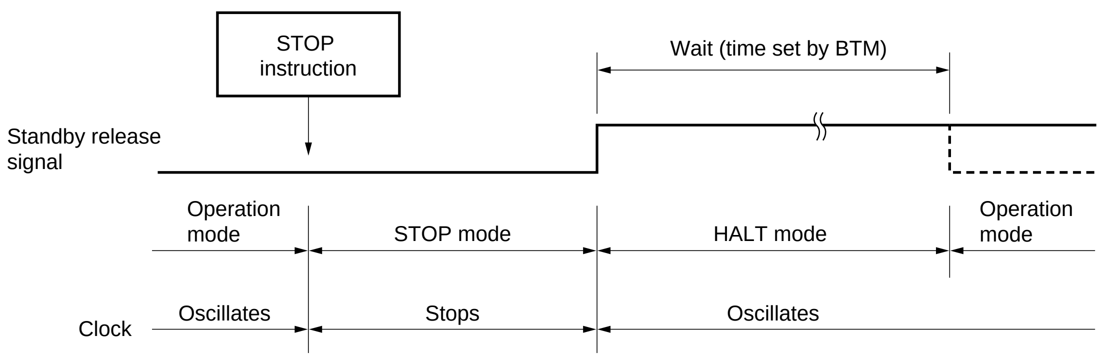
Both the STOP and HALT modes can be released when an interrupt request signal occurs that is enabled by the corresponding interrupt enable flag, or when the $\overline{\text{RESET}}$ signal is asserted. Fig. 7-1 illustrates how each mode is released.

Fig. 7-1 Releasing Standby Mode (1/2)

(a) Releasing STOP mode by $\overline{\text{RESET}}$ signal



(b) Releasing STOP mode by interrupt



Note The following two times can be selected by mask option:

$2^{17}/f_X$ (21.8 ms at 6.0 MHz, 31.3 ms at 4.19 MHz)

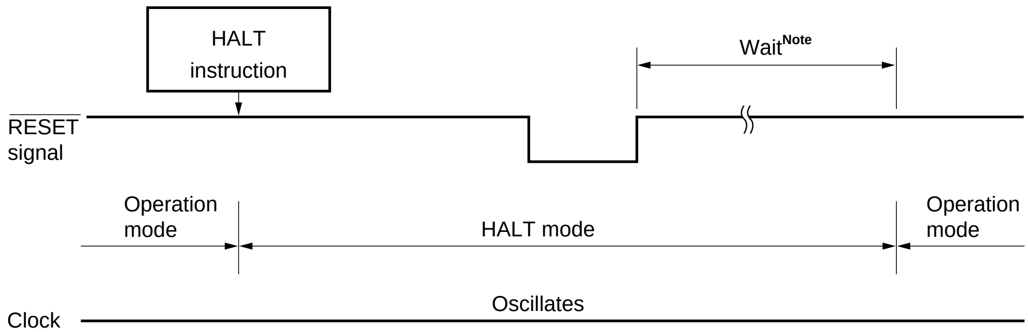
$2^{15}/f_X$ (5.46 ms at 6.0 MHz, 7.81 ms at 4.19 MHz)

★ However, the μPD75P0076 has no mask option, and the wait time is fixed at $2^{15}/f_X$.

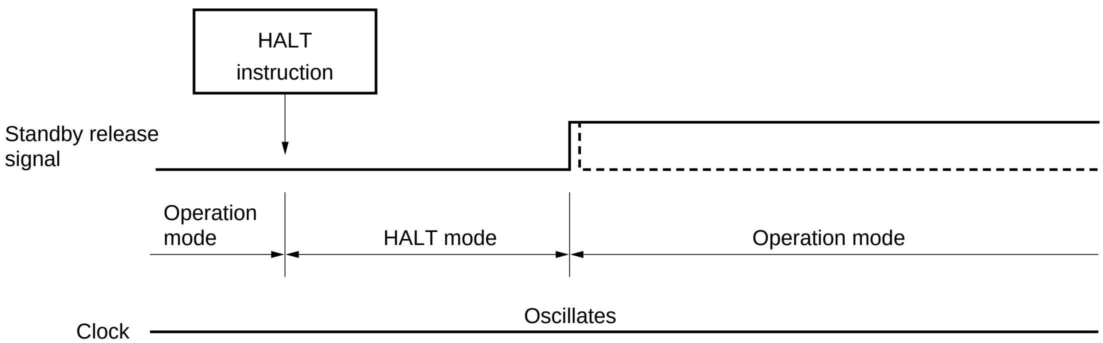
Remark The broken line indicates acknowledgment of the interrupt request that releases the standby mode.

Fig. 7-1 Releasing Standby Mode (2/2)

(c) Releasing HALT mode by $\overline{\text{RESET}}$ signal



(d) Releasing HALT mode by interrupt



Note The following two times can be selected by mask option:

$2^{17}/f_X$ (21.8 ms at 6.0 MHz, 31.3 ms at 4.19 MHz)

$2^{15}/f_X$ (5.46 ms at 6.0 MHz, 7.81 ms at 4.19 MHz)

★ However, the μPD75P0076 has no mask option, and the wait time is fixed at $2^{15}/f_X$.

Remark The broken line indicates acknowledgment of the interrupt request that releases the standby mode.

When the STOP mode has been released by an interrupt, the wait time is determined by the setting of BTM (refer to **Table 7-2**).

The time required for the oscillation to stabilize varies depending on the type of the oscillator used and the supply voltage when the STOP mode has been released. Therefore, you should select the appropriate wait time depending on the given conditions, and set BTM before setting the STOP mode.

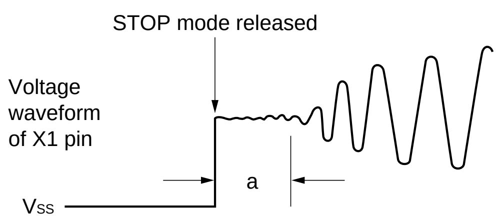
Table 7-2 Selecting Wait Time by BTM

BTM3	BTM2	BTM1	BTM0	Wait Time[Note]	
				f_X = 6.0 MHz	f_X = 4.19 MHz
–	0	0	0	About $2^{20}/f_X$ (about 175 ms)	About $2^{20}/f_X$ (about 250 ms)
–	0	1	1	About $2^{17}/f_X$ (about 21.8 ms)	About $2^{17}/f_X$ (about 31.3 ms)
–	1	0	1	About $2^{15}/f_X$ (about 5.46 ms)	About $2^{15}/f_X$ (about 7.81 ms)
–	1	1	1	About $2^{13}/f_X$ (about 1.37 ms)	About $2^{13}/f_X$ (about 1.95 ms)
Others				Setting prohibited	

Note This time does not include the time required to start oscillation after the STOP mode has been released.

Caution The wait time that elapses when the STOP mode has been released does not include the time that elapses until the clock oscillation is started after the STOP mode has been released (a in Fig. 7-2), regardless of whether the STOP mode has been released by the $\overline{\text{RESET}}$ signal or occurrence of an interrupt.

Fig. 7-2 Wait Time after Releasing STOP Mode



7.3 Operation After Release of Standby Mode

(1) When the standby mode has been released by the $\overline{\text{RESET}}$ signal, the normal reset operation is performed.
(2) When the standby mode has been released by an interrupt, whether or not a vector interrupt is executed when the CPU has resumed instruction execution is determined by the content of the interrupt master enable flag (IME).

(a) When IME = 0

Execution is started from the instruction next to the one that set the standby mode after the standby mode has been released. The interrupt request flag is retained.

(b) When IME = 1

A vector interrupt is executed after the standby mode has been released and then two instructions have been executed. However, if the standby mode has been released by INTW or INT2 (testable input), the processing same as (a) is performed because no vector interrupt is generated in this case.

★ 7.4 Selecting Mask Option

For the μPD750068 standby function, a wait time after the standby function is released by a $\overline{\text{RESET}}$ signal can be selected by mask option from the following two times.

<1> $2^{17}/f_X$ (21.8 ms: f_X = 6.0 MHz operation, 31.3 ms: f_X = 4.19 MHz operaiton)
<2> $2^{15}/f_X$ (5.46 ms: f_X = 6.0 MHz operation, 7.81 ms: f_X = 4.19 MHz operaiton)

The μPD75P0076 has no mask option and the wait time is fixed at $2^{15}/f_X$.

7.5 Application of Standby Mode

Use the standby mode in the following procedure:

<1> Detect the cause that sets the standby mode such as an interrupt input or power failure by port input (use of INT4 to detect a power failure is recommended).
<2> Process the I/O ports (process so that the current consumption is minimized).
Especially, do not open the input port. Be sure to input a low or high level to it.
<3> Specify an interrupt that releases the standby mode. (Note that use of INT4 is effective. Clear the interrupt enable flags of the interrupts that do not release the standby mode.)
<4> Specify the operation to be performed after the standby mode has been released (manipulate IME depending on whether interrupt processing is performed or not).
<5> Specify the CPU clock to be used after the standby mode has been released. (To change the clock, make sure that the necessary machine cycles elapse before the standby mode is set.)
<6> Select the wait time to elapse after the standby mode has been released.
<7> Set the standby mode (by using the STOP or HALT instruction).

By using the standby mode in combination with the system clock selecting function, low current consumption and low-voltage operation can be realized.

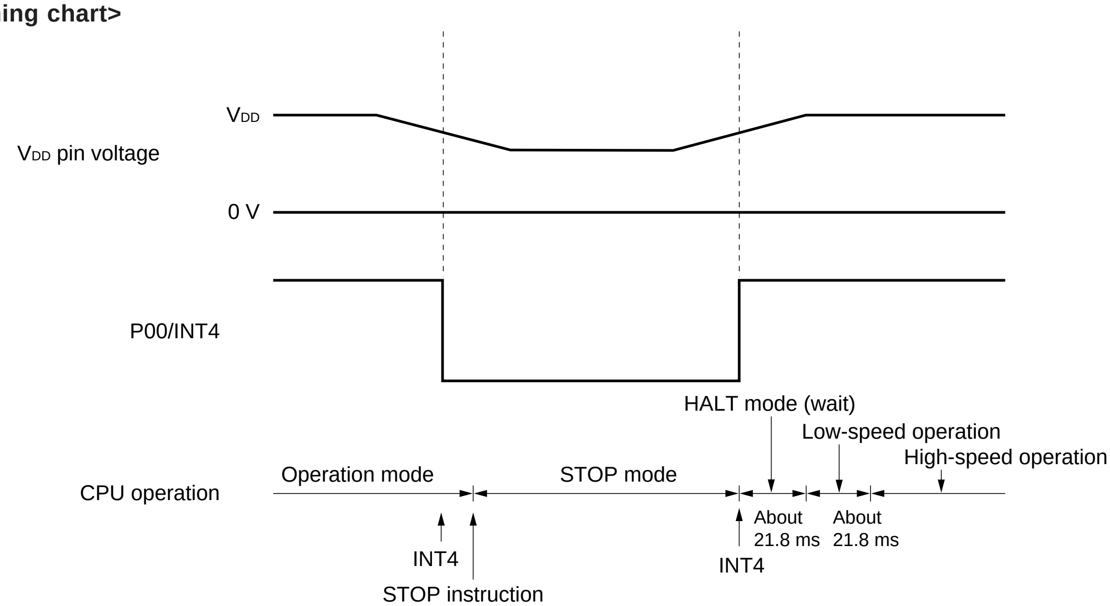
(1) Application example of STOP mode (f_X = 6.0 MHz)

<When using the STOP mode under the following conditions>

- The STOP mode is set at the falling edge of INT4 and released at the rising edge (INTBT is not used).
- All the I/O ports go into a high-impedance state (if the pins are externally processed so that the current consumption is reduced in a high-impedance state).
- Interrupts INT0 and INTT0 are used in the program. However, these interrupts are not used to release the STOP mode.
- The interrupts are enabled even after the STOP mode has been released.
- After the STOP mode has been released, operation is started with the slowest CPU clock.
- The wait time that elapses after the mode has been released is about 21.8 ms.
- A wait time of 21.8 ms elapses until the power supply stabilizes after the mode has been released. The P00/INT4 pin is checked two times to prevent chattering.

<Timing chart>

★



<Program example>

(INT4 processing program, MBE = 0)

```
VSUB4:   SKT    PORT0.0      ; P00 = 1?
         BR     PDOWN        ; Power down
         SET1   BTM.3        ; Power on
WAIT:    SKT    IRQBT        ; Waits for 21.8 ms
         BR     WAIT
         SKT    PORT0.0      ;  Checks chattering
         BR     PDOWN
         MOV    A, #0011B
         MOV    PCC, A       ; Sets high-speed mode
        (MOV    XA, #××H  )  ; Sets 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 in high-impedance state
         MOV    PMGB, XA
         DI     IE0          ; Disables INT0 and INTT0
         DI     IET0
         MOV    A, #1011B
         MOV    BTM, A       ; Wait time ≒ 21.8 ms
         NOP
         STOP                ; Sets STOP mode
         NOP
         RETI
```

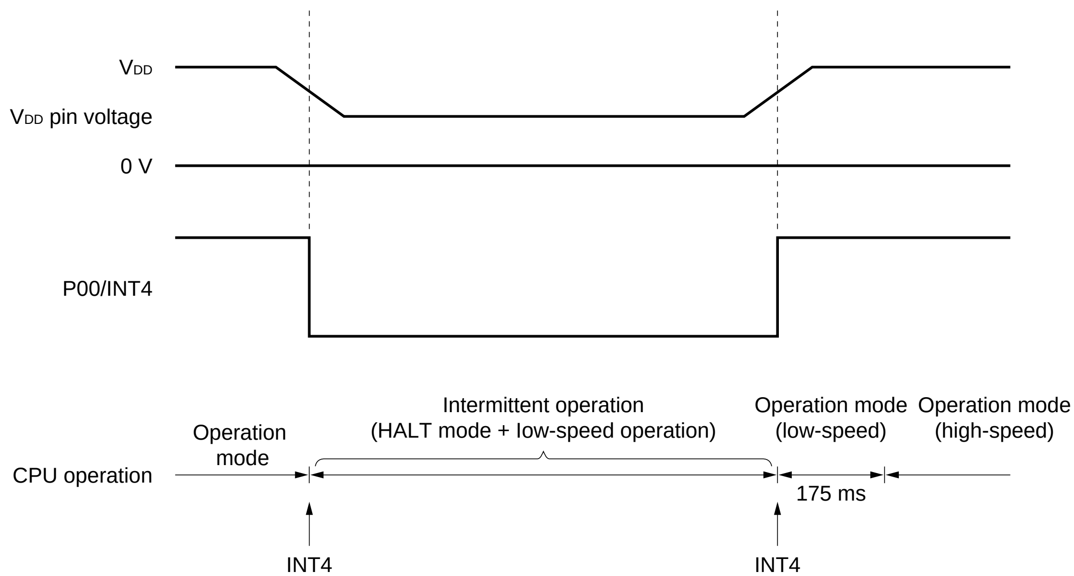
(2) Application example of HALT mode (f_X = 6.0 MHz)

<To perform intermittent operation under the following conditions>

- The standby mode is set at the falling edge of INT4 and released at the rising edge.
- In the standby mode, an intermittent operation is performed at intervals of 175 ms (INTBT).
- INT4 and INTBT are assigned with the lower priority.
- The slowest CPU clock is selected in the standby mode.

<Timing chart>

★



<Program example>

(Initial setting)

```
                MOV     A, #0011B
                MOV     PCC, A          ; High-speed mode
                MOV     XA, #05H
                MOV     WM, XA          ; Subsystem clock
                EI      IE4
                EI      IEW
                EI                      ; Enables interrupt
```

(Main routine)

```
                SKT     PORT0.0         ; Power supply OK?
                HALT                    ; Power down mode
                NOP                     ; Power supply OK?
                SKTCLR  IRQW            ; 0.5-sec flag?
                BR      MAIN            ; NO
                CALL    WATCH           ; Watch subroutine
    MAIN:         :
                  :
                  :
```

(INT4 processing routine)

```
    VINT4:      SKT     PORT0.0         ; Power supply OK?, MBE = 0
                BR3     PDOWN
                CLR1    SCC.3           ; Main system clock starts oscillating
                MOV     A, #1000B
                MOV     BTM,A
    WAIT1:      SKT     IRQBT           ; Waits for 175 ms
                BR      WAIT1
                SKT     PORT0.0         ; Checks chattering
                BR      PDOWN
                CLR1    SCC.0           ; Selects main system clock
                RETI
    PDOWN:      SET1    SCC.0           ; Selects subsystem clock
                MOV     A,#5
    WAIT2:      INCS    A               ; Waits for 46 machine cycles or more Note
                BR      WAIT2
                SET1    SCC.3           ; Main system clock oscillation stop
                RETI
```

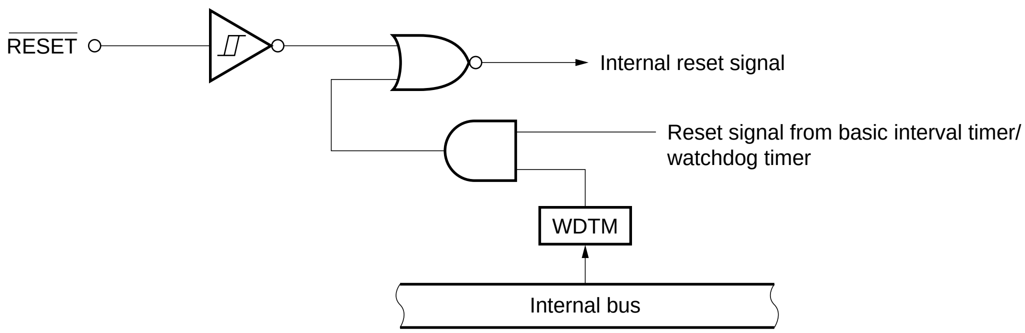
Note For how to select the system clock and CPU clock, refer to **5.2.3 Setting system clock and CPU clock.**

Caution To change the system clock from the main system clock to the subsystem clock, wait until the oscillation of the subsystem clock has stabilized.

