NEC

User's Manual

μPD7701x Family

Digital Signal Processor

Architecture

μPD77015
μPD77016
μPD77017
μPD77018
μPD77018A
μPD77019

Document No. U10503EJ4V0UM00 (4th edition)
Date Published May 1998 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 these products from Japan is regulated by the Japanese government. The export of some or all of these products may be prohibited without governmental license. To export or re-export some or all of these products from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

License not needed: μPD77016, μPD77019-013

The customer must judge the need for licence: μPD77015, μPD77017, μPD77018, μPD77018A, μPD77019

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.1.
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: 01-504-2787
Fax: 01-504-2860

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: 02-719-2377
Fax: 02-719-5951

NEC do Brasil S.A.
Cumbica-Guarulhos-SP, Brasil
Tel: 011-6465-6810
Fax: 011-6465-6829

J98. 2

MAJOR REVISIONS IN THIS EDITION

Page	Description
Throughout	Addition of descriptions of μPD77018A and 77019.
Throughout	Deletion of **Chapter 5 Assembly Instructions**
p.30	Addition of **2.1.2 (2) 116-pin plastic BGA**
p.41	Addition of BGA package numbers to **2.3.2 Pin function of μPD77015, 77017, 77018, 77018A, and 77019**
p.62	Addition of **Note 1** to **Table 3-7. Initialized Pins and Their Initial Statuses**
p.67	Addition of **Table 3-8. Pin Status in HALT Mode**
p.69	Addition of **Table 3-9. Pin Status in STOP Mode**
p.96	Change of **Caution** in **Figure 3-25. External Interrupt Timing**
p.97	Addition of **Caution** to **Table 3-17. Interrupt Vector Table**
p.116	Change of status during reset of DA0-DA15 (13) and $\overline{\text{X}}$/Y in **Table 3-20. Pin Status**
p.166	Addition of description to **3.7.3 (3) (C) I/O timing of non-standard serial clock**
p.175	Change of descriptions of HD0-HD7 in **Table 3-33. The Pin Status during and after Hardware Reset**
p.183, 184	Change of **3.7.4 (5) (c) Interrupts** and addition of **Caution** to this section
p.207	Addition of **4.3.1 (2) Parameters for self-boot of μPD77019-013**
p.214	Addition of description of high-speed simulator (SM77016-H) to **5.1.2 Software simulator**
p.216	Change of part number of **5.2.2 (3) Adapter for EB-77017 board**
p.230 to 232	Addition of **A.3 CPU Registers to Be Initialized and Initial Values** to **A.6 Status of Output Pins during Reset to Release STOP mode**

The mark ★ shows major revised points.

PREFACE

Readers: This manual should be read by engineers who wish to understand the functions of the μPD7701x family for designing software or hardware application systems.

Purpose: This document describes the hardware and software functions provided in the μPD7701x family products in the order shown below. This manual is designed to be used as a reference manual when developing application system hardware or software using μPD7701x products.

Organization: This manual consists of the following sections:

How to read: This manual assumes that readers possess basic knowledge on electric/electronic circuits, logic circuits, and microcomputers.

The μPD7701x family consists of the μPD77016, 77015 , 77017, 77018, 77018A, and 77019. Unless otherwise specified, "μPD7701x" refers to the entire family. If there are some differences in function or operation among family products, they are described under their respective names.

- **To understand all the μPD7701x functions**:
  Read this manual from Chapter 1 "Overview" through Chapter 4 "Boot Function" to gain a detailed understanding of the functions of this family.

- **If you are a hardware engineer**:
  Read this manual from Chapter 1 "Overview" through Chapter 4 "Boot Function". You will learn various useful points for configuring a hardware system as well as gain a detailed understanding of the functions of this family. Chapter 3 "Architecture" describes the related interface levels of the on-chip function blocks.

- **If you are a software engineer**:
  Read this manual from Chapter 1 "Overview" through Chapter 4 "Boot Function". You will learn various useful points for programming a software application as well as gain a detailed understanding of the functions of this family. Chapter 5 "Development Tools" introduces software development tools, additional tools for this family, and evaluation systems. Refer to "μPD7701x Family User's Manual Instructions."

- **If you use this document as a reference manual**:
  Note that an index is provided at the end of this manual. This index can be used to search a word based on a key word. Chapter 3 "Architecture" provides descriptions of internal device architecture on a top-down basis to facilitate searching a specific function.
  Appendix A "Device summary" summarizes various key points on the use of this device, such as instruction memory map, data memory map, and peripheral register map.

Legends:		
	Data weight	: Upper digit is left and lower is right.
	Active low	: $\overline{\text{XXX}}$ (a line is drawn over the name of pin or signal.)
	Memory map address	: Top-Higher, Bottom-Lower
	Note	: Explanation for the Note in the text.
	Caution	: Description that should be read carefully
	Remarks	: Complementary explanation for the text
	Bolded text	: Important items
	Numerical expression	: Binary ... 0bXXXX
		Decimal ... XXXX
		Hexadecimal ... 0xXXXX
	{ }	: Either of the items enclosed within { } can be selected.

Related Documents: Also use the following documents:

[Documents related to μPD7701x family]

- Data sheet

Part Number	μPD77016	μPD77015	μPD77017	μPD77018	μPD77018A	μPD77019	μPD77019-013
Document Number	U10891E	U10902E			U11849E		U13053E

- User's manual and brochure

Document Name		Document Number
Brochure		U12395E
User's Manual	Architecture	This manual
	Instructions	U13116E
Application Note	Basic software	U11958E

[Documents related to development tools]

Document Name		Document Number
IE-77016-98/PC User's Manual	Hardware	EEU-1541
IE77016-CM-EM6 User's Manual		EEU-1506
EB-77017 User's Manual		U12660E

Some of the above related documents are preliminary versions but are not so specified here.

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

[MEMO]

CONTENTS (1/3)

CONTENTS (2/3)

CONTENTS (3/3)

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)

Chapter 1

Overview

As the 21st century approaches, multi-media systems that do not only compute numeric data and process information as conventional computer systems do, but can also process images and sounds, which are important interfacing elements for human beings, have come in the limelight. Another important feature of multi-media systems is that they must provide a real-time processing capability. As a result, the quantity of information that must be communicated and processed is substantially increasing, the current architecture which heavily depends on the CPU for processing information, must be reviewed, and data processing systems that facilitate organizing multi-media systems are being increasingly demanded.

The μPD7701x family is a collection of 16-bit fixed-point digital signal processors (DSPs) of the new generation that have been developed for digital signal processing applications, including multi-media systems, which require high speed and high accuracy.

Their internal circuit consists of eight 40-bit general-purpose registers that are used to load/store or input/output data, a multiply accumulator that performs an operation of “16 bits $\times$ 16 bits + 40 bits —> 40 bits”, a 40-bit ALU, and a 40-bit barrel shifter.

By employing as the basic technology the Harvard architecture, in which the instruction memory space and the data memory space are separated, and by separating the data memory space into X and Y memory areas, the μPD7701x family can execute flexible, high-speed data transfer.

These DSPs are provided with two serial interfaces, a host interface, and general-purpose I/O ports as peripherals. Because these interface registers are mapped on the data memory space, various peripheral control operations can be performed by all addressing modes provided for the data memory.

The basic operation of the μPD7701x family is executed by using a pipeline of three stages: instruction fetch, instruction decode, and instruction execution. Some instructions, however, do not use this pipeline, but are designed so that their execution results can be used by the next instruction. Therefore, you can program your applications without having to pay any special attention to the pipeline.

1.1 Products of μPD7701x Family

The μPD7701x family consists of the following products:

- μPD77016
- μPD77015
- μPD77017
- μPD77018
- ★ μPD77018A
- ★ μPD77019

1.2 Features of μPD7701x Family

The μPD7701x family is provided with sophisticated next-generation DSP functions in addition to standard DSP technological features. The features common to all the models in the μPD7701x family and the features of each model are described below. Table 1-1 shows the differences among the μPD7701x family products.

1.2.1 Common features

- High-speed instruction cycle:
  30 ns (μPD77016, 77015, 77017, 77018)
  16.6 ns (μPD77018A, 77019)
- Harvard architecture eliminating bus neck
- Three stage pipeline architecture
- Rational combination of parallel instructions
- Multiply accumulator capable of executing 3-operand instructions (trinomial operation)
- Eight 40-bit general-purpose registers
- Eight data memory pointer registers (four each for X and Y memories)
- Dual data memory space promising flexible, high-speed data transfer
- Many addressing mode enabling flexible memory access
- Head room format eliminating operational overflow
- Various internal peripheral interfaces
- Many external interfaces
- Interrupt functions covering wide range of applications (internal: 6 levels, external: 4 levels)
- Hardware loop mechanism minimizing overhead
- Programmable external memory access wait
- Boot ROM
- Debug function (JTAG port)
- Standby function by HALT instruction

1.2.2 Features of μPD77016

The μPD77016 is the basic model of the μPD7701x family. Because it supports an external memory area of sufficient capacity for both instructions and data in addition to the common features of the μPD7701x family, the μPD77016 can cover a wide range of applications.

1.2.3 Features of μPD77015, 77017, 77018, 77018A, and 77019

The μPD77015, 77017, 77018, 77018A, and 77019 are ideally suited for compact and economical embedded systems, being provided with a clock multiplier circuit (mask option), a crystal oscillator circuit, a single, 3-V power supply, a power down function, and a 100-pin TQFP package in addition to the common features of the μPD7701x family. The only difference among these models is the capacity of the internal ROM and RAM. Select the model best suited to your application.

The μPD77019-013 invalidates the internal ROM of the μPD77019. Because mask processing is not necessary, use this model when only the internal RAM is used.

Table 1-1. Features of μPD7701x Family

Items		μPD7701x family					
		μPD77016	μPD77015	μPD77017	μPD77018	μPD77018A	μPD77019
Instruction cycle		30 ns(@ max. clock rate)				16.6 ns (@ max. clock rate)	
Clock rate (@ max. rate)		Ext. 66 MHz	Ext. 33 MHz (x1 mask option) Ext. 16.5 MHz (x2 mask option) Ext. 8.25 MHz (x4 mask option) Ext. 4.125 MHz (x8 mask option) Ext. 33 MHz crystal (x1 mask option)			Ext. 60 MHz (x1 mask option), Ext. 30 MHz (x2 mask option), Ext. 20 MHz (x3 mask option), Ext. 15 MHz (x4 mask option), Ext. 7.5 MHz (x8 mask option) Ext. 60 MHz crystal (x1 mask option)	
Parallel instruction execution		Trinomial operation & parallel load/store Binomial operation & parallel load/store Monomial operation & Conditional Register-to-register transfer & Conditional Branch & Conditional					
Hardware loop		Nesting of up to 4 levels					
Conditional instruction		Conditional operation, conditional transfer, conditional branch by combining standard conditional instructions with other instructions					
Multiply accumulator		16 bits $\times$ 16 bits + 40 bits —> 40 bits (such as trinomial operation: R0 = R0 + R1H * R2L)					
Accumulator		40-bit inputs and 40-bit output (binomial and monomial)					
General register		Eight 40-bits registers (R0-R7)					
Data memory pointer		Four pointers for X memory (DP0-DP3), Four pointers for Y memory (DP4-DP7)					
Interrupt		Internal: 6 levels (6 causes), External: 4 levels (4 causes)					
3-stage pipeline processing		Instruction fetch, instruction decode, instruction execution					
Instruction memory	Boot ROM	256 words (for boot function)					
(32 bits/word)	Int. ROM	n/a	4 K words	12 K words	24 K words	24 K words	24 K words
	Int. RAM	1.5 K words	256 words	256 words	256 words	256 words	4K words
	Ext. area	48 K words	n/a	n/a	n/a	n/a	n/a
X data memory	Int. ROM	n/a	2 K words	4 K words	12 K words	12 K words	12 K words
(16 bits/word)	Int. RAM	2 K words	1 K words	2 K words	3 K words	3 K words	3 K words
	Ext. area	48 K words	16 K words	16 K words	16 K words	16 K words	16 K words
Y data memory	Int. ROM	n/a	2 K words	4 K words	12 K words	12 K words	12 K words
(16 bits/word)	Int. RAM	2 K words	1 K words	2 K words	3 K words	3 K words	3 K words
	Ext. area	48 K words	16 K words	16 K words	16 K words	16 K words	16 K words
Serial interface		2 channels	2 channels (without SORQ2, SIAK2 signals)				
Host interface		8-bit parallel					
I/O port		Four ports (signal direction specifiable independently)					
Supply voltage		+5 V ± 10 %	+2.7 to +3.6 V				
Standby function		Entered when HALT instruction is executed.					
Powerdown function		n/a	Entered when STOP instruction is executed				
Package		160-pin plastic QFP	100-pin plastic TQFP			100-pin plastic TQFP, 116-pin plastic BGA	100-pin plastic TQFP
Miscellaneous		Debugging function (JTAG port) CMOS technology					

Remark n/a: not available

1.3 Main Applications of μPD7701x Family

As its name implies, a DSP is a device developed for digital signal processing. DSPs employing next generation technology, such as the μPD7701x family, are also provided with the functions of a general-purpose CPU, including a memory access capability and interrupt functions. Therefore, the μPD7701x can cover a wide range of applications. The main applications of these DSPs are listed below, by field.

General signal processing

- Digital filter (FIR filter, BIQUAD filter, etc.)
- High-speed Fourier transformation
- Hilbert transformation
- Relative processing
- Adaptive filter

Communication field

- High-speed modem (V.32, etc.)
- Digital cellular telephone (voice codec, equalizer, etc.)
- MPEG
- Echo canceler
- Adaptive equalizer
- Digital PBX
- DTMF encoder/decoder
- FAX
- Spread spectrum communication
- Multiplexed communication

Sound/acoustic

- Voice recognition
- Sound coding/decoding (ADPCM, PARCOR, etc.)
- Speech synthesis (phoneme synthesis, rule synthesis, etc.)
- Synthesizer
- Electronic musical instrument
- Sound field control
- Sound effects

Image processing/graphics

- Affine transformation
- 2-dimension orthogonal transformation (Fourier transformation, Hadamard transformation, KL transformation, etc.)
- Filtering (smoothing, median filter, etc.)
- Various operators (Laplacian, Sobel, etc.)
- Ray tracing, Mandelbrot
- CAD (3D graphics, etc.)
- Virtual reality
- Image compression/expansion (DCT, run length, variable-length coding)
- Image recognition
- Computer animation

Control

- Navigation system
- Disc control (CD, LD, etc.)
- Various servo systems (PID, AC servo, etc.)
- Control of laser printer and copier
- Robot
- NC control
- Fuzzy control

Measurement

- Spectrum analyzer
- Function generator
- Pattern matching
- Lock-in amplifier
- Box car integrator
- Various analysis systems (vibration analysis, transient analysis)

General numeric processing and others

- Data enciphering/deciphering
- Use as numerical processor
- Neural system

Chapter 2

Pin Functions

This chapter describes the pin configurations and pin functions of the μPD7701x family. The following are the pin names:

$\overline{\text{BSTB}}$	: Bus Strobe
CLKIN	: Clock Input
CLKOUT	: Clock Output
D0-D15	: 16-bit Data Bus
DA0-DA15	: External Data Memory Address Bus
GND	: Ground
HA0, HA1	: Host Data Access
$\overline{\text{HCS}}$	: Host Chip Select
HD0-HD7	: Host Data Bus
$\overline{\text{HOLDAK}}$	: Hold Acknowledge
$\overline{\text{HOLDRQ}}$	: Hold Request
$\overline{\text{HRD}}$	: Host Read
$\overline{\text{HRE}}$	: Host Read Enable
$\overline{\text{HWE}}$	: Host Write Enable
$\overline{\text{HWR}}$	: Host Write
IA0-IA15	: External Instruction Memory Address Bus
I.C.	: Internally Connected
ID0-ID31	: External Instruction Memory Data Bus
$\overline{\text{INT1}}$-$\overline{\text{INT4}}$	: Interrupt
$\overline{\text{MRD}}$	: Memory Read Output

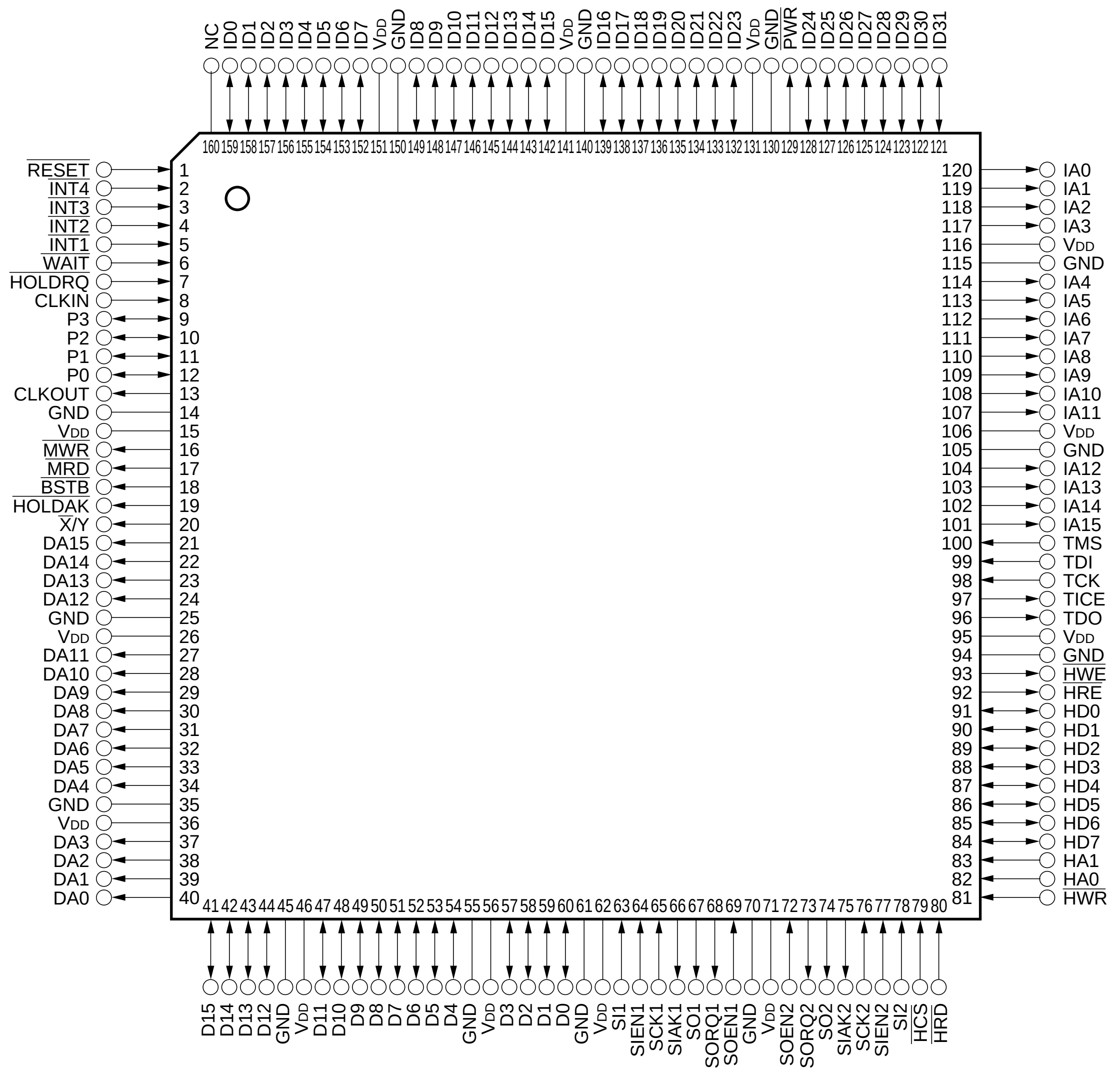
$\overline{\text{MWR}}$	: Memory Write Output
NC	: Non-connection
P0-P3	: Port
$\overline{\text{PWR}}$	: Program Memory Write Strobe
$\overline{\text{RESET}}$	: Reset
SCK1, SCK2	: Serial Clock Input
SI1, SI2	: Serial Data Input
SIAK1, SIAK2	: Serial Input Acknowledge
SIEN1, SIEN2	: Serial Input Enable
SO1, SO2	: Serial Data Output
SOEN1, SOEN2	: Serial Output Enable
SORQ1, SORQ2	: Serial Output Request
TCK	: Test Clock
TDI	: Test Data Input
TDO	: Test Data Output
TICE	: Test for In-circuit Emulator
TMS	: Test Mode Select
V_{DD}	: Power Supply
$\overline{\text{WAIT}}$	: Wait Input
$\overline{\text{X}}$ / Y	: X / Y Memory Select
X1, X2	: Crystal Connection

2.1 Pin Configurations

2.1.1 Pin configuration of μPD77016

- **μPD77016GM-KMD: 160-pin plastic QFP (fine pitch) (24 × 24 mm)**

Figure 2-1. 160-pin Plastic QFP



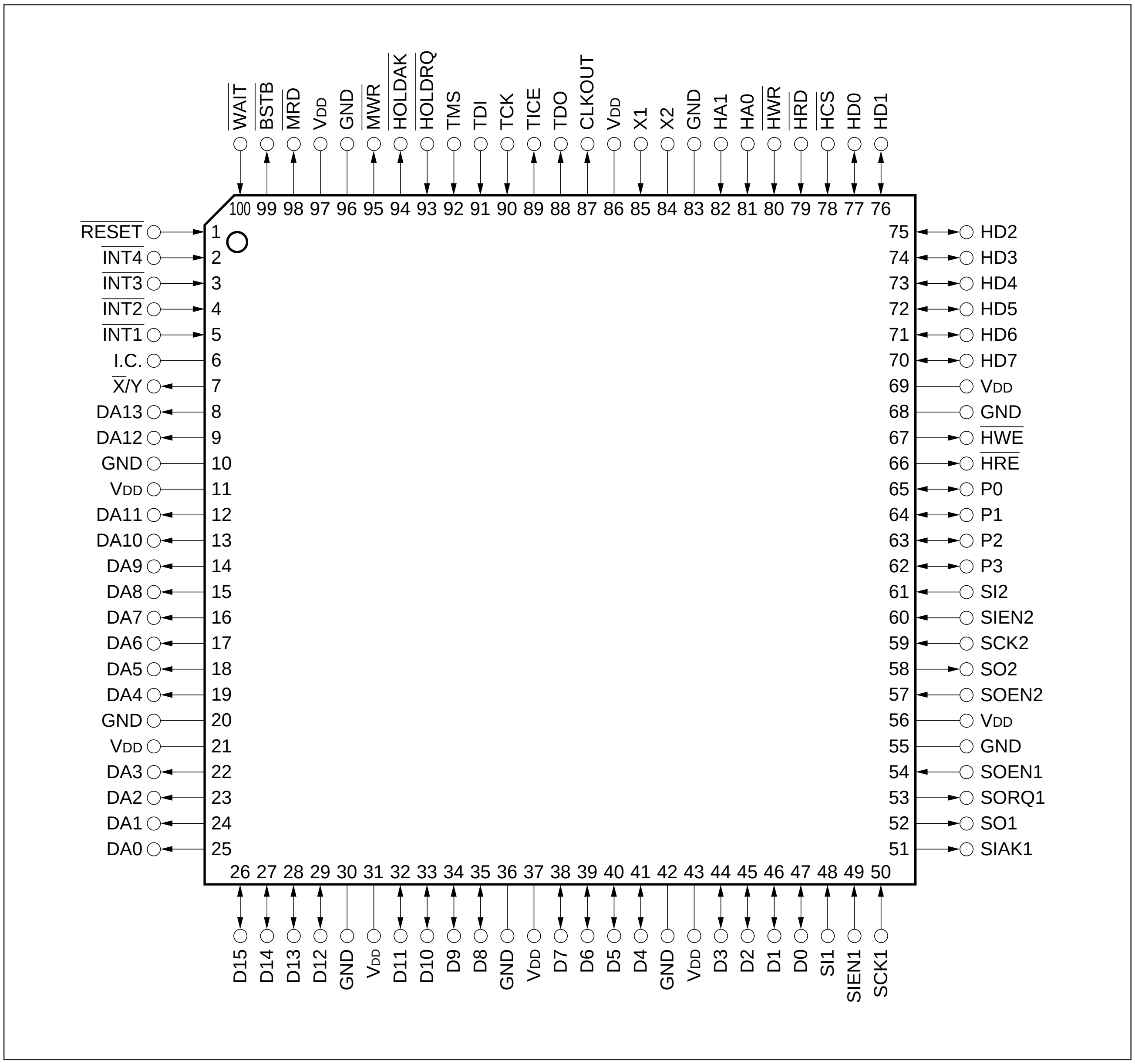
Pin No.	Pin name	Pin No.	Pin name	Pin No.	Pin name	Pin No.	Pin name
1	$\overline{\text{RESET}}$	41	D15	81	$\overline{\text{HWR}}$	121	ID31
2	$\overline{\text{INT4}}$	42	D14	82	HA0	122	ID30
3	$\overline{\text{INT3}}$	43	D13	83	HA1	123	ID29
4	$\overline{\text{INT2}}$	44	D12	84	HD7	124	ID28
5	$\overline{\text{INT1}}$	45	GND	85	HD6	125	ID27
6	$\overline{\text{WAIT}}$	46	V_{DD}	86	HD5	126	ID26
7	$\overline{\text{HOLDRQ}}$	47	D11	87	HD4	127	ID25
8	CLKIN	48	D10	88	HD3	128	ID24
9	P3	49	D9	89	HD2	129	$\overline{\text{PWR}}$
10	P2	50	D8	90	HD1	130	GND
11	P1	51	D7	91	HD0	131	V_{DD}
12	P0	52	D6	92	$\overline{\text{HRE}}$	132	ID23
13	CLKOUT	53	D5	93	$\overline{\text{HWE}}$	133	ID22
14	GND	54	D4	94	GND	134	ID21
15	V_{DD}	55	GND	95	V_{DD}	135	ID20
16	$\overline{\text{MWR}}$	56	V_{DD}	96	TDO	136	ID19
17	$\overline{\text{MRD}}$	57	D3	97	TICE	137	ID18
18	$\overline{\text{BSTB}}$	58	D2	98	TCK	138	ID17
19	$\overline{\text{HOLDAK}}$	59	D1	99	TDI	139	ID16
20	$\overline{\text{X}}$/Y	60	D0	100	TMS	140	GND
21	DA15	61	GND	101	IA15	141	V_{DD}
22	DA14	62	V_{DD}	102	IA14	142	ID15
23	DA13	63	SI1	103	IA13	143	ID14
24	DA12	64	SIEN1	104	IA12	144	ID13
25	GND	65	SCK1	105	GND	145	ID12
26	V_{DD}	66	SIAK1	106	V_{DD}	146	ID11
27	DA11	67	SO1	107	IA11	147	ID10
28	DA10	68	SORQ1	108	IA10	148	ID9
29	DA9	69	SOEN1	109	IA9	149	ID8
30	DA8	70	GND	110	IA8	150	GND
31	DA7	71	V_{DD}	111	IA7	151	V_{DD}
32	DA6	72	SOEN2	112	IA6	152	ID7
33	DA5	73	SORQ2	113	IA5	153	ID6
34	DA4	74	SO2	114	IA4	154	ID5
35	GND	75	SIAK2	115	GND	155	ID4
36	V_{DD}	76	SCK2	116	V_{DD}	156	ID3
37	DA3	77	SIEN2	117	IA3	157	ID2
38	DA2	78	SI2	118	IA2	158	ID1
39	DA1	79	$\overline{\text{HCS}}$	119	IA1	159	ID0
40	DA0	80	$\overline{\text{HRD}}$	120	IA0	160	NC

2.1.2 Pin configuration of μPD77015, 77017, 77018, 77018A, and 77019

(1) 100-pin plastic TQFP (FINE PITCH) (14 × 14 mm)

- μPD77015GC-×××-9EU
- μPD77017GC-×××-9EU
- μPD77018GC-×××-9EU
- μPD77018AGC-×××-9EU
- μPD77019GC-×××-9EU

Figure 2-2. 100-pin Plastic TQFP



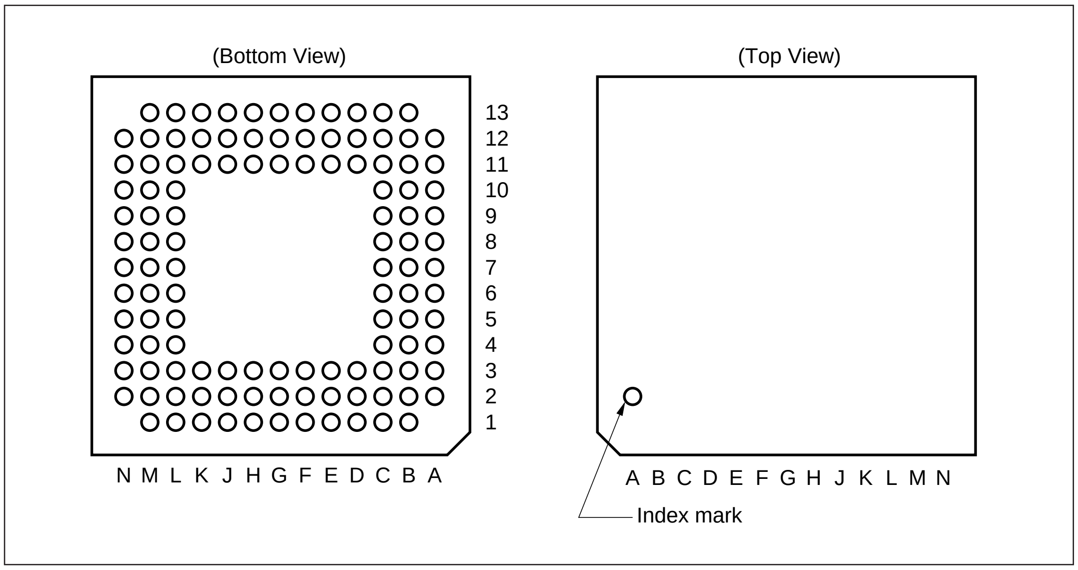
Pin No.	Pin name
1	$\overline{\text{RESET}}$
2	$\overline{\text{INT4}}$
3	$\overline{\text{INT3}}$
4	$\overline{\text{INT2}}$
5	$\overline{\text{INT1}}$
6	I.C.[Note]
7	$\overline{\text{X}}$/Y
8	DA13
9	DA12
10	GND
11	V_{DD}
12	DA11
13	DA10
14	DA9
15	DA8
16	DA7
17	DA6
18	DA5
19	DA4
20	GND
21	V_{DD}
22	DA3
23	DA2
24	DA1
25	DA0
26	D15
27	D14
28	D13
29	D12
30	GND
31	V_{DD}
32	D11
33	D10
34	D9
35	D8
36	GND
37	V_{DD}
38	D7
39	D6
40	D5
41	D4
42	GND
43	V_{DD}
44	D3
45	D2
46	D1
47	D0
48	SI1
49	SIEN1
50	SCK1
51	SIAK1
52	SO1
53	SORQ1
54	SOEN1
55	GND
56	V_{DD}
57	SOEN2
58	SO2
59	SCK2
60	SIEN2
61	SI2
62	P3
63	P2
64	P1
65	P0
66	$\overline{\text{HRE}}$
67	$\overline{\text{HWE}}$
68	GND
69	V_{DD}
70	HD7
71	HD6
72	HD5
73	HD4
74	HD3
75	HD2
76	HD1
77	HD0
78	$\overline{\text{HCS}}$
79	$\overline{\text{HRD}}$
80	$\overline{\text{HWR}}$
81	HA0
82	HA1
83	GND
84	X2
85	X1
86	V_{DD}
87	CLKOUT
88	TDO
89	TICE
90	TCK
91	TDI
92	TMS
93	$\overline{\text{HOLDRQ}}$
94	$\overline{\text{HOLDAK}}$
95	$\overline{\text{MWR}}$
96	GND
97	V_{DD}
98	$\overline{\text{MRD}}$
99	$\overline{\text{BSTB}}$
100	$\overline{\text{WAIT}}$

Note Leave this pin unconnected because the I.C. pin is connected to internal circuits.