CHAPTER 8 RESET FUNCTION

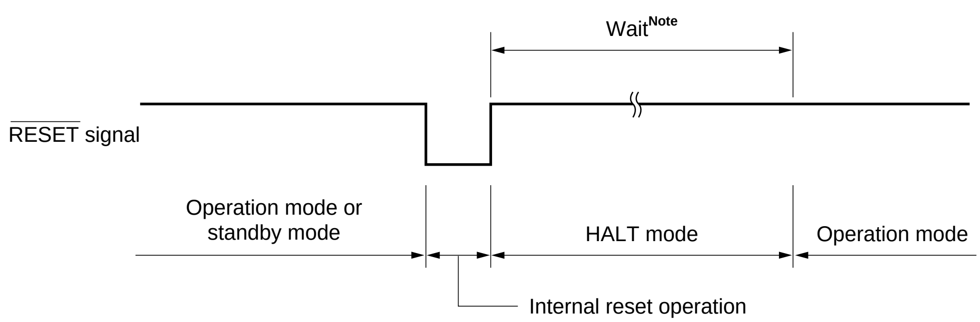
Two types of reset signals are used: the external reset signal ($\overline{\text{RESET}}$) and a reset signal from the basic interval timer/watchdog timer. When either of these reset signals is input, an internal reset signal is asserted. Fig. 8-1 shows the configuration of the reset circuit.

Fig. 8-1 Configuration of Reset Circuit



Each hardware unit is initialized when the $\overline{\text{RESET}}$ signal is asserted as shown in Table 8-1. Fig. 8-2 shows the timing of the reset operation.

Fig. 8-2 Reset Operation by $\overline{\text{RESET}}$ Signal



Note The following two times can be selected by the mask option:

$2^{17}/f_X$ (21.8 ms at 6.0 MHz, 31.3 ms at 4.19 MHz)

$2^{15}/f_X$ (5.46 ms at 6.0 MHz, 7.81 ms at 4.19 MHz)

★ However, the μPD75P0076 has no mask option, and the wait time is fixed at $2^{15}/f_X$.

Table 8-1 Status of Each Hardware Unit after Reset (1/2)

Hardware		When $\overline{\text{RESET}}$ Signal Asserted in Standby Mode	When $\overline{\text{RESET}}$ Signal Asserted during Operation
Program counter (PC)	μPD750064	Sets the lower 4 bits of program memory address 0000H to PC11-PC8, and contents of address 0001H to PC7-PC0	Same as left
	μPD750066, 750068	Sets the lower 5 bits of program memory address 0000H to PC12-PC8, and contents of address 0001H to PC7-PC0	
	μPD75P0076	Sets lower 6 bits of program memory address 0000H to PC13-PC8, and contents of address 0001H to PC7-PC0	
PSW	Carry flag (CY)	Retained	Undefined
	Skip flags (SK0-SK2)	0	0
	Interrupt status flags (IST0, IST1)	0	0
	Bank enable flags (MBE, RBE)	Sets bit 6 of program memory address 0000H to RBE and bit 7 to MBE	Same as left
Stack pointer (SP)		Undefined	Undefined
Stack bank select register (SBS)		1000B	1000B
Data memory (RAM)		Retained	Undefined
General-purpose register (X, A, H, L, D, E, B, C)		Retained	Undefined
Bank select registers (MBS, RBS)		0, 0	0, 0
Basic interval timer/watchdog timer	Counter (BT)	Undefined	Undefined
	Mode register (BTM)	0	0
	Watchdog timer enable flag (WDTM)	0	0
Timer/event counter (T0)	Counter (T0)	0	0
	Modulo register (TMOD0)	FFH	FFH
	Mode register (TM0)	0	0
	TOE0, TOUT F/F	0, 0	0, 0
Timer/event counter (T1)	Counter (T1)	0	0
	Modulo register (TMOD1)	FFH	FFH
	Mode register (TM1)	0	0
	TOE1, TOUT F/F	0, 0	0, 0
Watch timer	Mode register (WM)	0	0

Table 8-1 Status of Each Hardware Unit after Reset (2/2)

Hardware		When $\overline{RESET}$ Signal Asserted in Standby Mode	When $\overline{RESET}$ Signal Asserted during Operation
Serial interface	Shift register (SIO)	Retained	Undefined
	Operation mode register (CSIM)	0	0
Clock generation circuit, clock output circuit	Processor clock control register (PCC)	0	0
	System clock control register (SCC)	0	0
	Clock output mode register (CLOM)	0	0
Suboscillation circuit control register (SOS)		0	0
A/D converter	Mode register (ADM)	04H	04H
	SA register (SA)	7FH	7FH
Interrupt function	Interrupt request flag (IRQ×××)	Reset (0)	Reset (0)
	Interrupt enable flag (IE×××)	0	0
	Interrupt master enable flag (IME)	0	0
	Interrupt priority select register (IPS)	0	0
	INT0, 1, 2 mode registers (IM0, IM1, IM2)	0, 0, 0	0, 0, 0
Digital port	Output buffer	Off	Off
	Output latch	Cleared (0)	Cleared (0)
	I/O mode registers (PMGA, PMGB)	0	0
	Pull-up resistor specification register (POGA)	0	0
Bit sequential buffer (BSB0-BSB3)		Retained	Undefined

[MEMO]

CHAPTER 9 WRITING AND VERIFYING PROM (PROGRAM MEMORY)

The program memory of the μPD75P0076 is a one-time PROM. The memory capacity is as follows:

μPD75P0076: 16384 words × 8 bits

To write or verify this one-time PROM, the pins shown in Table 9-1 are used. Note that no address input pins are used and that the address is updated by inputting a clock from the X1 pin.

Table 9-1 Pins Used to Write or Verify Program Memory

Pin Name	Function
X1, X2	Inputs clock to update address when program memory is written or verified. Complement of X1 pin is input to X2 pin.
MD0/P30-MD3/P33	Select operation mode when program memory is written or verified
D0/P40-D3/P43 (lower 4 bits), D4/P50-D7/P53 (higher 4 bits)	Input or output 8-bit data when program memory is written or verified
V_{DD}	Supplies power supply voltage. Supplies 1.8 to 5.5 V for normal operation and +6 V when program memory is written or verified
V_{PP}	Applies program voltage for writing or verifying program memory (usually, V_{DD} potential)

Cautions 1. The program memory contents of the μPD75P0076 cannot be erased by ultraviolet rays because the μPD75P0076 is not provided with a window for erasure.

2. Process the pins not used for writing or verifying the program memory as follows:

- **Other than XT2 pin: Connect to V_{SS} via pull-down resistor**
- **XT2 pin: Open**

9.1 Operation Mode for Writing/Verifying Program Memory

When +6 V is applied to the V_{DD} pin of the μPD75P0076 and +12.5 V is applied to the V_{PP} pin, the program memory write/verify mode is set. In this mode, the following operation modes can be selected by using the MD0 through MD3 pins.

Table 9-2 Operation Mode

Specifies Operation Mode						Operation Mode
V_{DD}	V_{PP}	MD0	MD1	MD2	MD3	
+6 V	+12.5 V	H	L	H	L	Clears program memory address to 0
		L	H	H	H	Write mode
		L	L	H	H	Verify mode
		H	×	H	H	Program inhibit mode

Remark ×: L or H

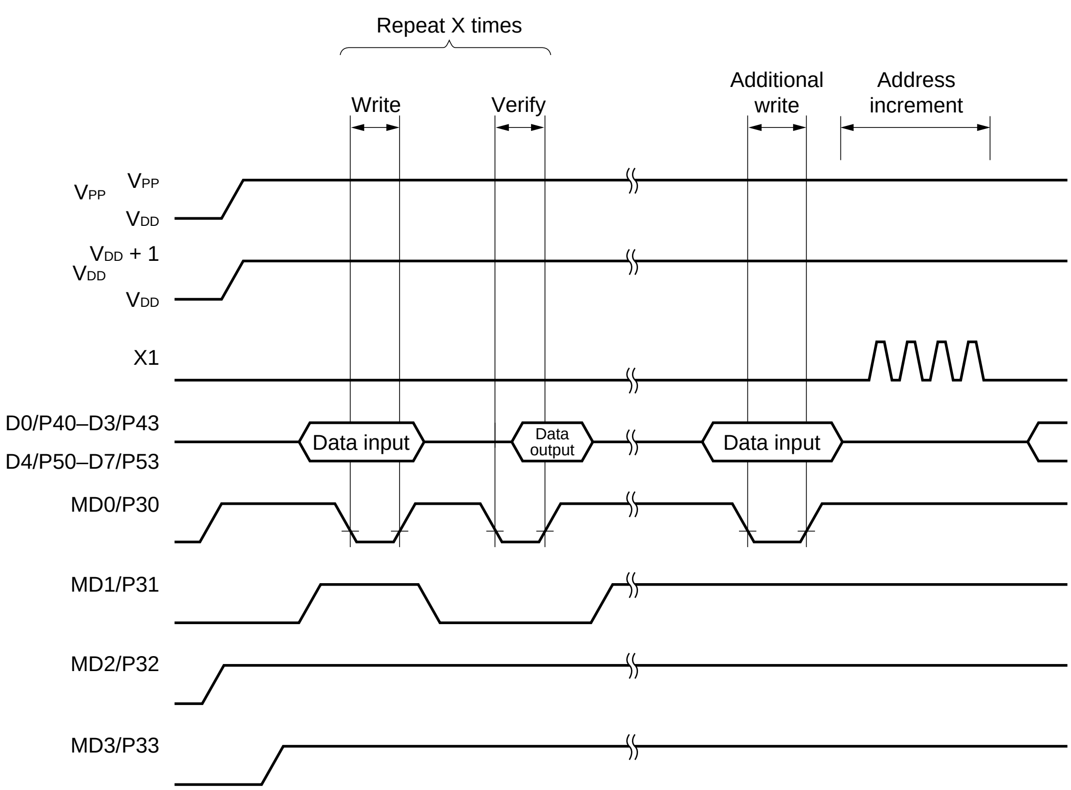
★

9.2 Writing Program Memory

The program memory can be written in the following procedure at high speed:

(1) Pull down the pins not used to V_{SS} with a resistor. The X1 pin is low.
(2) Supply 5 V to the V_{DD} and V_{PP} pins.
(3) Wait for 10 μs.
(4) Set the program memory address 0 clear mode.
(5) Supply +6 V to V_{DD} and +12.5 V to V_{PP}.
(6) Write data in the 1-ms write mode.
(7) Set the verify mode. If the data have been correctly written, proceed to (8). If not, repeat (6) and (7).
(8) Additional writing of (number of times data have been written in (6) and (7): X) × 1 ms
(9) Input a pulse four times to the X1 pin to update the program memory address (by one).
(10) Repeat (6) through (9) until the last address is written.
(11) Set the program memory address 0 clear mode.
(12) Change the voltage applied to the V_{DD} and V_{PP} pins to 5 V.
(13) Turn off the power supply.

Steps (2) through (9) above are illustrated below.

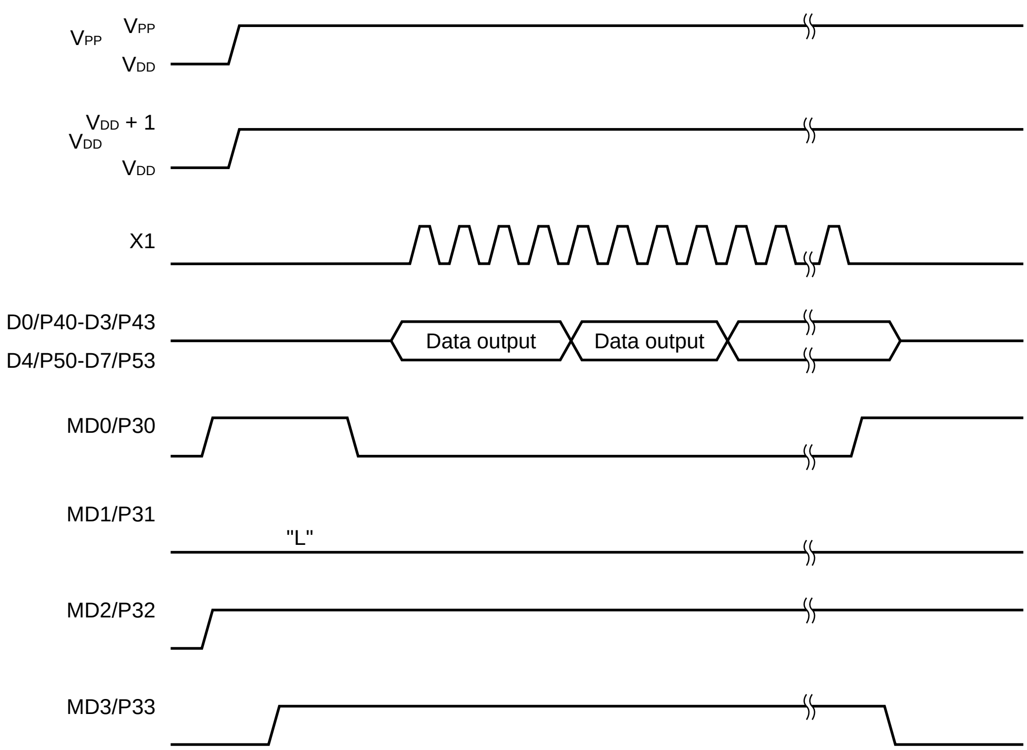


★ 9.3 Reading Program Memory

The contents of the program memory can be read in the following procedure:

(1) Pull down the pins not used to V_{SS} with a resistor. The X1 pin is low.
(2) Supply 5 V to the V_{DD} and V_{PP} pins.
(3) Wait for 10 μs.
(4) Set the program memory address 0 clear mode.
(5) Supply +6 V to V_{DD} and +12.5 V to V_{PP}.
(6) Verify mode. Data of each address is sequentially output at the cycle in which four clock pulses are input to the X1 pin.
(7) Set the program memory address 0 clear mode.
(8) Change the voltage applied to the V_{DD} and V_{PP} pins to 5 V.
(9) Turn off the power supply.

Steps (2) through (7) above are illustrated below.



9.4 One-time PROM Screening

Due to their structure, NEC cannot fully test one-time PROM products before shipment. After the required data has be written, we recommend that the PROMs be screened by being stored in the high temperature environment shown below, and then verified.

Storage temperature	Storage time
125 °C	24 hours

[MEMO]

CHAPTER 10 MASK OPTIONS

10.1 Pins

The μPD750068 pins have the following mask option.

Table 10-1. Selection of Pin Mask Option

Pins	Mask Option
P40-P43, P50-P53	Internal pull-up resistors specifiable bit-wise

Internal pull-up resistors are specifiable for P40 to P43 (port 4) and P50 to P53 (port 5) by mask option. The mask option can be specified bit-wise.

Ports 4 and 5 are set to high level after a reset when internal pull-up resistors are specified by mask option. When no internal pull-up resistor is specified, these ports go to high impedance.

The μPD75P0076 has no pull-up resistor by mask option.

10.2 Mask Option for Standby Function

For the μPD750068 standby function, a wait time can be selected by mask option. This is the time after the standby function is released by a $\overline{\text{RESET}}$ signal until the normal operating mode is set again (refer to **7.2 Releasing Standby Mode** for details). The wait time can be selected from the following two times.

<1> $2^{17}/f_X$ (21.8 ms: f_X = 6.0 MHz operation, 31.3 ms: f_X = 4.19 MHz operation)
<2> $2^{15}/f_X$ (5.46 ms: f_X = 6.0 MHz operation, 7.81 ms: f_X = 4.19 MHz operation)

The μPD75P0076 has no mask option and the wait time is fixed at $2^{15}/f_X$.

★ 10.3 Subsystem Clock Feedback Resistor Mask Options

With the mask option settings, you can choose whether or not to use the feedback resistor in the sub-system clock of the μPD750068.

<1> Feedback resistor can be used (switched ON or OFF via software)
<2> Feedback resistor cannot be used (switched out in hardware)

To use the feedback resistor after selecting <1>, set SOS.0 to 0 via software, and the feedback resistor is turned on (for details, refer to **5.2.2 (6) Suboscillation circuit control register (SOS)**).

When using the subsystem clock, select <1>.

In the μPD75P0076, there is no mask option setting, and the feedback resistor can always be used.