★ **(2) 116-pin plastic BGA (FINE PITCH) (12 × 12 mm)**

- **μPD77018AS9-×××-YJC**

Figure 2-3. 116-Pin Plastic BGA



Pin No.	Pin name	Pin No.	Pin name	Pin No.	Pin name	Pin No.	Pin name
A2	$\overline{\text{BSTB}}$	C6	TMS	G11	P1	L9	D2
A3	V_{DD}	C7	TICE	G12	P3	L10	D0
A4	$\overline{\text{MWR}}$	C8	X1	G13	P2	L11	GND
A5	$\overline{\text{HOLDAK}}$	C9	GND	H1	DA7	L12	SIAK1
A6	TDI	C10	$\overline{\text{HWR}}$	H2	DA9	L13	SORQ1
A7	TDO	C11	V_{DD}	H3	DA8	M1	DA2
A8	V_{DD}	C12	V_{DD}	H11	SI2	M2	DA1
A9	HA1	C13	HD5	H12	SCK2	M3	DA0
A10	$\overline{\text{HRD}}$	D1	$\overline{\text{INT2}}$	H13	SIEN2	M4	GND
A11	$\overline{\text{HCS}}$	D2	$\overline{\text{INT4}}$	J1	DA4	M5	D11
A12	HD0	D3	$\overline{\text{INT3}}$	J2	DA6	M6	D9
B1	$\overline{\text{WAIT}}$	D11	HD4	J3	DA5	M7	D7
B2	V_{DD}	D12	V_{DD}	J11	SO2	M8	D5
B3	V_{DD}	D13	V_{DD}	J12	SOEN2	M9	D3
B4	GND	E1	$\overline{\text{X}}$/Y	J13	V_{DD}	M10	GND
B5	$\overline{\text{HOLDRQ}}$	E2	$\overline{\text{INT1}}$	K1	V_{DD}	M11	GND
B6	TCK	E3	I.C.[Note]	K2	V_{DD}	M12	SCK1
B7	CLKOUT	E11	HD6	K3	GND	M13	SO1
B8	X2	E12	HD7	K11	GND	N2	D15
B9	HA0	E13	$\overline{\text{HWE}}$	K12	GND	N3	D13
B10	V_{DD}	F1	V_{DD}	K13	SOEN1	N4	D12
B11	HD1	F2	DA13	L1	DA3	N5	V_{DD}
B12	HD2	F3	DA12	L2	V_{DD}	N6	D8
B13	HD3	F11	GND	L3	D14	N7	V_{DD}
C1	$\overline{\text{RESET}}$	F12	P0	L4	GND	N8	D4
C2	V_{DD}	F13	$\overline{\text{HRE}}$	L5	D10	N9	V_{DD}
C3	V_{DD}	G1	DA10	L6	GND	N10	D1
C4	$\overline{\text{MRD}}$	G2	GND	L7	D6	N11	SI1
C5	GND	G3	DA11	L8	GND	N12	SIEN1

Note Leave this pin unconnected because the I.C. pin is connected to internal circuits.

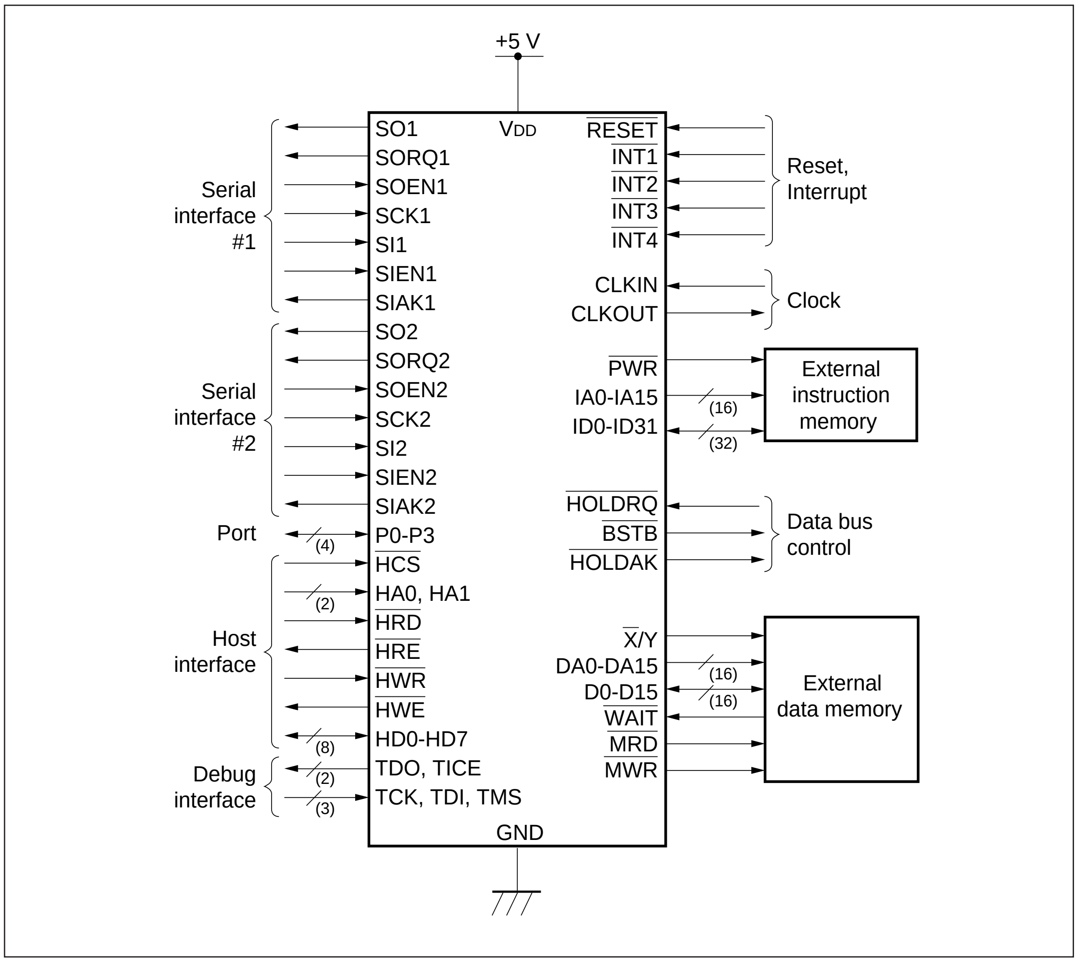
2.2 Pin Organizations

This section describes the pin connections shown in section 2.1 by function.

2.2.1 Pin organization of μPD77016

The figure below shows the pin organization of the μPD77016.

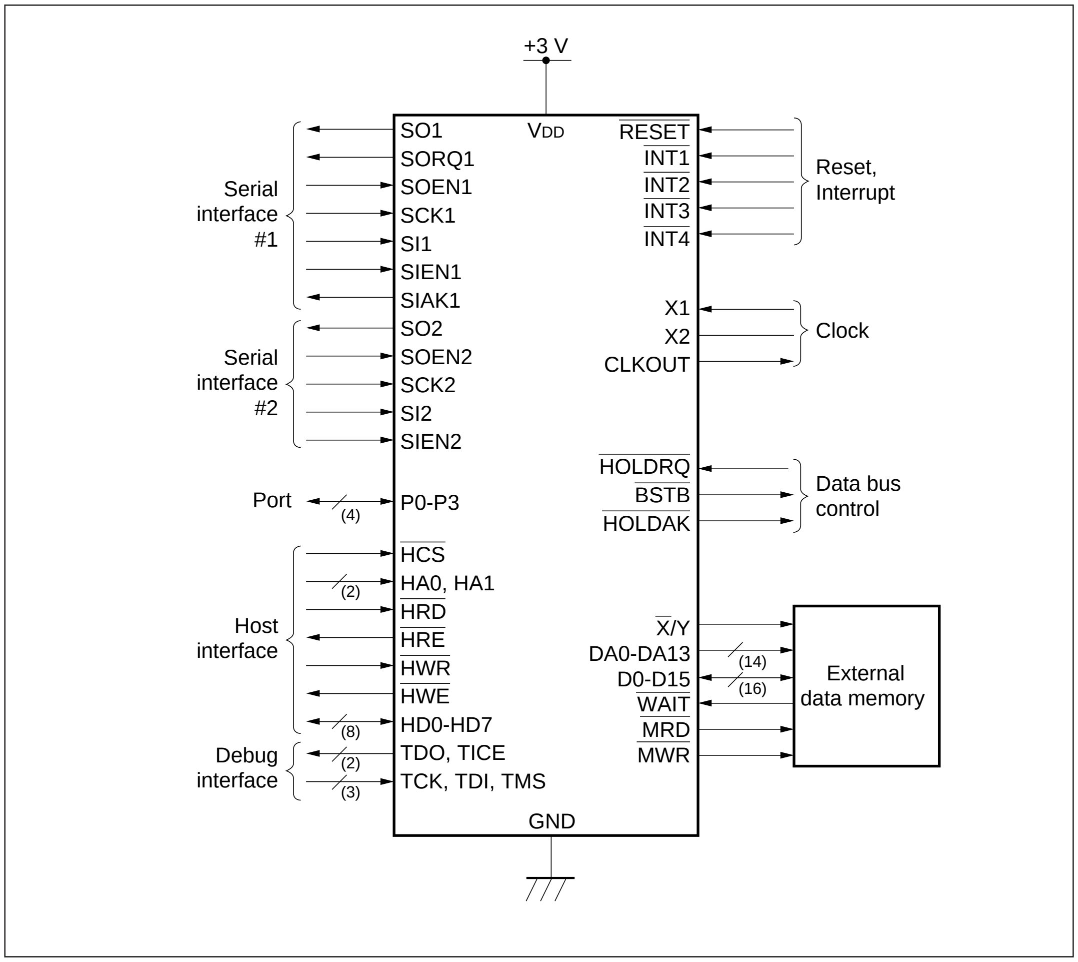
Figure 2-4. Pin Organization of μPD77016



2.2.2 Pin organization of μPD77015, 77017, 77018, 77018A, and 77019

The figure below shows the pin organization of the μPD77015, 77017, 77018, 77018A, and 77019.

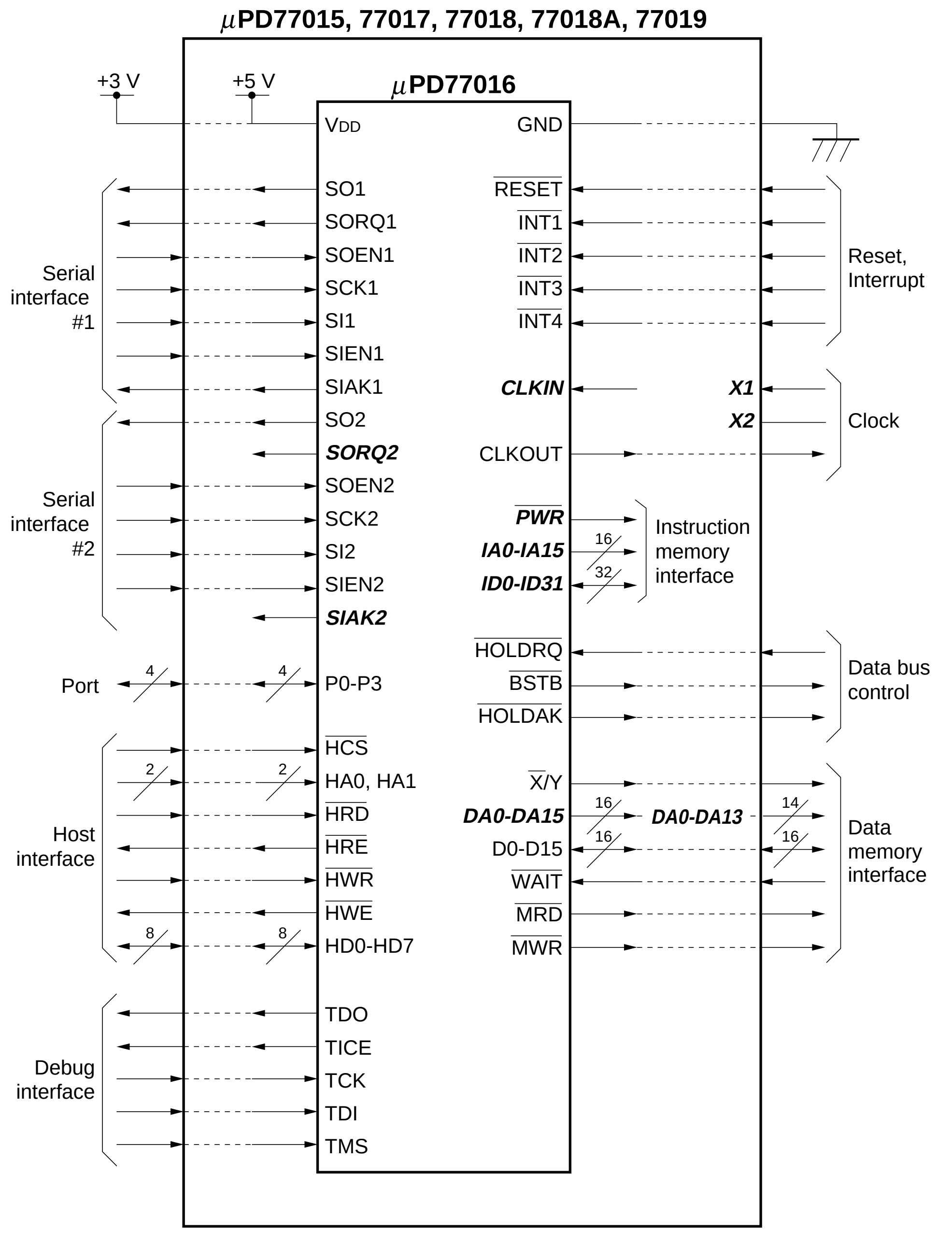
Figure 2-5. Pin Organization of μPD77015, 77017, 77018, 77018A, and 77019



2.2.3 Comparison in pin configurations of μPD7701x family

Some pins of the μPD77016 are different in configuration from those of the μPD77015, 77017, 77018, 77018A, and 77019 as shown in the comparative figure below.

Figure 2-6. Comparison in Pin Configurations of μPD7701x Family



Remark Italicized pins are those which separate the μPD77016 from the other family members.

2.3 Pin Functions

2.3.1 Pin function of μPD77016

(1) Power supply

Pin name	Pin No.	I/O	Function
V_{DD}	15, 26, 36, 46, 56, 62, 71, 95, 106, 116, 131, 141, 151	—	+5-V power supply. Connect all these pins to a +5-V power supply.
GND	14, 25, 35, 45, 55, 61, 70, 94, 105, 115, 130, 140, 150	—	Ground. Connect all these pins to 0 V.

(2) System control

Pin name	Pin No.	I/O	Function
CLKIN	8	Input	Clock input. Always supply the clock in normal device operation. In the standby mode, however, clock supply may be stopped.
CLKOUT	13	Output	Internal system clock output. Clock divided by two in synchronization with CLKIN. Used for processing by an external circuit in synchronization with the instruction cycle of the μPD77016.
$\overline{\text{RESET}}$	1	Input	Internal system reset signal input. Initializes the hardware of the device. Be sure to input the $\overline{\text{RESET}}$ signal after power application to the device.

(3) Interrupt

Pin name	Pin No.	I/O	Function
$\overline{\text{INT1}}$-$\overline{\text{INT4}}$	5-2	Input	External interrupt inputs. These interrupt inputs can be masked and are detected at the falling edge. If interrupts conflict, a one level recording function is available for each input.

(4) External data memory interface

Pin name	Pin No.	I/O	Function
$\overline{X}$ / Y	20	Output (3S)	Memory space select signal output. • 0: Selects X memory space. • 1: Selects Y memory space. The X and Y memory spaces of an external memory cannot be accessed at the same time.
DA0-DA15	40 - 37 34 - 27 24 - 21	Output (3S)	Address bus for external data memory. • Accesses an external memory. When external memory is not accessed, this bus keeps outputting the address of the external memory memory location accessed last. If the external has never been accessed after reset, it outputs a low level (0x0000).
D0-D15	60 - 57, 54 - 47, 44 - 41	I/O (3S)	16-bit data bus. • Accesses an external memory.
$\overline{MRD}$	17	Output (3S)	Read output. • External memory read.
$\overline{MWR}$	16	Output (3S)	Write output. • External memory write.
$\overline{WAIT}$	6	Input	Wait signal input. Wait cycles specified by DWTR (data memory wait cycle register) are inserted when the external data memory is accessed, and this signal is sampled at the end of the wait cycles. While the $\overline{WAIT}$ signal is active (low level), wait cycles are unconditionally inserted. • 0: Wait • 1: No wait (If the wait cycle specified by DWTR has been completed.)
$\overline{HOLDRQ}$	7	Input	Hold request signal input. An external circuit asserts this signal active (low level) when it uses the data memory bus. • 0: Hold request • 1: No request
$\overline{BSTB}$	18	Output	Bus strobe signal output. This signal is asserted active (low level) when the μPD77016 uses the external data bus. • 0: μPD77016 uses the bus. • 1: μPD77016 does not use the bus.
$\overline{HOLDAK}$	19	Output	Hold acknowledge signal output. This signal is asserted active (low level) when an external circuit requests $\overline{HOLDRQ}$ and then the external circuit is enabled to use the data memory bus. • 0: Enables the external circuit to use the data memory bus. • 1: Prevents the external circuit from using the data memory bus.

Remark "3S" in the "I/O" column of the above table stands for three-state pin, and these pins go into a high-impedance state when the external data memory is not accessed and when the bus is released (HOLDAK = 0).

(5) Serial interface

Pin name	Pin No.	I/O	Function
SCK1	65	Input	Serial #1 channel clock input. Signals related to serial input/output are sampled in synchronization with this signal.
SORQ1	68	Output	Serial #1 channel output request signal output. This signal is asserted active (high level) before serial data is output. • 0: Serial data is not output. • 1: Serial data is output.
SOEN1	69	Input	Serial #1 channel output enable signal input. This signal is asserted active (high level) to inform the μPD77016 that the external circuit is ready to accept serial output data. • 0: Not ready to accept serial data output. • 1: Ready to accept serial data output
SO1	67	Output (3S)	Serial #1 channel data output. This signal is output in synchronization with the rising of SCK1.
SIEN1	64	Input	Serial #1 channel input enable signal input. This signal is asserted active (high level) to inform the μPD77016 that the external circuit is ready to supply serial data. • 0: Not ready to supply serial data. • 1: Ready to supply serial data.
SI1	63	Input	Serial #1 channel data input. This signal is input in synchronization with the falling of SCK1.
SIAK1	66	Output	Serial #1 channel input acknowledge signal output. This signal informs the external circuit that the μPD77016 is ready to input serial data. • 0: Not ready to accept serial data input. • 1: Ready to accept serial data input.
SCK2	76	Input	Serial #2 channel clock input. Signals related to serial input/output are sampled in synchronization with this signal.
SORQ2	73	Output	Serial #2 channel output request signal output. This signal is asserted active (high level) before serial data is output. • 0: Serial data is not output. • 1: Serial data is output.

Remark "3S" in the "I/O" column of the above table stands for three-state pin, and these pins go into a high-impedance state when data output have been finished or $\overline{\text{RESET}}$ is input.

Pin name	Pin No.	I/O	Function
SOEN2	72	Input	Serial #2 channel output enable signal input. This signal is asserted active (high level) to inform the μPD77016 that the external circuit is ready to accept serial data output. • 0: Not ready to accept serial data output. • 1: Ready to accept serial data input
SO2	74	Output (3S)	Serial #2 channel data output. This signal is output in synchronization with the rising of SCK2.
SIEN2	77	Input	Serial #2 channel input enable signal. This signal is asserted active (high level) to inform the μPD77016 that the external circuit is ready for supplying serial data. • 0: Not ready to supply serial data • 1: Ready to supply serial data
SI2	78	Input	Serial #2 channel data input. This signal is input in synchronization with the falling of SCK2.
SIAK2	75	Output	Serial #2 channel input acknowledge signal output. This signal informs the external circuit that the μPD77016 is ready to input serial data. • 0: Not ready to accept serial data input. • 1: Ready to accept serial data input.

Remark "3S" in the "I/O" column of the above table stands for three-state pin, and these pins go into a high-impedance state when data output have been finished or $\overline{\text{RESET}}$ is input.

(6) Host interface

Pin name	Pin No.	I/O	Function
HA1	83	Input	Specifies the register to be accessed by HD0 through HD7. • 0: When the μPD77016 is read ($\overline{HRD}$ = 0), the host transmission data register (HDT (out)) is accessed: when it is written ($\overline{HWR}$ = 0), the host receive data register (HDT (in)) is accessed. • 1: The host interface status register (HST) is accessed.
HA0	82	Input	Specifies the register to be accessed by HD0 through HD7. • 0: Bits 7 through 0 of HST, HDT (out), and HDT (in) are accessed. • 1: Bits 15 through 8 of HST, HDT (out), and HDT (in) are accessed.
$\overline{HCS}$	79	Input	Chip select input.
$\overline{HRD}$	80	Input	Host read input. Pulse read from the host. Data is output in synchronization with the falling of this signal.
$\overline{HWR}$	81	Input	Host write input. Pulse written from the host. Data is input in synchronization with the rising of this signal.
$\overline{HRE}$	92	Output	Host read enable output. When this signal is active (low level), the host can access the μPD77016 for read. • 0: Host can access for read. • 1: Host cannot access for read.
$\overline{HWE}$	93	Output	Host write enable output. When this signal is active (low level), the host can access the μPD77016 for write. • 0: Host can access for write. • 1: Host cannot access for write
HD0-HD7	91-84	I/O (3S)	8-bit host data bus

Remark "3S" in the "I/O" column of the above table stands for three-state pin, and these pins go into a high-impedance state when host does not access the μPD7701x's host interface.

(7) I/O port

Pin name	Pin No.	I/O	Function
P0-P3	12-9	I/O	General-purpose I/O port. Each pin can be set in the input or output mode independently by PCD (port command register). Data is input or output via PCD and PDT (port data register).

(8) External instruction memory interface

Pin name	Pin No.	I/O	Function
IA0-IA15	120 - 117, 114 - 107, 104 - 101	Output (3S)	External instruction memory address bus. Even when the internal instruction memory of the μPD77016 is accessed, the address internally accessed is output to an external device. At this time, the data output from the external instruction memory is ignored.
ID0-ID31	159 - 152, 149 - 142, 139 - 132, 128 - 121,	I/O (3S)	32-bit instruction input
$\overline{\text{PWR}}$	129	Output (3S)	Program memory write strobe. While the μPD77016 is performing a boot operation, this signal is used as a write strobe to the external instruction memory when the program is loaded to the external instruction memory, instead of the internal memory.

Remark "3S" in the "I/O" column of the above table stands for three-state pin, and these pins go into a high-impedance state when the hardware reset signal is input.

(9) Debug interface

Pin name	Pin No.	I/O	Function
TDO	96	Output	Used for debugging.
TICE	97	Output	Used for debugging.
TCK	98	Input	Used for debugging.
TDI	99	Input	Used for debugging.
TMS	100	Input	Used for debugging.

(10) Other

Pin name	Pin No.	I/O	Function
NC	160	—	Non-connection

★ 2.3.2 Pin function of μPD77015, 77017, 77018, 77018A, and 77019

Because the pin numbers differ depending on the package, refer to the pin numbers of the package used.

(1) Power supply

Pin name	Pin No.		I/O	Function
	TQFP	BGA		
V_{DD}	11, 21, 31, 37, 43, 56, 69, 86, 97	A3, A8, B2 B3, B10, C2, C3, C11, C12, D12, D13, F1, J13, K1, K2, L2, N5, N7, N9	—	+3-V power supply. Connect all these pins to a +3-V power supply.
GND	10, 20, 30, 36, 42, 55, 68, 83, 96	B4, C5, C9, F11, G2, K3, K11, K12, L4, L6, L8, L11, M4, M10, M11	—	Ground. Connect all these pins to 0 V.

(2) System control

Pin name	Pin No.		I/O	Function
	TQFP	BGA		
X1	85	C8	Input	Clock input or crystal oscillator connection. Use this pin to input an external clock. Always supply the clock in normal device operation. In the stop mode, however, clock supply may be stopped.
X2	84	B8	—	Crystal oscillator connection. When using an external clock, leave this pin open. Always leave this pin on the μPD77019-013 open.
CLKOUT	87	B7	Output	Internal system clock output. Output synchronized with the clock input or crystal oscillation frequency supplied to X1. Use this pin when processing in synchronization with the instruction cycle of the μPD77015, 77017, 77018, 77018A, or 77019 is necessary for the external circuit. Output of the internal system clock can be also suppressed (fixed to low level) by mask option.
$\overline{\text{RESET}}$	1	C1	Input	Internal system reset signal input. Initializes the hardware of the device. Be sure to input the $\overline{\text{RESET}}$ signal after power application to the device.

(3) Interrupt

Pin name	Pin No.		I/O	Function
	TQFP	BGA		
$\overline{\text{INT1}}$-$\overline{\text{INT4}}$	5-2	E2, D1, D3, D2	Input	External interrupt inputs. These interrupt inputs can be masked and are detected at the falling edge. If interrupts conflict, a one level recording function is available for each input.

(4) External data memory interface

Pin name	Pin No. TQFP	Pin No. BGA	I/O	Function
$\overline{X}$ / Y	7	E1	Output (3S)	Memory space select signal output. • 0: Selects X memory space. • 1: Selects Y memory space. The X and Y memory spaces of an external memory cannot be accessed at the same time.
DA0-DA13	25 - 22, 19 - 12 8, 9	M3 - M1, L1, J1, J3, J2, H1, H3, H2, G1, G3, F3, F2	Output (3S)	Address bus for external data memory. • Accesses an external memory. When the external memory is not accessed, this bus keeps outputting the address of the external memory location accessed last. If the external memory has never been accessed after reset, it outputs a low level (0x0000).
D0-D15	47 - 44, 41 - 38, 35 - 32, 29 - 26	L10, N10, L9, M9, N8, M8, L7, M7, N6, M6, L5, M5, N4, N3, L3, N2	I/O (3S)	16-bit data bus. Accesses an external memory.
$\overline{\text{MRD}}$	98	C4	Output (3S)	Read output. External memory read.
$\overline{\text{MWR}}$	95	A4	Output (3S)	Write output. External memory write.
$\overline{\text{WAIT}}$	100	B1	Input	Wait signal input. Wait cycles specified by DWTR (data memory wait cycle register) are inserted when the external data memory is accessed, and this signal is sampled at the end of the wait cycles. While the $\overline{\text{WAIT}}$ signal is active (low level), wait cycles are unconditionally inserted. • 0: Wait • 1: No wait (However, if the wait cycle specified by DWTR has been completed.)
$\overline{\text{HOLDRQ}}$	93	B5	Input	Hold request signal input. An external circuit asserts this signal active (low level) when it uses the data memory bus. • 0: Hold request • 1: No wait
$\overline{\text{BSTB}}$	99	A2	Output	Bus strobe signal output. This signal is asserted active (low level) when the μPD77015, 77017, 77018, 77018A, or 77019 uses the external data bus. • 0: μPD77015, 77017, 77018, 77018A, or 77019 uses the bus. • 1: μPD77015, 77017, 77018, 77018A, or 77019 does not use the bus.
$\overline{\text{HOLDAK}}$	94	A5	Output	Hold acknowledge signal output. This signal is asserted active (low level) when an external circuit requests $\overline{\text{HOLDRQ}}$ and then the external circuit is enabled to use the data memory bus. • 0: Enables the external circuit to use the data memory bus. • 1: Does not enable the external circuit to use the data memory bus.

Remark "3S" in the "I/O" column of the above table stands for three-state pin, and these pins go into a high-impedance state when the external data memory is not accessed and when the bus is released ($\overline{\text{HOLDAK}}$ = 0).

(5) Serial interface

Pin name	Pin No. TQFP	Pin No. BGA	I/O	Function
SCK1	50	M12	Input	Serial #1 channel clock input. Signals related to serial input/output are sampled in synchronization with this signal.
SORQ1	53	L13	Output	Serial #1 channel output request signal output. This signal is asserted active (high level) before serial data is output. • 0: Serial data is not output. • 1: Serial data is output.
SOEN1	54	K13	Input	Serial #1 channel output enable signal input. This signal is asserted active (high level) to inform the μPD77015, 77017, 77018, 77018A, or 77019 that the external circuit is ready to accept serial output data. • 0: Not ready to accept serial data output. • 1: Ready to accept serial data output
SO1	52	M13	Output (3S)	Serial #1 channel data output. This signal is output in synchronization with the rising of SCK1.
SIEN1	49	N12	Input	Serial #1 channel input enable signal input. This signal is asserted active (high level) to inform the μPD77015, 77017, 77018, 77018A, or 77019 that the external circuit is ready to supply serial data. • 0: Not ready to supply serial data. • 1: Ready to supply serial data.
SI1	48	N11	Input	Serial #1 channel data input. This signal is input in synchronization with the falling of SCK1.
SIAK1	51	L12	Output	Serial #1 channel input acknowledge signal output. This signal informs the external circuit that the μPD77015, 77017, 77018, 77018A or 77019 is ready to input serial data. • 0: Not ready to accept serial data input. • 1: Ready to accept serial data input.
SCK2	59	H12	Input	Serial #2 channel clock input. Signals related to serial input/output are sampled in synchronization with this signal.

Remark "3S" in the "I/O" column of the above table stands for three-state pin, and these pins go into a high-impedance state when data output have been finished or $\overline{\text{RESET}}$ is input.

Pin name	Pin No.		I/O	Function
	TQFP	BGA		
SOEN2	57	J12	Input	Serial #2 channel output enable signal input. This signal is asserted active (high level) to inform the μPD77015, 77017, 77018, 77018A, or 77019 that the external circuit is ready to accept serial data output. • 0: Not ready to accept serial data output. • 1: Ready to accept serial data input
SO2	58	J11	Output (3S)	Serial #2 channel data output. This signal is output in synchronization with the rising of SCK2.
SIEN2	60	H13	Input	Serial #2 channel input enable signal. This signal is asserted active (high level) to inform the μPD77015, 77017, 77018, 77018A, or 77019 that the external circuit is ready to supply serial data. • 0: Not ready to supply serial data • 1: Ready to supply serial data
SI2	61	H11	Input	Serial #2 channel data input. This signal is input in synchronization with the falling of SCK2.

Remark "3S" in the "I/O" column of the above table stands for three-state pin, and these pins go into a high-impedance state when data output have been finished or $\overline{\text{RESET}}$ is input.

(6) Host interface

Pin name	Pin No.		I/O	Function
	TQFP	BGA		
HA1	82	A9	Input	Specifies the register to be accessed by HD0 through HD7. • 0: When the μPD77015, 77017, 77018, 77018A, or 77019 is read ($\overline{\text{HRD}}$ = 0), the host transmission data register (HDT (out)) is accessed: when it is written ($\overline{\text{HWR}}$ = 0), the host receive data register (HDT (in)) is accessed. • 1: The host interface status register (HST) is accessed.
HA0	81	B9	Input	Specifies the register to be accessed by HD0 through HD7. • 0: Bits 7 through 0 of HST, HDT (out), and HDT (in) are accessed. • 1: Bits 15 through 8 of HST, HDT (out), and HDT (in) are accessed.
$\overline{\text{HCS}}$	78	A11	Input	Chip select input
$\overline{\text{HRD}}$	79	A10	Input	Host read input. Pulse read from the host. Data is output in synchronization with the falling of this signal.
$\overline{\text{HWR}}$	80	C10	Input	Host write input. Pulse written from the host. Data is input in synchronization with the rising of this signal.
$\overline{\text{HRE}}$	66	F13	Output	Host read enable output. When this signal is active (low level), the host can access the μPD77015, 77017, 77018, 77018A, or 77019 for read. • 0: Host can access for read. • 1: Host cannot access for read.
$\overline{\text{HWE}}$	67	E13	Output	Host write enable output. When this signal is active (low level), the host can access the μPD77015, 77017, 77018, 77018A, or 77019 for write. • 0: Host can access for write. • 1: Host cannot access for write
HD0-HD7	77-70	A12, B11 - B13, D11, C13, E11, E12	I/O (3S)	8-bit host data bus

Remark "3S" in the "I/O" column of the above table stands for three-state pin, and these pins go into a high-impedance state when the host does not access the μPD7701x's host interface.

(7) I/O port

Pin name	Pin No.		I/O	Function
	TQFP	BGA		
P0-P3	65-62	F12, G11, G13, G12	I/O	General-purpose I/O port. Each pin can be set in the input or output mode independently by PCD (port command register). Data is input or output via PCD and PDT (port data register).

(8) Debug interface

Pin name	Pin No.		I/O	Function
	TQFP	BGA		
TDO	88	A7	Output	Used for debugging.
TICE	89	C7	Output	Used for debugging.
TCK	90	B6	Input	Used for debugging.
TDI	91	A6	Input	Used for debugging.
TMS	92	C6	Input	Used for debugging.

(9) Others

Pin name	Pin No.		I/O	Function
	TQFP	BGA		
I.C.	6	E3	—	Internally connected: Open this pin.

Caution Be sure not to connect the I.C. pin. If any signal is applied to the I.C. pin or if the I.C. pin status is read out, normal operation of the device is not guaranteed.

2.4 Handling of Unused Pins

It is recommended that unused pins be connected as shown in the table below.

Table 2-1. Handling of Unused Pins

Pin	Direction	Recommended Connection
$\overline{\text{INT1}}$-$\overline{\text{INT4}}$	I	Connect to V_{DD}.
$\overline{\text{X}}$ / Y	O	Open
DA0-DA15	O	Open
D0-D15[Note 1]	I/O	Connect to V_{DD} via pull-up resistor, or to GND via pull-down resistor.
$\overline{\text{MRD}}$, $\overline{\text{MWR}}$	O	Open
$\overline{\text{WAIT}}$	I	Connect to V_{DD}.
$\overline{\text{HOLDRQ}}$	I	Connect to V_{DD}.
$\overline{\text{BSTB}}$	O	Open
$\overline{\text{HOLDAK}}$	O	Open
SCK1, SCK2	I	Connect to V_{DD} or GND.
SI1, SI2	I	Connect to V_{DD} or GND.
SIEN1, SIEN2	I	Connect to GND.
SOEN1, SOEN2	I	Connect to GND.
SORQ1, SORQ2	O	Open
SO1, SO2	O	Open
SIAK1, SIAK2	O	Open
HA0, HA1	I	Connect to V_{DD} or GND.
$\overline{\text{HCS}}$, $\overline{\text{HRD}}$, $\overline{\text{HWR}}$	I	Connect to V_{DD}.
$\overline{\text{HRE}}$, $\overline{\text{HWE}}$	O	Open
HD0-HD7[Note 2]	I/O	Connect to V_{DD} via pull-up resistor, or to GND via pull-down resistor.
P0-P3	I/O	Connect to V_{DD} via pull-up resistor, or to GND via pull-down resistor.
ID0-ID31	I/O	Connect to V_{DD} via pull-up resistor, or to GND via pull-down resistor.
IA0-IA15	O	Open
$\overline{\text{PWR}}$	O	Open
TCK	I	Connect to GND via pull-down resistor.
TDO,TICE	O	Open
TMS, TDI	I	Open (pulled up internally)
CLKOUT	O	Open

Remark I: Input pin, O: Output pin, I/O: I/O pin

Notes **1.** These pins may be left open if the external data memory is not accessed by program. To reduce the current consumption by using the halt mode or stop mode, however, observe the recommended connection.

2. These pins may be left open if $\overline{\text{HCS}}$, $\overline{\text{HRD}}$, and $\overline{\text{HWR}}$ are fixed to high level.
To reduce the current consumption by using the halt mode or stop mode, however, observe the recommended connection.

Chapter 3
Architecture

This chapter describes the architecture of the μPD7701x family by dividing it into several physical blocks and explaining the functions of each block. The overall organization is described in section 3.1, and the details (units) are then described in section 3.2 and following sections.

3.1 Overall Block Organization

This section divides the physical structure of the μPD7701x family into several functional blocks for explanation.

The μPD7701x family consists of the following internal units:

- Buses (main bus, X data bus, and Y data bus)

  Refer to section 3.2 “Buses.”
- System control units

  Refer to section 3.3 “System Control Units.”
- Program control unit

  Refer to section 3.4 “Program Control Unit.”
- Data addressing unit

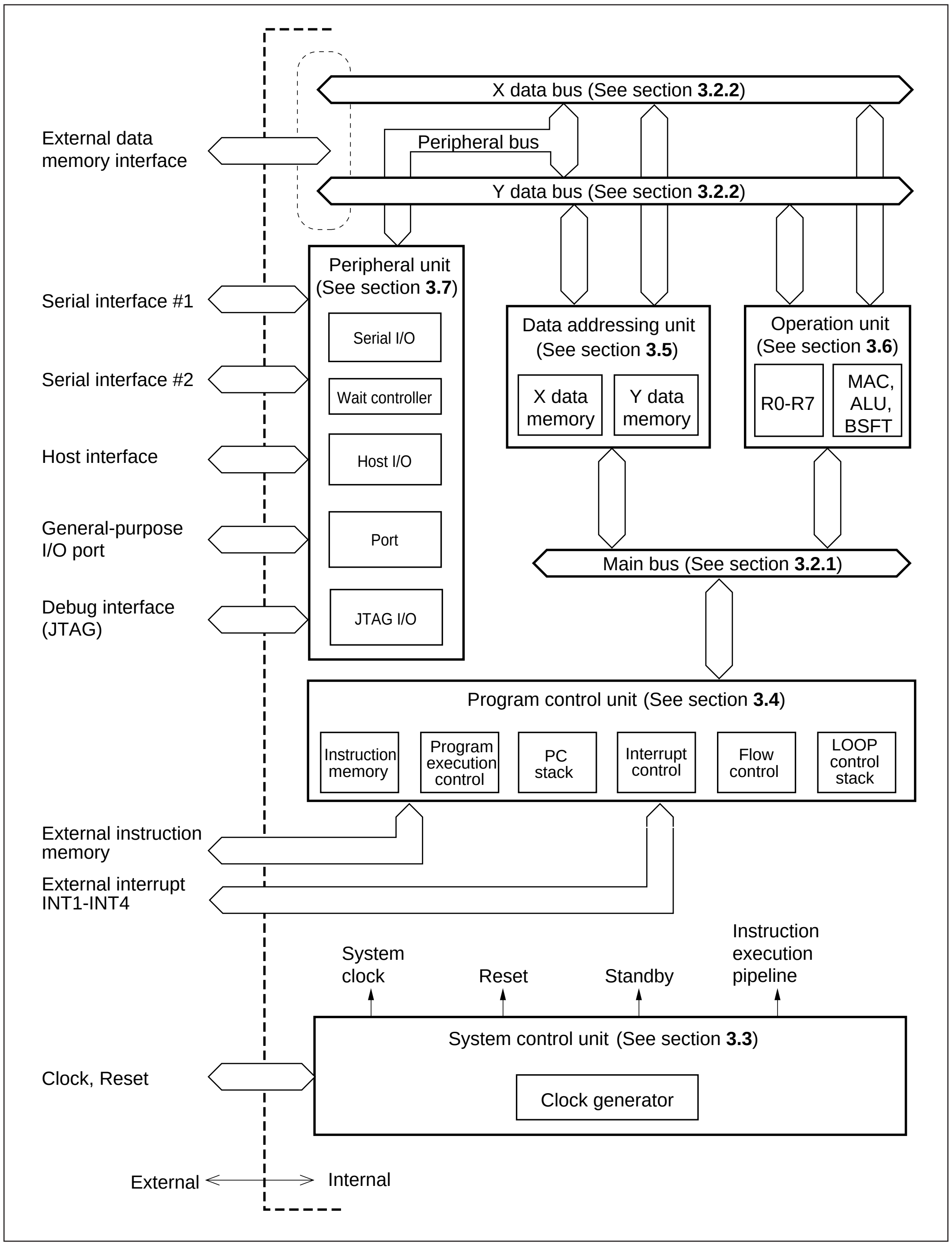
  Refer to section 3.5 “Data Addressing Unit.”
- Operation unit

  Refer to section 3.6 “Operation Unit.”
- Peripheral unit

  Refer to section 3.7 “Peripheral Unit.”

Figure 3-1 illustrates the block organization. Refer to the corresponding sections for the functions of the respective blocks.

Figure 3-1. Overall Block Organization


X data bus (See section **3.2.2**)
Peripheral bus
Y data bus (See section **3.2.2**)
External data memory interface
Serial interface #1
Serial interface #2
Host interface
General-purpose I/O port
Debug interface (JTAG)
Peripheral unit (See section **3.7**)
Serial I/O
Wait controller
Host I/O
Port
JTAG I/O
Data addressing unit (See section **3.5**)
X data memory
Y data memory
Operation unit (See section **3.6**)
R0-R7
MAC, ALU, BSFT
Main bus (See section **3.2.1**)
Program control unit (See section **3.4**)
Instruction memory
Program execution control
PC stack
Interrupt control
Flow control
LOOP control stack
External instruction memory
External interrupt INT1-INT4
System clock
Reset
Standby
Instruction execution pipeline
System control unit (See section **3.3**)
Clock generator
Clock, Reset
External
Internal

3.2 Buses

A bus transfers data between external devices and the processor. The μPD7701x family is provided with the following three types of buses:

- Main bus
- X data bus
- Y data bus

3.2.1 Main bus

(1) Function

This 16-bit bus connects the general-purpose registers (R0-R7) and control registers, etc. It transfers data when the following instructions are executed:

- Register-to-register transfer instruction

This instruction transfers data between the L part of a general-purpose register and a non-general-purpose register. These registers are listed in Table 3-1. Note that this instruction transfers only the L part of a general-purpose register.

For details, refer to "μPD7701x Family User's Manual Instructions."

Caution A general-purpose register consists of 40 bits. These 40 bits are divided into three parts: L (lower 16 bits), H (16 bits in the middle), and E (higher 8 bits). For details, refer to section 3.6.2 "General-purpose registers and data formats."

- Immediate value setting instruction

This instruction sets immediate data to a specified register. Of the registers listed in Table 3-1, the following can be specified.

- General-purpose registers (L part (R0L-R7L) only)
- Data pointers (DP0-DP7)
- Index registers (DN0-DN7)
- Modulo registers (DMX, DMY)

For the details of this instruction, refer to "μPD7701x Family User's Manual Instructions."

(2) Registers connected to main bus

The table shown below lists the registers connected to the main bus.

Table 3-1. Registers Connected to Main Bus

Register name	Assembler-reserved name	Load (L)/store (S)
General-purpose register	R0L-R7L (L part of R0-R7)	L/S
Data pointer	DP0-DP7	L/S
Index register	DN0-DN7	L/S
Modulo register	DMX, DMY	L/S
Stack	STK	L/S
Stack pointer	SP	L/S
Loop counter	LC	L/S
Loop stack (LSTK)	LSR1, LSR2, LSR3	L/S
Loop stack pointer	LSP	L/S
Status register	SR	L/S
Interrupt enable flag stack register	EIR	L/S
Error status register	ESR	L/S

3.2.2 Data bus

(1) Function

This 16-bit bus connects the general-purpose registers, X and Y data memories, and internal peripherals. It transfers data when the following instructions are executed.

- Parallel load/store instruction
- Partial load/store instruction
- Direct addressing load/store instruction
- Immediate value index load/store instruction

For the details of the load/store instruction, refer to "μPD7701x Family User's Manual Instructions."

The data bus is classified into X data bus, Y data bus, and peripheral bus. The logical and physical relations among these buses are shown in Table 3-2.

Table 3-2. Functional Block and Bus

Functional block	Relations among X data bus, Y data bus, and peripheral bus
Internal memory peripherals	X and Y data buses are logically and physically separated. Therefore, both the X and Y data buses are validated for transfer by a single instruction.
Internal peripheral	X and Y data buses are logically and physically connected. Even when a peripheral-related register is accessed from X or Y memory space, therefore, the same peripheral register is accessed as long as the address is the same. At this time, however, the peripheral register cannot be accessed simultaneously from the X and Y data memory spaces with a single instruction.
External memory	Although the X and Y data buses are logically separated, they are physically common. Therefore, the X and Y external memories cannot be accessed simultaneously with a single instruction.

(2) X data bus

This 16-bit bus connects the general-purpose registers, X data memory, and the bus from the internal peripherals. This bus transfers data when the following instructions are executed:

- Parallel load/store instruction (for X memory)
- Partial load/store instruction (for X memory)
- Direct addressing load/store instruction (for X memory)
- Immediate value index load/store instruction (for X memory)

Cautions 1. Although the X and Y data buses are separated inside the device, a single data bus is commonly used externally. Thus, an instruction that accesses both external memories cannot be executed in the same instruction cycle.

2. The same peripheral register is accessed for internal peripheral regardless of whether the X or Y memory is accessed, as long as the address is the same.

3. Even in the case of 2 above, a peripheral register cannot be accessed from both the X and Y memory spaces in the same instruction cycle.

The table shown below shows the registers and memories connected to the X data bus.

Table 3-3. Registers and Memories Connected to X Data Bus

Register/memory name	Assembler-reserved name	Load (L)/store (S)
General-purpose register	R0-R7 R0E-R7E R0H-R7H R0L-R7L R0EH-R7EH	L/S
X internal RAM	—	L/S
X internal ROM (not for the μPD77016)	—	from ROM to bus only
External memory	—	L/S
Internal peripheral	—	L/S

Caution A general-purpose register consists of 40 bits. These 40 bits are divided into three parts: L (lower 16 bits), H (16 bits in the middle), and E (higher 8 bits). Any of these parts can be specified for transfer. For details, refer to section 3.6.2 "General-purpose registers and data formats."

(3) Y data bus

This 16-bit bus connects the general-purpose registers, Y data memory, and the bus from the internal peripherals. This bus transfers data when the following instructions are executed:

- Parallel load/store instruction (for Y memory)
- Partial load/store instruction (for Y memory)
- Direct addressing load/store instruction (for Y memory)
- Immediate value index load/store instruction (for Y memory)

Cautions 1. Although the X and Y data buses are separated inside the device, a single data bus is commonly used externally. Thus, an instruction that accesses both external memories cannot be executed in the same instruction cycle.

2. The same peripheral register is accessed for internal peripheral units regardless of whether the X or Y memory is accessed, as long as the address is the same.

3. Even in the case of 2 above, a peripheral register cannot be accessed from both the X and Y memory spaces in the same instruction cycle.

Table 3-4 shows the registers and memories connected to the Y data bus.

Table 3-4. Registers and Memories Connected to Y Data Bus

Register/memory name	Assembler-reserved name	Load (L)/store (S)
General-purpose register	R0 - R7 R0E - R7E R0H - R7H R0L - R7L R0EH - R7EH	L/S
Y internal RAM	—	L/S
Y internal ROM (not for the μPD77016)	—	from ROM to bus only
External memory	—	L/S
Internal peripheral	—	L/S

Caution A general-purpose register consists of 40 bits. These 40 bits are divided into three parts: L (lower 16 bits), H (16 bits in the middle), and E (higher 8 bits). Any of these parts can be specified for transfer. For details, refer to section 3.6.2 "General-purpose registers and data formats."

(4) Peripheral bus

This 16-bit bus connects the internal peripheral registers and the X/Y data buses. The peripheral registers are commonly mapped on the X/Y memory spaces. Data is transferred by executing the following instructions.

- Parallel load/store instruction (for peripheral register)
- Partial load/store instruction (for peripheral register)
- Direct addressing load/store instruction (for peripheral register)
- Immediate value index load/store instruction (for peripheral register)

For the details of the peripheral bus, refer to section 3.7 "Peripheral Units."

Cautions 1. The same peripheral register is accessed for internal peripheral regardless of whether the X or Y memory is accessed, as long as the address is the same.

2. Even in the case of 1 above, a peripheral register cannot be accessed from both the X and Y memory spaces in the same instruction cycle.

3.3 System Control Units

The following basic functions, which support the digital signal processor operations of the μPD7701x family, are called system control units:

- Clock generator
- Reset function
- Pipeline architecture
- Standby function

3.3.1 Clock generator

The clock generator is a circuit that generates and controls the system clock supplied to the CPU. The configuration of this circuit differs between the μPD77016 and μPD77015/77017/77018/77018A/77019.

(1) μPD77016

An internal system clock is generated from an external clock input to the CLKIN pin. This internal system clock serves as a reference of the internal basic timing of the device. The internal system clock is also output from the CLKOUT pin to establish synchronization between external devices and the μPD77016. At this time, the frequency ratio of the external clock to the internal system clock is 2 : 1.

Figure 3-2 shows the clock circuit, and Figure 3-3 shows the timing requirements.

Figure 3-2. Clock Circuit of μPD77016

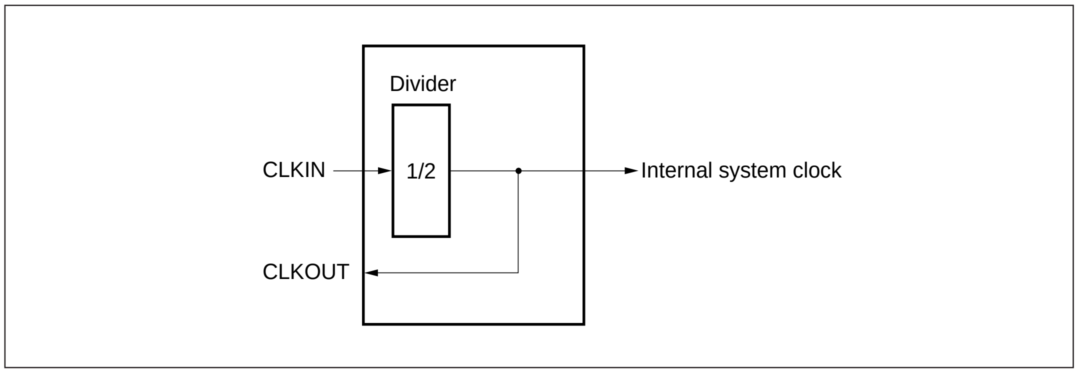
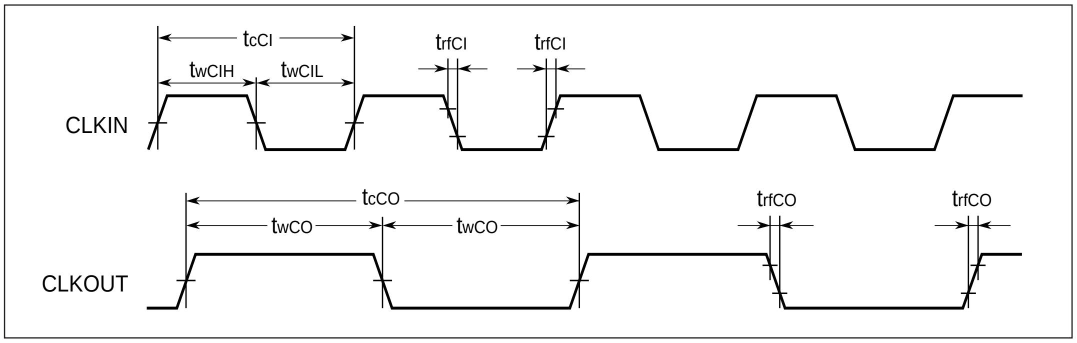


Figure 3-3. Clock Timing of μPD77016



(2) μPD77015, 77017, 77018, 77018A, 77019

[External clock input]

An internal system clock is generated from an external clock input to the X1 pin. This internal system clock serves a reference of the internal basic timing.

The internal system clock is also output from the CLKOUT pin to establish synchronization between external devices and μPD77015, 77017, 77018, 77018A, 77019 (this function can be disabled by mask option).

The external clock is multiplied by PLL. The multiple can be specified by mask option. At this time, the frequency ratio of the external clock to the internal system clock is selected from the following alternatives:

- 1 (external) : 1 (internal)
- 1 (external) : 2 (internal)
- 1 (external) : 3 (internal) (for μPD77018A and 77019 only)
- 1 (external) : 4 (internal)
- 1 (external) : 8 (internal)

Assuming that the internal system clock frequency is 33 MHz, for example, the input clock frequencies are as follows:

- 1 : 1 —> external 33 MHz
- 1 : 2 —> external 16.5 MHz
- 1 : 3 —> external 11 MHz (for μPD77018A and 77019 only)
- 1 : 4 —> external 8.25 MHz
- 1 : 8 —> external 4.125 MHz

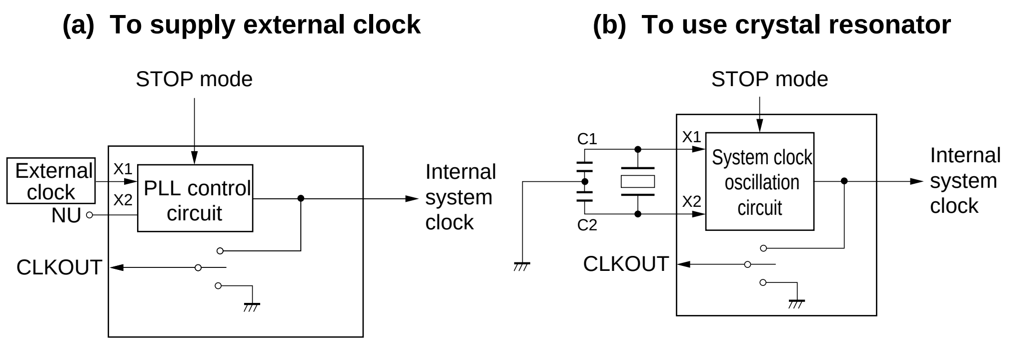
[When using crystal resonator]

If a frequency ratio of 1 : 1 is selected by mask option, a crystal resonator can be connected across the X1 and X2 pins to configure a self-oscillation circuit.

Figure 3-4 shows the clock circuit of the μPD77015, 77017, 77018, 77018A, and 77019 and Figure 3-5 shows the timing.

For how to order mask option, refer to Appendix B "Ordering Information."

Figure 3-4. Clock Circuit of μPD77015, 77017, 77018, 77018A, 77019



Remark NU: Not use (leave this pin open.)