[MEMO]

CHAPTER 11 INSTRUCTION SET

The instruction set of the μPD750068 is based on the instruction set of the 75X series and therefore, maintains compatibility with the 75X series, but has some improved features. They are:

(1) Bit manipulation instructions for various applications
(2) Efficient 4-bit manipulation instructions
(3) 8-bit manipulation instructions comparable to those of 8-bit microcontrollers
(4) GETI instruction reducing program size
(5) String-effect and base number adjustment instructions enhancing program efficiency
(6) Table reference instructions ideal for successive reference
(7) 1-byte relative branch instruction
(8) Easy-to-understand, well-organized NEC's standard mnemonics

For the addressing modes applicable to data memory manipulation and the register banks valid for instruction execution, refer to **3.2 Bank Configuration of General-Purpose Registers**.

11.1 Unique Instructions

This section describes the unique instructions of the μPD750068's instruction set.

11.1.1 GETI instruction

The GETI instruction converts the following instructions into 1-byte instructions:

(a) Subroutine call instruction to 16K-byte space (0000H-3FFFH)
(b) Branch instruction to 16-byte space (0000H-3FFFH)
(c) Any 2-byte, 2-machine cycle instruction (except BRCB and CALLF instructions)
(c) Combination of two 1-byte instructions

The GETI instruction references a table at addresses 0020H through 007FH of the program memory and executes the referenced 2-byte data as an instruction of (a) to (d). Therefore, 48 types of instructions can be converted into 1-byte instructions.

If instructions that are frequently used are converted into 1-byte instructions by using this GETI instruction, the number of bytes of the program can be substantially decreased.

11.1.2 Bit manipulation instruction

The μPD750068 has reinforced bit test, bit transfer, and bit Boolean (AND, OR, and XOR) instruction, in addition to the ordinary bit manipulation (set and clear) instructions.

The bit to be manipulated is specified in the bit manipulation addressing mode. Three types of bit manipulation addressing modes can be used. The bits manipulated in each addressing mode are shown in Table 11-1.

Table 11-1 Types of Bit Manipulation Addressing Modes and Specification Range

Addressing	Peripheral Hardware That Can Be Manipulated	Addressing Range of Bit That Can be Manipulated
fmem. bit	RBE, MBE, IST1, IST0, SCC, IE×××, IRQ×××	FB0H-FBFH
	PORT0-6, 11	FF0H-FFFH
pmem. @L	BSB0-3, PORT0-6, 11	FC0H-FFFH
@H+mem. bit	All peripheral hardware units that can be manipulated bitwise	All bits of memory bank specified by MB that can be manipulated bitwise

Remarks **1.** ×××: 0, 1, 2, 4, BT, T0, T1, W, CSI
2. MB = MBE MBS

11.1.3 String-effect instruction

The μPD750068 has the following two types of string-effect instructions:

(a) MOV A, #n4 or MOV XA, #n8
(b) MOV HL, #n8

“String effect” means locating 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 shown in this example, and if the address executed first is A0, the two instructions following this address are replaced with the NOP instructions. If the address executed first is A1, the following one instruction is replaced with the NOP instruction. In other words, only the instruction that is executed first is valid, and all the string-effect instructions that follow are processed as NOP instructions.

By using these string-effect instructions, constants can be efficiently set to the accumulator (A register or register pair XA) and data pointer (register pair HL).

11.1.4 Base number adjustment instruction

Some application requires that the result of addition or subtraction of 4-bit data (which is carried out in binary number) be converted into a decimal number or into a number with a base of 6, such as time.

Therefore, the μPD750068 is provided with base number adjustment instructions that adjusts the result of addition or subtraction of 4-bit data into a number with any base.

(1) Base adjustment of result of addition

Where the base number to which the result of addition executed is to be adjusted is m, the contents of the accumulator and memory are added in the following combination, and the result is adjusted to a number with a base of m:

```
ADDS A, #16-m
ADDC A, @HL    ; A, CY ← A + (HL) + CY
ADDS A, #m
```

Occurrence of an overflow is indicated by the carry flag.

If a carry occurs as a result of executing the ADDC A, @HL instruction, the ADDS A, #n4 instruction is skipped. If a carry does not occur, the ADDS A, #n4 instruction is executed. At this time, however, the skip function of the instruction is disabled, and the following instruction is not skipped even if a carry occurs as a result of addition. Therefore, a program can be written after the ADDS A, #n4 instruction.

Example To add accumulator and memory in decimal

```
ADDS A,  #6
ADDC A,  @HL    ; A, CY ← A + (HL) + CY
ADDS A,  #10
          ⋮
```

(2) Base adjustment of result of subtraction

Where the base number into which the result of subtraction executed is to be adjusted is m, the contents of memory (HL) are subtracted from those of the accumulator in the following combination, and the result of subtraction is adjusted to a number with a base of m:

```
SUBC  A, @HL
ADDS  A, #m
```

Occurrence of an underflow is indicated by the carry flag.

If a borrow does not occur as a result of executing the SUBC A, @HL instruction, the following ADDS A, #n4 instruction is skipped. If a borrow occurs, the ADDS A, #n4 instruction is executed. At this time, the skip function of this instruction is disabled, and the following instruction is not skipped even if a carry occurs as a result of addition. Therefore, a program can be written after the ADDS A, #n4 instruction.

11.1.5 Skip instruction and number of machine cycles required for skipping

The instruction set of the μPD750068 configures a program where instructions may be or may not be skipped if a given condition is satisfied.

If a skip condition is satisfied when a skip instruction is executed, the instruction next to the skip instruction is skipped and the instruction after next is executed.

When a skip occurs, the number of machine cycles required for skipping is:

(a) If the instruction that follows the skip instruction (i.e., the instruction to be skipped) is a 3-byte instruction (BR !addr, BRA !addr1, CALL !addr, or CALLA !addr1 instruction): 2 machine cycles

(b) Instruction other than (a): 1 machine cycle

11.2 Instruction Set and Operation

(1) Operand representation and description

Describe an operand in the operand field of each instruction according to the operand description method of the instruction (for details, refer to **RA75X Assembler Package User's Manual - Language (EEU-1363)**. If two or more operands are shown, select one of them. The uppercase letters, +, and – are keywords and must be described as is.

The symbols of register flags can be described as labels, instead of mem, fmem, pmem, and bit. (However, the number of labels described for fmem and pmem are limited. For details, refer to **Table 3-1 Addressing Modes** and **Fig. 3-7 μPD750068 I/O Map**).

Representation	Description
reg	X, A, B, C, D, E, H, L
reg1	X, B, C, D, E, H, L
rp	XA, BC, DE, HL
rp1	BC, DE, HL
rp2	BC, DE
rp'	XA, BC, DE, HL, XA', BC', DE', HL'
rp'1	BC, DE, HL, XA', BC', DE', HL'
rpa	HL, HL+, HL–, DE, DL
rpa1	DE, DL
n4	4-bit immediate data or label
n8	8-bit immediate data or label
mem	8-bit immediate data or label**Note**
bit	2-bit immediate data or label
fmem	Immediate data FB0H-FBFH, FF0H-FFFH or label
pmem	Immediate data FC0H-FFFH or label
addr, addr1	Immediate data 0000H-FFFH or label (μPD750064)
(MKII mode only)	Immediate data 0000H-17FFH or label (μPD750066)
	Immediate data 0000H-1FFFH or label (μPD750068)
	Immediate data 0000H-3FFFH or label (μPD75P0076)
caddr	12-bit immediate data or label
faddr	11-bit immediate data or label
taddr	Immediate data 20H-7FH (where bit0 = 0) or label
PORTn	PORT0-PORT6, PORT11
IE×××	IEBT, IET0, IET1, IE0-IE2, IE4, IECSI, IEW
RBn	RB0-RB3
MBn	MB0, MB1, MB15

Note mem can be described only for an even address for 8-bit data processing.

(2) Legend for explanation of operation

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'	: Expansion register pair (XA')
BC'	: Expansion register pair (BC')
DE'	: Expansion register pair (DE')
HL'	: Expansion 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 = 0-6, 11)
IME	: Interrupt master enable flag
IPS	: Interrupt priority select register
IE×××	: Interrupt enable flag
RBS	: Register bank select flag
MBS	: Memory bank select flag
PCC	: Processor clock control register
.	: Address or bit delimiter
(××)	: Contents addressed by ××
××H	: Hexadecimal data

(3) Symbols in addressing area field

<table>
<tr><td>*1</td><td colspan="2">MB = MBE·MBS
(MBS = 0, 1, 15)</td><td rowspan="4">Data memory addressing</td></tr>
<tr><td>*2</td><td colspan="2">MB = 0</td></tr>
<tr><td>*3</td><td colspan="2">MBE = 0 : MB = 0 (000H-07FH)
MB = 15 (F80H-FFFH)
MBE = 1 : MB = MBS (MBS = 0, 1, 15)</td></tr>
<tr><td>*4</td><td colspan="2">MB = 15, fmem = FB0H-FBFH,
FF0H-FFFH</td></tr>
<tr><td>*5</td><td colspan="2">MB = 15, pmem = FC0H-FFFH</td><td rowspan="14">Program memory addressing</td></tr>
<tr><td rowspan="4">*6</td><td>μPD750064</td><td>addr, addr1 = 000H-FFFH</td></tr>
<tr><td>μPD750066</td><td>addr, addr1 = 0000H-17FFH</td></tr>
<tr><td>μPD750068</td><td>addr, addr1 = 0000H-1FFFH</td></tr>
<tr><td>μPD75P0076</td><td>addr, addr1 = 0000H-3FFFH</td></tr>
<tr><td>*7</td><td colspan="2">addr, addr1 = (Current PC) – 15 to (Current PC) –1
(Current PC) + 2 to (Current PC) +16</td></tr>
<tr><td rowspan="4">*8</td><td>μPD750064</td><td>caddr = 000H-FFFH</td></tr>
<tr><td>uPD750066</td><td>caddr = 0000H-0FFFH (PC_{12} = 0) or
1000H-17FFH (PC_{12} = 1)</td></tr>
<tr><td>μPD750068</td><td>caddr = 0000H-0FFFH (PC_{12} = 0) or
1000H-1FFFH (PC_{12} = 1)</td></tr>
<tr><td>μPD75P0076</td><td>caddr = 0000H-0FFFH ($PC_{13, 12}$ = 00B) or
1000H-1FFFH ($PC_{13, 12}$ = 01B) or
2000H-2FFFH ($PC_{13, 12}$ = 10B) or
3000H-3FFFH ($PC_{13, 12}$ = 11B)</td></tr>
<tr><td>*9</td><td colspan="2">faddr = 000H-07FFH</td></tr>
<tr><td>*10</td><td colspan="2">taddr = 0020H-007FH</td></tr>
<tr><td>*11</td><td colspan="2">MkII mode only
addr1 = 0000H-0FFFH (μPD750064)
0000H-17FFH (μPD750066)
0000H-1FFFH (μPD750068)
0000H-3FFFH (μPD75P0076)</td></tr>
</table>

Remarks 1. MB indicates a memory bank that can be accessed.

2. In *2, MB = 0 regardless of MBE and MBS.

3. In *4 and *5, MB = 15 regardless of MBE and MBS.

4. *6 through *11 indicate areas that can be addressed.

(4) Explanation for machine cycle field

S indicates the number of machine cycles required for an instruction with skip to execute the skip operation. The value of S varies as follows:

- When skip is executed .... S = 0
- When 1- or 2-byte instruction is skipped .... S = 1
- When 3-byte instructionNote is skipped .... S = 2

Note 3-byte instructions: BR !addr, BRA !addr1, CALL !addr, CALLA !addr1

Caution The GETI instruction is skipped in one machine cycle.

One machine cycle is equal to one cycle of CPU clock Φ (=t_{CY}), and four times can be set by PCC (refer to **Fig. 5-12 Format of Processor Clock Control Register**).

Instructions	Mnemonic	Operand	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	
		@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'		

Instructions	Mnemonic	Operand	Bytes	Machine Cycle	Operation	Addressing Area	Skip Condition
Table reference	**MOVT**	XA, @PCDE	1	3	• μPD750064 XA ← (PC_{11-8} + DE)$_{ROM}$		
					• μPD750066, 750068 XA ← (PC_{12-8} + DE)$_{ROM}$		
					• μPD75P0076 XA ← (PC_{13-8} + DE)$_{ROM}$		
		XA, @PCXA	1	3	• μPD750064 XA ← (PC_{11-8} + XA)$_{ROM}$		
					• μPD750066, 750068 XA ← (PC_{12-8} + XA)$_{ROM}$		
					• μPD75P0076 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	
Operation	**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', 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 When using the μPD750064, set “0” in the B register.
When using the μPD750066, and 750068, only the lower 1 bit of the B register is valid.
When using the μPD75P0076, only the lower 2 bits of the B register are valid.

Instructions	Mnemonic	Operand	Bytes	Machine Cycle	Operation	Addressing Area	Skip Condition
Operation	**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 manipula-tion	**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 ← $\overline{CY}$		

Instructions	Mnemonic	Operand	Bytes	Machine Cycle	Operation	Addressing Area	Skip Condition
Memory bit manipula-tion	**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(mem.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	

Instructions	Mnemonic	Operand	Bytes	Machine Cycle	Operation	Addressing Area	Skip Condition
Branch	BR	addr	–	–	• μPD750064 $PC_{11-0} \leftarrow addr$ (Optimum instruction is selected by assembler from following: • BR !addr • BRCB ! caddr • BR $addr)	*6	
					• μPD750066, 750068 $PC_{12-0} \leftarrow addr$ (Optimum instruction is selected by assembler from following: • BR ! addr • BRCB ! caddr • BR $addr)		
					• μPD75P0076 $PC_{13-0} \leftarrow addr$ (Optimum instruction is selected by assembler from following: • BR ! addr • BRCB ! caddr • BR $addr)		

Instructions	Mnemonic	Operand	Bytes	Machine Cycle	Operation	Addressing Area	Skip Condition
Branch	BR[Note]	addr1	–	–	• μPD750064 PC_{11-0} ← addr1 Optimum instruction is selected by assembler from following: • BR !addr • BRA ! addr1 • BRCB ! caddr • BR $addr1	*11	
					• μPD750066, 750068 PC_{12-0} ← addr1 Optimum instruction is selected by assembler from following: • BR ! addr • BRA ! addr1 • BRCB ! caddr • BR $addr1		
					• μPD75P0076 PC_{13-0} ← addr1 Optimum instruction is selected by assembler from following: • BR ! addr • BRA ! addr1 • BRCB ! caddr • BR $addr1		
		! addr	3	3	• μPD750064 PC_{11-0} ← addr	*6	
					• μPD750066, 750068 PC_{12-0} ← addr		
					• μPD75P0076 PC_{13-0} ← addr		
		$addr	1	2	• μPD750064 PC_{11-0} ← addr	*7	
					• μPD750066, 750068 PC_{12-0} ← addr		
					• μPD75P0076 PC_{13-0} ← addr		

Note The shaded area is applicable only to the Mk II mode. The rest is applicable only to the Mk I mode.

Instructions	Mnemonic	Operand	Bytes	Machine Cycle	Operation	Addressing Area	Skip Condition
Branch	BR[Note 1]	$addr1	1	2	• μPD750064 $PC_{11-0} \leftarrow$ addr1	*7	
					• μPD750066, 750068 $PC_{12-0} \leftarrow$ addr1		
					• μPD75P0076 $PC_{13-0} \leftarrow$ addr1		
		PCDE	2	3	• μPD750064 $PC_{11-0} \leftarrow PC_{11-8}$ + DE		
					• μPD750066, 750068 $PC_{12-0} \leftarrow PC_{12-8}$ + DE		
					• μPD75P0076 $PC_{13-0} \leftarrow PC_{13-8}$ + DE		
		PCXA	2	3	• μPD750064 $PC_{11-0} \leftarrow PC_{11-8}$ + XA		
					• μPD750066, 750068 $PC_{12-0} \leftarrow PC_{12-8}$ + XA		
					• μPD75P0076 $PC_{13-0} \leftarrow PC_{13-8}$ + XA		
		BCDE	2	3	• μPD750064 $PC_{11-0} \leftarrow$ BCDE[Note 2]	*6	
					• μPD750066, 750068 $PC_{12-0} \leftarrow$ BCDE[Note 3]		
					• μPD75P0076 $PC_{13-0} \leftarrow$ BCDE[Note 4]		
		BCXA	2	3	• μPD750064 $PC_{11-0} \leftarrow$ BCXA[Note 2]	*6	
					• μPD750066, 750068 $PC_{12-0} \leftarrow$ BCXA[Note 3]		
					• μPD75P0076 $PC_{13-0} \leftarrow$ BCXA[Note 4]		

Notes
1. The shaded area is applicable only to the Mk II mode. The rest is applicable only to the Mk I mode.
2. Be sure to place “0” in the B register.
3. Only the lower 1 bit of the B register is valid.
4. Only the lower 2 bits of the B register are valid.