Cautions 1. Be sure to specify a multiple of 1 by mask option when using a crystal resonator with the μPD77015, 77017, 77018, 77018A, or 77019. The processor does not operate with any other multiples.

2. The multiplication factor of the mask option of the μPD77019-013 is fixed to 4. The crystal resonator cannot be used with this model.

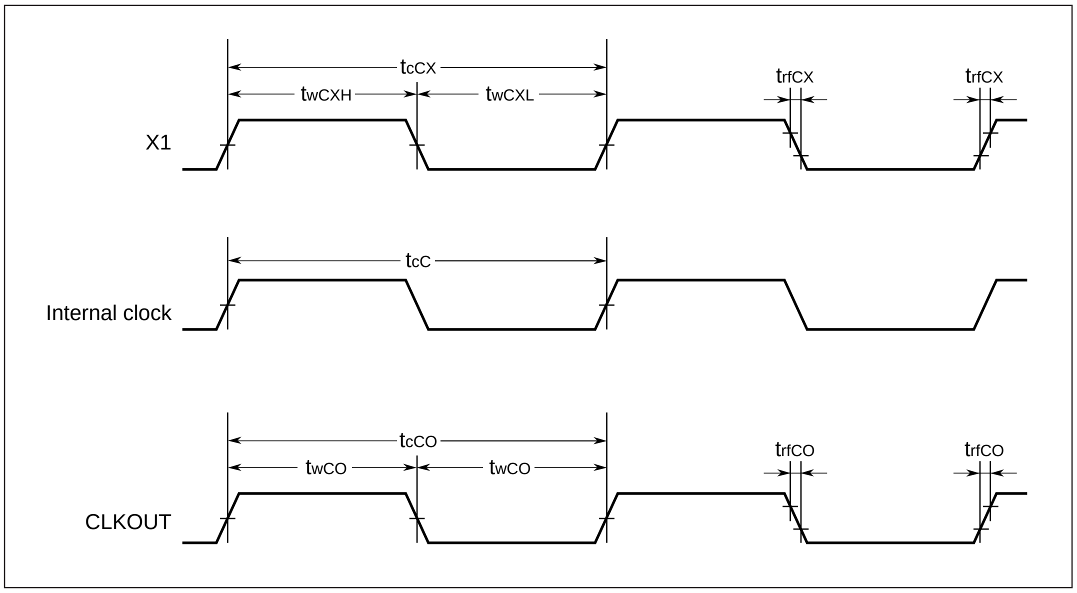
---

(3) Clock operation in standby mode

The operating status of the system clock is in the HALT/STOP mode is as follows:

		STOP mode	HALT mode
μPD77016		—	enable to stop
μPD77015/77017/77018 /77018A/77019	ext. clock + PLL	stops	1/8 of ext. clock × PLL factor
	crystal resonator	stops	1/8 of crystal resonator frequency

Figure 3-5. Clock Timing of μPD77015, 77017, 77018, 77018A, 77019



3.3.2 Reset function

The hardware of the device is reset when the signal input to the $\overline{\text{RESET}}$ pin is activated (low level). The purpose of resetting is to correctly initialize the device before program execution. The registers and pins to be initialized, and their initial values are shown in Tables 3-5 to 3-7. Figure 3-6 shows the reset timing.

For details of how the value of each pin and each register changes depending on the respective boot operations, refer to Chapter 4 “Boot Function.”

Table 3-5. CPU Registers to Be Initialized and Their Initial Values

Register name	Initial value	Description
SR	0xF000	Interrupts of all sources are enabled but interrupts are disabled generally at all the present and past levels. Loop operation is not performed.
PC	0	Address 0 is a boot area and execution branches to address 0x200 after boot processing has ended. Therefore, the reset entry as a user area is at address 0x200.
SP	0	—
LC	0b1xxx xxxx xxxx xxxx	Indicates that loop operation is not performed. The count value itself is undefined.
LSP	0	—
RC	0b1xxx xxxx xxxx xxxx	Indicates that repeat operation is not performed. The count value itself is undefined.
EIR	0xFFFF	Indicates that all the interrupts are disabled at all the present and past levels.
ESR	0	—

Table 3-6. Initialized Memory-mapped Registers and Their Initial Values

Register name	Initial value	Description
SST1, SST2	0x0002	The serial interface is initialized as follows: • MSB first for both input and output • 16-bit length for both input and output • Wait is not used for load/store of SDT • Status transition mode • Clears error flag of SDT load/store • Data store to SDT enabled • No data to be loaded from SDT
PCD	0x0000	The I/O ports are initialized as follows: • No bit manipulation • No mode setting
HST	0x0301	The host interface is initialized as follows: • Wait is not used for HDT access • Disables HRE and HWE functions • Clears UF0 and UF1 to zero • Clears error flag for host read/write • Clears error flag of HDT load/store • Disables read from host • Enables write to host

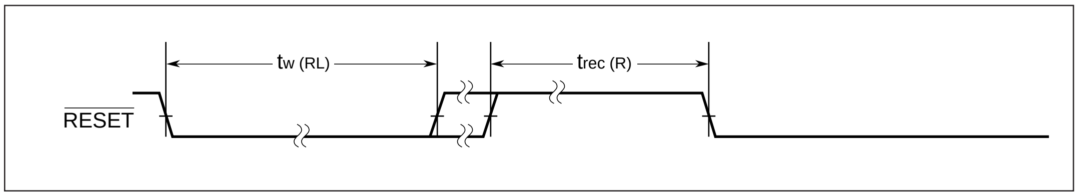
Table 3-7. Initialized Pins and Their Initial Statuses

Pin name	Initial Status
$\overline{\text{X}}$/Y	Low-level output**Note 1**
DA0-DA15	Low-level output**Note 1**
D0-D15	High-impedance
IA0-IA15 **Note 2**	Low-level output (high-impedance during reset)
ID0-ID31 **Note 2**	High-impedance
$\overline{\text{PWR}}$ **Note 2**	High-level output (high-impedance during reset)
$\overline{\text{MRD}}$, $\overline{\text{MWR}}$, $\overline{\text{BSTB}}$	High-level output **Note 1**
SORQ1, SORQ2, SIAK1, SIAK2	Low-level output
SO1, SO2	High-impedance
$\overline{\text{HRE}}$, $\overline{\text{HWE}}$	High-level output
P0-P3	Input status
TICE	Low-level output

★ **Notes** **1.** These pins go into a high-impedance state when the bus is released ($\overline{\text{HOLDAK}}$ = 0). The bus can be released even during reset when $\overline{\text{HOLDRQ}}$ = 0.

2. μPD77016 only.

Figure 3-6. Reset Timing



3.3.3 Pipeline architecture

The μPD7701x family employs pipeline architecture to enhance the execution speed. Generally, one instruction completes its processing via several machine cycles each of which performs elemental processing. The instructions of the μPD7701x family have the following three machine cycles:

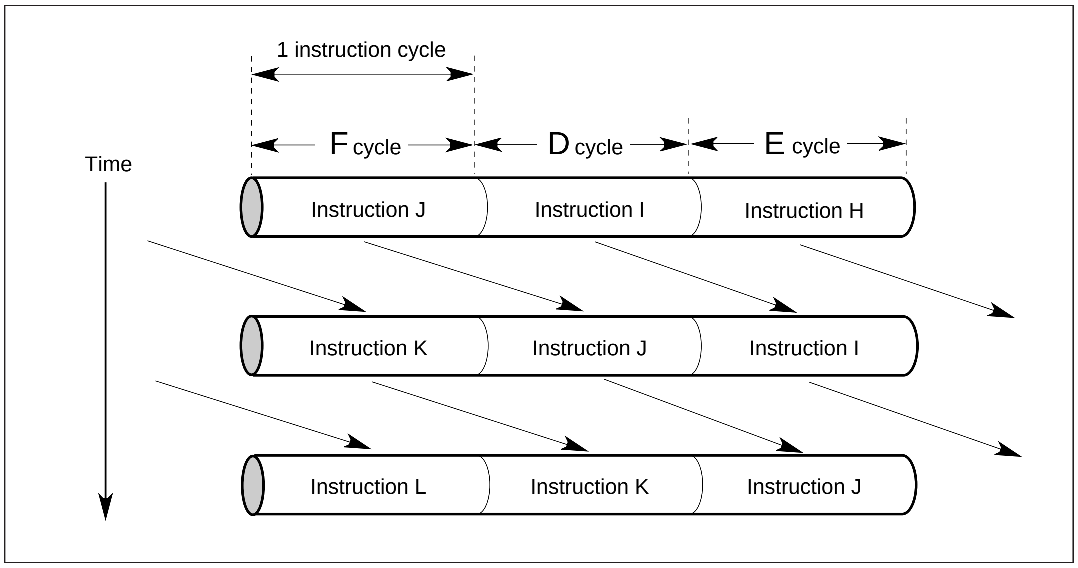
F	: instruction fetch cycle Reads an op code from the instruction memory.
D	: decode cycle Decodes the read op code.
E	: execution cycle Executes the decoded result.

The part that executes each machine cycle is called a pipeline stage. Each stage independently completes processing with the same number of clocks (1 cycle). Therefore, an instruction under execution enters stages one after another without wait time. In addition, three instructions can exist in the respective three stages at the same time. In other words, it seems as if one instruction were processed with the execution time of one stage as long as the instruction passes through the pipelines without any instruction stream fault. The number of clock cycles in one stage is called one instruction cycle, which is 30 ns in the case that operates with a 33-MHz clock.

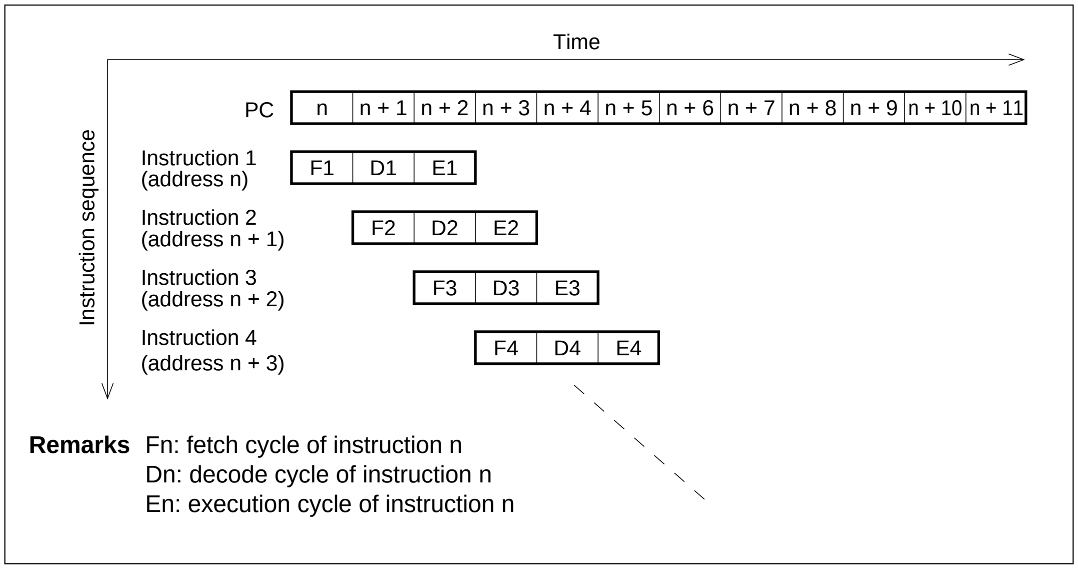
Figure 3-7 provides images of pipeline operation. Figure 3-7(a) is a conceptional illustration that shows the flow of executed instructions when viewed from each pipeline stage. Figure 3-7(b) shows the sequence in which instructions are executed in pipeline, from the viewpoint of each instruction.

Figure 3-7. Pipeline Image

(a) Pipeline image 1



(b) Pipeline image 2



Remarks Fn: fetch cycle of instruction n
Dn: decode cycle of instruction n
En: execution cycle of instruction n

(1) Successive MAC, ALU, Barrelshifter operations

When an instruction performing arithmetic/logic operations uses the result of the operation executed by the preceding instruction as an input operand, the result of the operation is written to a general-purpose register and, at the same time, input to the operation unit for the operation by the subsequent instruction. Consequently, programming can be done without having to be aware of the pipeline.

(2) Branch instruction

If a pipeline hazard occurs as a result of executing a branch instruction, the pipeline is replenished again with a NOP instruction inserted into the delay slot. Though the execution time is consequently extended, this does not cause erroneous application operation, and there is no need for users to consider the pipeline operation even in programming branch instructions.

For further details about pipeline timing with branch instructions refer to section 3.4.2 “Program execution control block”.

Caution **The delay due to the processing of the pipeline must be taken into account in the following cases:**

- **Instructions that control interrupts (by setting EIR, etc.) requires two instruction cycles to update the interrupt control information (refer to section 3.4.4 “Interrupt”).**
- **When a value is set to DPn by using a Inter-register transfer instruction or immediate value setting instruction, the memory cannot be accessed by using the value set to DPn as an address until the instruction that follow the instruction that has set a value to DPn.**

  Example:
  ```
  inst#1   DP0 = 0x0100     ;
  inst#2   NOP              ;  DP0 cannot be used here!
  inst#3   R0L = *DP0       ;
  ```
- **The branch instruction cannot be written within three instructions before the loop end (refer to section 3.4.3 “Flow control block”).**

3.3.4 Standby function

The μPD7701x family is provided with a standby function that stops the device to reduce the current consumption. The device enters a standby status when an appropriate instruction is executed. This status is called a standby mode. The standby mode is set by two types of instructions: HALT and STOP. The μPD77016 does not have the STOP mode.

(1) Standby mode by HALT instruction

This standby mode is common to all models of the μPD7701x family and is called HALT mode. This mode is set by the HALT instruction. The current consumption of the device in the HALT mode is reduced. The HALT mode is set or released as follows:

(a) The HALT mode is set by executing the HALT instruction.

(b) At this time, the registers and internal memory retain the status immediately before the HALT mode is set, and the current consumption of the device decreases. The status of each pin of the device is shown in Table 3-8.

(c) This mode is released by using an external/internal interrupt (which is not masked) or hardware reset (refer to section 3.4.4 “Interrupt”).

(d) When the HALT mode has been released by using an interrupt, the return address to which execution returns after the interrupt processing is the address of the instruction next to the HALT instruction. Before the HALT mode is released, a heat-up cycle (NOP) of one instruction cycle is inserted so that the system can restore from the power-down status.

(e) For the μPD77016 only in this mode, the external clock may be stopped (fixed to high or low level). To release this mode by using an interrupt, however, the clock must be restored before the interrupt is executed.

★ ***Table 3-8. Pin Status in HALT Mode***

Pin name	When $\overline{\text{HOLDRQ}}$ is Active (low level)	When $\overline{\text{HOLDRQ}}$ is Inactive (high level)
CLKOUT[Note]	×1/8 clock output	
$\overline{\text{X}}$/Y	High impedance	Low level
IA0 - IA15	Retains status immediately before	
DA13 - DA0	High impedance	Retains status immediately before
D15 - D0	High impedance	
$\overline{\text{MRD}}$/$\overline{\text{MWR}}$	High impedance	High level
$\overline{\text{HOLDAK}}$	Low level	High level
$\overline{\text{BSTB}}$	High level	
SORQ1, SIAK1, SO1, SO2	Retains status immediately before	
$\overline{\text{HRE}}$, $\overline{\text{HWE}}$, HD7 - HD0		
P3 - P0		
TDO, TICE		

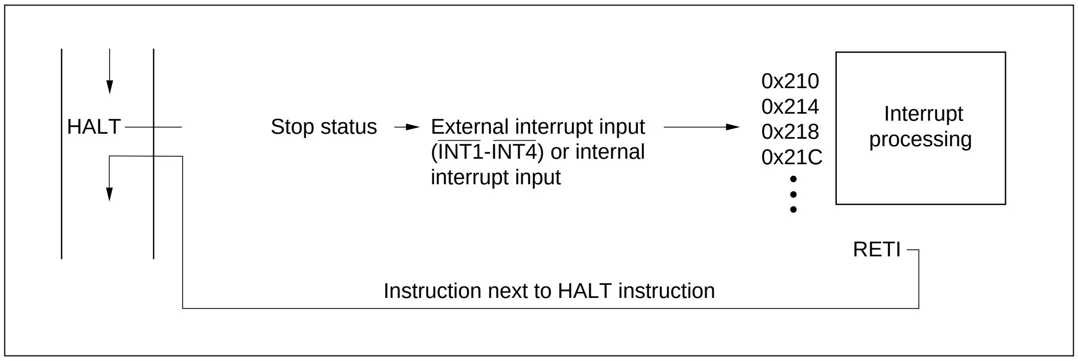
Caution Fix the input pins and pins that go into a high-impedance state in the HALT mode to the high or low level.

Note With the μPD77015, 77017, 77018, 77018A, and 77019, if CLKOUT is activated low by mask option, it remains low.

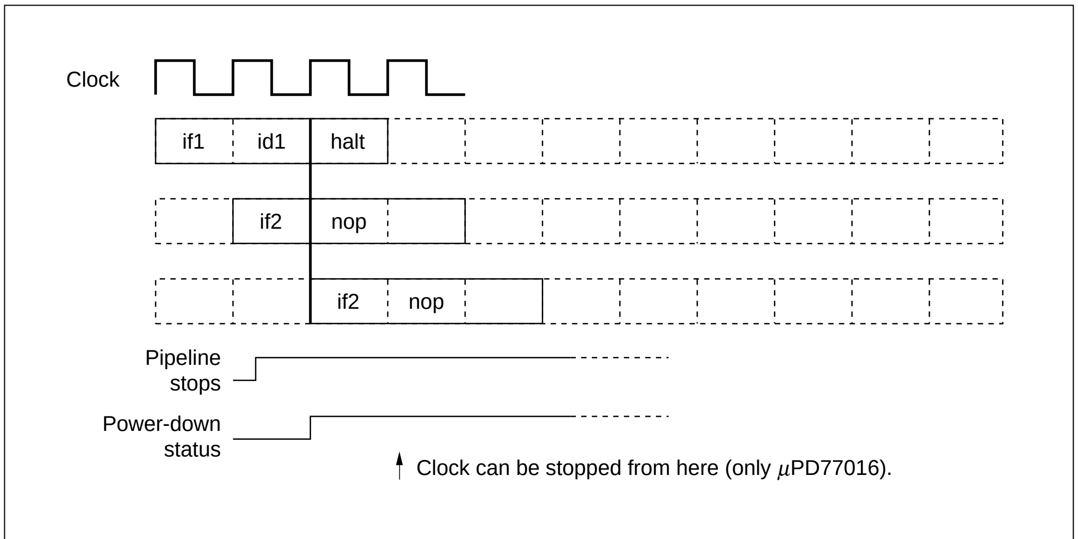
Figure 3-8 (a) illustrates how the HALT mode is released by using an interrupt. Figure 3-8 (b) and (c) show the timings of setting and releasing the HALT mode, respectively.

Figure 3-8. HALT Mode

(a) Releasing from HALT mode (by using interrupt)

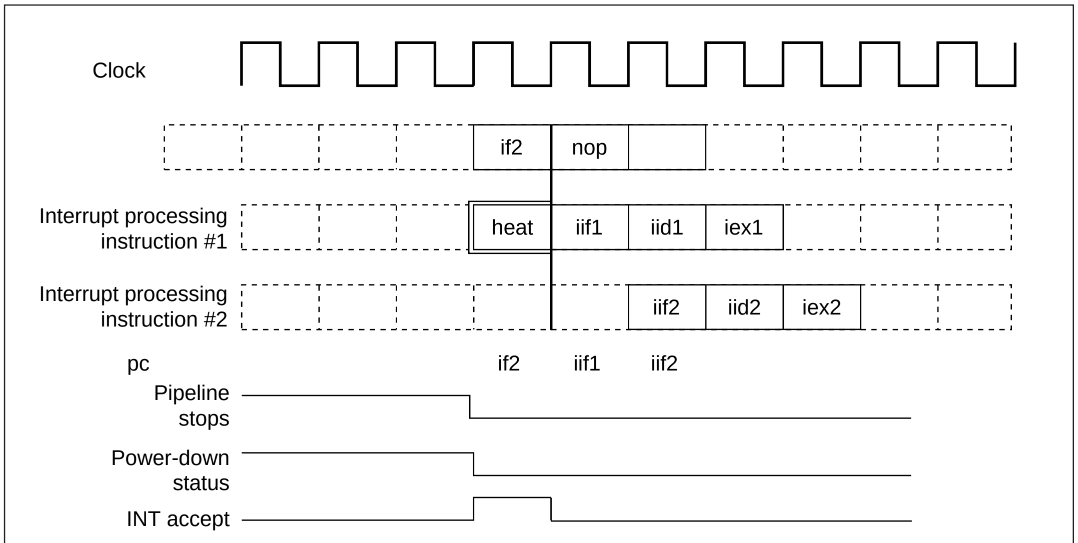

HALT
Stop status
External interrupt input ($\overline{\text{INT1}}$-$\overline{\text{INT4}}$) or internal interrupt input
0x210
0x214
0x218
0x21C
Interrupt processing
RETI
Instruction next to HALT instruction

(b) Timing of setting HALT mode



Remarks if : instruction fetch
id: instruction decode

(c) Timing of releasing HALT mode



Remarks ifn : instruction fetch
iifn : interrupt instruction fetch
iidn : interrupt instruction decode
iexn: interrupt instruction execution

(2) Standby mode by STOP instruction

This mode, called STOP mode, can be set with the μPD77015/77017/77018/77018A/77019 only. The STOP mode is set by executing the STOP instruction. In this mode, the current consumption of the device is reduced to several tens of μA. The STOP mode is set or released as follows:

1. The STOP mode is set when the STOP instruction is executed.
2. At this time, the status of each pin of the device is as shown in Table 3-9.
3. The clock circuit and PLL stop, and the current consumption of the device is reduced to several tens of μA or less.
4. The device is released from the STOP mode only by hardware reset. At this time, PLL takes some time to be released from the mode. Therefore, assert the reset signal active for at least 1 ms.

★ ***Table 3-9. Pin Status in STOP Mode***

Pin name	When $\overline{\text{HOLDRQ}}$ is Active (low level)	When $\overline{\text{HOLDRQ}}$ is Inactive (high level)
CLKOUT	Low level	
$\overline{\text{X}}$/Y	High impedance	Low level
DA13 - DA0	High impedance	Retains status immediately before
D15 - D0	High impedance	
$\overline{\text{MRD}}$/$\overline{\text{MWR}}$	High impedance	High level
$\overline{\text{HOLDAK}}$	Low level	High level
$\overline{\text{BSTB}}$	High level	
SORQ1, SIAK1, SO1, SO2	Retains status immediately before	
$\overline{\text{HRE}}$, $\overline{\text{HWE}}$, HD7 - HD0		
P3 - P0		
TDO, TICE		

Caution Fix the input pins and pins that go into a high-impedance state in the STOP mode to the high or low level.

When the STOP mode is released by means of hardware reset, the output pins are initialized. Some output pins, however, are in the undefined status until the PLL of the device is stabilized, and their operation is not guaranteed.

Table 3-10 shows the status of each output pin during the reset period following releasing the STOP mode.

Table 3-10. Output Pin Status during Reset Period after Releasing STOP Mode

Pin name	Initialized status	Status during reset of STOP mode release
CLKOUT	system clock Note	undefined Note
$\overline{X}$/Y	low level	undefined
DA0-DA13	0x0000	
D0-D15	high-impedance	
$\overline{MRD}$	high level	
$\overline{MWR}$		
$\overline{BSTB}$		
$\overline{HOLDAK}$		
HD0-HD7	high-impedance	
$\overline{HRE}$	high level	
$\overline{HWE}$		
SO1, SO2	high-impedance	
SIAK1	low level	
SORQ1		
P0-P3	input mode	

Note When CLKOUT is fixed to low level by mask option, low level is output in even status of Initializing or status during reset for releasing STOP mode.

3.4 Program Control Unit

This unit controls program execution. Data can be loaded from or stored to the registers in this unit via the main bus. This unit plays a role in execution of the following instructions:

- General instruction execution
- Branch instruction
- Hardware loop instruction
- Interrupt (Although an interrupt is not an instruction, PC, STK, SP, SR, and EIR are automatically controlled by INTC.)

Execution of these instructions is controlled by the following three blocks of the program control unit:

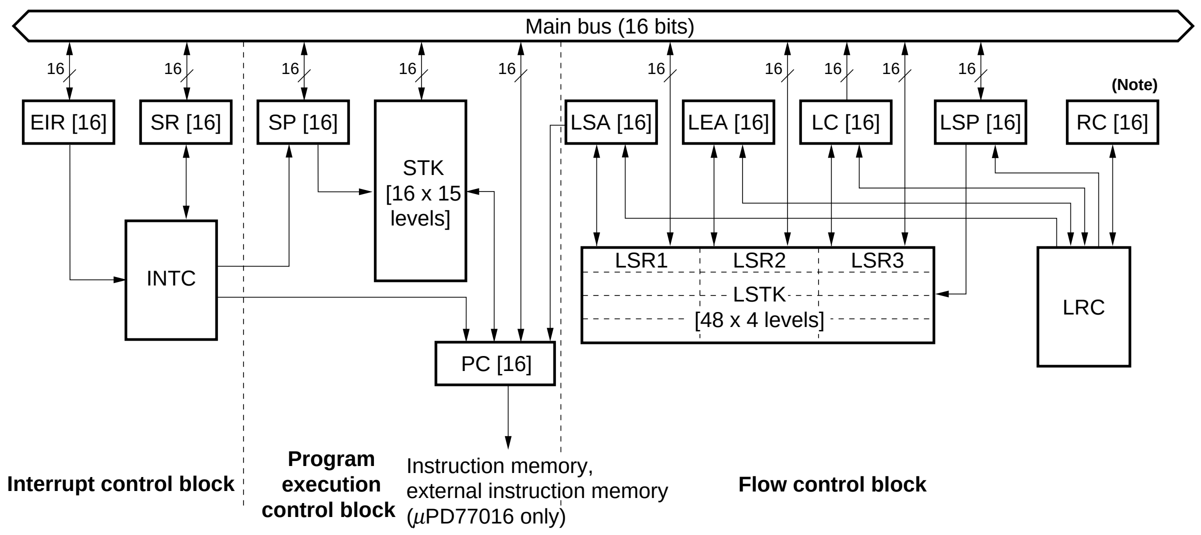
- Program execution control block
- Flow control block
- Interrupt control block

Section 3.4.1 “Block configuration” shows a detailed block diagram of the program control unit. Section 3.4.2 “Program execution control block” through section 3.4.4 “Interrupt” describe the details of the functions of the respective blocks.

3.4.1 Block configuration

Figure 3-9 shows the block configuration of the program control unit.

Figure 3-9. Program Control Unit



Note RC cannot directly transfer data via the main bus.

3.4.2 Program execution control block

Program execution is controlled by the following registers:

- Program counter (PC)
- Stack (STK)
- Stack pointer (SP)

(1) Program counter (PC)

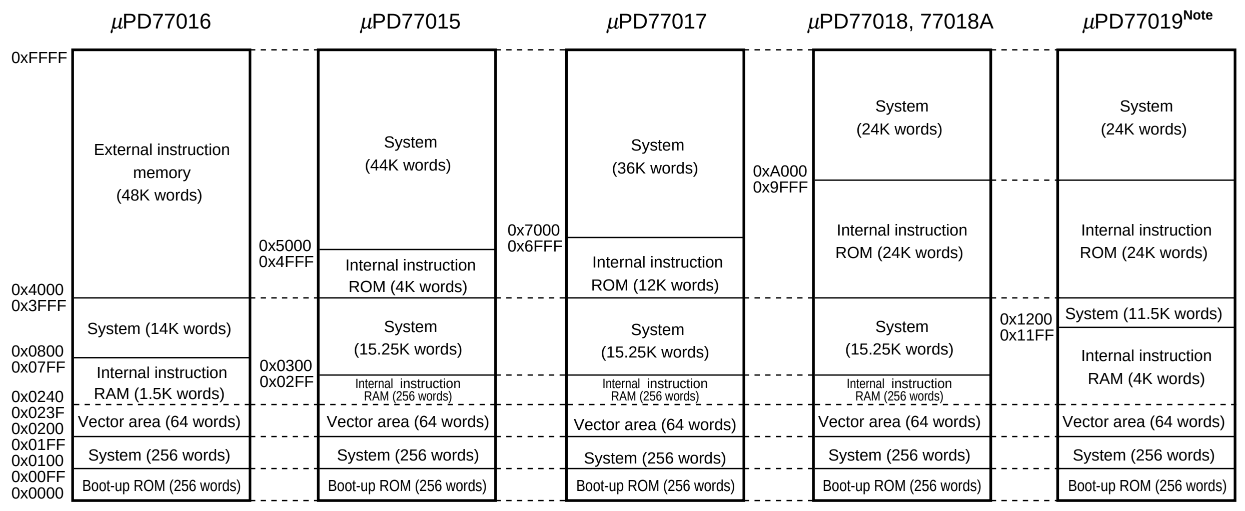
This is a 16-bit register that holds the address of the instruction currently under execution when the program is executed. Therefore, the range of the value the PC can take is the entire memory space.

Caution The PC can take any value as long as it is in the range of 16 bits, but the portion that is not defined as the instruction memory space or the portion that is reserved for the system must not be accessed.

(a) Instruction memory

The instruction memory space of the μPD7701x family is shown below.

★ ***Figure 3-10. Instruction Memory Space***



Caution No program or data must be stored to the addresses reserved for the system, nor must these addresses be accessed. If any of these addresses is accessed, normal operation of the μPD7701x family is not guaranteed.

Note The μPD77019-013 does not have the internal ROM of the μPD77019.

(b) Internal instruction memory

The *μ*PD7701x family is provided with ROM or RAM as an internal instruction memory. The capacity of the internal instruction memory differs depending on the model, as shown in Table 3-11. The internal ROM of the *μ*PD77019-013 cannot be used.

Table 3-11. Capacity of Internal Instruction Memory

Part number	Internal ROM capacity	Internal RAM capacity
*μ*PD77016	None	1.5K words
*μ*PD77015	4K words	256 words
*μ*PD77017	12K words	
*μ*PD77018	24K words	
*μ*PD77018A		
*μ*PD77019		4 K words

(c) External instruction memory

The *μ*PD7701x can be connected with an external instruction memory. The capacity of this external memory is shown in Table 3-12.

Table 3-12. Capacity of External Memory

Part number	External instruction memory capacity
*μ*PD77016	48K words
*μ*PD77015	None
*μ*PD77017	
*μ*PD77018	
*μ*PD77018A	
*μ*PD77019	

(d) Interfacing external instruction memory

The *μ*PD77016 can be connected with an external instruction memory. The application program, however, cannot handle the external instruction memory as data. The application program can access the external instruction memory only in the following two cases:

- When the program has been booted from the data memory or the host interface to the external memory by using the internal reboot function (Refer to Chapter 4 “Boot Function.”)
- When execution is branched to the external instruction memory space by using a branch instruction

The interface that connects an external instruction memory to the *μ*PD77016 is explained in the following sections.

(e) Hardware for external instruction memory expansion

The following pins are used for interfacing the external instruction memory:

- IA0-IA15 : 16 bits of address lines that provide an instruction memory address. IA15 is the MSB, and IA0 is the LSB.
- ID0-ID31 : 32-bit data bus that transfers instruction codes. ID31 is the MSB, and ID0 is the LSB.
- $\overline{\text{PWR}}$: Signal line that generates a write strobe signal to the instruction memory. Active low.

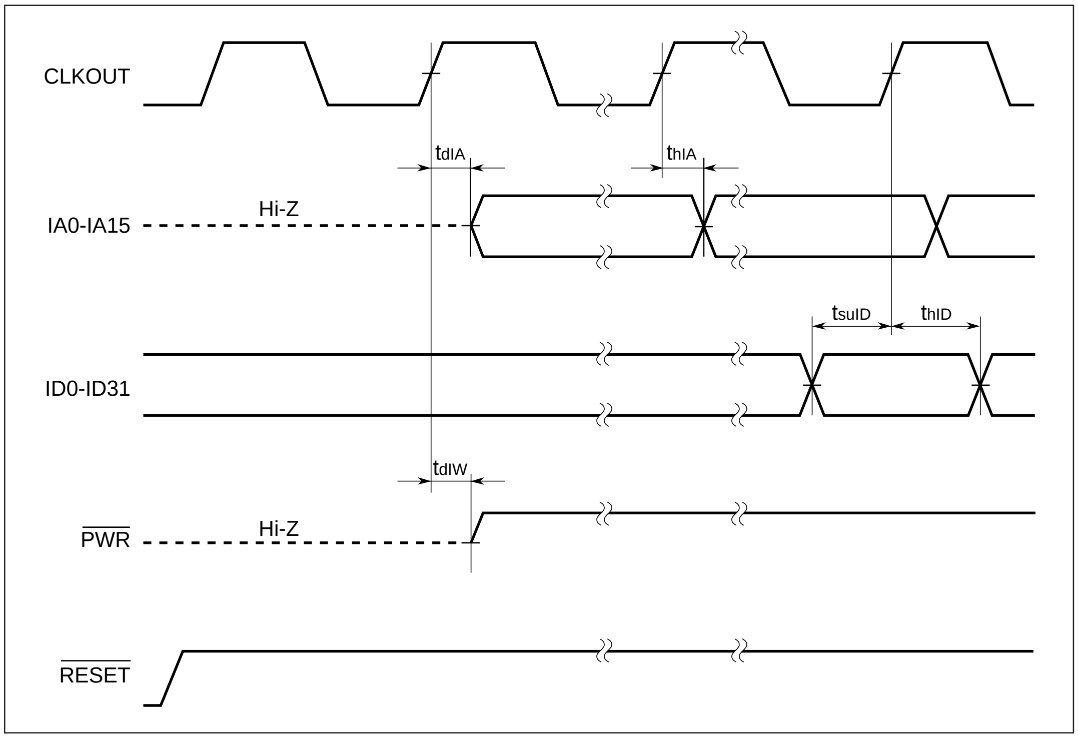
Table 3-13. Pin Statuses

Pin	I/O	During reset	Initial after reset	No external memory accesses
IA0-IA15	O	Hi-Z	L	Internal memory address currently accessed
ID0-ID31	I/O	Hi-Z	Hi-Z	Hi-Z
$\overline{\text{PWR}}$	O	Hi-Z	H	H

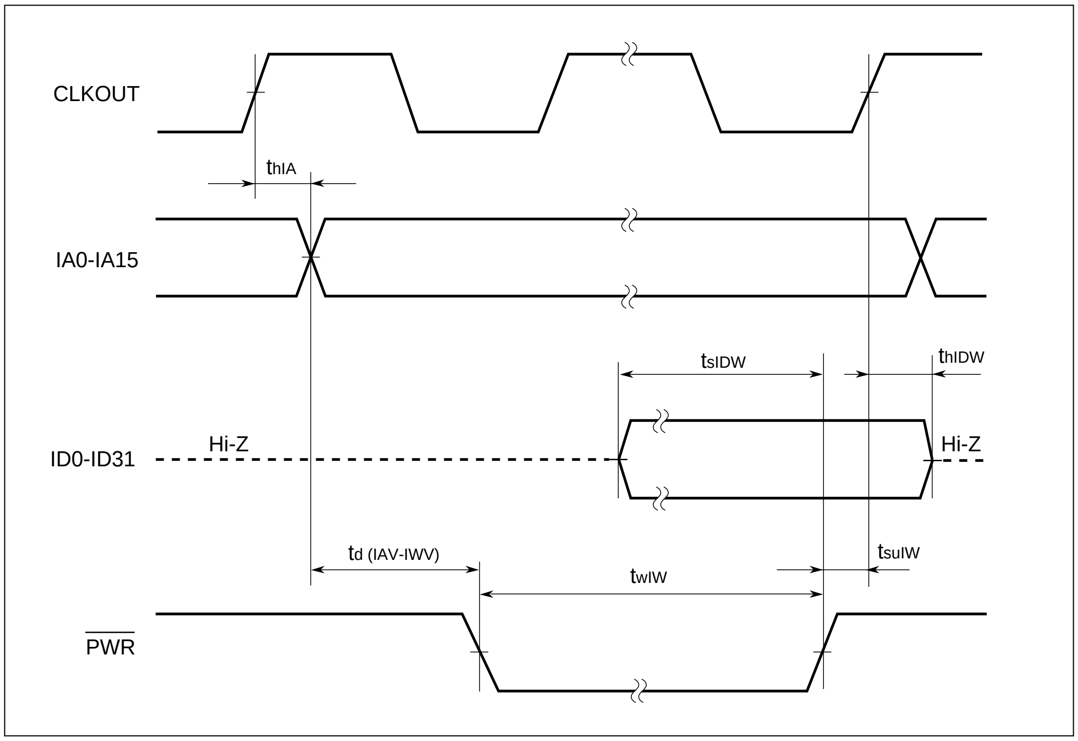
Figure 3-11 shows the operation timing.

Figure 3-11. Instruction Memory Operation Timing

(a) Read operation timing



(b) Write operation timing



(f) Wait function of external instruction memory

• IWTR (instruction memory wait cycle register)

The external instruction memory interface does not have a hardware wait function that is effected through handshaking, but is provided with a programmable wait function that is controlled via software. An instruction memory wait cycle register (IWTR) is provided as one of the internal peripheral registers, and the predetermined number of wait cycles can be selected and set to this register by the application program.

IWTR is a 16-bit register. The number of wait cycles, at what one wait cycle corresponds to one internal system clock cycle can be specified by setting bits 2 to 7 of this register. The three fields of this register, IB-ID fields, each consisting of 2 bits, correspond to three 16K-word banks which correspond to the external memory that accounts for the 3/4 of the 64K-word memory space, and wait cycles can be independently inserted to each of the three banks.

Figure 3-12 illustrates the image of this control. Table 3-14 shows the relations between the value set to each field of IWTR and the number of wait cycles.

Figure 3-12. Instruction Memory Control Banks and IWTR Field Configuration

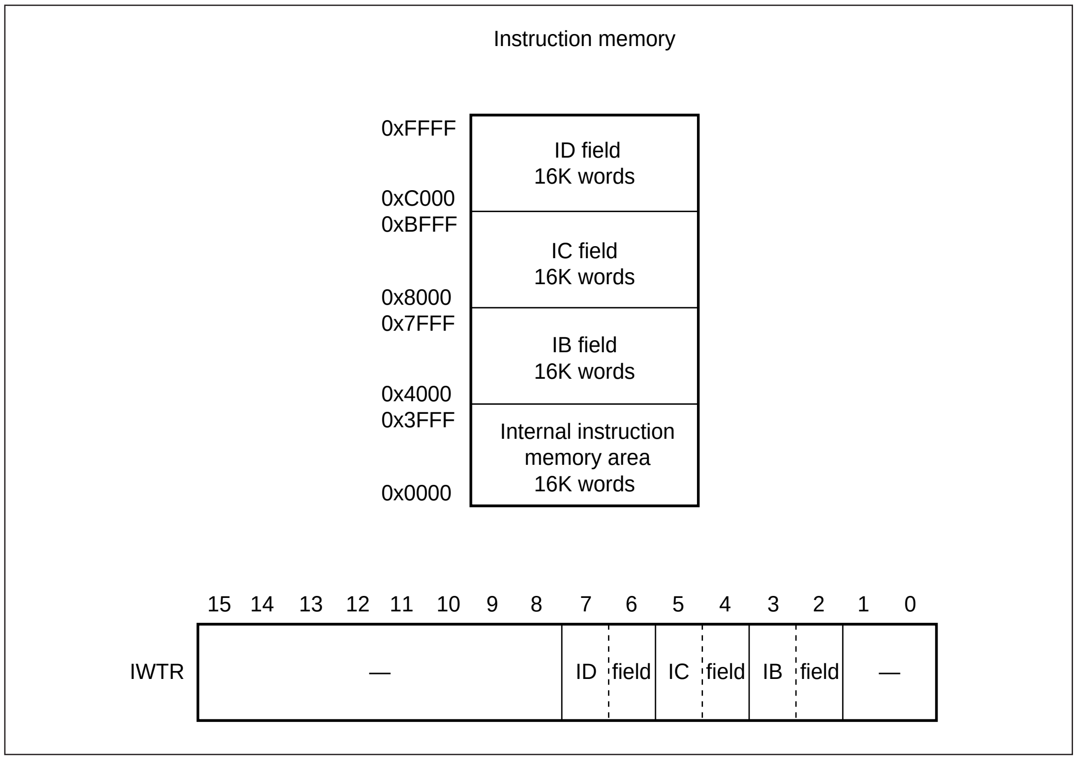


Table 3-14. Set Values of IWTR Fields and Number of Wait Cycles

Bits		No. of wait cycles	Remarks
0	0	0	1 cycle access: Connects SRAM with access time of 8 ns (at 33 MHz)
0	1	1	2 cycle access: Connects SRAM with access time of 35 ns (at 33 MHz)
1	0	3	4 cycle access: Connects SRAM with access time of 85 ns (at 33 MHz)
1	1	7	8 cycle access: Connects mask ROM with access time of 150 ns (at 33 MHz)

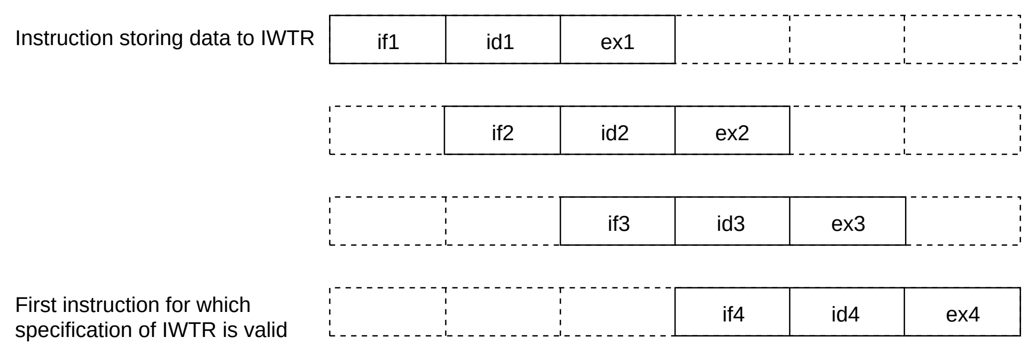
Cautions 1. Data written to bits 15 through 8, 1, and 0 are ignored. The contents of bits 15 through 8, 1, and 0 are undefined when they are read.

2. With the μPD77015/77017/77018/77018A/77019, data written to IWTR is ignored, and undefined data is read from IWTR.

- Timing of instruction memory wait controller

Wait control to IWTR by using a store instruction becomes valid starting from instruction fetch immediately after the execution cycle of the store instruction. Figure 3-13 shows the timing.

Figure 3-13. Valid Timing of Instruction Memory Wait Control

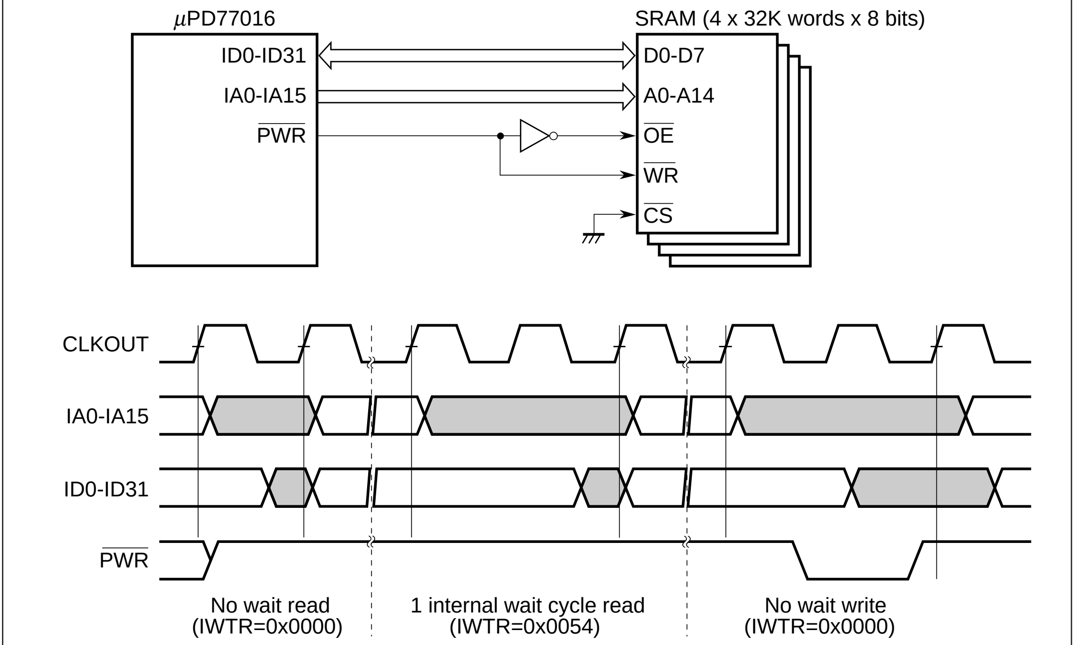

Instruction storing data to IWTR
if1
id1
ex1
if2
id2
ex2
if3
id3
ex3
First instruction for which specification of IWTR is valid
if4
id4
ex4

Remarks if : instruction fetch
id : instruction decode
ex: instruction execution

- External instruction memory interface example

Based on the above description, Figure 3-14 shows an example of a simple instruction memory interface. In this example, the external instruction memory has a capacity of 32K words, and the read/write timing without wait control and 1-wait read timing are shown.

Figure 3-14. Example of External Instruction Memory Interface



(2) Stack (STK) and stack pointer (SP)

Stack (STK) is a register file dedicated to saving/restoring PC and consists of 16 bits by 15 levels.

It is used to:

- Save return address when a subroutine is called
- Save the current address under execution when an interrupt occurs

For the details of the interrupt, refer to section 3.4.4 “Interrupt.”

A pointer register that points to the stack level (called stack top) that is currently to be accessed is called stack pointer (SP). SP consists of 16 bits, but setting a value other than 0 to 15 to this pointer is prohibited. The stack top and SP are connected to the main bus; therefore, data can be exchanged with a general-purpose register via the main bus.

When the stack overflows or underflows, stack error flag (ste) of ESR is set to 1.

Remark Do not write the RET or RETI instruction just after the inter-register transfer instruction to load from/ store to STK or SP.

(3) Related instructions

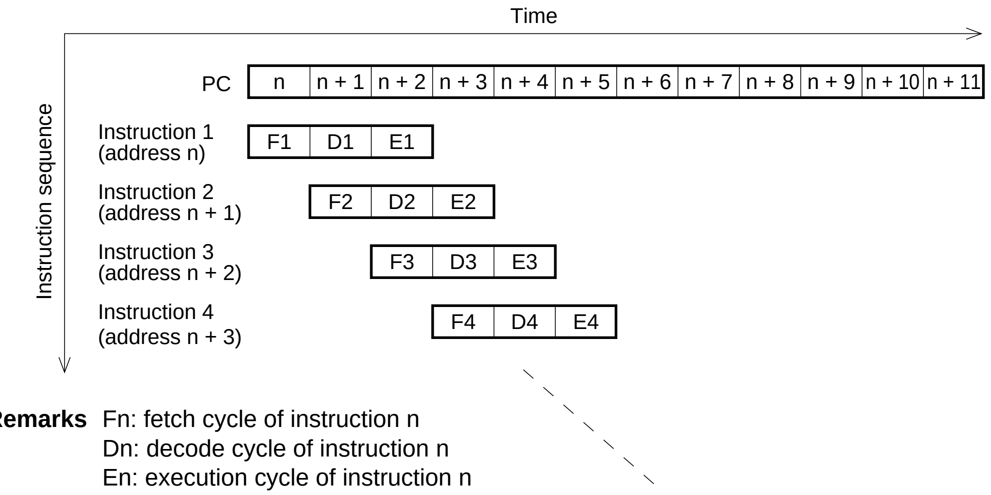
The operations of the program counter (PC), stack (STK), and stack pointer (SP) can be viewed from the following two points:

- Instruction execution and PC operation
- Branch instruction and operations of PC, SP, and STK

(a) Instruction execution and PC (normal operation)

The value of PC is incremented each time an instruction is fetched. Figure 3-15 shows the image when this PC operation is combined with pipeline execution.

Figure 3-15. Normal Operation of PC



Remarks Fn: fetch cycle of instruction n
Dn: decode cycle of instruction n
En: execution cycle of instruction n

(b) Branch instruction and operations of PC, SP, and STK

The branch instructions are classified into the following three types:

<1> Jump and subroutine call

The branch instructions are further subdivided into these two types of instructions, depending on whether the address of the instruction under opcode fetch (value of the PC) is saved to the stack or not.

• JMP instruction

Does not save the address of the instruction under opcode fetch to the stack. Therefore, program flow cannot automatically return to the branch source address from the branch destination address.

• Subroutine call instruction :

Saves the address of the instruction under opcode fetch (address of the instruction next to the subroutine call instruction) to the stack. To return program flow from the branch destination address to the branch source address, the return instruction is used.

<2> Branch viewed from PC setting format

The branch instructions can be classified into the following two types when viewed from the format in which the branch destination address is set to the PC:

• Immediate jump/call

This format is called immediate jump or immediate call. The JMP/CALL instructions for which a numeric value is coded as an operand execute branch in this format. At this time, the numeric value is added to or subtracted from the current PC value as 16-bit 2's complement. Therefore this is in fact a relative branch, relative to the current PC value. Program flow can be branched in the range of ±32K words, i.e., in the entire 64K-word space.

Caution When this instruction is written in assembler, write a direct branch destination address or label as the operand. The assembler calculates the correct relative branch distance to the current PC value automatically.

• Register indirect jump/call

This format is called register indirect jump or register indirect call. The JMP/CALL instructions for which DPn register is described as an operand execute branch in this format. At this time, the value of the DPn register is directly set to the PC.

<3> Conditional or unconditional branch

The μPD7701x family does not have dedicated conditional branch or conditional return instructions. Conditional branch is realized by combining conditional instructions and branch instructions, and conditional return is realized by combining conditional instructions and return instructions. These are classified into the following two types:

- Unconditional JMP/CALL/RET instructions

  These instructions always (unconditionally) branch (JMP/CALL/RETurn).

- Conditional JMP/CALL/RET instructions

  These instructions branch (JMP/CALL/RETurn) only when the condition of the combined conditional instructions is true.

Table 3-15 summarizes the above discussion. Note that, although the processing execution sequence when branch takes place does not differ depending on whether the instruction is conditional or unconditional, the actual execution time is 1 instruction cycle longer when a conditional instruction is used in combination. Although this is not indicated in the table, if the condition of a conditional branch instruction is not satisfied, delay due to pipeline hazard does not occur (refer to Figures 3-16 to 3-19).

Table 3-15. Classification of Branch Instructions

Instruction name	Condition judgment	Address specification	Word length	Instruction cycles
Jump instruction	Unconditional	PC relative	1	2
	Conditional			3
	Unconditional	Register indirect absolute	1	3
	Conditional			
Subroutine call instruction	Unconditional	PC relative	1	2
	Conditional			3
Indirect subroutine call instruction	Unconditional	Register indirect absolute	1	3
	Conditional			
Return instruction	Unconditional	—	1	2
	Conditional			3
Interrupt return instruction	Unconditional	—	1	2
	Conditional			3

Caution **Above number of instruction cycles are valid if the condition is satisfied and program flow branches. If the condition is not satisfied, even the conditional branch occupies one instruction cycle because the branch is not performed and no pipeline hazard occurs.**

Figures 3-16 to 3-19 show the timing of the following instructions:

- Unconditional immediate jump
- Unconditional indirect jump
- Conditional immediate jump (condition satisfied: jump)
- Conditional immediate jump (condition not satisfied: pass)

The meanings of the symbols in each figure are as follows (n=0,1,2,..):

ifn	: instruction fetch	jifn	: jump destination instruction fetch
idn	: instruction decode	exn	: instruction execution
ia	: instruction address operation	addr	: address output
p	: purge	push	: stack push
pop	: stack pop	jdec	: jump destination decode
popi	: interrupt pop		

Figure 3-16. Timing of Unconditional Immediate Jump

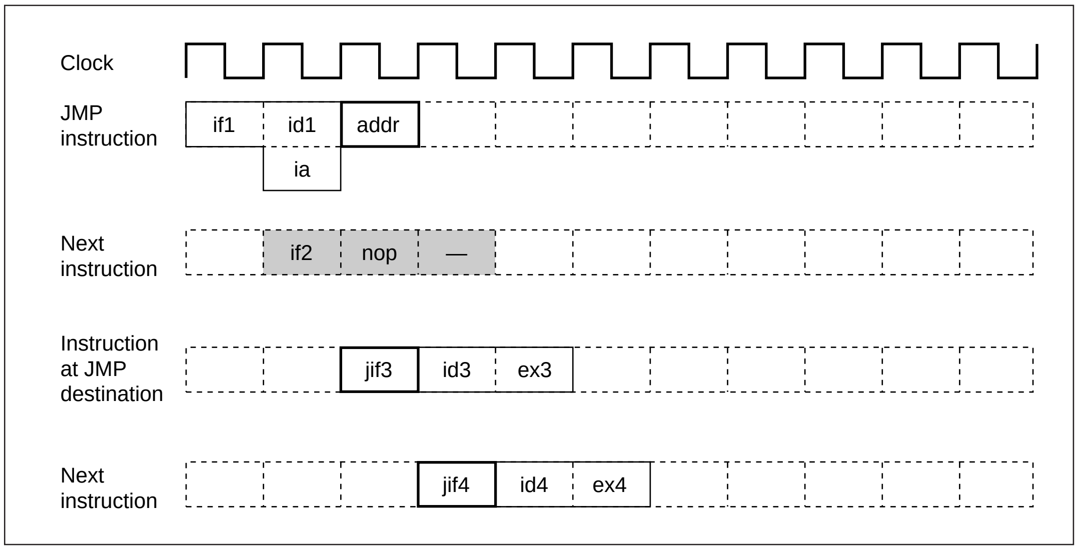


Figure 3-17. Timing of Unconditional Indirect Jump

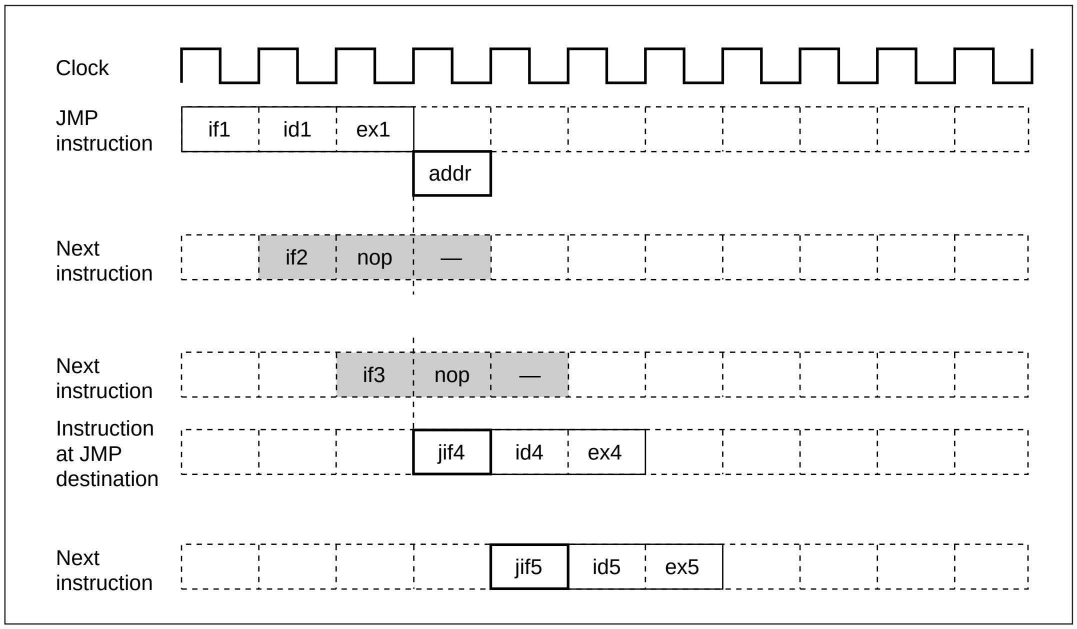


Figure 3-18.   Timing of Conditional Immediate Jump (condition satisfied: branch)

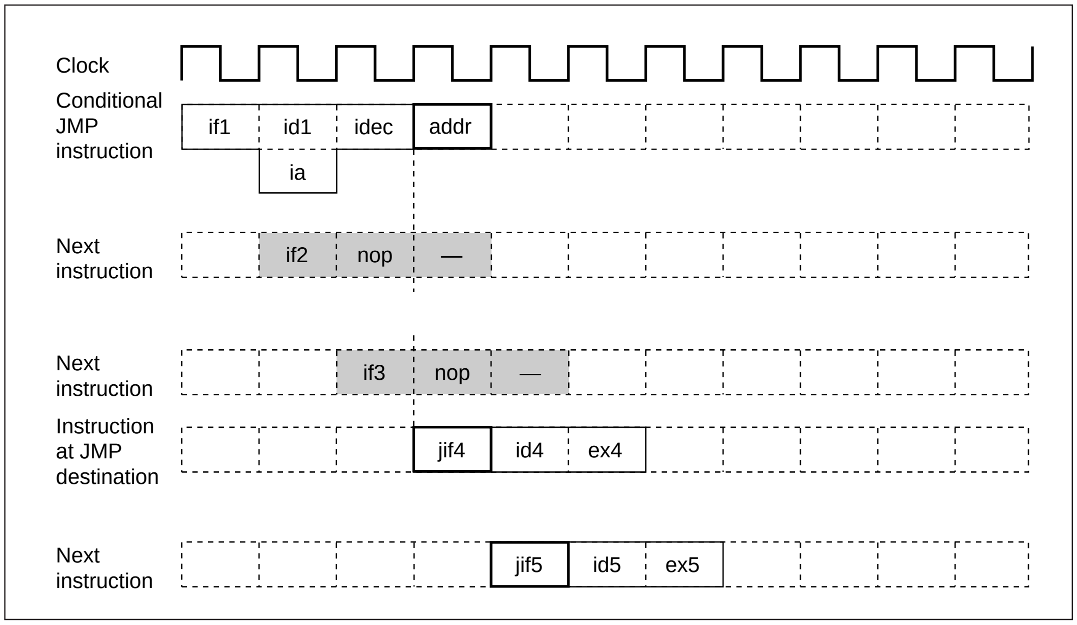
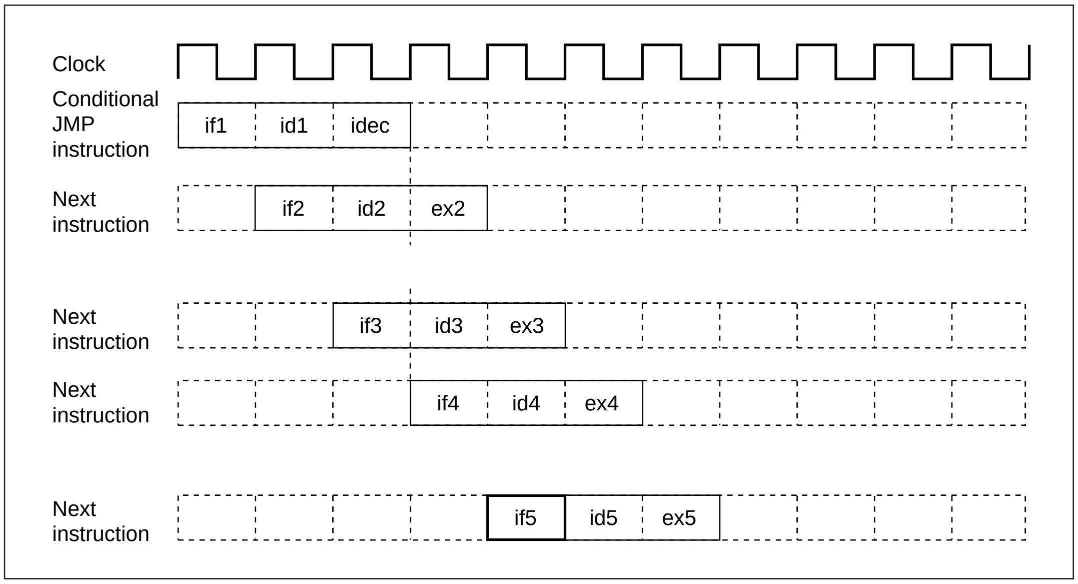


Figure 3-19.   Timing of Conditional Immediate Jump (condition not satisfied: pass)



(c) Operation of subroutine call/return

Subroutine call is executed by the CALL instruction. When the CALL instruction is executed, execution branches in the following procedure:

1. The value of SP is incremented (pre-increment).
2. The value of PC (address next to the CALL instruction) is saved to the STK indicated by SP.
3. The branch destination address is set to the PC. At this time, if the branch destination is given as a numeric value, the numeric value is added to or subtracted from the current PC value as 2's complement. If the branch destination is given by the DPn register, the value of the DPn register is directly set to the PC.