Instructions	Mnemonic	Operand	Bytes	Machine Cycle	Operation	Addressing Area	Skip Condition
Branch	BRA[Note]	! addr1	3	3	• μPD750064 $PC_{11-0} \leftarrow$ addr1	*11	
					• μPD750066, 750068 $PC_{12-0} \leftarrow$ addr1		
					• μPD75P0076 $PC_{13-0} \leftarrow$ addr1		
	BRCB	! caddr	2	3	• μPD750064 $PC_{11-0} \leftarrow caddr_{11-0}$	*8	
					• μPD750066, 750068 $PC_{12-0} \leftarrow caddr_{11-0}$		
					• μPD75P0076 $PC_{13-0} \leftarrow caddr_{11-0}$		
Subroutine/ stack control	CALLA[Note]	! addr1	3	3	• μPD750064 (SP–2) ← ×, ×, MBE, RBE (SP–6) (SP–3) (SP–4) ← PC_{11-0} (SP–5) ← 0, 0, 0, 0 $PC_{11-0} \leftarrow$ addr1, SP ← SP–6	*11	
					• μPD750066, 750068 (SP–2) ← ×, ×, MBE, RBE (SP–6) (SP–3) (SP–4) ← PC_{11-0} (SP–5) ← 0, 0, 0, PC_{12} $PC_{12-0} \leftarrow$ addr1, SP ← SP–6		
					• μPD75P0076 (SP–2) ← ×, ×, MBE, RBE (SP–6) (SP–3) (SP–4) ← PC_{11-0} (SP–5) ← 0, 0, $PC_{13, 12}$ $PC_{13-0} \leftarrow$ addr1, SP ← SP–6		

Note The shaded area is applicable only to the Mk II mode. The rest is applicable only to the Mk I mode.

Instructions	Mnemonic	Operand	Bytes	Machine Cycle	Operation	Addressing Area	Skip Condition
Subroutine/ stack control	CALL[Note]	! addr	3	3	• μPD750064 (SP–3) ← MBE, RBE, 0, 0 (SP–4) (SP–1) (SP–2) ← PC_{11-0} PC_{11-0} ← addr, SP ← SP–4	*6	
					• μPD750066, 750068 (SP–3) ← MBE, RBE, 0, PC_{12} (SP–4) (SP–1) (SP–2) ← PC_{11-0} PC_{12-0} ← addr, SP ← SP–4		
					• μPD75P0076 (SP–2) ← MBE, RBE, $PC_{13, 12}$ (SP–4) (SP–1) (SP–2) ← PC_{11-0} PC_{13-0} ← addr, SP ← SP–4		
				4	• μPD750064 (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		
					• μPD750066, 750068 (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		
					• μPD75P0076 (SP–2) ← X, X, MBE, RBE (SP–6) (SP–3) (SP–4) ← PC_{11-0} (SP–5) ← 0, 0, $PC_{13, 12}$ PC_{13-0} ← addr, SP ← SP–6		

Note The shaded area is applicable only to the Mk II mode. The rest is applicable only to the Mk I mode.

Instructions	Mnemonic	Operand	Bytes	Machine Cycle	Operation	Addressing Area	Skip Condition
Subroutine/ stack control	CALLF[Note]	! faddr	2	2	• μPD750064 (SP–3) ← MBE, RBE, 0, 0 (SP–4) (SP–1) (SP–2) ← PC_{11-0} PC_{11-0} ← 0 + faddr, SP ← SP–4	*9	
					• μPD750066, 750068 (SP–3) ← MBE, RBE, 0, PC_{12} (SP–4) (SP–1) (SP–2) ← PC_{11-0} PC_{12-0} ← 00 + faddr, SP ← SP–4		
					• μPD75P0076 (SP–3) ← MBE, RBE, $PC_{13, 12}$ (SP–4) (SP–1) (SP–2) ← PC_{11-0} PC_{13-0} ← 000 + faddr, SP ← SP–4		
				3	• μPD750064 (SP–2) ← ×, ×, 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		
					• μPD750066, 750068 (SP–2) ← ×, ×, 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		
					• μPD75P0076 (SP–2) ← ×, ×, MBE, RBE (SP–6) (SP–3) (SP–4) ← PC_{11-0} (SP–5) ← 0, 0, $PC_{13, 12}$ PC_{13-0} ← 000 + faddr, SP ← SP–6		

Note The shaded area is applicable only to the Mk II mode. The rest is applicable only to the Mk I mode.

Instructions	Mnemonic	Operand	Bytes	Machine Cycle	Operation	Addressing Area	Skip Condition
Subroutine/ stack control	RET[Note]		1	3	• μPD750064 PC_{11-0} ← (SP) (SP+3) (SP+2) MBE, RBE, 0, 0 ← (SP+1), SP ← SP+4		
					• μPD750066, 750068 PC_{11-0} ← (SP) (SP+3) (SP+2) MBE, RBE, 0, PC_{12} ← (SP+1), SP ← SP+4		
					• μPD75P0076 PC_{11-0} ← (SP) (SP+3) (SP+2) MBE, RBE, $PC_{13, 12}$ ← (SP+1) SP ← SP+4		
					• μPD750064 ×, ×, MBE, RBE ← (SP+4) 0, 0, 0, 0 ← (SP+1) PC_{11-0} ← (SP) (SP+3) (SP+2), SP ← SP+6		
					• μPD750066, 750068 ×, ×, MBE, RBE ← (SP+4) 0, 0, 0, PC_{12} ← (SP+1) (SP–5) ← 0, 0, 0, PC_{12} PC_{11-0} ← (SP) (SP+3) (SP+2), SP ← SP+6		
					• μPD75P0076 ×, ×, MBE, RBE ← (SP+4) 0, 0, $PC_{13, 12}$ ← (SP+1) PC_{11-0} (SP) (SP+3) (SP+2) SP ← SP+6		

Note The shaded area is applicable only to the Mk II mode. The rest is applicable only to the Mk I mode.

Instructions	Mnemonic	Operand	Bytes	Machine Cycle	Operation	Addressing Area	Skip Condition
Subroutine/ stack control	RETS[Note]		1	3+S	• μPD750064 MBE, RBE, 0, 0 ← (SP+1) PC_{11-0} ← (SP) (SP+3) (SP+2) SP ← SP+4 the skip unconditionally		
					• μPD750066, 750068 MBE, RBE, 0, PC_{12} ← (SP+1) PC_{11-0} ← (SP) (SP+3) (SP+2) SP ← SP+4 the skip unconditionally		
					• μPD75P0076 MBE, RBE, $PC_{13, 12}$ ← (SP+1) PC_{11-0} ← (SP) (SP+3) (SP+2) SP ← SP+4 the skip unconditionally		
					• μPD750064 0, 0, 0, 0 ← (SP+1) PC_{11-0} ← (SP) (SP+3) (SP+2) ×, ×, MBE, RBE ← (SP+4) SP ← SP+6 the skip unconditionally		
					• μPD750066, 750068 0, 0, 0, PC_{12} ← (SP+1) PC_{11-0} ← (SP) (SP+3) (SP+2) ×, ×, MBE, RBE ← (SP+4) SP ← SP+6 the skip unconditionally		
					• μPD75P0076 0, 0, $PC_{13, 12}$ ← (SP+1) PC_{11-0} ← (SP) (SP+3) (SP+2) ×, ×, MBE, RBE ← (SP+4) SP ← SP+6 the skip unconditionally		

Note The shaded area is applicable only to the Mk II mode. The rest is applicable only to the Mk I mode.

Instructions	Mnemonic	Operand	Bytes	Machine Cycle	Operation	Addressing Area	Skip Condition
Subroutine/ stack control	RETI[Note]		1	3	• μPD750064 MBE, RBE, 0, 0 ← (SP+1) PC_{11-0} ← (SP) (SP+3) (SP+2) PSW ← (SP+4) (SP+5), SP ← SP+6		
					• μPD750066, 750068 MBE, RBE, 0, PC_{12} ← (SP+1) PC_{11-0} ← (SP) (SP+3) (SP+2) PSW ← (SP+4) (SP+5), SP ← SP+6		
					• μPD75P0076 MBE, RBE, 0, $PC_{13,\ 12}$ ← (SP+1) PC_{11-0} ← (SP) (SP+3) (SP+2) PSW ← (SP+4) (SP+5), SP ← SP+6		
					• μPD750064 0, 0, 0, 0 ← (SP+1) PC_{11-0} ← (SP) (SP+3) (SP+2) PSW ← (SP+4) (SP+5), SP ← SP+6		
					• μPD750066, 750068 0, 0, 0, PC_{12} ← (SP+1) PC_{11-0} ← (SP) (SP+3) (SP+2) PSW ← (SP+4) (SP+5), SP ← SP+6		
					• μPD75P0076 0, 0, $PC_{13,\ 12}$ ← (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		
		IE×××	2	2	IE××× ← 1		
	DI		2	2	IME (IPS, 3) ← 0		
		IE×××	2	2	IE××× ← 0		

Note The shaded area is applicable only to the Mk II mode. The rest is applicable only to the Mk I mode.

Instructions	Mnemonic	Operand	Bytes	Machine Cycle	Operation	Addressing Area	Skip Condition
I/O	IN[Note]	A, PORTn	2	2	A ← PORTn (n = 0–6, 11)		
		XA, PORTn	2	2	XA ← PORTn+1, PORTn (n = 4)		
	OUT[Note]	PORTn, A	2	2	PORTn ← A (n = 2–6)		
		PORTn, XA	2	2	PORTn+1, PORTn ← XA (n = 4)		
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		
Special	SEL	RBn	2	2	RBS ← n (n = 0–3)		
		RBn	2	2	MBS ← n (n = 0, 1, 15)		

Note To execute IN/OUT instruction, it is necessary that MBS = 0 or MBE = 1, MBS = 15.

Instructions	Mnemonic	Operand	Bytes	Machine Cycle	Operation	Addressing Area	Skip Condition
Special	GETI[Note]	taddr	1	3	• μPD750064 With TBR instruction PC_{11-0} (taddr) $_{3-0}$ + (taddr+1)	*10	
					With TCALL instruction (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		
					Other than TBR and TCALL instructions Executes instruction of (taddr) (taddr+1)		Depends on referenced instruction
					• μPD750066, 750068 With TBR instruction PC_{12-0} (taddr) $_{4-0}$ + (taddr+1)		
					With TCAR instruction (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		
					Other than TBR and TCALL instructions Executes instruction of (taddr) (taddr+1)		Depends on referenced instruction
					• μPD75P0076 With TBR instruction PC_{13-0} (taddr) $_{5-0}$ + (taddr+1)		
					With TCAR instruction (SP–4) (SP–1) (SP–2) ← PC_{11-0} (SP–3) ← MBE, RBE, $PC_{13, 12}$ PC_{13-0} ← (taddr) $_{5-0}$ + (taddr+1) SP ← SP–4		
					Other than TBR and TCALL instructions Executes instruction of (taddr) (taddr+1)		Depends on referenced instruction

Note The TBR and TCALL instructions are the assembler directives for table definition for the GETI instruction.

Instructions	Mnemonic	Operand	Bytes	Machine Cycle	Operation	Addressing Area	Skip Condition
Special	GETI[Note1, 2]	taddr	1	3	• μPD750064 With TBR instruction PC_{11-0} (taddr)$_{3-0}$ + (taddr+1)	*10	
				4	With TCALL instruction (SP–6) (SP–3) (SP–4) ← PC_{11-0} (SP–5) ← 0, 0, 0, 0 (SP–2) ← ×, ×, MBE, RBE PC_{11-0} ← (taddr)$_{3-0}$ + (taddr+1) SP ← SP–6		
				3	Other than TBR and TCALL instructions Executes instruction of (taddr) (taddr+1)		Depends on referenced instruction
				3	• μPD750066, 750068 With TBR instruction PC_{12-0} (taddr)$_{4-0}$ + (taddr+1)		
				4	With TCAR instruction (SP–6) (SP–3) (SP–4) ← PC_{11-0} (SP–5) ← 0, 0, 0, PC_{12} (SP–2) ← ×, ×, MBE, RBE PC_{12-0} ← (taddr)$_{4-0}$ + (taddr+1) SP ← SP–6		
				3	Other than TBR and TCALL instructions Executes instruction of (taddr) (taddr+1)		Depends on referenced instruction
				3	• μPD75P0076 With TBR instruction PC_{13-0} (taddr)$_{5-0}$ + (taddr+1)		
				4	With TCAR instruction (SP–6) (SP–3) (SP–4) ← PC_{11-0} (SP–5) ← 0, 0, $PC_{13, 12}$ (SP–2) ← ×, ×, MBE, RBE PC_{13-0} ← (taddr)$_{5-0}$ + (taddr+1) SP ← SP–6		
				3	Other than TBR and TCALL instructions Executes instruction of (taddr) (taddr+1)		Depends on referenced instruction

Notes 1. The TBR and TCALL instructions are the assembler directives for table definition for the GETI instruction.

2. The shaded area is applicable only to the Mk II mode. The rest is applicable only to the Mk I mode.

11.3 Op Code of Each Instruction

(1) Description of symbol of op code

R_2	R_1	R_0	reg
0	0	0	A
0	0	1	X
0	1	0	L
0	1	1	H
1	0	0	E
1	0	1	D
1	1	0	C
1	1	1	B

reg reg1

P_2	P_1	P_0	reg-pair
0	0	0	XA
0	0	1	XA'
0	1	0	HL
0	1	1	HL'
1	0	0	DE
1	0	1	DE'
1	1	0	BC
1	1	1	BC'

rp' rp'1

Q_2	Q_1	Q_0	addressing
0	0	0	@HL
0	1	0	@HL+
0	1	1	@HL–
1	0	0	@DE
1	0	1	@DL

@rpa @rpa1

P_2	P_1	reg-pair
0	0	XA
0	1	HL
1	0	DE
1	1	BC

rp2 rp1 rp

N_5	N_2	N_1	N_0	IE×××
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 IE×××
T_n : immediate data for taddr × 1/2
A_n : immediate data for [relative address distance from branch destination address (2-16)] – 1
S_n : immediate data for 1's complement of [relative address distance from branch destination address (15-1)]

(2) Op code for bit manipulation addressing

*1 in the operand field indicates the following three types:

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

The second byte *2 of the op code corresponding to the above addressing is as follows:

*1	2nd Byte of Op Code								Accessible Bit
fmem. bit	1	0	B_1	B_0	F_3	F_2	F_1	F_0	Bit of FB0H-FBFH that can be manipulated
	1	1	B_1	B_0	F_3	F_2	F_1	F_0	Bit of FF0H-FFFH that can be manipulated
pmem. @L	0	1	0	0	G_3	G_2	G_1	G_0	Bit of FC0H-FFFH that can be manipulated
@H+mem. bit	0	0	B_1	B_0	D_3	D_2	D_1	D_0	Bit of accessible memory bank that can be manipulated

B_n : immediate data for bit

F_n : immediate data for fmem
(indicates lower 4 bits of address)

G_n : immediate data for pmem
(indicates bits 5-2 of address)

D_n : immediate data for mem

Instruction	Mnemonic	Operand	Op 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, *1	1 0 1 1 1 1 0 1	*2	
		*1, CY	1 0 0 1 1 0 1 1	*2	

Instruction	Mnemonic	Operand	Op Code		
			B_1	B_2	B_3
Operation	**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	Mnemonic	Operand	Op 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 P_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	
		*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	
		*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	
		*1	1 0 0 1 1 1 0 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	
		*1	1 0 1 1 1 1 1 0	*2	
	SKTCLR	*1	1 0 0 1 1 1 1 1	*2	
	AND1	CY, *1	1 0 1 0 1 1 0 0	*2	
	OR1	CY, *1	1 0 1 0 1 1 1 0	*2	
	XOR1	CY, *1	1 0 1 1 1 1 0 0	*2	

Instruction	Mnemonic	Operand	Op Code B1	Op Code B2	Op Code B3
Branch	**BR**	!addr	1 0 1 0 1 0 1 1	0 0 ←	addr →
		\$addr1 (+16) ~ (+2)	0 0 0 0 A_3 A_2 A_1 A_0		
		(–1) ~ (–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	1 0 0 1 1 0 0 1	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 →	
Subroutine/stack	**CALLA**	!addr1	1 0 1 1 1 0 1 1	0 ←	addr1 →
	CALL	!addr	1 0 1 0 1 0 1 1	0 1 ←	addr →
	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	
Interrupt control	**EI**		1 0 0 1 1 1 0 1	1 0 1 1 0 0 1 0	
		IE×××	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	
		IE×××	1 0 0 1 1 1 0 1	1 0 N_5 1 1 N_2 N_1 N_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	
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 1 1 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 Instruction Function and Application

This section describes the functions and applications of the respective instructions. The instructions that can be used and the functions of the instructions differ between the MkI and MkII modes of the μPD750064, 750066, 750068, and 75P0076. Read the descriptions on the following pages according to the following guidance:

How to read

◯ : This instruction can be used commonly to all the following:

μPD750064
μPD750066
μPD750068
μPD75P0076
} In MkI and MkII modes

(I) : This instruction can be used only in the MkI mode of the μPD750064, 750066, 750068, and 753P0076.

(II) : This instruction can be used only in the MkII mode of the μPD750064, 750066, 750068, and 75P0076.

(I/II) : This instruction can be used commonly in the MkI and MkII modes of the μPD750064, 750066 and 750068, and 75P0076, but the function may differ between the MkI and MkII modes.
In the MkI mode, refer to the description under the heading [MkI mode]. In the MkII mode, read the description under the heading [MkII mode].

Remark The functions described in this section are explained by using the μPD750066, and 750068, whose program counter has 13 bits, as an example. When this manual is used for the μPD750064 or μPD75P0076, whose program counters have 12 bits and 14 bits respectively, please make the required substitutions.

11.4.1 Transfer instructions

◯ MOV A, #n4

Function: A ← n4 n4 = I_{3-0}: 0-FH

Transfers 4-bit immediate data n4 to the A register (4-bit accumulator). This instruction has a string effect (group A), and if MOV A, #n4 or MOV XA, #n8 follows this instruction, the string-effect instruction following the instruction executed is processed as NOP.

Application example

(1) To set 0BH to the accumulator

```
MOV A, #0BH
```

(2) To select data output to port 3 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_{3-0} 0-FH

Transfers 4-bit immediate data n4 to A register reg1 (X, H, L, D, E, B, or C).

◯ MOV XA, #n8

Function: XA ← n8 n8 = I_{7-0}: 00H-FFH

Transfers 8-bit immediate data n8 to register pair XA. This instruction has a string effect and if the same instruction or the MOV A, #n4 follows this instruction, the string-effect instruction following the instruction executed is processed as NOP.

◯ MOV HL, #n8

Function: HL ← n8 n8 = I_{7-0}: 00H-FFH

Transfers 8-bit immediate data n8 to register pair HL. This instruction has a string effect and if the same instruction follows this instruction, the string-effect instruction following the instruction executed is processed as NOP.

◯ MOV rp2, #n8

Function: np2 ← n8 n8 = I_{7-0}: 00H-FFH

Transfers 8-bit immediate data n8 to register pair rp2 (BC, DE).

◯ MOV A, @HL

Function: A ← (HL)

Transfers the data memory content addressed by the register pair HL to the A register.

◯ MOV A, @HL+

Function: A ← (HL), L ← L + 1
skip if L = 0H

Transfers the data memory content addressed by the register pair HL to the A register. Then automatically increments (+1) the L register content and skips the following instruction if the result is 0H.

◯ MOV A, @HL–

Function: A ← (HL), L ← L – 1
skip if L = FH

Transfers the data memory content addressed by the register pair HL to the A register. Then automatically decrements (–1) the L register content and skips the following instruction if the result is FH.

◯ MOV A, @rpa1

Function: A ← (rpa)
Where rpa = HL+: skip if L = 0
where rpa = HL–: skip if L = FH

Transfers the contents of the data memory addressed by register pair rpa (HL, HL+, HL–, DE, or DL) to the A register.

If autoincrement (HL+) is specified as rpa, the contents of the L register are automatically incremented by one after the data has been transferred. If the contents of the L register become 0 as a result, the next one instruction is skipped.

If autodecrement (HL–) is specified as rpa, the contents of the L register are automatically decremented by one after the data has been transferred. If the contents of the L register become FH as a result, the next one instruction is skipped.

⬭ MOV XA, @HL

Function: A ← (HL), X ← (HL+1)

Transfers the contents of the data memory addressed by register pair HL to the A register, and the contents of the next memory address to the X register.

If the contents of the L register are a odd number, an address whose least significant bit is ignored is transferred.

Application example

To transfer the data at addresses 3EH and 3FH to register pair XA

```
MOV HL, #3EH
MOV XA, @HL
```

⬭ MOV @HL, A

Function: (HL) ← A

Transfers the contents of the A register to the data memory addressed by register pair HL.

⬭ MOV @HL, XA

Function: (HL) ← A, (HL+1) ← X

Transfers the contents of the A register to the data memory addressed by register pair HL, and the contents of the X register to the next memory address.

However, if the contents of the L register are a odd number, an address whose least significant bit is ignored is transferred.

⬭ MOV A, mem

Function: A ← (mem) mem = D_{7-0}: 00H-FFH

Transfers the contents of the data memory addressed by 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 contents of the data memory addressed by 8-bit immediate data mem to the A register and the contents of the next address to the X register.

The address that can be specified by mem is an even address.

Application example

To transfer the data at addresses 40H and 41H to register pair XA

```
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 addressed by 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 addressed by 8-bit immediate data mem and the contents of the X register to the next memory address.

The address that can be specified by mem is an even address.

◯ MOV A, reg

Function: A ← reg

Transfers the contents of register reg (X, A, H, L, D, E, B, or 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', or BC') to register pair XA.

Application example

To transfer the data of register pair XA' to register pair XA

```
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, or C).

⬭ MOV rp'1, XA

Function: rp'1 ← XA

Transfers the contents of register pair XA to register pair rp'1 (HL, DE, BC, XA', HL', DE', or BC').

◯ XCH A, @HL

Function: A ↔ (HL)

Exchanges the contents of the A register with the contents of the data memory addressed by register pair HL.

◯ XCH A, @HL+

Function: A ↔ (HL), L ← L + 1
skip if L = 0H

Exchanges the contents of the A register with the contents of the data memory addressed by register pair HL. Then automatically increments (+1) the L register content and skips the following instruction if the result is 0H.

◯ XCH A, @HL–

Function: A ↔ (HL), L ← L – 1
skip if L = FH

Exchanges the contents of the A register with the contents of the data memory addressed by register pair HL. Then automatically decrements (–1) the L register content and skips the following instruction if the result is FH.

◯ XCH A, @rpa1

Function: A ↔ (rpa)
Where rpa = HL+: skip if L = 0
Where rpa = HL–: sKIP if L = FH

Exchanges the contents of the A register with the contents of the data memory addressed by register pair rpa (HL, HL+, HL–, DE, or DL). If autoincrement (HL+) or autodecrement (HL–) is specified as rpa, the contents of the L register are automatically incremented or decremented by one after the data have been exchanged. If the result is 0 in the case of HL+ and FH in the case of HL–, the next one instruction is skipped.

Application example

To exchange the data at data memory addresses 20H through 2FH with the data at addresses 30H through 3FH

```
        SEL    MB0
        MOV    D, #2
        MOV    HL, #30H
LOOP:   XCH    A, @HL    ; A ↔ (3×)
        XCH    A, @DL    ; A ↔ (2×)
        XCH    A, @HL+   ; A ↔ (3×)
        BR     LOOP
```

◯ XCH XA, @HL

Function: A ↔ (HL), X ↔ (HL+1)

Exchanges the contents of the A register with the contents of the data memory addressed by register pair HL, and the contents of the X register with the contents of the next address.

If the contents of the L register are an odd number, however, an address whose least significant bit is ignored is specified.

◯ XCH A, mem

Function: A ↔ (mem) mem = D_{7-0}: 00H-FEH

Exchanges the contents of the A register with the contents of the data memory addressed by 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 memory contents addressed by 8-bit immediate data mem, and the contents of the X register with the contents of the next memory address.

The address that can be specified by mem is an even address.

◯ XCH A, reg1

Function: A ↔ reg1

Exchanges the contents of the A register with the contents of register reg1 (X, H, L, D, E, B, or C).

◯ XCH XA, rp’

Function: XA ↔ rp’

Exchanges the contents of register pair XA with the contents of register pair rp’ (XA, HL, DE, BC, XA’, HL’, DE’, or BC’).

11.4.2 Table reference instruction

◯ MOVT XA, @PCDE

Function: In the case the μPD750066, 750068: XA ← ROM (PC_{12-8}+DE)

Transfers the lower 4 bits of the table data in the program memory addressed when the lower 8 bits (PC_{7-0}) of the program counter (PC) are replaced with the contents of register pair DE, to the A register, and the higher 4 bits to the X register.

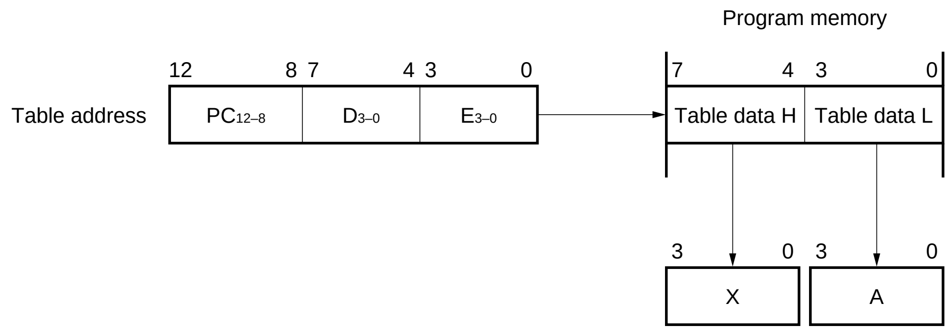
The table address is determined by the contents of the program counter (PC) when this instruction is executed.

The necessary data must be programmed to the table area in advance by using an assembler directive (DB instruction).

The program counter is not affected by execution of this instruction.

This instruction is useful for successively referencing table data.

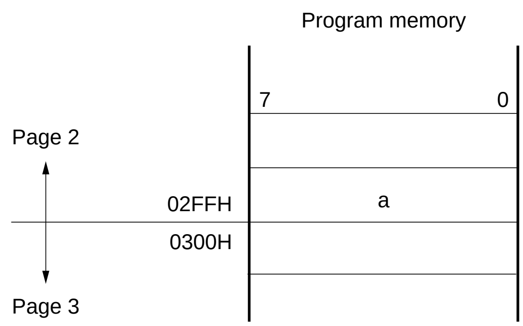
Example In the case of the μPD750066, 750068:



Remark The functions described in this section are explained by using the μPD750066 and 750068, whose program counter has 13 bits, as an example. When this manual is used for the μPD750064 and μPD75P0076, whose program counters have 12 bits and 14 bits respectively, please make the required substitutions.

Caution

The MOVT XA, @PCDE instruction usually references the table data in page where the instruction exists. If the instruction is at address ××FFH, however, not the table data in the page where the instruction exists, but the table data in the next page is referenced.



For example, if the MOVT XA, @PCDE instruction is located at position a in the above figure, the table data in page 3, not page 2, specified by the contents of register pair DE is transferred to register pair XA.

Application example

To transfer the 16-byte data at program memory addresses ××F0H through ××FFH to data memory addresses 30H through 4FH

```
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     ××F0H
        DB      ××H, ××H, ...   ; table data
```

◯ MOVT XA, @PCXA

Function: In the case of the μPD750066, 750068: XA ← ROM ($PC_{12\text{-}8}$+XA)

Transfers the lower 4 bits of the table data in the program memory addressed when the lower 8 bits ($PC_{7\text{-}0}$) of the program counter (PC) are replaced with the contents of register pair XA, to the A register, and the higher 4 bits to the X register.

The table address is determined by the contents of the PC when this instruction is executed.

The necessary data must be programmed to the table area in advance by using an assembler directive (DB instruction).

The PC is not affected by execution of this instruction.

Caution If an instruction exists at address ××FFH, the table data of the next page is transferred, in the same manner as MOVT XA, @PCDE.

Remark The functions described in this section are explained by using the μPD750066 and 750068, whose program counter has 13 bits, as an example. When this manual is used for the μPD750064 and μPD75P0076, whose program counters have 12 bits and 14 bits respectively, please make the required substitutions.

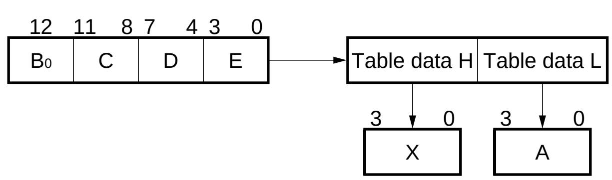
◯ MOVT XA, @BCDE

Function: In the case of the μPD750066, 750068: XA ← $(BCDE)_{ROM}$

Transfers the lower 4 bits of the table data (8-bit) in the program memory addressed by the lower 1 bit of register B and the contents of registers C, D, and E, to the A register, and the higher 4 bits to the X register.

The necessary data must be programmed to the table area in advance by using an assembler directive (DB instruction). The PC is not affected by execution of this instruction.

Example



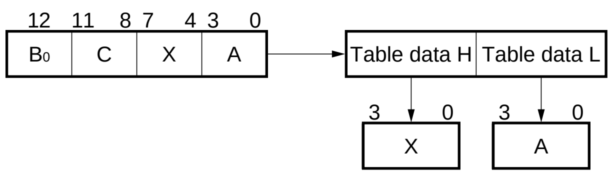
◯ MOVT XA, @BCXA

Function: In the case of the μPD750066, 750068: XA ← $(BCDE)_{ROM}$

Transfers the lower 4 bits of the table data (8-bit) in the program memory addressed by the lower 1 bit of register B and the contents of registers C, X, and A, to the A register, and the higher 4 bits to the X register.

The necessary data must be programmed to the table area in advance by using an assembler directive (DB instruction). The PC is not affected by execution of this instruction.

Example



Remark The functions described in this section are explained by using the μPD750066 and 750068, whose program counter has 13 bits, as an example. When this manual is used for the μPD750064 and μPD75P0076, whose program counters have 12 bits and 14 bits respectively, please make the required substitutions.

11.4.3 Bit transfer instruction

◯ MOV1 CY, fmem.bit

◯ MOV1 CY, pmem.@L

◯ MOV1 CY, @H+mem.bit

Function: CY ← (bit specified by operand)

Transfers the contents of the data memory addressed in the bit manipulating addressing mode (fmem.bit, pmem.@L, or @H+mem.bit) to the carry flag (CY).

◯ MOV1 fmem.bit, CY

◯ MOV1 pmem.@L, CY

◯ MOV1 @H+mem.bit, CY

Function: (Bit specified by operand) ← CY

Transfers the contents of the carry flag (CY) to the data memory bit addressed in the bit manipulation addressing mode (fmem.bit, pmem.@L, or @H+mem.bit).

Application example

To output the flag of bit 3 at data memory address 3FH to the bit 2 of port 3

```
FLAG    EQU 3FH.3
SEL     MB0
MOV     H, #FLAG SHR6  ; H ← higher 4 bits of FLAG
MOV1    CY, @H+FLAG    ; CY ← FLAG
MOV1    PORT3.2, CY    ; P32 ← CY
```

11.4.4 Operation instruction

◯ ADDS A, #n4

Function: A ← A+n4; Skip if carry. n4 = I_{3-0}: 0-FH

Adds 4-bit immediate data n4 to the contents of the A register. If a carry occurs as a result, the next one instruction is skipped. The carry flag is not affected.

If this instruction is used in combination with ADDC A, @HL or SUBC A, @HL instruction, it can be used as a base number adjustment instruction (refer to **11.1.4 Base number adjustment instruction**).

◯ ADDS XA, #n8

Function: XA ← XA+n8; Skip if carry. n8 = I_{7-0}: 00H-FFH

Adds 8-bit immediate data n8 to the contents of register pair XA. If a carry occurs as a result, the next one instruction is skipped. The carry flag is not affected.

◯ ADDS A, @HL

Function: A ← A + (HL); Skip if carry.

Adds the contents of the data memory addressed by register pair HL to the contents of the A register. If a carry occurs as a result, the next one instruction is skipped. 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', or BC') to the contents of register pair XA. If a carry occurs as a result, the next one instruction is skipped. The carry flag is not affected.

◯ ADDS rp'1, XA

Function: rp' ← rp'1 + XA; Skip if carry.

Adds the contents of register pair XA to register pair rp'1 (HL, DE, BC, XA', HL', DE', or BC'). If a carry occurs as a result, the next one instruction is skipped. The carry flag is not affected.

Application example

To shift a register pair to the left

```
MOV     XA, rp'1
ADDS    rp'1, XA
NOP
```

◯ ADDC A, @HL

Function: A, CY ← A+ (HL) +CY

Adds the contents of the data memory addressed by register pair HL to the contents of the A register, including the carry flag. If a carry occurs as a result, the carry flag is set; if not, the carry flag is reset.

If the ADDS A, #n4 instruction is placed next to this instruction, and if a carry occurs as a result of executing this instruction, the ADDS A, #n4 instruction is skipped. If a carry does not occur, the ADDS A, #n4 instruction is executed, and a function that disables the skip function of the ADDS A, #n4 instruction is effected. Therefore, these instructions can be used in combination for base number adjustment (refer to **11.1.4 Base number adjustment instruction**).

◯ ADDC XA, rp'

Function: XA, CY ← XA + rp' + CY

Adds the contents of register pair rp' (XA, HL, DE, BC, XA', HL', DE', or BC') to the contents of register pair XA, including the carry. If a carry occurs as a result, the carry flag is set; if not, the carry flag is reset.

◯ ADDC rp'1, XA

Function: rp'1, CY ← rp'1+XA+CY

Adds the contents of register pair XA to the contents of register pair rp'1 (HL, DE, BC, XA', HL', DE', or BC'), including the carry flag. If a carry occurs as a result, the carry flag is set; if not, the carry flag is reset.

◯ SUBS A, @HL

Function: A ← A – (HL); Skip if borrow.

Subtracts the contents of the data memory addressed by register pair HL from the contents of the A register, and sets the result to the A register. If a borrow occurs as a result, the next one 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’, or BC’) from the contents of register pair XA, and sets the result to register pair XA. If a borrow occurs as a result, the next one instruction is skipped.

The carry flag is not affected.

Application example

To compare specified data memory contents with the contents of a register pair

```
MOV     XA, mem
SUBS    XA, rp’
                ; (mem) ≥ rp’
                ; (mem) < rp’
```

◯ SUBS rp’1, XA

Function: rp’ ← rp’1 – XA; Skip if borrow.