To return execution from a subroutine, the RET instruction is used. This instruction is executed in the following procedure:

1. The value in the STK indicated by the SP is restored to the PC.
2. The value of SP is decremented (post-decrement).

Remark For the timing of the CALL instruction, refer to the timing of the JMP instruction.
The timing of the CALL instruction is the same as that of the JMP instruction, except that the return address is saved to the stack. The timing of the return instruction is the same as that of the immediate jump instruction, i.e. it takes two instruction cycles.

(d) Operation when interrupt occurs

When an interrupt occurs, the address of the instruction under opcode fetch (address of the instruction when the interrupt is acknowledged) is saved to the stack, and the branch destination address is set to the PC. To return from the interrupt, the RETI (return from Interrupt) instruction is used.

For the operation of the interrupt, refer to section 3.4.4 "Interrupt."

3.4.3 Flow control block

In general, a high-level language provides sophisticated flow control syntax (e.g., for loop and while loop of the C language). The μPD7701x family is provided with hardware that allows this flow control to be directly described as assembly instructions, and performs loop/repeat operation without any timing overhead. The loop/repeat control circuit controls the loop/repeat operations.

Flow control is managed by the following registers and functional blocks:

- Repeat counter (RC)

  This 16-bit counter register holds the number of repetitions of a repeat instruction.

- Loop start address register (LSA)

  This 16-bit register holds the loop start address during loop execution.

- Loop end address register (LEA)

  This 16-bit register holds the loop end address during loop execution.

- Loop counter (LC)

  An initial value is set to this 16-bit register when execution of the LOOP instruction is started. Each time loop is executed once, the value of this register is decremented. When the current value of the register reaches 0, it indicates the end of the loop.

- Loop stack (LSTK)

  The LSTK is a register file with 3 $\times$ 16 bits $\times$ 4 levels to save and store the LSA, LEA and LC values. It saves the LSA, LEA and LC values by the loop instruction. The values are restored to the LSA, LEA and LC upon loop termination or by the loop pop instruction.

  This file serves as one of the following three 16-bit registers for input/output to/from the main bus with inter-register transfer instruction.

  - LSR1: Saves/restores loop start address (stack for LSA)
  - LSR2: Saves/restores loop end address (stack for LEA)
  - LSR3: Saves/restores loop counter (stack for LC)

  If LSR1 is specified for the inter-register transfer instruction source, the LSP is decremented after transfer. If LSR1 is specified for the inter-register transfer instruction destination, data is transferred after the LSP is incremented.

  [Software loop stack]

  If a loop over 4 levels causes a loop stack overflow, the return address is lost and processing cannot return normally.

  When you know in advance that there is a loop over 4 levels, the contents of the stack should be saved to memory prior to stack overflow, so that a normal return can be executed even though another loop operation is performed. This method is called software loop stack.

  In this case, note that the contents stored should be rewritten to the loop stack in correspondence with stack level, when stack contents are saved to memory. A software loop stack programming example is shown below.

Software loop stack example:

- Push (DP0: Save address)
  ```
  R0L = *DP0– –
  R0L = LSR3;
  *DP0– – = R0L;
  R0L = LSR2;
  *DP0– – = R0L;
  R0L = LSR1;
  *DP0– – = R0L;
  ```
- Pop (DP0: Restore address)
  ```
  R0L = *DP0++;
  LSR1 = R0L;
  R0L = *DP0++;
  LSR2 = R0L;
  R0L = *DP0++;
  LSR3 = R0L;
  ```

- Loop stack pointer (LSP)

  This pointer indicates the current position of LSTK. Although this is a 16-bit register, the value that can be set to it is 0 to 4.

  The LSP value can be input/output to/from the main bus with inter-register transfer instruction. The LSP value becomes 0 by reset.

  The LSP is incremented/decremented by 3 bits (bits 2 to 0). Bits 15 to 3 are fixed to 0.

  The LSP is incremented in the following cases:

  - When the LSA, LEA, and LC values are saved to the LSTK by the loop instruction
  - When the LSR1 is specified for the inter-register transfer instruction destination

  The LSP is decremented in the following cases:

  - When the LSTK value is returned to the LSA, LEA, and LC upon loop termination or by the loop pop instruction
  - When LSR1 is specified for the inter-register transfer instruction source

Cautions 1. When the value of LSP is not between 0 to 4, a stack overflow or underflow occured indicating an error.

2. Do not set the LSP value between 5 and 0xFFFF.

- Loop/repeat controller (LRC)

  This circuit controls the loop and repeat instructions.

Caution **All the above registers, except RC, are connected to the main bus, so that data can be transferred between them and general-purpose registers.**

Flow control has the following two functions:

- Repeat function (REP instruction)
- Loop function (LOOP instruction, LPOP instruction)

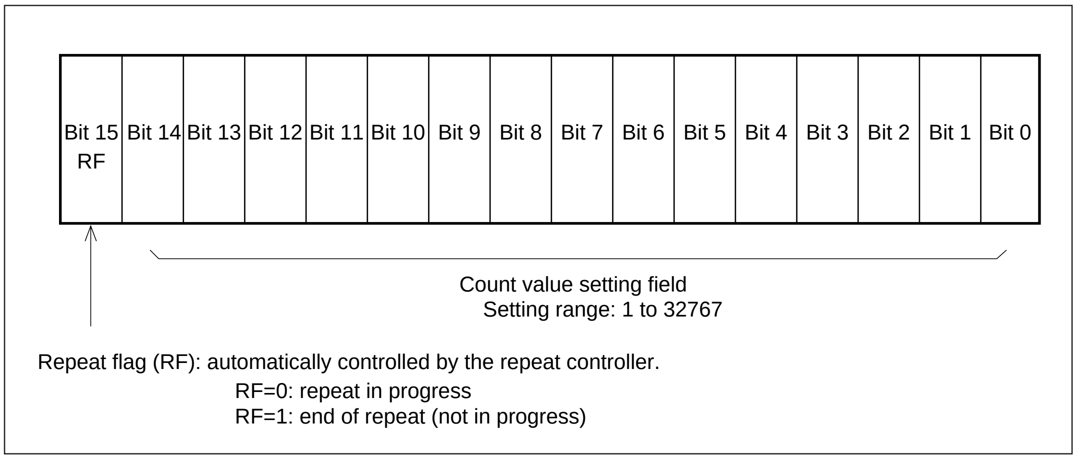
(1) Repeat function

The repeat function that is written by the REP instruction realizes repetition of one instruction on a count basis. The instruction to be repeated the repeat target instruction, follows immediately the REP instruction itself.

(a) Format of repeat counter (RC)

Figure 3-20 shows the format of the repeat counter (RC).

Figure 3-20. Format of RC


Bit 15 RF
Bit 14
Bit 13
Bit 12
Bit 11
Bit 10
Bit 9
Bit 8
Bit 7
Bit 6
Bit 5
Bit 4
Bit 3
Bit 2
Bit 1
Bit 0
Count value setting field
Setting range: 1 to 32767
Repeat flag (RF): automatically controlled by the repeat controller.
RF=0: repeat in progress
RF=1: end of repeat (not in progress)

Caution **During the entire repeat operation no interrupt will be acknowledged. For further details refer to section 3.4.4 “Interrupts”.**

(b) Summary of repeat function

The repeat function can be summarized as follows:

- A single instruction is repeated.
- The number of repetitions can be directly given as a numeric value or by using a general-purpose register (R0L-R7L).
- The number of repetitions ranges from 1 to 32767.
- The value of PC is not incremented during repeat operation.
- RC is decremented each time the instruction is repeated, and repeat ends when the instruction has been repeated the specified number of times.
- The repeat function depends on RC only, and is not counted as nesting of loop instructions.

(c) Procedure of repeat function execution

When the REP instruction is executed, the repeat function is implemented in the following procedure.

1. The number of repetitions given as the parameter of the REP instruction is set to RC.
2. The value of PC is incremented, and the instruction immediately after the REP instruction is repeated. At this time, an invalid cycle of one instruction cycle is generated.
3. During repeat operation, PC holds a next address of this instruction that has been repeated.
4. The value of RC is decremented each time the instruction has been repeated once. After the instruction has been repeated the specified number of times, repeat ends.
5. When repeat ends, the value of PC is incremented. When execution shifts from the instruction that has been repeated to the next instruction, the pipeline stages are successive. Therefore, no overhead occurs when repeat ends.

For the repeat instruction, refer to “μPD7701x Family User’s Manual Instructions”.

(d) Repeat execution timing

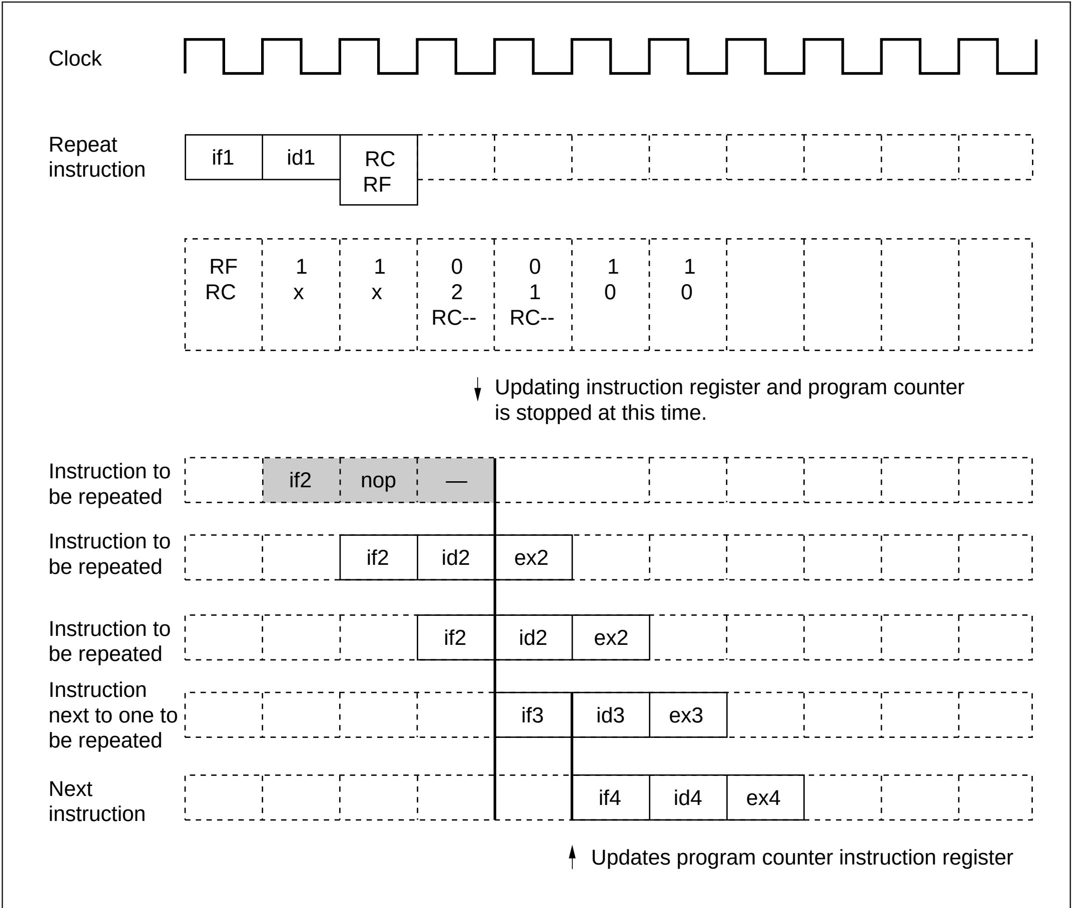
The following figures show an example in which the REP instruction is repeated two times.

Figure 3-21 shows the assembly program, and Figure 3-22 shows the execution timing.

Figure 3-21. Example of Repeat Instruction (repetition of 2 times)

```
REP              2;
R0       /=      R1;
```

Figure 3-22. Repeat Execution Timing (repetition of 2 times)



Clock
Repeat instruction
if1
id1
RC RF
RF RC
1 x
1 x
0 2 RC--
0 1 RC--
1 0
1 0
Updating instruction register and program counter is stopped at this time.
Instruction to be repeated
if2
nop
—
Instruction to be repeated
if2
id2
ex2
Instruction to be repeated
if2
id2
ex2
Instruction next to one to be repeated
if3
id3
ex3
Next instruction
if4
id4
ex4
Updates program counter instruction register

Remarks RF: repeat flag RC: repeat counter

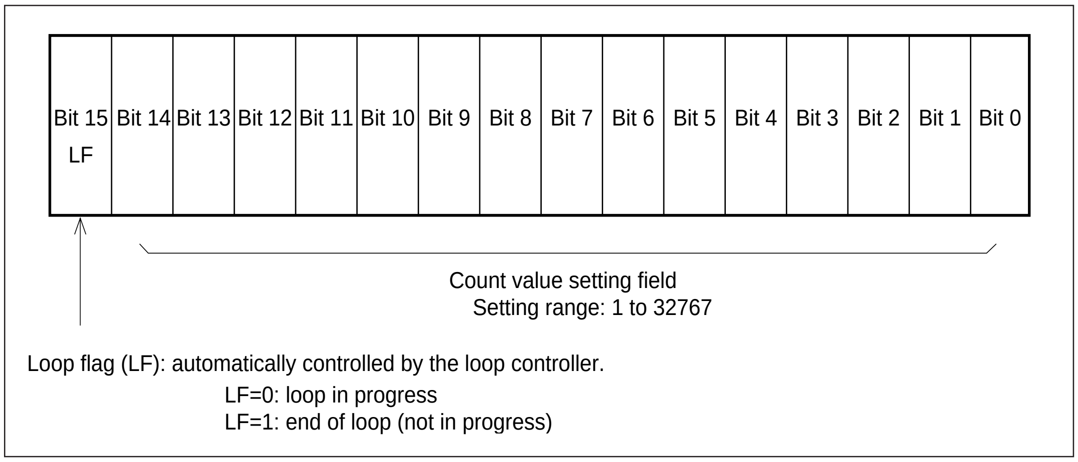
(2) Loop function

The loop function that is described by using the LOOP instruction realizes loop flow of an instruction group consisting of 2 to 255 instructions on a count basis. Nesting of loop is supported by a four level hardware loop stack. To escape from the loop at any point, the LPOP instruction is provided, so that flexible loop control is performed.

(a) Format of loop counter (LC)

Figure 3-23 shows the format of the loop counter (LC).

Figure 3-23. Format of LC


Bit 15
LF
Bit 14
Bit 13
Bit 12
Bit 11
Bit 10
Bit 9
Bit 8
Bit 7
Bit 6
Bit 5
Bit 4
Bit 3
Bit 2
Bit 1
Bit 0
Count value setting field
Setting range: 1 to 32767
Loop flag (LF): automatically controlled by the loop controller.
LF=0: loop in progress
LF=1: end of loop (not in progress)

Remark The loop flag LC is also contained in the status register (SR) (refer to section 3.4.4 "Interrupt").

(b) Summary of loop function

The loop function can be summarized as follows:

- Groups 2 to 255 instructions as a loop element.
- The number of loops can be given directly by a numeric value or by using a general-purpose register (R0L-R7L).
- The number of loops ranges from 1 to 32767.
- Nesting of up to 4 levels can be realized by the loop stack.
- Execution can be escaped from the loop when:

  (1) The count value reaches 1

  (2) The LPOP instruction and then JMP instruction are executed

Remark For interrupt processing in conjunction with loop operations (refer to section 3.4.4 "Interrupt").

(c) Loop function execution procedure

When the LOOP instruction is executed, the loop function is implemented in the following procedure:

<1> When loop is started

1. The value of LSP is incremented (pre-increment).
2. The current LSA, LEA, and LC are saved to LSTK indicated by LSP.
3. The loop start address is set to LSA.
4. The loop end address is calculated and set to LEA.
5. The number of loops is set to LC.

<2> During loop operation

1. The value of LC is decremented if the values of PC and LEA are equal.
2. The value of LSA is set to PC if LC is not 1. If LC is 1, the loop end processing is executed.

<3> Loop end processing

1. The value of PC is incremented.
2. The value of LSTK indicated by LSP is restored to LSA, LEA, and LC.
3. The value of LSP is decremented (post-decrement).

<4> Loop end processing by LPOP instruction

The LPOP instruction discards one level of loop by performing the following processing.

1. Restores the value of LSTK indicated by LSP to LSA, LEA, and LC.
2. Decrements the value of LSP (post-decrement).

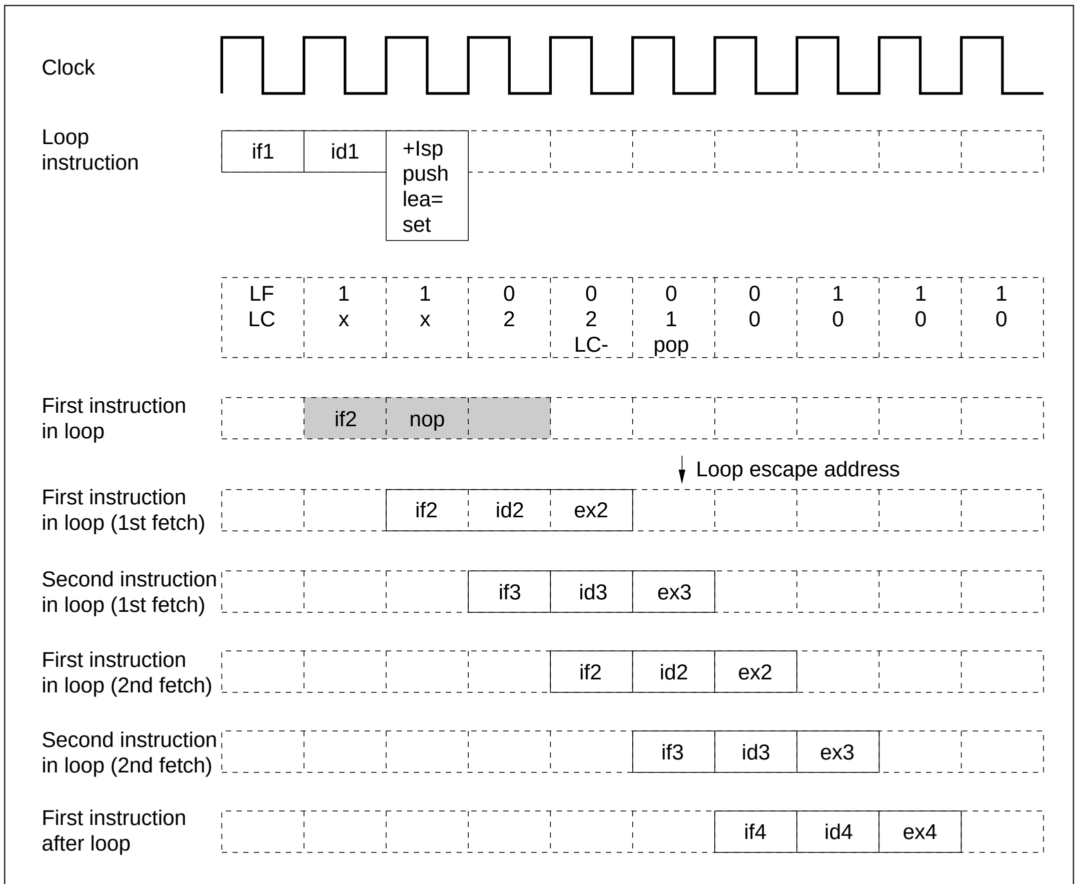
For the LOOP and LPOP instructions, refer to “μPD7701x Family User’s Manual Instructions”.

Caution The LPOP instruction does not automatically control PC for escaping from the loop. Therefore, execute the LPOP instruction after escaping from the loop by using the JMP instruction, or execute the LPOP instruction and then escape from the loop by using the JMP instruction (Refer to “μPD7701x Family User’s Manual Instructions”).

(d) Timing of loop execution (example of two loops operation)

Figure 3-24 shows an example of the LOOP instruction execution timing. In this example, two loops operation in which a group of two instructions is executed only once is performed.

Figure 3-24. Loop Execution Timing (example of 2 loops operation)


Clock
Loop instruction
if1
id1
+lsp push lea= set
LF LC
1 x
1 x
0 2
0 2 LC-
0 1 pop
0 0
1 0
1 0
1 0
First instruction in loop
if2
nop
Loop escape address
First instruction in loop (1st fetch)
if2
id2
ex2
Second instruction in loop (1st fetch)
if3
id3
ex3
First instruction in loop (2nd fetch)
if2
id2
ex2
Second instruction in loop (2nd fetch)
if3
id3
ex3
First instruction after loop
if4
id4
ex4

Remarks LF: loop flag LC: loop counter

3.4.4 Interrupt

The μPD7701x family has powerful interrupt functions. This section describes the following functions:

- Interrupt cause
- Interrupt control function
- Interrupt acknowledgment condition
- Hardware condition of external interrupt
- Interrupt vector

(1) Interrupt cause

There is a total of 10 interrupt causes available including internal and external interrupts.

- Internal interrupt : Caused by events specified by internal peripherals.
  Six internal causes are available.
- External interrupt: Triggered by external causes via hardware signal pins.
  Four external causes are available.

Table 3-16 lists all the interrupt causes.

Table 3-16. Interrupt Causes

Internal/external	Interrupt cause
Internal	SI1 input
	Completion of serial interface #1 input
	SO1 output
	Enabling output of serial interface #1
	SI2 input
	Completion of serial interface #2 input
	SO2 output
	Enabling output of serial interface #2
	HI input
	Completion of host interface input
	HO output
	Enabling output of host interface
External	INT1
	Falling edge of external signal pin $\overline{\text{INT1}}$
	INT2
	Falling edge of external signal pin $\overline{\text{INT2}}$
	INT3
	Falling edge of external signal pin $\overline{\text{INT3}}$
	INT4
	Falling edge of external signal pin $\overline{\text{INT4}}$

(2) Interrupt control function

All interrupt causes, regardless of whether they are internal or external, are handled as independent events and at independent levels. Here is the summary of the functions to control the interrupts:

- Each interrupt cause can be enabled or disabled independently.
- All interrupts can be enabled or disabled as one group (global enable).
- A stack for global interrupt enable function is provided, so that multiple (nesting of) interrupts can be handled.
- The interrupt vectors (entry points of interrupt servicing routine at interrupt acknowledgement) for all interrupt causes are fixed.
- When an interrupt has been acknowledged, the current instruction is aborted, and program execution control is transferred to the specified entry point.
- After the interrupt servicing routine is executed completely, control is returned to the instruction that was suspended by the interrupt.
- When an interrupt request is issued during the execution of Jump or some other instructions, a delay cycle is inserted before the interrupt is acknowledged.

(3) Interrupt acknowledgment condition

When an interrupt request is generated by an interrupt cause, the interrupt will be acknowledged if both following conditions are satisfied:

- Global interrupt enable (EI) flag value is 0 (enable).
- Interrupt cause enable flag value corresponding to the requested interrupt is 0 (enable).

Note, however, that acknowledging the interrupt is delayed in any of the following cases:

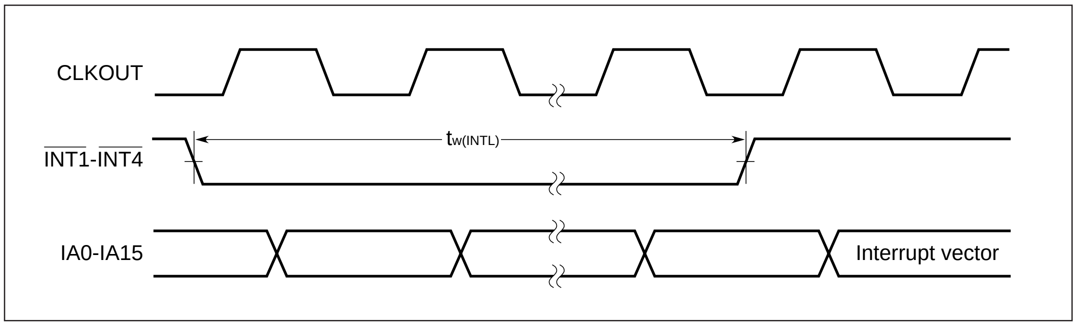
- While a jump instruction is fetched, decoded, or executed
- While a repeat instruction or a repeat target instruction is fetched, decoded, or executed
- While a loop instruction is fetched, decoded, or executed
- While a loop termination instruction (instruction at loop end address) is fetched

(4) Hardware conditions of external interrupt

External interrupts ($\overline{\text{INT1}}$-$\overline{\text{INT4}}$) are acknowledged when the falling edges of the corresponding pins have been detected. To issue several interrupt requests successively, make the corresponding pin to high and then to low, for each interrupt request, to create a falling edge. Note that each of the high and low levels must have enough duration for the system to recognize level changes.

Figure 3-25 shows external interrupt timing.

Figure 3-25. External Interrupt Timing



★

Caution In case of the μPD77015, 77017, 77018, 77018A, and 77019, if the processor is in HALT mode, the active (low) time of an external interrupt $\overline{\text{INT1}}$-$\overline{\text{INT4}}$ has to be extended to minimum 8×$t_{W(INTL)}$, since the CLKOUT period during HALT mode is extended to 8 times longer.