Subtracts the contents of register pair XA from register pair rp’1 (HL, DE, BC, XA’, HL’, DE’, or BC’), and sets the result to specified register pair rp’1. If a borrow occurs as a result, the next one instruction is skipped.

The carry flag is not affected.

◯ SUBC A, @HL

Function: A, CY ← A – (HL) – CY

Subtracts the contents of the data memory addressed by register pair HL to the contents from the A register, including the carry flag, and sets the result to the A register. If a borrow occurs as a result, the carry flag is set; if not, the carry flag is reset.

If the ADDS A, #n4 instruction is placed next to this instruction, and if a borrow does not occur as a result of executing this instruction, the ADDS A, #n4 instruction is skipped. If a borrow occurs, the ADDS A, #n4 instruction is executed, and a function that disables the skip function of the ADDS A, #n4 instruction is effected. Therefore, these instructions can be used in combination for base number adjustment (refer to **11.1.4 Base number adjustment instruction**).

◯ SUBC XA, rp'

Function: XA, CY ← XA – rp' – CY

Subtracts the contents of register pair rp' (XA, HL, DE, BC, XA', HL', DE', or BC') from the contents of register pair XA, including the carry, and sets the result to register pair XA. If a borrow occurs as a result, the carry flag is set; if not, the carry flag is reset.

◯ SUBC rp'1, XA

Function: rp'1, CY ← rp'1 – XA – CY

Subtracts the contents of register pair XA from the contents of register pair rp'1 (HL, DE, BC, XA', HL', DE', or BC'), including the carry flag, and sets the result to specified register pair rp'1. If a borrow occurs as a result, the carry flag is set; if not, the carry flag is reset.

◯ AND A, #n4

Function: A ← A ∧ n4   n4 = I_{3-0}: 0-FH

ANDs 4-bit immediate data n4 with the contents of the A register, and sets the result to the A register.

Application example

To clear the higher 2 bits of the accumulator to 0

```
AND A, #0011B
```

◯ AND A, @HL

Function: A ← A ∧ (HL)

ANDs the contents of the data memory addressed by register pair HL with the contents of the A register, and sets the result to the A register.

◯ AND XA, rp’

Function: XA ← XA ∧ rp’

ANDs the contents of register pair rp’ (XA, HL, DE, BC, XA’, HL’, DE’, or BC’) with the contents of register pair XA, and sets the result to register pair XA.

◯ AND rp’1, XA

Function: rp’1 ← rp’1 ∧ XA

ANDs the contents of register pair XA with register pair rp’1 (HL, DE, BC, XA’, HL’, DE’, or BC’), and sets the result to a specified register pair.

◯ OR A, #n4

Function: A ← A ∨ n4 n4 = I_{3-0}: 0-FH

ORs 4-bit immediate data n4 with the contents of the A register, and sets the result to the A register.

Application example

To set the lower 3 bits of the accumulator to 1

```
OR A, #0111B
```

◯ OR A, @HL

Function: A ← A ∨ (HL)

ORs the contents of the data memory addressed by register pair HL with the contents of the A register, and sets the result to the A register.

◯ OR XA, rp'

Function: XA ← XA ∨ rp'

ORs the contents of register pair rp' (XA, HL, DE, BC, XA', HL', DE', or BC') with the contents of register pair XA, and sets the result to register pair XA.

◯ OR rp'1, XA

Function: rp'1 ← rp'1 ∨ XA

ORs the contents of register pair XA with register pair rp'1 (HL, DE, BC, XA', HL', DE', or BC'), and sets the result to a specified register pair.

XOR A, #n4

Function: $A \leftarrow A \forall n4$ $n4 = I_{3-0}$: 0-FH

Exclusive-ORs 4-bit immediate data n4 with the contents of the A register, and sets the result to the A register.

Application example

To invert the higher 4 bits of the accumulator

```
XOR A, #1000B
```

XOR A, @HL

Function: $A \leftarrow A \forall (HL)$

Exclusive-ORs the contents of the data memory addressed by register pair HL with the contents of the A register, and sets the result to the A register.

XOR XA, rp'

Function: $XA \leftarrow XA \forall rp'$

Exclusive-ORs the contents of register pair rp' (XA, HL, DE, BC, XA', HL', DE', or BC') with the contents of register pair XA, and sets the result to register pair XA.

XOR rp'1, XA

Function: $rp'1 \leftarrow rp'1 \forall XA$

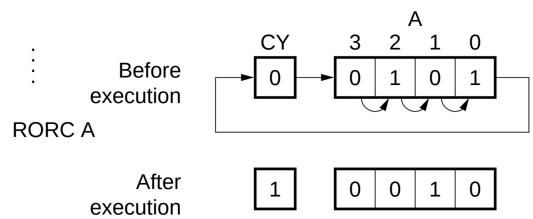
Exclusive-ORs the contents of register pair XA with register pair rp'1 (HL, DE, BC, XA', HL', DE', or BC'), and sets the result to a specified register pair.

11.4.5 Accumulator manipulation instruction

◯ 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) 1 bit to the left with the carry flag.



◯ NOT A

Function: A ← $\overline{A}$

Takes 1’s complement of the A register (4-bit accumulator) (inverts the bits of the accumulator).

11.4.6 Increment/decrement instruction

◯ INCS reg

Function: reg ← reg+1; Skip if reg = 0

Increments the contents of register reg (X, A, H, L, D, E, B, or C). If reg = 0 as a result, the next one instruction is skipped.

◯ INCS rp1

Function: rp1 ← rp1+1; Skip if rp1 = 00H

Increments the contents of register pair rp1 (HL, DE, or BC). If rp1 = 00H as a result, the next one instruction is skipped.

◯ INCS @HL

Function: (HL) ← (HL)+1; Skip if (HL) = 0

Increments the contents of the data memory addressed by pair register HL. If the contents of the data memory become 0 as a result, the next one instruction is skipped.

◯ INCS mem

Function: (mem) ← (mem) + 1; Skip if (mem) = 0, mem = D_{7-0}: 00H-FFH

Increments the contents of the data memory addressed by 8-bit immediate data mem. If the contents of the data memory become 0 as a result, the next one 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, or C). If reg = FH as a result, the next one 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', or BC'). If rp’ = FFH as a result, the next one instruction is skipped.

11.4.7 Compare instruction

○ SKE reg, #n4

Function: Skip if reg = n4 n4 = I_{3-0}: 0-FH

Skips the next one instruction if the contents of register reg (X, A, H, L, D, E, B, or C) are equal to 4-bit immediate data n4.

○ SKE @HL, #n4

Function: Skip if (HL) = n4 n4 = I_{3-0}: 0-FH

Skips the next one instruction if the contents of the data memory addressed by register pair HL are equal to 4-bit immediate data n4.

○ SKE A, @HL

Function: Skip if A = (HL)

Skips the next one instruction if the contents of the A register are equal to the contents of the data memory addressed by register pair HL.

○ SKE XA, @HL

Function: Skip if A = (HL) and X = (HL + 1)

Skips the next one instruction if the contents of the A register are equal to the contents of the data memory addressed by register pair HL and if the contents of the X register are equal to the contents of the next memory address.

However, if the contents of the L register are an odd number, an address whose least significant address is ignored is specified.

○ SKE A, reg

Function: Skip if A = reg

Skips the next one instruction if the contents of the A register are equal to register reg (X, A, H, L, D, E, B, or C).

○ SKE XA, rp'

Function: Skip if XA = rp'

Skips the next one instruction if the contents of register pair XA are equal to the contents of register pair rp' (XA, HL, DE, BC, XA', HL', DE', or BC').

11.4.8 Carry flag manipulation instruction

◯ 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 next one instruction if the carry flag is 1.

◯ NOT1 CY

Function: CY ← $\overline{CY}$

Inverts the carry flag. Therefore, sets the carry flag to 1 if it is 0, and clears the flag to 0 if it is 1.

11.4.9 Memory bit manipulation instruction

◯ SET1 mem.bit

Function: (mem.bit) ← 1 mem = D_{7-0}: 00H-FFH, bit = B_{1-0}: 0-3

Sets the bit specified by 2-bit immediate data bit at the address specified by 8-bit immediate data mem.

◯ SET1 fmem.bit

◯ SET1 pmem.@L

◯ SET1 @H+mem.bit

Function: (bit specified by operand) ← 1

Sets the bit of the data memory addressed in the bit manipulation addressing mode (fmem.bit, pmem.@L, or @H+mem.bit).

◯ CLR1 mem.bit

Function: (mem.bit) ← 0 mem = D_{7-0}: 00H-FFH, bit = B_{1-0}: 0-3

Clears the bit specified by 2-bit immediate data bit at the address specified by 8-bit immediate data mem.

◯ CLR1 fmem.bit

◯ CLR1 pmem.@L

◯ CLR1 @H+mem.bit

Function: (bit specified by operand) ← 0

Clears the bit of the data memory addressed in the bit manipulation addressing mode (fmem.bit, pmem.@L, or @H+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 next one instruction if the bit specified by 2-bit immediate data bit at the address specified by 8-bit immediate data mem is 1.

◯ SKT fmem.bit

◯ SKT pmem.@L

◯ SKT @H+mem.bit

Function: Skip if (bit specified by operand) = 1

Skips the next one instruction if the bit of the data memory addressed in the bit manipulation addressing mode (fmem.bit, pmem.@L, or @H+mem.bit) is 1.

◯ SKF mem.bit

Function: Skip if (mem.bit) = 0
mem = D_{7-0}: 00H-FFH, bit = B_{1-0}: 0-3

Skips the next one instruction if the bit specified by 2-bit immediate data bit at the address specified by 8-bit immediate data mem is 0.

◯ SKF fmem.bit

◯ SKF pmem.@L

◯ SKF @H+mem.bit

Function: Skip if (bit specified by operand) = 0

Skips the next one instruction if the bit of the data memory addressed in the bit manipulation addressing mode (fmem.bit, pmem.@L, or @H+mem.bit) is 0.

○ SKTCLR fmem.bit

○ SKTCLR pmem.@L

○ SKTCLR @H+mem.bit

Function: Skip if (bit specified by operand) = 1 then clear

Skips the next one instruction if the bit of the data memory addressed in the bit manipulation addressing mode (fmem.bit, pmem.@L, or @H+mem.bit) is 1, and clears the bit to “0”.

○ AND1 CY, fmem.bit

○ AND1 CY, pmem.@L

○ AND1 CY, @H+mem.bit

Function: $CY \leftarrow CY \wedge$ (bit specified by operand)

ANDs the content of the carry flag with the contents of the data memory addressed in the bit manipulation addressing mode (fmem.bit, pmem.@L, or @H+mem.bit), and sets the result to the carry flag.

○ OR1 CY, fmem.bit

○ OR1 CY, pmem.@L

○ OR1 CY, @H+mem.bit

Function: $CY \leftarrow CY \vee$ (bit specified by operand)

ORs the content of the carry flag with the contents of the data memory addressed in the bit manipulation addressing mode (fmem.bit, pmem.@L, or @H+mem.bit), and sets the result to the carry flag.

○ XOR1 CY, fmem.bit

○ XOR1 CY, pmem.@L

○ XOR1 CY, @H+mem.bit

Function: $CY \leftarrow CY \veebar$ (bit specified by operand)

Exclusive-ORs the content of the carry flag with the contents of the data memory addressed in the bit manipulation addressing mode (fmem.bit, pmem.@L, or @H+mem.bit), and sets the result to the carry flag.

11.4.10 Branch instruction

(I) BR addr

Function: In the case of the μPD750068: $PC_{12-0} \leftarrow$ addr
addr = 0000H-1FFFH

Branches to an address specified by immediate data addr.

This instruction is an assembler directive and is replaced by the assembler at assembly time with the optimum instruction from the BR !addr, BRCB !caddr, and BR $addr instructions.

(II) BR addr1

Function: In the case of the μPD750068: $PC_{12-0} \leftarrow$ addr1
addr1 = 0000H-1FFFH

Branches to an address specified by immediate data addr1.

This instruction is an assembler directive and is replaced by the assembler at assembly time with the optimum instruction from the BRA !addr1, BR !addr, BRCB !caddr, and BR $addr instructions.

(II) BRA !addr1

Function: In the case of the μPD750068: $PC_{12-0} \leftarrow$ addr1

( ) BR !addr

Function: In the case of the μPD750068: $PC_{12-0} \leftarrow$ addr
addr = 0000H-3FFFH

Transfers immediate data addr to the program counter (PC) and branches to an address specified by the PC.

Remark The functions described in this section are explained by using the μPD750068, with a 13-bit program counter and addr = 0000H to 1FFFH, as an example. When this manual is used for the μPD750064 with a 12-bit program counter, addr = 000H to FFFH, the μPD750066 with a 13-bit program counter, addr = 0000H to 17FFH, or the μPD75P0076 with a 14-bit program counter addr = 0000H to 3FFFH, please make the required substitutions.

(I) BR $addr

Function: In the case of the μPD750068: $PC_{12-0} \leftarrow$ addr
addr = (PC–15) to (PC–1), (PC+2) to (PC+16)

This is a relative branch instruction that has a branch range of (–15 to –1) and (+2 to +16) from the current address. It is not affected by a page boundary or block boundary.

(II) BR $addr1

Function: In the case of the μPD750068: $PC_{12-0} \leftarrow$ addr1
addr1 = (PC–15) to (PC–1), (PC+2) to (PC+16)

This is a relative branch instruction that has a branch range of (–15 to –1) and (+2 to +16) from the current address. It is not affected by a page boundary or block boundary.

Remark The functions described in this section are explained by using the μPD750068, with a 13-bit program counter and addr = 0000H to 1FFFH, as an example. When this manual is used for the μPD750064 with a 12-bit program counter, addr = 000H to FFFH, the μPD750066 with a 13-bit program counter, addr = 0000H to 17FFH, or the μPD75P0076 with a 14-bit program counter addr = 0000H to 3FFFH, please make the required substitutions.

◯ BRCB !caddr

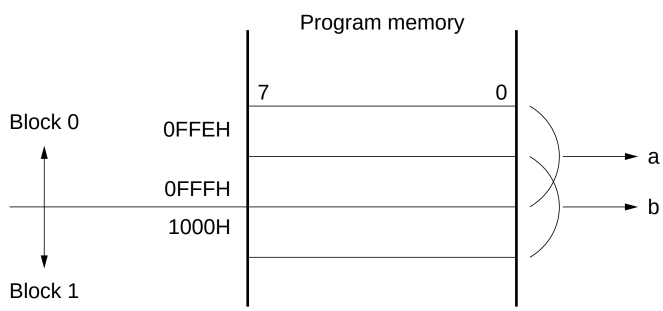
Function: In the case of the μPD750068: $PC_{12-0} \leftarrow PC_{12} + caddr_{11-0}$
caddr = n000H-nFFFH
n = PC_{12} = 0, 1

Branches to an address specified by the lower 12 bits of the program counter (PC_{11-0}) replaced with 12-bit immediate data caddr.

★ The program counter of the μPD750064 has an 11-bit configuration, therefore, BRCB ! caddr can be used to branch to all areas. PC_{12} of the μPD750066 and 750068, and $PC_{12,13}$ of the μPD75P0076 are fixed, therefore BRCB ! caddr can only be used to branch within a block.

Caution

The BRCB !caddr instruction usually branches execution in a block where the instruction exists. If the first byte of this instruction is at address 0FFEH or 0FFFH, however, execution does not branch to block 0 but to block 1.



If the BRCB !caddr instruction is at position b in the figure above, execution branches to block 1, not block 0.

Remark The functions described in this section are explained by using the μPD750068, with a 13-bit program counter and addr = 0000H to 1FFFH, as an example. When this manual is used for the μPD750064 with a 12-bit program counter, addr = 000H to FFFH, the μPD750066 with a 13-bit program counter, addr = 0000H to 17FFH, or the μPD75P0076 with a 14-bit program counter addr = 0000H to 3FFFH, please make the required substitutions.

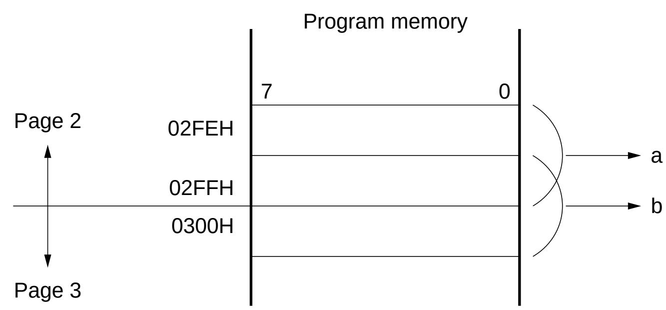
◯ BR PCDE

Function: In the case of the μPD750068: $PC_{12-0} \leftarrow PC_{12-8} + DE$
$PC_{7-4} \leftarrow D$, $PC_{3-0} \leftarrow E$

Branches to an address specified by the lower 8 bits of the program counter (PC_{7-0}) replaced with the contents of register pair DE. The higher bits of the program counter are not affected.

Caution

The BR PCDE instruction usually branches execution to the page where the instruction exists. If the first byte of the op code is at address ××FE or ××FFH, however, execution does not branch in that page, but to the next page.



For example, if the BR PCDE instruction is at position a or b in the above figure, execution branches to the lower 8-bit address specified by the contents of register pair DE in page 3, not in page 2.

◯ BR PCXA

Function: In the case of the μPD750068: $PC_{12-0} \leftarrow PC_{12-8} + XA$
$PC_{7-4} \leftarrow X$, $PC_{3-0} \leftarrow A$

Branches to an address specified by the lower 8 bits of the program counter (PC_{7-0}) replaced with the contents of register pair XA. The higher bits of the program counter are not affected.

Caution

This instruction branches execution to the next page, not to the same page, if the first byte of the op code is at address ××FEH or ××FFH, in the same manner as the BR PCDE instruction.

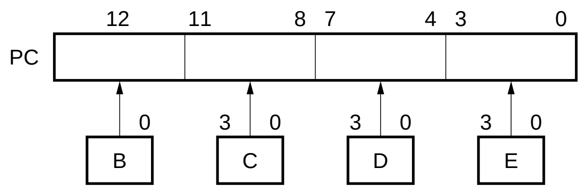
Remark The functions described in this section are explained by using the μPD750068, with a 13-bit program counter and addr = 0000H to 1FFFH, as an example. When this manual is used for the μPD750064 with a 12-bit program counter, addr = 000H to FFFH, the μPD750066 with a 13-bit program counter, addr = 0000H to 17FFH, or the μPD75P0076 with a 14-bit program counter addr = 0000H to 3FFFH, please make the required substitutions.

◯ BR BCDE

Function: In the case of the μPD750068: PC_{12-0} ← BCDE

Example

To branch to an address specified by the contents of the program counter replaced by the contents of registers $B_{1, 0}$, C, D, and E

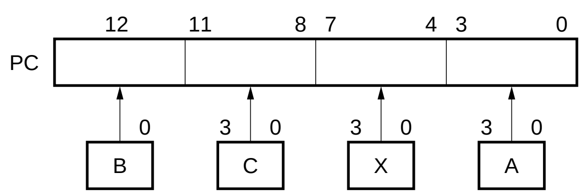


◯ BR BCXA

Function: In the case of the μPD750068: PC_{12-0} ← BCXA

Example

To branch to an address specified by the contents of the program counter replaced by the contents of registers B_0, C, X, and A



◯ TBR addr

Function:

This is an assembler directive for table definition by the GETI instruction. It is used to replace a 3-byte BR !addr instruction with a 1-byte GETI instruction. Describe 12-bit address data as addr. For details, refer to **RA75X Assembler Package User's Manual - Language (EEU-1363)**.

Remark The functions described in this section are explained by using the μPD750068, with a 13-bit program counter and addr = 0000H to 1FFFH, as an example. When this manual is used for the μPD750064 with a 12-bit program counter, addr = 000H to FFFH, the μPD750066 with a 13-bit program counter, addr = 0000H to 17FFH, or the μPD75P0076 with a 14-bit program counter addr = 0000H to 3FFFH, please make the required substitutions.

11.4.11 Subroutine/stack control instruction

(II) CALLA !addr1

Function: In the case of the μPD750068:
(SP–2) ← ×, ×, 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: In the case of the μPD750068:
[MkI 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

[MkII mode]
(SP–2) ← ×, ×, 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), MBE, and RBE to the data memory (stack) addressed by the stack pointer (SP), decrements the SP, and then branches to an address specified by 14-bit immediate data addr.

Remark The functions described in this section are explained by using the μPD750068, with a 13-bit program counter and addr = 0000H to 1FFFH, as an example. When this manual is used for the μPD750064 with a 12-bit program counter, addr = 000H to FFFH, the μPD750066 with a 13-bit program counter, addr = 0000H to 17FFH, or the μPD75P0076 with a 14-bit program counter addr = 0000H to 3FFFH, please make the required substitutions.

(I/II) CALLF !faddr

Function: In the case of the μPD750068:

[MkI mode]

(SP–1) ← PC_{7-4}, (SP–2) ← PC_{3-0}

(SP–3) ← MBE, RBE, 0, PC_{12}

(SP–4) ← PC_{11-8}, SP ← SP–4

PC_{12-0} ← 00+faddr

faddr = 0000H-07FFH

[MkII mode]

(SP–2) ← ×, ×, MBE, RBE

(SP–3) ← PC_{7-4}, (SP–4) ← PC_{3-0}

(SP–5) ← 0, 0, 0, PC_{12}, (SP–6) ← PC_{11-8}

SP ← SP–6

PC_{12-0} ← 00+faddr

faddr = 0000H-07FFH

Saves the contents of the program counter (return address), MBE, and RBE to the data memory (stack) addressed by the stack pointer (SP), decrements the SP, and then branches to an address specified by 11-bit immediate data faddr. The address range from which a subroutine can be called is limited to 0000H to 07FFH (0 to 2047).

TCALL !addr

Function

This is an assembler directive for table definition by the GETI instruction. It is used to replace a 3-byte CALL !addr instruction with a 1-byte GETI instruction. Describe 12-bit address data as addr. For details, refer to **RA75X Assembler Package User's Manual - Language (EEU-1363)**.

Remark The functions described in this section are explained by using the μPD750068, with a 13-bit program counter and addr = 0000H to 1FFFH, as an example. When this manual is used for the μPD750064 with a 12-bit program counter, addr = 000H to FFFH, the μPD750066 with a 13-bit program counter, addr = 0000H to 17FFH, or the μPD75P0076 with a 14-bit program counter addr = 0000H to 3FFFH, please make the required substitutions.

(I/II) RET

Function: In the case of the μPD750068:

[MkI mode] $PC_{11-8} \leftarrow$ (SP), MBE, RBE, 0, $PC_{12} \leftarrow$ (SP+1)
$PC_{3-0} \leftarrow$ (SP+2)
$PC_{7-4} \leftarrow$ (SP+3), SP $\leftarrow$ SP+4

[MkII mode] $PC_{11-8} \leftarrow$ (SP), 0, 0, 0, $PC_{12} \leftarrow$ (SP+1)
$PC_{3-0} \leftarrow$ (SP+2), $PC_{7-4} \leftarrow$ (SP+3)
×, ×, MBE, RBE $\leftarrow$ (SP+4), SP $\leftarrow$ SP+6

Restores the contents of the data memory (stack) addressed by the stack pointer (SP) to the program counter (PC), memory bank enable flag (MBE), and register bank enable flag (RBE), and then increments the contents of the SP.

Caution
All the flags of the program status word (PSW) other than MBE and RBE are not restored.

(I/II) RETS

Function: In the case of the μPD750068:

[MkI mode] $PC_{11-8} \leftarrow$ (SP), MBE, 0, 0, $PC_{12} \leftarrow$ (SP+1)
$PC_{3-0} \leftarrow$ (SP+2), $PC_{7-4} \leftarrow$ (SP+3), SP $\leftarrow$ SP+4
Then skip unconditionally

[MkII mode] $PC_{11-8} \leftarrow$ (SP), 0, 0, 0, $PC_{12} \leftarrow$ (SP+1)
$PC_{3-0} \leftarrow$ (SP+2), $PC_{7-4} \leftarrow$ (SP+3)
×, ×, MBE, RBE $\leftarrow$ (SP+4), SP $\leftarrow$ SP+6
Then skip unconditionally

Restores the contents of the data memory (stack) addressed by the stack pointer (SP) to the program counter (PC), memory bank enable flag (MBE), and register bank enable flag (RBE), increments the contents of the SP, and then skips unconditionally.

Caution
All the flags of the program status word (PSW) other than MBE and RBE are not restored.

Remark The functions described in this section are explained by using the μPD750068, with a 13-bit program counter and addr = 0000H to 1FFFH, as an example. When this manual is used for the μPD750064 with a 12-bit program counter, addr = 000H to FFFH, the μPD750066 with a 13-bit program counter, addr = 0000H to 17FFH, or the μPD75P0076 with a 14-bit program counter addr = 0000H to 3FFFH, please make the required substitutions.

(I/II) RETI

Function: In the case of the μPD750068:

[MkI mode] $PC_{11-8} \leftarrow$ (SP), MBE, RBE, 0, $PC_{12} \leftarrow$ (SP+1)
$PC_{3-0} \leftarrow$ (SP+2), $PC_{7-4} \leftarrow$ (SP+3)
$PSW_L \leftarrow$ (SP+4), $PSW_H \leftarrow$ (SP+5)
SP $\leftarrow$ SP+6

[MkII mode] $PC_{11-8} \leftarrow$ (SP), 0, 0, 0, $PC_{12} \leftarrow$ (SP+1)
$PC_{3-0} \leftarrow$ (SP+2), $PC_{7-4} \leftarrow$ (SP+3)
$PSW_L \leftarrow$ (SP+4), $PSW_H \leftarrow$ (SP+5)
SP $\leftarrow$ SP+6

Restores the contents of the data memory (stack) addressed by the stack pointer (SP) to the program counter (PC) and program status word (PSW), and then increments the contents of the SP.

This instruction is used to return execution from an interrupt processing routine.

Remark The functions described in this section are explained by using the μPD750068, with a 13-bit program counter and addr = 0000H to 1FFFH, as an example. When this manual is used for the μPD750064 with a 12-bit program counter, addr = 000H to FFFH, the μPD750066 with a 13-bit program counter, addr = 0000H to 17FFH, or the μPD75P0076 with a 14-bit program counter addr = 0000H to 3FFFH, please make the required substitutions.

◯ PUSH rp

Function: (SP–1) ← rp_H, (SP–2) ← rp_L, SP ← SP–2

Saves the contents of register pair rp (XA, HL, DE, or BC) to the data memory (stack) addressed by the stack pointer (SP), and then decrements the contents of the SP.

The higher 4 bits of the register pair (rp_H, X, H, D, or B) are saved to the stack addressed by (SP–1), and the lower 4 bits (rp_L: A, L, E, or C) are saved to the stack 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 register bank select register (RBS) to the data memory (stack) addressed by the stack pointer (SP), and then decrements the contents of the SP.

◯ POP rp

Function: rp_L ← (SP), rp_H ← (SP+1), SP ← SP+2

Restores the contents of the data memory addressed by the stack pointer (SP) to register pair rp (XA, HL, DE, or BC), and then decrements the contents of the stack pointer.

The contents of (SP) are restored to the higher 4 bits of the register pair (rp_H, X, H, D, or B), and the contents of (SP+1) are restored to the lower 4 bits (rp_L: A, L, E, or C).

◯ POP BS

Function: RBS ← (SP), MBS ← (SP+1), SP ← SP+2

Restores the contents of the data memory (stack) addressed by the stack pointer (SP) to the register bank select register (RBS) and memory bank select register (MBS), and then increments the contents of the SP.

11.4.12 Interrupt control instruction

○ EI

Function: IME (IPS.3) ← 1

Sets the interrupt mask enable flag (bit 3 of the interrupt priority select register) to “1” to enable interrupts. Acknowledging an interrupt is controlled by an interrupt enable flag corresponding to the interrupt.

○ EI IE×××

Function: IE××× ← 1 ××× = N_5, $N_{2\text{-}0}$

Sets a specified interrupt enable flag (IE×××) to “1” to enable acknowledging the corresponding interrupt (××× = BT, CSI, T0, T1, W, 0, 1, 2, or 4).

○ DI

Function: IME (IPS.3) ← 0

Resets the interrupt mask enable flag (bit 3 of the interrupt priority select register) to “0” to disable all interrupts, regardless of the contents of the respective interrupt enable flags.

○ DI IE×××

Function: IE××× ← 1 ××× = N_5, $N_{2\text{-}0}$

Resets a specified interrupt enable flag (IE×××) to “0” to disable acknowledging the corresponding interrupt (××× = BT, CSI, T0, T1, W, 0, 1, 2, or 4).

11.4.13 Input/output instruction

◯ IN A, PORTn

Function: A ← PORTn n = N_{3-0}: 0-6, 11

Transfers the contents of a port specified by PORTn (n = 0-6, 11) to the A register.

Caution

When this instruction is executed, it is necessary that MBE = 0 or (MBE = 1, MBS = 15). n can be 0 to 6, 11.

The data of the output latch is loaded to the A register in the output mode, and the data of the port pins are loaded to the register in the input mode.

◯ IN XA, PORTn

Function: A ← PORTn, X ← PORTn+1 n = N_{3-0}: 4

Transfers the contents of the port specified by PORTn (n = 4) to the A register, and transfers the contents of the next port to the X register.

Caution

Only 4 can be specified as n. When this instruction is executed, it is necessary that MBE = 0 or (MBE = 1, MBS = 15).

The data of the output latch is loaded to the A and X registers in the output mode, and the data of the port pins are loaded to the registers in the input mode.

◯ OUT PORTn, A

Function: PORTn ← A n = N_{3-0}: 2-6

Transfers the contents of the A register to the output latch of a port specified by PORTn (n = 2-6).

Caution

When this instruction is executed, it is necessary that MBE = 0 or (MBE = 1, MBS = 15).

Only 2 to 6 can be specified as n.

◯ OUT PORTn, XA

Function: PORTn ← A, PORTn+1 ← X n = N_{3-0}: 4

Transfers the contents of the A register to the output latch of a port specified by PORTn (n = 4), and the contents of the X register to the output latch of the next port.

Caution

When this instruction is executed, it is necessary that MBE = 0 or (MBE = 1, MBS = 15).

Only 4 can be specified as n.

11.4.14 CPU control instruction

◯ HALT

Function: PCC.2 ← 1

Sets the HALT mode (this instruction sets the bit 2 of the processor clock control register).

Caution
Make sure that an NOP instruction follows the HALT instruction.

◯ STOP

Function: PCC.3 ← 1

Sets the STOP mode (this instruction sets the bit 3 of the processor clock control register).

Caution
Make sure that an NOP instruction follows the STOP instruction.

◯ NOP

Function: Executes nothing but consumes 1 machine cycle.

11.4.15 Special instruction

SEL RBn

Function: RBS ← n n = N_{1-0}: 0-3

Sets 2-bit immediate data n to the register bank select register (RBS).

SEL MBn

Function: MBS ← n n = N_{3-0}: 0, 1, 15

Transfers 4-bit immediate data n to the memory bank select register (MBS).

(I/II) GETI taddr

Function: In the case of the μPD750068:
taddr = T_{5-0}, 0: 20H-7FH
[MkI mode]

- **When table defined by TBR instruction is referenced**

PC_{12-0} ← $(taddr)_{4-0}$ + (taddr+1)

- **When table defined by 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 table defined by instruction other than TBR and TCALL is referenced**
  Executes instruction with (taddr) (taddr+1) as op code

Remark The functions described in this section are explained by using the μPD750068, with a 13-bit program counter and addr = 0000H to 1FFFH, as an example. When this manual is used for the μPD750064 with a 12-bit program counter, addr = 000H to FFFH, the μPD750066 with a 13-bit program counter, addr = 0000H to 17FFH, or the μPD75P0076 with a 14-bit program counter addr = 0000H to 3FFFH, please make the required substitutions.

- **When table defined by TBR instruction is referencedNote**

$PC_{12-0} \leftarrow (taddr)_{4-0} + (taddr+1)$

- **When table defined by TCALL instruction is referencedNote**

  $(SP-2) \leftarrow \times, \times$, MBE, RBE

  $(SP-3) \leftarrow PC_{7-4}$, $(SP-4) \leftarrow PC_{3-0}$

  $(SP-5) \leftarrow 0, 0, 0, PC_{14}$, $(SP-6) \leftarrow PC_{11-8}$

$PC_{12-0} \leftarrow (taddr)_{4-0} + (taddr+1)$, $SP \leftarrow SP-6$

- **When table defined by instruction other than TBR and TCALL is referenced**

  Executes instruction with (taddr) (taddr+1) as op code

Note The address specified by the TBR and TCALL instructions is limited to 0000H to 3FFFH.

References the 2-byte data at the program memory address specified by (taddr), (taddr+1) and executes it as an instruction.

The area of the reference table consists of addresses 0020H through 007FH. Data must be written to this area in advance. Write the mnemonic of a 1-byte or 2-byte instruction as the data as is.

When a 3-byte call instruction and 3-byte branch instruction is used, data is written by using an assembler directive (TCALL or TBR).

Only an even address can be specified by taddr.

Remark The functions described in this section are explained by using the μPD750068, with a 13-bit program counter and addr = 0000H to 1FFFH, as an example. When this manual is used for the μPD750064 with a 12-bit program counter, addr = 000H to FFFH, the μPD750066 with a 13-bit program counter, addr = 0000H to 17FFH, or the μPD75P0076 with a 14-bit program counter addr = 0000H to 3FFFH, please make the required substitutions.

Caution

Only the 2-machine cycle instruction can be set to the reference table as a 2-byte instruction (except the BRCB and CALLF instructions). Two 1-byte instructions can be set only in the following combinations:

Instruction of 1st Byte	Instruction of 2nd Byte

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 contents of the PC are not incremented while the GETI instruction is executed. Therefore, after the reference instruction has been executed, processing continues from the address next to that of the GETI instruction.

If the instruction preceding the GETI instruction has a skip function, the GETI instruction is skipped in the same manner as the other 1-byte instructions. If the instruction referenced by the GETI instruction has a skip function, the instruction that follows the GETI instruction is skipped.