(5) Interrupt vector

Every interrupt cause has a dedicated entry point (also called vector). These vectors for interrupt causes are sequentially set from the start position (address 0x200) of the internal instruction area, creating a 64-word table. Each cause is assigned four instruction addresses. If interrupt servicing is not completed within four instructions including the interrupt return instruction (RETI), execution must branch beyond address 0x240 for service completion.

(a) Interrupt vector table

Table 3-17 shows the interrupt vector table.

Table 3-17. Interrupt Vector Table

Vector	Internal/external	Interrupt cause
0x200	Internal	Reset
0x204	—	Reserved
0x208	—	Reserved
0x20C	—	Reserved
0x210	External	INT1
0x214	External	INT2
0x218	External	INT3
0x21C	External	INT4
0x220	Internal	SI1 input
0x224	Internal	SO1 output
0x228	Internal	SI2 input
0x22C	Internal	SO2 output
0x230	Internal	HI input
0x234	Internal	HO output
0x238	—	Reserved
0x23C	—	Reserved

Cautions 1. Although the reset signal is not an interrupt, it is treated as a vector entry as if it is an interrupt.

2. It is recommended that the vector of unused interrupt causes be branched to an abnormality processing routine.

★ **3. Because the vector area of the mask ROM model also exists in the internal RAM area, this area must be booted up. Also because the entry after reset is address 0x200, booting up address 0x200 is necessary even when the internal instruction RAM and interrupts are not used.**

(b) Example of processing of interrupt vector

See the following example.

```
; Definitions
#define SI1 0x3800     ; address of serial input register
#define SO1 0x3800     ; address of serial output register

; Interrupt vector table
int_vec imseg at 0x200 ; start of vector table
  :
  :
        org 0x220
                       ; serial input #1 interrupt vector
(0x220) JMP INPUT      ; branches to application area for
(0x221) NOP            ; more than 4 instructions
(0x222) NOP            ;
(0x223) NOP            ;
                       ; serial output #1 interrupt vector
                       ; interrupt service less than 4 instructions
(0x224) R0H=*DP4++     ; fetch data from y-memory
(0x225) *SO1:y=R0H     ; transter it to serial output #1
(0x226) RETI           ; return from interrupt
(0x227) NOP            ;
                       ; serial input #2 interrupt
(0x228) NOP            ; serial input #2 is not used (RETI
(0x229) RETI           ; instruction cannot be written at
(0x22A) NOP            ; brginning of vector)
(0x22B) NOP            ;
  :
  :
; Main program segment
main    imseg          ; start at 0x240 (automatically located
                       ; to 0x240 by the Linker)
  :
  :
                       ; serial input #1 interrupt servicing routine
INPUT: R0H=*SI1:y      ; fetch data from serial input #1
       R1=*DP0         ;
       R1=R1+R0H*R2H   ;
       *DP0=R1H        ;
       RETI            ; return from interrupt
```

(6) Interrupt control software

Interrupts are controlled by the following registers (refer to Figure 3-9 “Program Control Unit”):

- Status register (SR)
- Interrupt enable flag stack register (EIR)

(a) Status register (SR)

This is a 16-bit register that enables or disables all the interrupts (general interrupt enable/ disable), and enables or disables each interrupt cause separately. When the value of a bit of this register is 0, the corresponding interrupt is enabled; when the bit is 1, the interrupt is disabled.

The values of SR can be read and written by executing the register-to-register transfer instruction. This register is set to 0xF000 at reset.

Interrupt enable flag				Reserved		Interrupt enable flag for each cause									
						On-chip I/O device						External interrupt master			
EI	EP	EB	LF			ho	hi	so2	si2	so1	si1	int4	int3	int2	int1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

<1> Interrupt enable flags (EI: enable interrupt, EP: enable interrupt previous, EB: enable interrupt before)

The EI, EP, and EB flags enable or disable all the interrupts. When the value of these flags is 0, the interrupts are enabled; when it is 1, the interrupts are disabled. These three flags, EI (enable interrupt), EP (enable interrupt previous), and EB (enable interrupt before), enable or disable the current interrupts, and interrupts one levels before and two levels before.

These flags are the same as the EI, EP, EB flags at bits 15-13 of the EIR register (refer to section (b) “Interrupt enable flag stack register”). Therefore, the values of bits 15-13 of the SR are always the same as those of the EIR register.

The following nesting of interrupt and stack manipulation are handled by the EI, EP, and EB flags and E3 through E15 flags of the EIR register (refer to section (b) “Interrupt enable flag stack register”).

When interrupt has been acknowledged;

value of EB —> E3 of EIR register

value of EP —> shifted to EB

value of EI —> shifted to EP

EI —> set to 1 (all interrupt disable)

Vice versa at RETI instruction;

value of EI —> wasted

value of EP —> shifted to EI

value of EB —> shifted to EP

value of E3 of EIR register —> EB

For multiple interrupts, refer to section (b) “Interrupt enable flag stack register”.
The interrupt flag before updating is valid while a transfer instruction which specifies SR as the destination is fetched and executed, that is, between the transfer instruction and the next instruction, and between the next instruction and the instruction that follows.

Example of changing interrupt enable flag (enabled —> disabled)

```
Initial status:EI=0; (interrupt enabled)
  R0L=EIR;
  R0=R0|0x8000;
  EIR=R0L;                  }
  Next instruction;         } May branch to interrupt servicing
  Instruction that follows; }
```

Caution   To rewrite the EP and EB flag, be sure to disable all the interrupts (EI = 1).

<2> Loop flag (LF)

This flag indicates whether execution is in a loop or not. The value “0” shows that the execution is in a loop, and “1” for not in a loop.

Caution   Do not change this flag when modifying any interrupt mask flags. Modify interrupt mask flags always by reading the current SR contents and mask only the dedicted flags (refer to following examples).

<3> Reserved flags

A write to these flags is ignored. Undefined values are returned when these flags are read.

<4> Interrupt enable flags for each cause

These flags enable or disable the corresponding interrupt causes. When the value of any of these flags is 0, the corresponding interrupt is enabled; when it is 1, the interrupt is disabled. The values of these flags are not affected even when the respective interrupts have been acknowledged. There are the following five types of these flags, totaling 10.

- External interrupts 1-4 : Interrupts from external interrupt pins ($\overline{\text{INT1}}$-$\overline{\text{INT4}}$).
- SI1, SI2 : Interrupts that occur when serial input has been completed and data has been received by the serial data register (SDT: for input).

• SO1, SO2	:	Interrupts that occur when serial output has been completed and transmit data can be written to the serial data register (SDT: for output).
• HI	:	Interrupt that occurs when host interface input has been completed and data has been received by the host data register (HDT: for reception).
• HO	:	Interrupt that occurs when host interface output has been completed and transmit data can be written to the host data register (HDT: for transmission).

Caution To rewrite any of the above flags, be sure to disable all interrupts (EI = 1).

Example of rewriting interrupt enable flag for each cause (enabled —> disabled)

```
R0L=EIR         ; disable interrupts generally here:  via EIR register
R0=R0|0x8000    ; set EI = 1 (general interrupt disabled)
EIR=R0L         ; write back to EIR
NOP             ; wait until interrupt disable becomes valid
R0L=SR          ; disable external INT1 via SR register
R0=R0|0x0001    ; set INT1 = 1 (disabled)
SR=R0L          ; write back to SR
```

(b) Interrupt enable flag stack register (EIR)

This 16-bit register stacks the general interrupt enable flags. When a bit of this register is 0, the corresponding interrupt is enabled; when the bit is 1, the interrupt is disabled.

The values of EIR can be read and written by executing the register-to-register transfer instruction.

The value of this register is set to 0xFFFF at reset.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EI	EP	EB	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14	E15

When an interrupt has been acknowledged, the contents of this register are shifted 1 bit to the right, and the bit EI is set to 1 to disable all interrupts generally. The register contents are shifted 1 bit to the left by execution of the interrupt return instruction RETI where E15 is set to 1 simultaneously. Cause of them, multiple interrupts of up to 16 levels are guaranteed.

Bits 15-13 (EI, EP, EB) are the same as the bits 15-13 of the SR register.

The interrupt enable/disable status can be changed by writing EIR with the register-to-register transfer instruction. However, note that this change will be valid three instructions after writing EIR.

Example of enabling interrupt (disabled status —> enabled)

```
Initial status:EI=1  (interrupt disabled)

  R0L=EIR;
  R0=R0&0x7FFF;
  EIR=R0L;        ⎫
  Instruction 1;  ⎬ Interrupt disabled during this period
  Instruction 2;  ⎭
  Instruction 3;  } Interrupt enabled
```

<1> EIR and multiple interrupts

As described earlier, a multiple interrupt system can be configured by using the EIR register. This paragraph describes the concept of multiple interrupts, taking an example shown in Figure 3-26 and focusing on EIR.

[Prerequisite]

All interrupts are enabled by the corresponding interrupt enable flags.

[Process]

(1) Clear the EI bit to 0 to enable all interrupts.

(2) INT1 is acknowledged, and control is transferred to the INT1 servicing routine. At this time, the contents of the EIR register are shifted 1 bit to the right, and one level of interrupt status is stacked. At the same time, bit 15 (EI) is set to 1, disabling the other interrupts.

(3) The interrupts are enabled (EI=0) in the INT1 servicing routine.

(4) INT2 is acknowledged, and control is transferred to the INT2 servicing routine. In the same manner as before, the contents of EIR are shifted 1 bit to the right, and EI is set to 1, disabling the interrupts.

(5) INT3 request is generated while the INT2 servicing routine is executed. However, this interrupt is not acknowledged because it is disabled, but recorded.

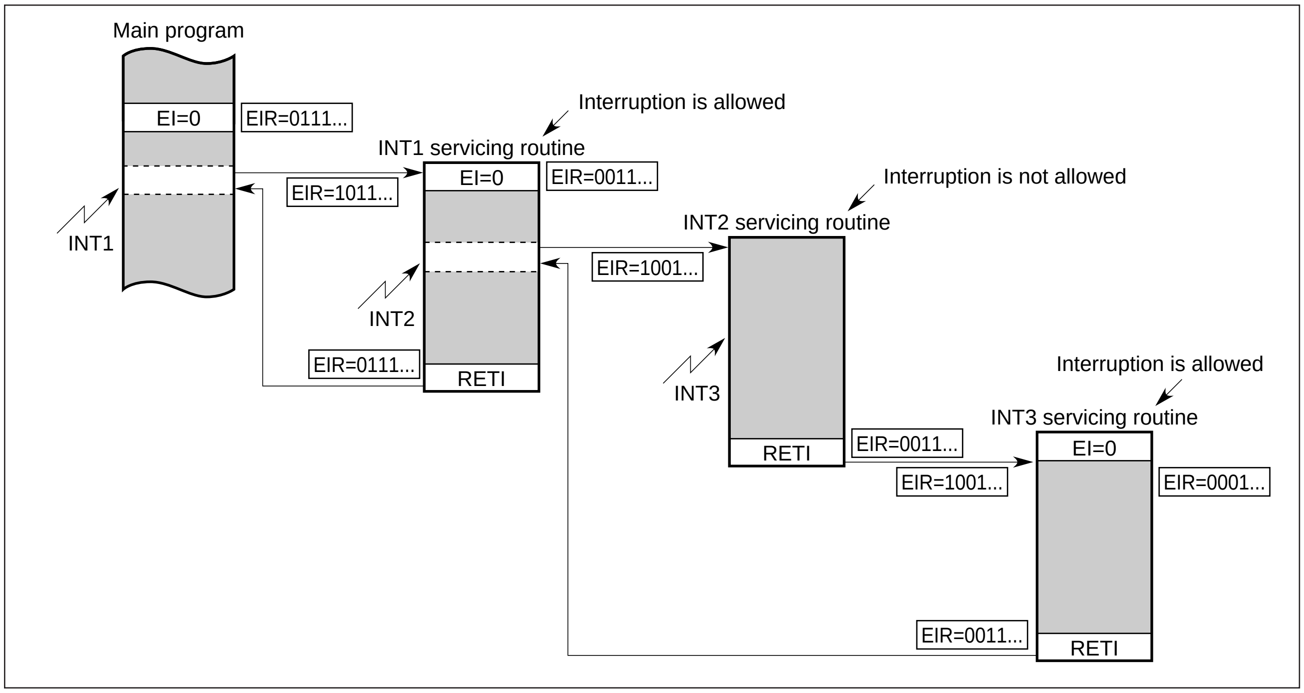
(6) When the INT2 servicing routine is ended in the RETI instruction, the contents of EIR are shifted 1 bit to the left. Consequently, the status before acknowledging the INT2 interrupt is restored. In this status, EI=0, enabling the interrupts.

(7) The recorded INT3 is now acknowledged, and control is transferred to the INT3 servicing routine. The contents of EIR are shifted 1 bit to the right again, and EI is set to 1. If necessary, clear EI to 0.

(8) When the INT3 servicing routine is ended in the RETI instruction, the contents of EIR are shifted 1 bit to the left, and the status before INT3 was acknowledged is restored (INT1 is being processed).

(9) Execution of the INT1 servicing routine continues. When the RETI instruction is executed at the end of this routine, the status before INT1 was acknowledged is restored.

Figure 3-26. Multiple Interrupt Processings



<2> Differences between SR and EIR

The most significant three bits of the SR and EIR registers (EI, EP, and EB) are accessed as common bits. The EI bit directly enables or disables the current interrupt, and therefore care must be exercised in manipulating this bit. The differences between SR and EIR are as follows, when the EI bit is manipulated:

- To enable the interrupts, the EI bit of either the SR or EIR register can be used.
- To disable the interrupts, use of the EI bit of the EIR register is recommended.

There is no problem when the interrupts are enabled by the EI bit because the interrupts have been disabled up to that point. When the interrupts are disabled, however, the following situation may arise:

```
R0L=SR   ; disable interrupt generally here:  via SR register

     <– Interrupt occurs and jump to interrupt servicing routine:

         ; Interrupt servicing routine
         ; This routine disables INT1 interrupt individually
         R1L=SR          ;
         R1=R1|0x0001    ; set INT1=1 (disabled)
         SR=R1L          ; write back to SR
         RETI            ; return from interrupt

     <– SR has changed meanwhile

R0=R0|0x8000             ; set EI=1
SR=R0L                   ; write back to SR
   :
   :
```

In this case, writing data to the SR register is ignored while the interrupt is serviced. To avoid this situation, it is recommended to use the EI bit of the EIR register, rather than that of the SR register, to disable the interrupts.

(7) Interrupt sequence

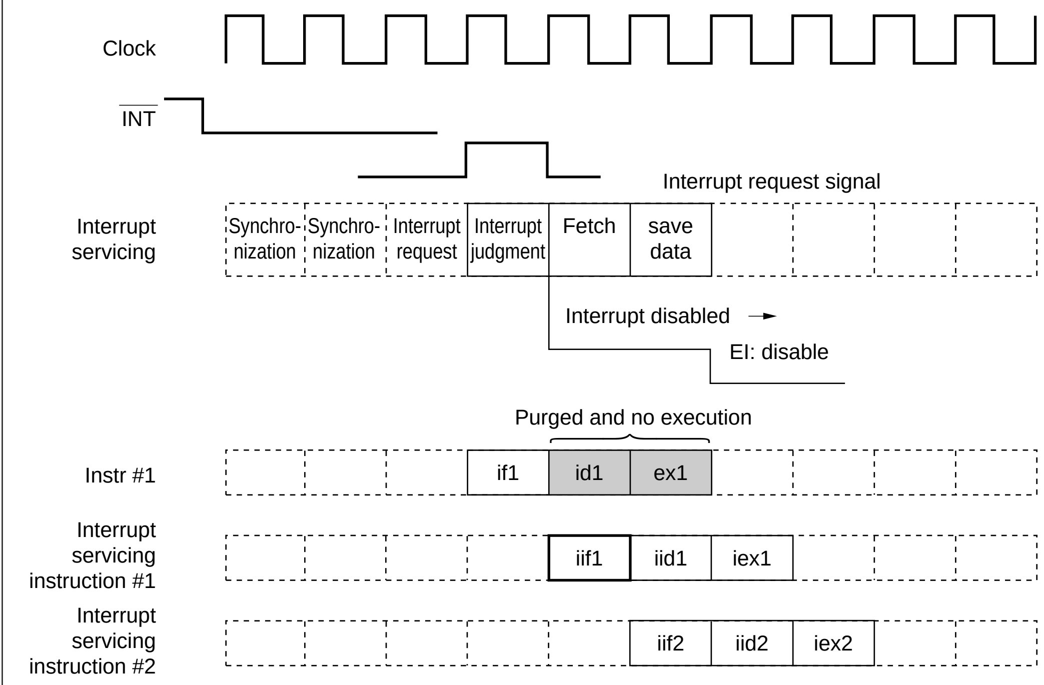
(a) Acknowledging an interrupt

When an interrupt has been acknowledged, the following operations are performed:

- An instruction that was fetched immediately before the interrupt has been acknowledged is kept pending.
- The EIR register is shifted 1 bit to the right to stack 1 level.
- The EI bit is set to 1 to disable the interrupts.
- SP is incremented.
- The address of the pending instruction is saved to STK specified by SP.
- A specified interrupt vector address is set to PC, and execution branches to interrupt servicing routine.

Figure 3-27 shows timing of acknowledging an interrupt.

Figure 3-27. Interrupt Acknowledging Timing



Remarks		
	ifn : instruction fetch	iifn : interrupt instruction fetch
	idn : instruction decode	iidn : interrupt instruction decode
	exn: instruction execution	iexn: interrupt instruction execution

(b) Returning from interrupt

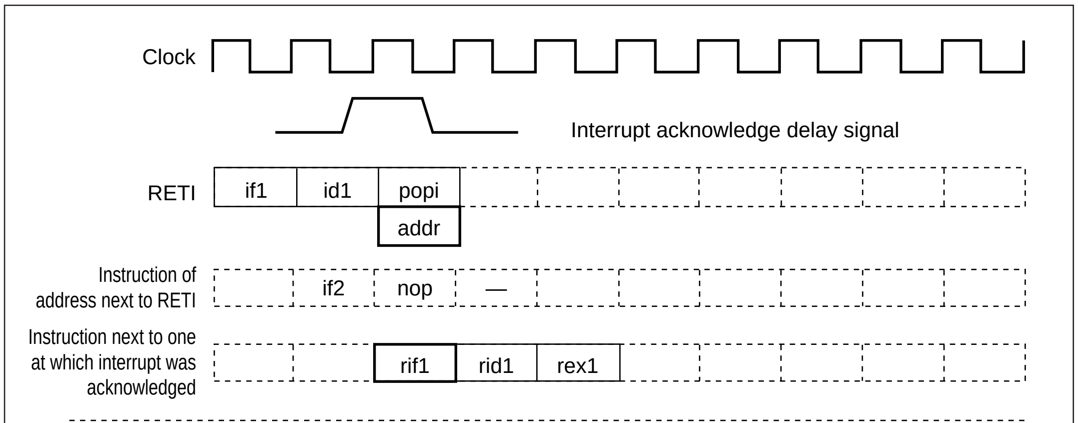
When the RETI instruction (interrupt return) is executed, the following are processed in two to three instruction cycles, and execution returns from the interrupt servicing routine.

- The value of STK indicated by SP is restored to PC.
- SP is decremented.
- The EIR register is shifted to the left, and interrupt enable flags are restored.
- Execution branches to the return address (the instruction that was kept pending when the interrupt was acknowledged).

Figure 3-28 (a) and (b) shows the return timings by using an unconditional RETI instruction and a conditional RETI instruction with the condition satisfied, respectively.

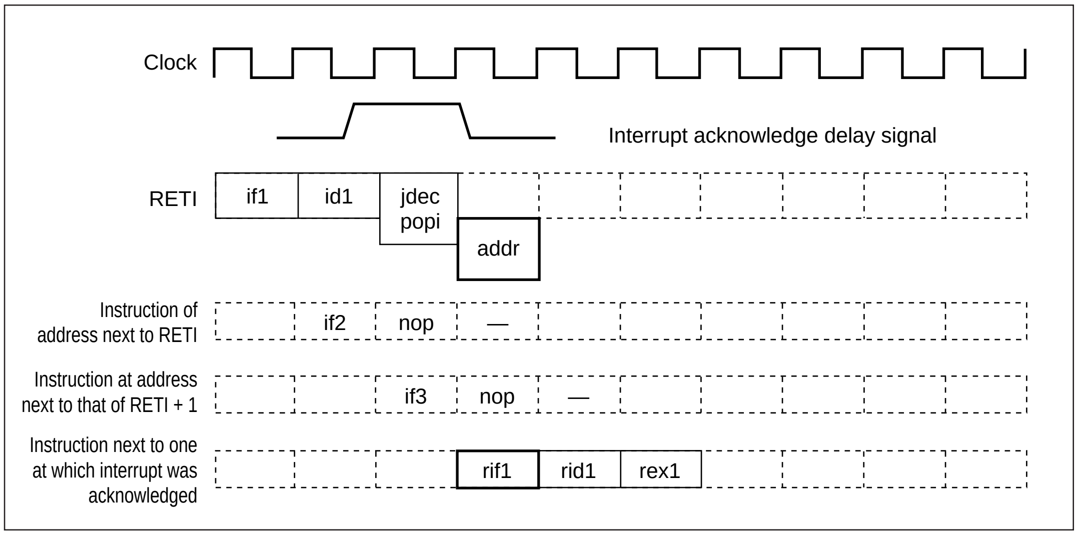
Figure 3-28. Timing by RETI Instruction

(a) Unconditional


Clock
Interrupt acknowledge delay signal
RETI
if1
id1
popi
addr
Instruction of address next to RETI
if2
nop
—
Instruction next to one at which interrupt was acknowledged
rif1
rid1
rex1

ifn : instruction fetch idn: instruction decode rifn : interrupt return destination instruction fetch
ridn : interrupt return destination instruction decode
rexn: interrupt return destination instruction execution

(b) Conditional instruction: Condition satisfied


Clock
Interrupt acknowledge delay signal
RETI
if1
id1
jdec popi
addr
Instruction of address next to RETI
if2
nop
—
Instruction at address next to that of RETI + 1
if3
nop
—
Instruction next to one at which interrupt was acknowledged
rif1
rid1
rex1

(8) Delaying interrupt acknowledgment

In the course of acknowledging an interrupt, registers SP, STK, and PC are automatically managed. To prevent conflicts with instructions that address these registers, acknowledging an interrupt is delayed when any of the following instructions that may cause such a conflict is executed. Note that the interrupt acknowledgement itself (branching to the interrupt servicing routine) still introduces only a single delay cycle.

Caution Interrupt is not acknowledged under following conditions and interrupt request is held until interrupt enables;
- **During peripheral I/O wait function**
- **During external memory access wait cycles**
- **During repeat process**

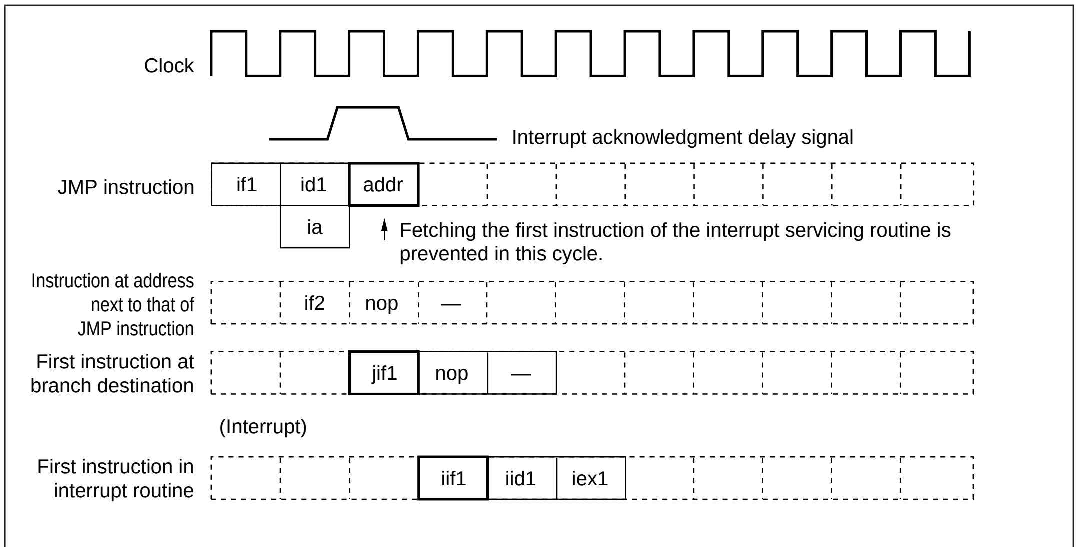
(a) Instructions generating delay of one instruction cycle

The following instructions cause a delay of interrupt acknowledgment of one instruction cycle:

- Decoding of unconditional JMP instruction (PC-relative jump by immediate data)
- Decoding of unconditional CALL instruction (PC-relative jump by immediate data)
- Decoding of unconditional RET instruction
- Decoding of unconditional RETI instruction
- Decoding of FINT instruction
- Fetching of loop end instruction

Figure 3-29 illustrates how an interrupt is delayed that occurs during the processing of any of these instructions.

Figure 3-29. Interrupt Delay Timing (one-cycle delay)


Clock
Interrupt acknowledgment delay signal
JMP instruction
if1
id1
addr
ia
Fetching the first instruction of the interrupt servicing routine is prevented in this cycle.
Instruction at address next to that of JMP instruction
if2
nop
—
First instruction at branch destination
jif1
nop
—
(Interrupt)
First instruction in interrupt routine
iif1
iid1
iex1

Remarks jifn, jidn, jexn: fetch, decode, or execution of instruction at branch destination
iifn, iidn, iexn: fetch, decode, or execution of interrupt routine
ia: branch destination address calculation addr: address

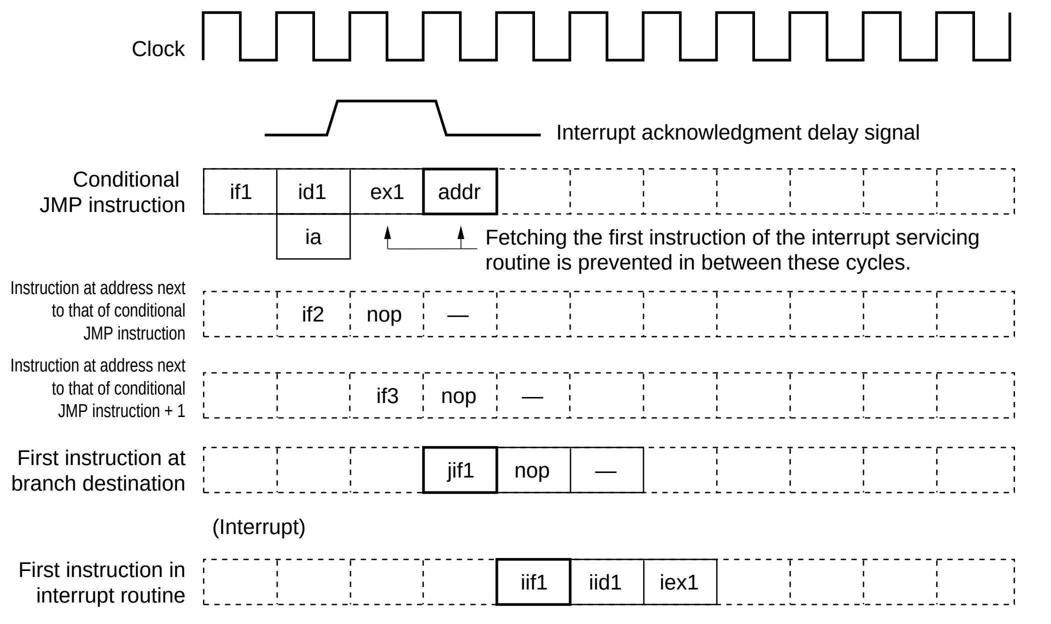
(b) Instructions generating delay of two instruction cycles

The following instructions cause a delay of interrupt acknowledgment of two instruction cycles:

- Decoding of onditional JMP instruction (PC-relative jump by immediate data)
- Decoding of conditional CALL instruction (PC-relative jump by immediate data)
- Decoding of conditional RET instruction
- Decoding of conditional RETI instruction
- Decoding of unconditional/conditional register-indirect JMP instruction
- Decoding of unconditional/conditional register-indirect CALL instruction
- Decoding of REP instruction
- Decoding of LOOP instruction

Figure 3-30 illustrates how an interrupt is delayed that occurs during the execution of any of these instructions.