If an instruction having a string effect is referenced by the GETI instruction, it is executed as follows:

- If the instruction preceding the GETI instruction has the string effect of the same group as the referenced instruction, the string effect is lost and the referenced instruction is not skipped when GETI is executed.
- If the instruction next to GETI has the string effect of the same group as the referenced instruction, the string effect by the referenced instruction is valid, and the instruction following that instruction is skipped.

Application example

```
MOV   HL, #00H  ⎫
MOV   XA, #FFH  ⎬ Replaced by GETI
CALL  SUB1      ⎪
BR    SUB2      ⎭
        ORG     20H
HL00:   MOV     HL, #00H
XAFF:   MOV     XA, #FFH
CSUB1:  TCALL   SUB1
BSUB2:  TBR     SUB2
          :
        GETI    HL00        ; MOV HL, #00H
          :
        GETI    BSUB2       ; BR SUB2
          :
        GETI    CSUB1       ; CALL SUB1
          :
        GETI    XAFF        ; MOV XA, #FFH
```

APPENDIX A FUNCTIONS OF μPD75068, 750068, AND 75P0076

(1/2)

Item		μPD75068	μPD750068	μPD75P0076
Program memory		Mask ROM 0000H-1F7FH (8064 × 8 bits)	Mask ROM 0000H-1FFFH (8192 × 8 bits)	One-time PROM 0000H-3FFFH (16384 × 8 bits)
Data memory		000H-1FFH (512 × 4 bits)		
CPU		75X Standard CPU	75XL CPU	
General-purpose register		4 bits × 8 or 8 bits × 4	(4 bits × 8 or 8 bits × 4) 4 banks	
Instruction execution time	With main system clock	0.95, 1.91, 3.81, 15.3 μs (at 4.19 MHz)	• 0.67, 1.33, 2.67, 10.7 μs (at 6.0 MHz) • 0.95, 1.91, 3.81, 15,3 μs (at 4.19 MHz)	
	With subsystem clock	122 μs (at 32.768 kHz)		
I/O port	CMOS input	12 (connections of internal pull-up resistors specifiable by software: 7)		
	CMOS input/output	12 (connections of internal pull-up resistors specifiable by software)		
	N-ch open-drain input/output	8 (internal pull-up resistors specifiable by mask option) withstand voltage 10 V	8 (internal pull-up resistors specifiable by mask option) withstand voltage 13 V	8 (no mask option) withstand voltage 13 V
	Total	32		
Timer		3 channels • 8-bit timer/event counter • 8-bit basic interval counter • Watch timer	4 channels • 8-bit timer/event counter 0 (addition of watch timer output) • 8-bit timer/event counter 1 (can be used as 16-bit timer/ event counter) • 8-bit basic interval timer/watchdog timer • Watch timer	
A/D converter		• 8-bit resolution × 8 channels (successive approximation type) • Can be operated from V_{DD} = 2.7 V	• 8-bit resolution × 8 channels (successive approximation type) • Can be operated from V_{DD} = 1.8 V	
Clock output (PCL)		Φ, 524, 262, 65.5 kHz (main system clock at 4.19 MHz)	• Φ, 1.05MHz, 262 kHz, 65.5 kHz (main system clock at 4.19 MHz) • Φ, 1.5 MHz, 375 kHz, 93.8 kHz (main system clock at 6.0 MHz)	
BUZ output (BUZ)		2, 4, 32 kHz (main system clock at 4.19 MHz, subsystem clock at 32.768 kHz)	• 2, 4, 32 kHz (main system clock at 4.19 MHz or subsystem clock: 32.768 kHz) • 2.93, 5.86, 46.9 kHz (main system clock at 6.0 MHz)	

(2/2)

Item	μPD75068	μPD750068	μPD75P0076
Serial interface	Three modes are supported • 3-line serial I/O mode ... MSB/LSB first selectable • 2-line serial I/O mode • SBI mode	Two modes are supported • 3-line serial I/O mode ... MSB/LSB first selectable • 2-line serial I/O mode	
Vector interrupt	External: 3, internal: 3	External: 3, internal: 4	
Test input	External : 1, internal : 1		
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 × 10 mil)	• 42-pin plastic shrink DIP (600 mil, 1.778 mm pitch) • 42-pin plastic shrink SOP (375 mil, 0.8 mm pitch)	

The following development tools are available to support development of systems using the μPD750068. With the 75XL series, a relocatable assembler that can be used in common with any models in the series is used in combination with a device file dedicated to the model being used.

Language processor

RA75X relocatable assembler	Host machine			Order code
		OS	Supply media	
	PC-9800 series	MS-DOS™ (Ver.3.30 to Ver.6.2[Note]) ★	3.5"2HD	μS5A13RA75X
			5"2HD	μS5A10RA75X
	IBM PC/AT™ or compatible machine	Refer to **OS of IBM PC.**	3.5" 2HC	μS7B13RA75X
			5"2HC	μS7B10RA75X

Device file	Host machine			Order code
		OS	Supply media	
	PC-9800 series	MS-DOS (Ver.3.30 to Ver.6.2[Note]) ★	3.5"2HD	μS5A13DF750068
			5"2HD	μS5A10DF750068
	IBM PC/AT or compatible machine	Refer to **OS of IBM PC.**	3.5" 2HC	μS7B13DF750068
			5"2HC	μS7B10DF750068

Note Although Ver.5.00 or above has a task swap function, this function cannot be used with this software.

Remark The operations of the assembler and device file are guaranteed only on the above host machines and OS.

PROM writing tool

<table>
<tr><td rowspan="2">Hardware</td><td>PG-1500</td><td colspan="4">This is a PROM programmer that can program a built-in PROM single-chip microcontroller in a stand-alone mode, or under control of a host computer when connected with an accessory board and an optional programmer adapter. It can also program typical PROMs from 256K-bit to 4M-bit models.</td></tr>
<tr><td>PA-75P0076CU</td><td colspan="4">PROM programmer adapter dedicated to the μPD75P0076CU and 75P0076GT and connected to the PG-1500.</td></tr>
<tr><td rowspan="7">Software</td><td rowspan="7">PG-1500 controller</td><td colspan="4">Connects the PG-1500 and a host machine with a parallel or serial interface to control the PG-1500 on the host machine.</td></tr>
<tr><td colspan="3">Host machine</td><td rowspan="2">Order code</td></tr>
<tr><td></td><td>OS</td><td>Supply media</td></tr>
<tr><td rowspan="2">PC-9800 series</td><td rowspan="2">MS-DOS
(Ver.3.30
to
Ver.6.2 Note) ★</td><td>3.5"2HD</td><td>μS5A13PG1500</td></tr>
<tr><td>5"2HD</td><td>μS5A10PG1500</td></tr>
<tr><td rowspan="2">IBM PC/AT or compatible machine</td><td rowspan="2">Refer to OS of IBM PC.</td><td>3.5" 2HC</td><td>μS7B13PG1500</td></tr>
<tr><td>5"2HC</td><td>μS7B10PG1500</td></tr>
</table>

Note Although Ver.5.00 or above has a task swap function, this function cannot be used with this software.

Remark The operation of the PG-1500 controller is guaranteed only on the above host machines and OS.

Debugging Tools

As the debugging tools for the μPD750068, in-circuit emulators (IE-75000-R and IE-75001-R) are available. The following table shows the system configuration of the in-circuit emulators.

<table>
<tr><td rowspan="6">Hardware</td><td>IE-75000-R[Note1]</td><td colspan="4">The IE-75000-R is an in-circuit emulator that debugs the hardware and software of an application system using the 75X series or 75XL series. To develop the μPD750068 subseries, use this in-circuit emulator with an optional emulation board IE-75300-R-EM and emulation probe.
The in-circuit emulator is connected with a host machine or PROM programmer for efficient debugging.
The IE-75000-R contains the emulation board IE-75000-R-EM.</td></tr>
<tr><td>IE-75001-R</td><td colspan="4">The IE-75001-R is an in-circuit emulator that debugs the hardware and software of an application system using the 75X series or 75XL series. To develop the μPD750068 subseries, use this in-circuit emulator with an optional emulation board IE-75300-R-EM and emulation probe.
The in-circuit emulator is connected with a host machine or PROM programmer to provide efficient debugging.</td></tr>
<tr><td>IE-75300-R-EM</td><td colspan="4">This is an emulation board to evaluate an application system using the μPD750068 subseries. It is used with the IE-75000-R or IE-75001-R.</td></tr>
<tr><td>EP-750068CU-R</td><td colspan="4">This is an emulation probe for the μPD750068CU.
It is connected to the IE-75000-R or IE-75001-R and IE-75300-R-EM.</td></tr>
<tr><td>EP-750068GT-R</td><td colspan="4" rowspan="2">This is an emulation probe for the μPD750068GT.
It is connected to the IE-75000-R or IE-75001-R and IE-75300-R-EM.
A flexible board, EV-9500GT-42, that facilitates connection with the target system is also supplied.</td></tr>
<tr><td>EV-9500GT-42</td></tr>
<tr><td rowspan="7">Software</td><td rowspan="7">IE control program</td><td colspan="4">This program connects the IE-75000-R or IE-75001-R and a host machine with an RS-232C or Centronics interface to control the IE-75000-R or IE-75001-R on the host machine.</td></tr>
<tr><td colspan="3">Host machine</td><td rowspan="2">Order code</td></tr>
<tr><td></td><td>OS</td><td>Supply media</td></tr>
<tr><td rowspan="2">PC-9800 series</td><td rowspan="2">MS-DOS
(Ver.3.30 to Ver.6.2[Note2])</td><td>3.5"2HD</td><td>μS5A13IE75X</td></tr>
<tr><td>5"2HD</td><td>μS5A10IE75X</td></tr>
<tr><td rowspan="2">IBM PC/AT or compatible machine</td><td rowspan="2">Refer to OS of IBM PC.</td><td>3.5" 2HC</td><td>μS7B13IE75X</td></tr>
<tr><td>5"2HC</td><td>μS7B10IE75X</td></tr>
</table>

Notes 1. This is a maintenance part.

2. Although Ver.5.00 or above has a task swap function, this function cannot be used with this software.

Remark The operation of the IE control program is guaranteed only on the above host machines and OS.

OS of IBM PC

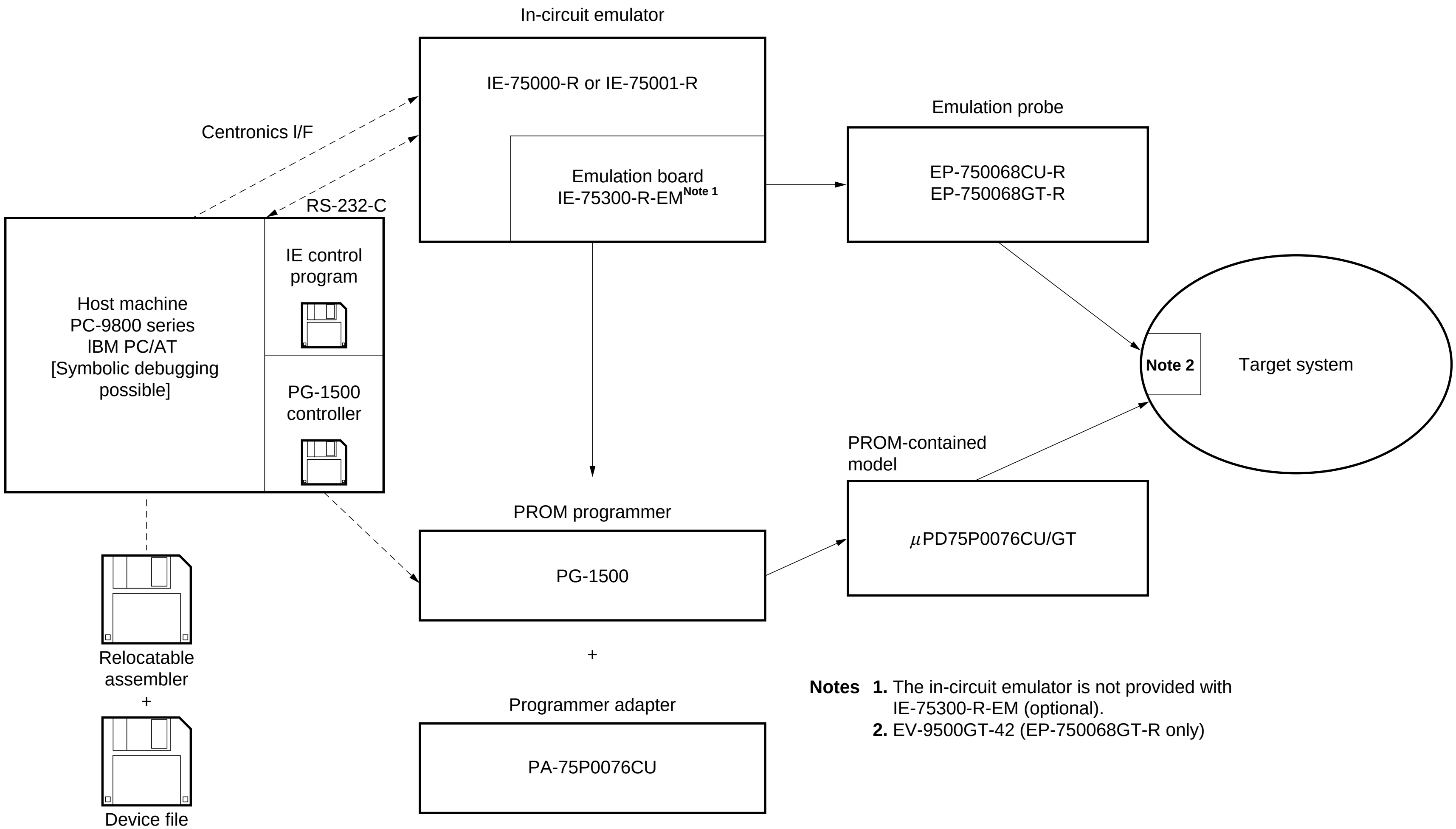
The following OS is supported as the OS for IBM PC.

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 the English mode is supported.

Caution Although Ver.5.00 or above has a task swap function, this function cannot be used with this software.

Development Tool Configuration



Notes **1.** The in-circuit emulator is not provided with IE-75300-R-EM (optional).

2. EV-9500GT-42 (EP-750068GT-R only)

[MEMO]

APPENDIX C ORDERING MASK ROM

After your program has been developed, you can place an order for a mask ROM using the following procedure:

<1> Reservation for mask ROM ordering

Inform NEC of when you intend to place an order for the mask ROM. (NEC's response may be delayed if we are not informed in advance.)

★ **<2> Preparation of ordering media**

Following three mediums are available for ordering mask ROM.

- UV-EPROM**Note**
- 3.5-inch IBM format floppy disk (outside Japan only)
- 5-inch IBM format floppy disk (outside Japan only)

Note Prepare three UV-EPROMs with the same contents. (For the product with mask option, write down the mask option data on the mask option information sheet.)

<3> Preparation of necessary documents

Fill out the following documents when ordering the mask ROM:

- Mask ROM Ordering Sheet
- Mask ROM Ordering Check Sheet
- Mask Option Information Sheet (necessary for product with mask option)

<4> Ordering

Submit the media prepared in <2> and documents prepared in <3> to NEC distributer or NEC sales department by the order reservation date.

Caution For details, refer to the information document "ROM Code Ordering Procedure (IEM-1366)."

[MEMO]

APPENDIX D INSTRUCTION INDEX

D.1 Instruction Index (by function)

[Subroutine/stack control instruction]

[Interrupt control instruction]

[Input/output instruction]

[CPU control instruction]

[Special instruction]

D.2 Instruction Index (alphabetical order)

APPENDIX E HARDWARE INDEX

[R]

[S]

[T]

[W]

★ # APPENDIX F REVISION HISTORY

Version	Contents	Page
2nd	The μPD750064, 750068 and 75P0076 changed from "under development" to "development completed".	Throughout
	The μPD750066 added.	
	Data bus pins (D0-D7) added.	
	The XT2 open changed to the XT1 complement input when an external clock is used.	
	Processing of Unused Pins changed.	CHAPTER 2 PIN FUCNTIONS
	Table and caution of the maximum time required to select system clock and CPU clock changed.	CHAPTER 5 PERIPHERAL HARDWARE FUNCTION
	Selecting mask option added.	CHAPTER 7 STANDBY FUNCTION
	Writing Program Memory changed.	CHAPTER 9 WRITING AND VERIFYING PROM (PROGRAM MEMORY)
	Reading Program Memory changed.	
	Subsystem clock feedback resistor mask option added.	CHAPTER 10 MASK OPTIONS
	Modification of the instruction list	CHAPTER 11 INSTRUCTION SET
	Version of the supported OS updated.	APPENDIX B DEVELOPMENT TOOLS
	Procedure for ordering supply media changed.	APPENDIX C ORDERING MASK ROM

[MEMO]

NEC

Facsimile Message

From:

___
Name

___
Company

___
Tel. FAX

___
Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288
1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6465-6829

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: 02-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: 044-548-7900

I would like to report the following error/make the following suggestion:

Document title: ___

Document number: ___ Page number: ___

___

___

___

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❏	❏	❏	❏
Technical Accuracy	❏	❏	❏	❏
Organization	❏	❏	❏	❏

CS 99.1