Figure 3-30. Interrupt Delay Timing (two-cycle delay)


Clock
Interrupt acknowledgment delay signal
Conditional JMP instruction
if1
id1
ex1
addr
ia
Fetching the first instruction of the interrupt servicing routine is prevented in between these cycles.
Instruction at address next to that of conditional JMP instruction
if2
nop
—
Instruction at address next to that of conditional JMP instruction + 1
if3
nop
—
First instruction at branch destination
jif1
nop
—
(Interrupt)
First instruction in interrupt routine
iif1
iid1
iex1

Remarks ifn: fetch of instruction n
idn: decode of instruction n
exn: execution of instruction n
jifn, jidn, jexn: fetch, decode, or execution of instruction at branch destination
iifn, iidn, iexn: fetch, decode, or execution of interrupt routine
ia: branch destination address calculation
addr: address

(9) Conflict and recording of interrupt

(a) Recording interrupt

When an interrupt has been acknowledged, an interrupt servicing program is executed. During the execution, the global interrupt enable flag “EI” is automatically set to 1 (disable). Therefore, if another interrupt occurs during this period, it is not acknowledged immediately, but is recorded classified by the cause. When the interrupt servicing program has been ended in the RETI (return from interrupt) instruction, the EI flag is cleared to 0, enabling other interrupts. Consequently, the recorded interrupt is acknowledged and processed. This interrupt recording function works not only when EI is 1, but also when the corresponding interrupt enable flags are set to the disable state.

Cautions 1. All interrupt request are recorded, disregarding the settings of all interrupt enable/disable flags.

2. Only one level of interrupt can be recorded per cause.

3. The internal flag that records the occurrence of an interrupt is not cleared unless the corresponding interrupt is acknowledged.

4. The FINT instruction discards all interrupt requests. For further details refer to μPD7701x Family User’s Manual Instructions.

(b) Priority of interrupt

It is undefined which interrupt is served first if two or more interrupts occur at the same time.

3.4.5 Error status register (ESR)

This 16-bit register holds error flags which indicate some error status's of the processor. A write to bits 15-4 of this register is ignored. Undefined values are returned when these flags are read.

Bits 3-0 of ESR are set to 1 when an error occurs. The values of these bits are not clear to 0 unless a hardware reset is applied or they are rewritten by program (inter-register instruction).

The values of ESR can be read and written by executing the inter-register transfer instruction.

The value of this register is cleared to 0 at reset.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ESR													ovf	ste	lse	bac

(a) ovf: Overflow error flag

This flag is set to 1 if an overflow occurs while the operation unit calculates data in the 40-bit two's complement format.

(b) ste: Stack error flag

This flag is set to 1 when the stack overflows or underflows.

(c) lse: Loop stack error flag

This flag is set to 1 when the loop stack overflows or underflows.

(d) bac: Bus access error flag

This flag is set to 1 when prohibited parallel data memory access combination is executed. Combinations of prohibited parallel data memory access are as follows;

- Internal ROM and Internal ROM
- Internal ROM and External area
- External area and External area
- Peripheral area and Peripheral area

3.5 Data Addressing Unit

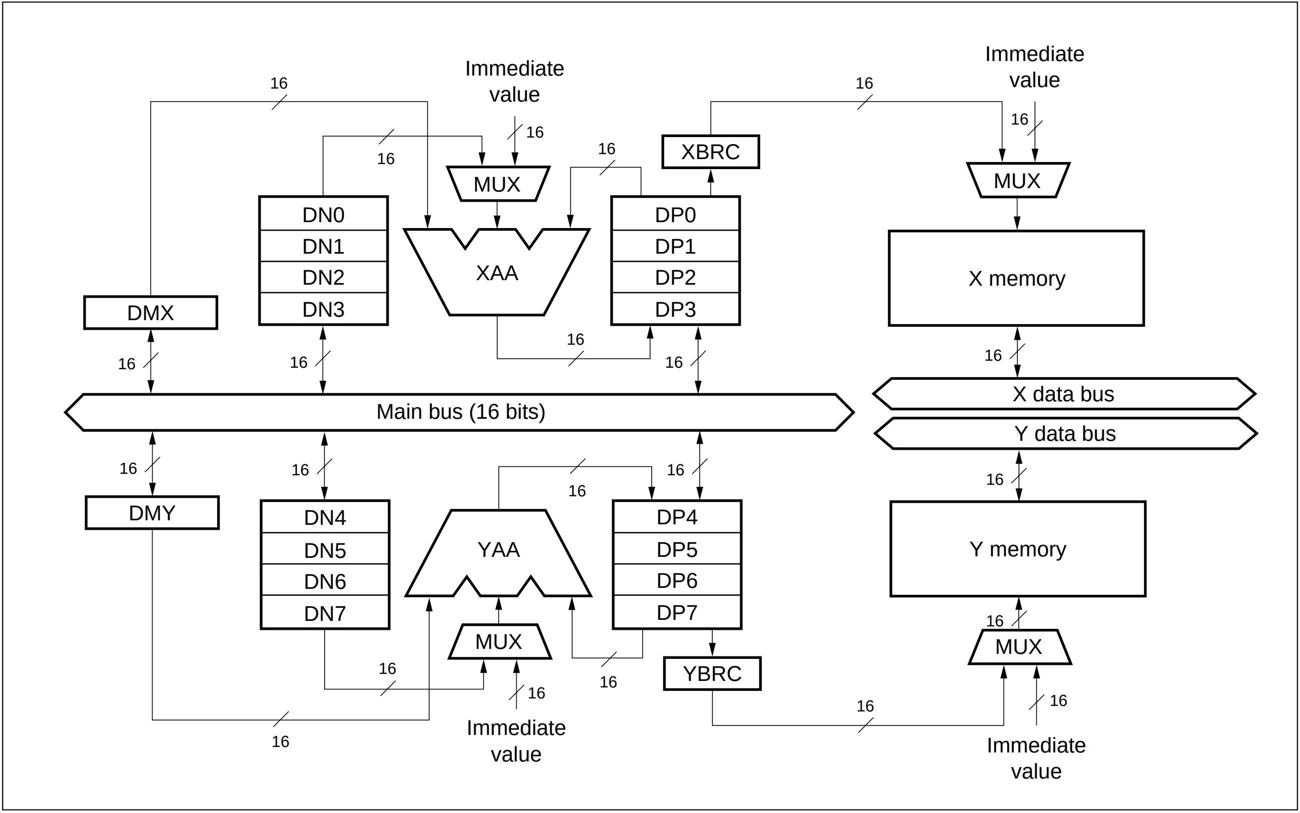
Generally, a DSP is required to access a large quantity of data flexibly and efficiently.

The μPD7701x family is provided with dedicated data addressing units to efficiently access the data memory spaces.

3.5.1 Block configuration

Figure 3-31 is the block diagram of the data address unit.

Figure 3-31. Data Addressing Unit



3.5.2 Data memory space

The devices of the μPD7701x family have two independent data memory spaces, X and Y, to which data can be accessed flexibly. Each of the X and Y data memory spaces is divided into internal memory and external memory areas. The internal memory area can always be accessed at high speeds as an internal resource of the device. The internal memory areas of both the X and Y memory spaces can be accessed simultaneously. The external memory area allows connection of memories of various speed range, using the incorporated software and hardware wait functions. In addition, the internal memory area is further divided into ROM and RAM areas.

This subsection describes the memory spaces.

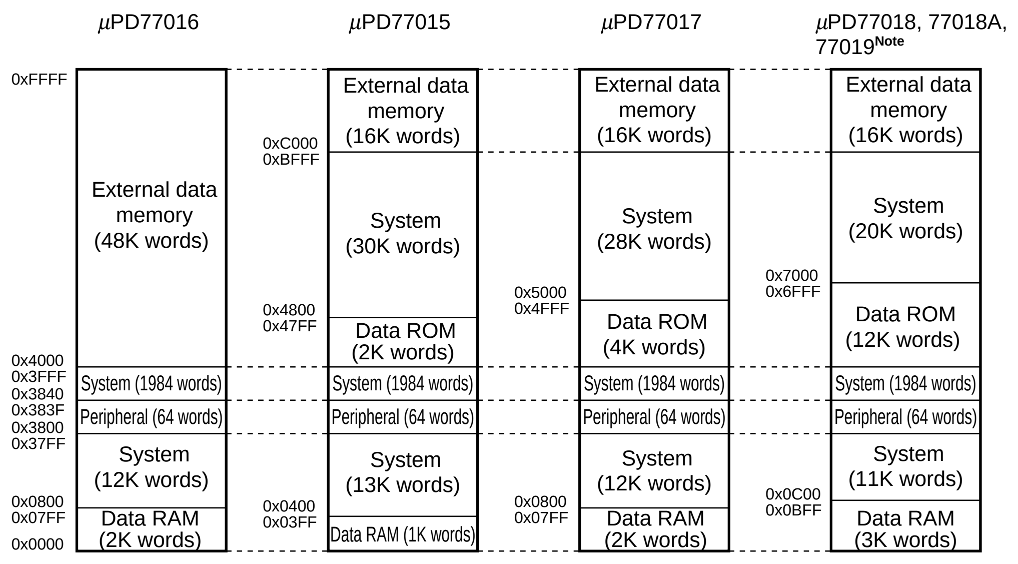
(1) X and Y memory spaces

The devices of the μPD7701x family have two independent data memory spaces: X and Y. These spaces are respectively accessed via the X and Y data buses described earlier in this chapter (refer to section 3.2.2 "Data bus"). The features of these memory spaces are as follows:

- One word consists of 16 bits.
- Both X and Y spaces have 64K words.

Although the memory maps of the X and Y memory spaces are the same, there are some differences among the products in the μPD7701x family. The following figure shows the X and Y memory maps of each product in the family.

★ ***Figure 3-32. X/Y Data Memory Map***



Caution No program or data must be stored to the addresses reserved for the system, nor must these addresses be accessed. If any of these addresses is accessed, normal operation of the μPD7701x family is not guaranteed.

Note The μPD77019-013 does not have the internal ROM of the μPD77019.

(2) Internal data memory

As shown by the memory map in Figure 3-32, a 16K-word area starting from address 0 functions as an internal area of the μPD77016. With the μPD77015, 77017, 77018, 77018A, and 77019, a 48K-word area starting from address 0 is used as the internal area. The internal area is divided into a ROM area, RAM area, peripheral area, and system area, of which the ROM and RAM areas are used as a data memory. The capacity of the internal data memory differs by processor type, which allows users to select the device best suited to their application.

For the details of the peripheral area, refer to section 3.7.2 “Peripheral registers”.

Caution Accessing system areas is prohibited.

(a) Internal ROM and RAM

As described above, the capacities of the internal ROM and RAM areas can be selected. The table below lists the available capacity options.

Note that the internal ROM of the μPD77019-013 cannot be used.

Table 3-18. ROM and RAM Capacities

	ROM		RAM	
Part number	X	Y	X	Y
μPD77016	None	None	2K words	2K words
μPD77015	2K words	2K words	1K words	1K words
μPD77017	4K words	4K words	2K words	2K words
μPD77018 μPD77018A μPD77019	12K words	12K words	3K words	3K words

(3) External data memory interface

(a) External data memory capacity

As shown in Figure 3-32, the μPD7701x family can expand the memory capacity by using an external data memory. The expandable capacity differs by processor type, as shown in the table below.

Table 3-19. Capacity of External Data Memory

Part number	X external data memory	Y external data memory
μPD77016	48K words	48K words
μPD77015 μPD77017 μPD77018 μPD77018A μPD77019	16K words	16K words

(b) Interface signals of external data memory

When using the external data memory, bear in mind the following differences from the internal data memory:

- As described in section 3.2.2 “Data bus”, the external data bus is shared by the X and Y data buses. Therefore, the X and Y spaces, which are logically separated when viewed from the program, can be considered as a single memory space if the $\overline{\text{X}}$/Y select signal is regarded as one of the address bits.
- An wait function is available which is effected by the data memory wait cycle register (DWTR) and the $\overline{\text{WAIT}}$ pin.

The external data memory interface uses the following pins:

<1> DA0-DA15 (address output pins)

These output pins constitute a 16-bit address bus. Note, however, that the μPD77015, 77017, 77018, 77018A, and 77019 are not provided with the DA14 and DA15 pins.

All these pins go into a high-impedance state while the bus is released.

They output a low level immediately after reset.

The statuses of these pins are not changed when the external data memory is not accessed (address output continues).

<2> $\overline{\text{X}}$/Y ($\overline{\text{X}}$/Y output pin)

This pin outputs a low level if the address bus DA0-DA15 accesses the X memory space; it outputs a high level if the Y memory is accessed.

This pin goes into a high-impedance state while the bus is released.

<3> D0-D15 (data input/output pin)

These pins constitute a 16-bit data bus.

They go into a high-impedance state while the bus is released, or when the external data memory is not accessed.

<4> $\overline{\text{MRD}}$ (memory read output pin)

This pin outputs the read strobe signal for the external data memory.

It goes into a high-impedance state while the bus is released.

Data is latched and MRD is switched in synchronization with the rising edge of CLKOUT.

<5> $\overline{\text{MWR}}$ (memory write output pin)

This pin outputs the write strobe signal for the external data memory.

It goes into a high-impedance state while the bus is released.

Data is output and MWR is switched in synchronization with the rising edge of CLKOUT.

<6> $\overline{\text{WAIT}}$ (wait input pin)

The μPD7701x inserts wait cycle(s) if this pin is made low when the external data memory is accessed.

<7> $\overline{\text{HOLDRQ}}$ (bus hold request input pin)

This pin inputs a signal requesting occupancy of the bus.

It is used to arbitrate the bus in a system where two or more CPUs, including the μPD7701x, shares the bus.

When this signal is made low, the bus is released to an external device after the current bus cycle has been completed.

<8> $\overline{\text{BSTB}}$ (bus strobe output pin)

This pin outputs a signal requesting use of the external data bus.

When the bus is controlled by the μPD7701x serving as a bus master, this signal functions as a bus strobe signal and indicates that the bus is accessed.

In the bus slave status, this signal functions as an external data bus request signal in response to the $\overline{\text{HOLDRQ}}$ signal output by a bus master.

<9> $\overline{\text{HOLDAK}}$ (bus hold acknowledge output pin)

This pin outputs a signal permitting an external device to use the bus.

It outputs a low level while the bus is released to an external device.

Table 3-20. Pin Status

Pin	I/O	During reset	Initial after reset	No external memory access	During bus release
★ DA0-DA15 (13[Note])	O	L	L	previous	Hi-Z
$\overline{\text{X}}$/Y	O	H	L	previous	Hi-Z
D0-D15	I/O	Hi-Z	Hi-Z	Hi-Z	Hi-Z
$\overline{\text{MRD}}$	O	H	H	H	Hi-Z
$\overline{\text{MWR}}$	O	H	H	H	Hi-Z
$\overline{\text{WAIT}}$	I	–	–	–	–
$\overline{\text{HOLDRQ}}$	I	–	–	–	L
$\overline{\text{BSTB}}$	O	H	H	H	H/L
$\overline{\text{HOLDAK}}$	O	previous	previous	H	L

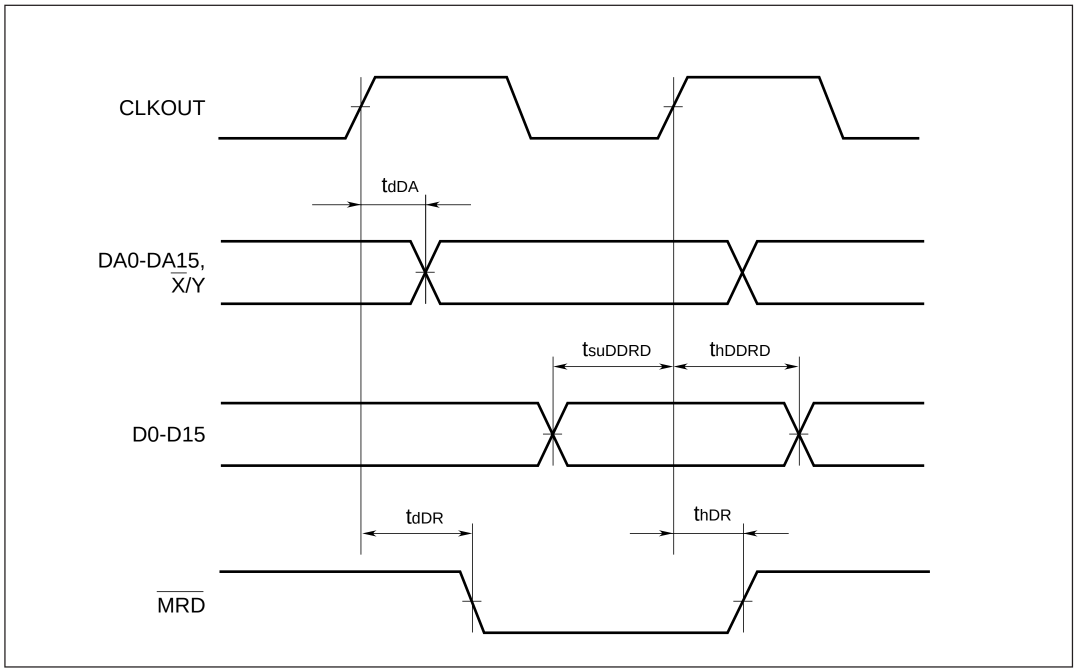
Note The μPD77015, 77017, 77018, 77018A, and 77019 are not provided with DA14 and DA15.

(c) Data memory access timing

Figure 3-33 (a) shows a read cycle without wait cycles, and Figure 3-34 (a) shows a write cycle without wait cycles. Figure 3-33 (b) shows a read cycle with wait cycles, and Figure 3-34 (b) shows a write cycle with wait cycles. If data memory read cycles are successively generated, $\overline{\text{MRD}}$ remains low.

Figure 3-33. Timing of Data Memory Read Cycle

(a) Without wait cycles



(b) With wait cycles

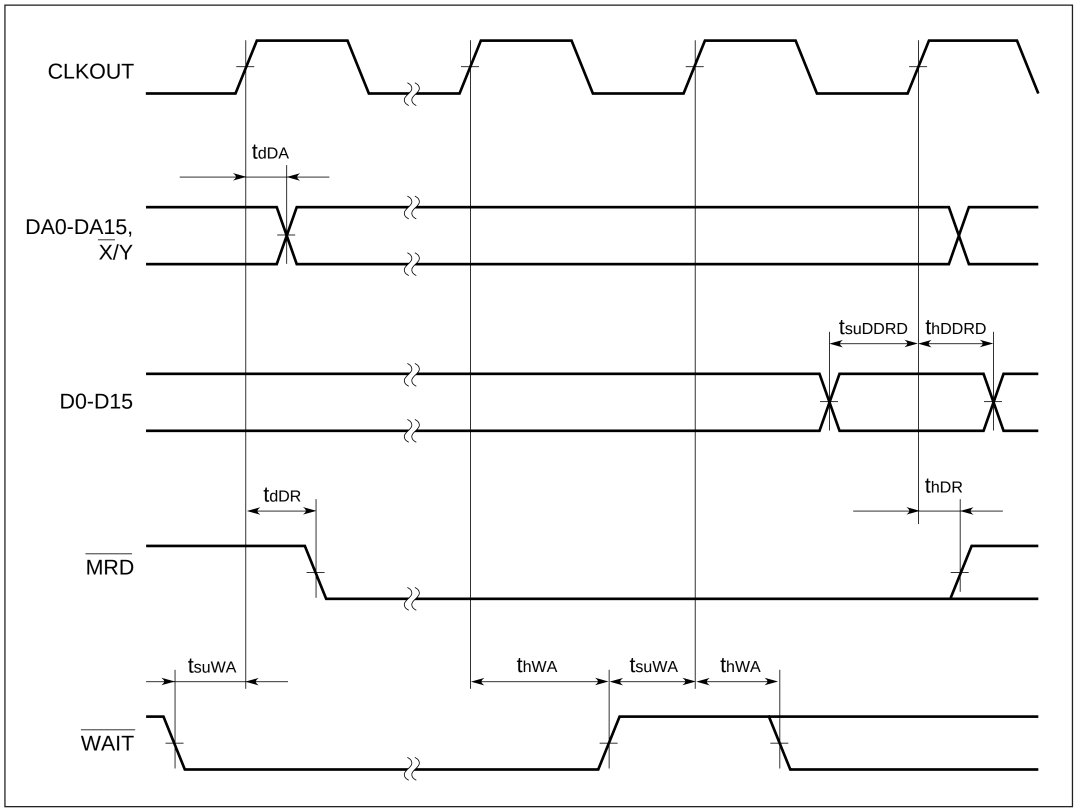
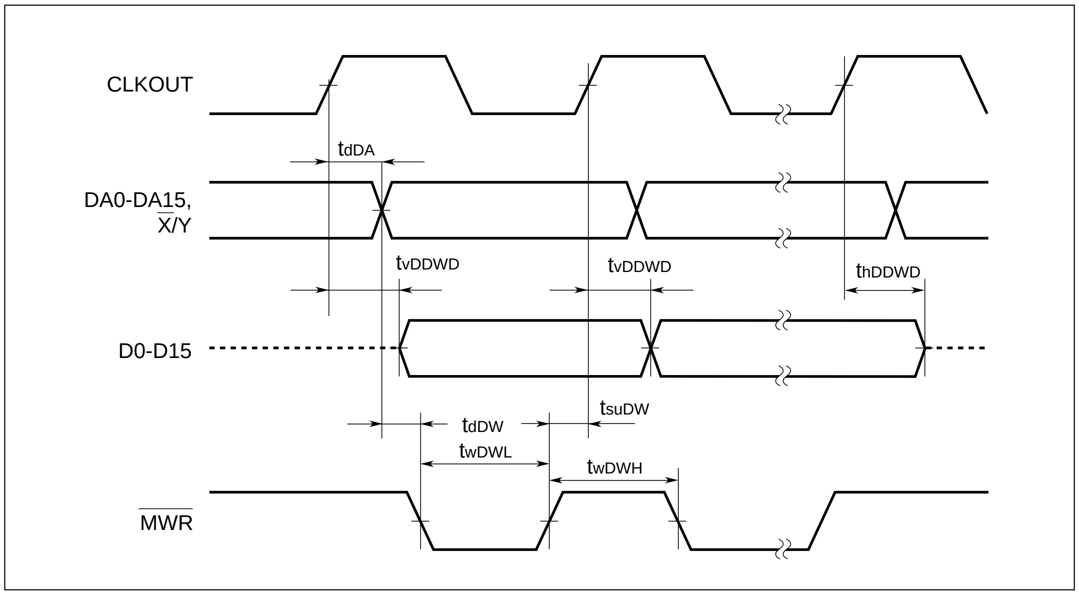
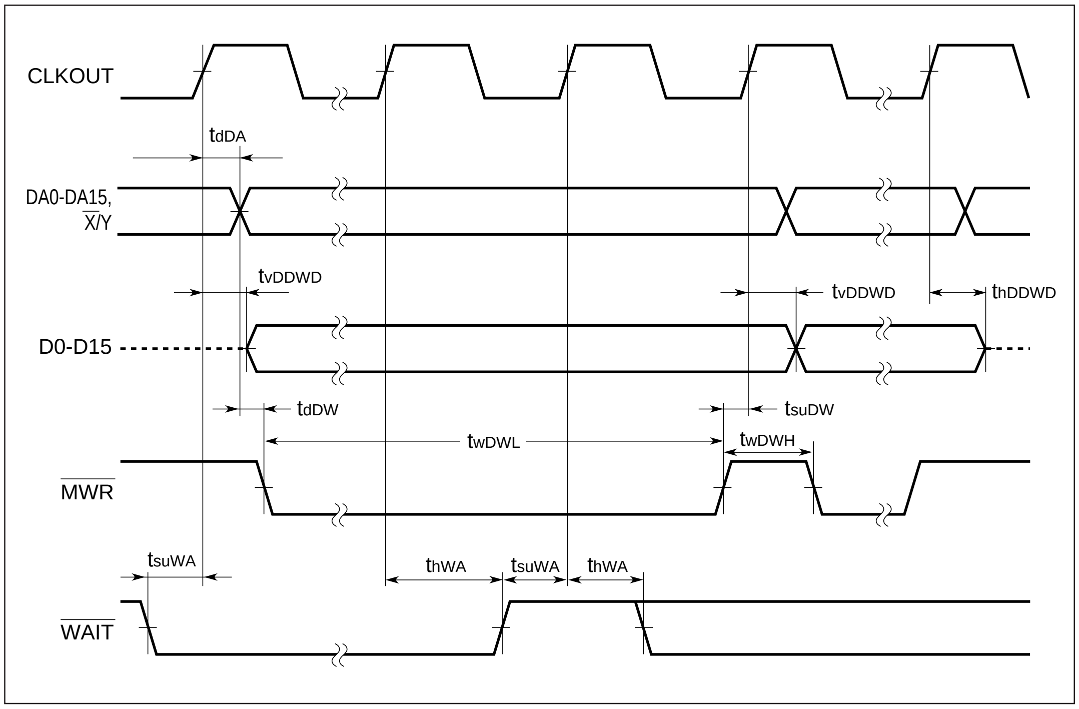


Figure 3-34. Timing of Data Memory Write Cycle

(a) Without wait cycles



(b) With wait cycles



(d) Wait controller

The wait controller enables access of an external data memory with a long access time, and inserts wait cycle(s) during external data memory access cycles. This wait function can be controlled in the following two ways:

• By hardware signal pin

The $\overline{\text{WAIT}}$ pin is provided, so that any number of wait cycles can be inserted by means of handshaking by hardware.

• By DWTR (data memory wait cycle register)

The number of wait cycles specified in advance by software can be set to DWTR which is mapped to the memory space as a peripheral register.

The $\overline{\text{WAIT}}$ signal is a negative logic input signal and is sampled at the rising edge of CLKOUT when the external data memory is accessed. While this signal is active (low level), the corresponding cycle serves as a wait cycle (refer to (c) "Data memory access timing").

<1> DWTR (data memory wait cycle register)

DWTR is a register that implements a programmable wait function controlled by software. This register is provided as one of the peripheral registers. The number of wait cycles specified in advance can be selected and set to this register by the application program.

DWTR is a 16-bit register and is divided into fields as follows, for each processor type:

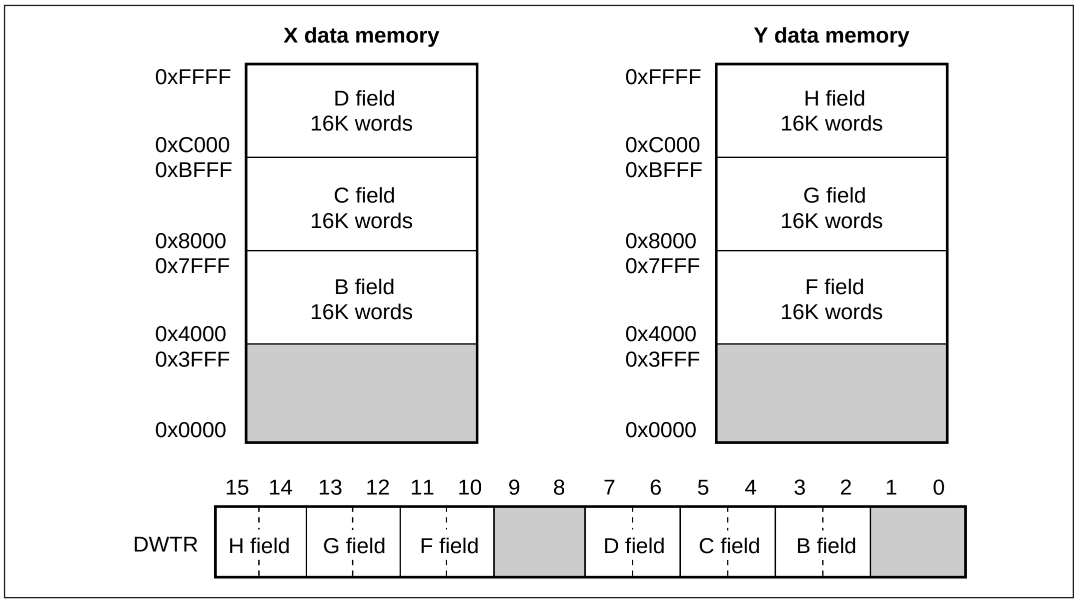
• The DWTR of the μPD77016 consists of six fields of two bits each. These six fields correspond to six 16K-word banks corresponding to the external memory. The six 16K-word banks are created by dividing each of the 64K-word external X and Y memory spaces by four. For each bank, the number of wait cycles can be set independently.

• The DWTR of the μPD77015, 77017, 77018, 77018A, and 77019 consists two 2-bit fields, which correspond to the two 16K-word banks external X and Y memory. The two 16K-word banks are created by dividing each of the 64K-word external X and Y memory space by four. For each bank, the number of wait cycles can be set independently.

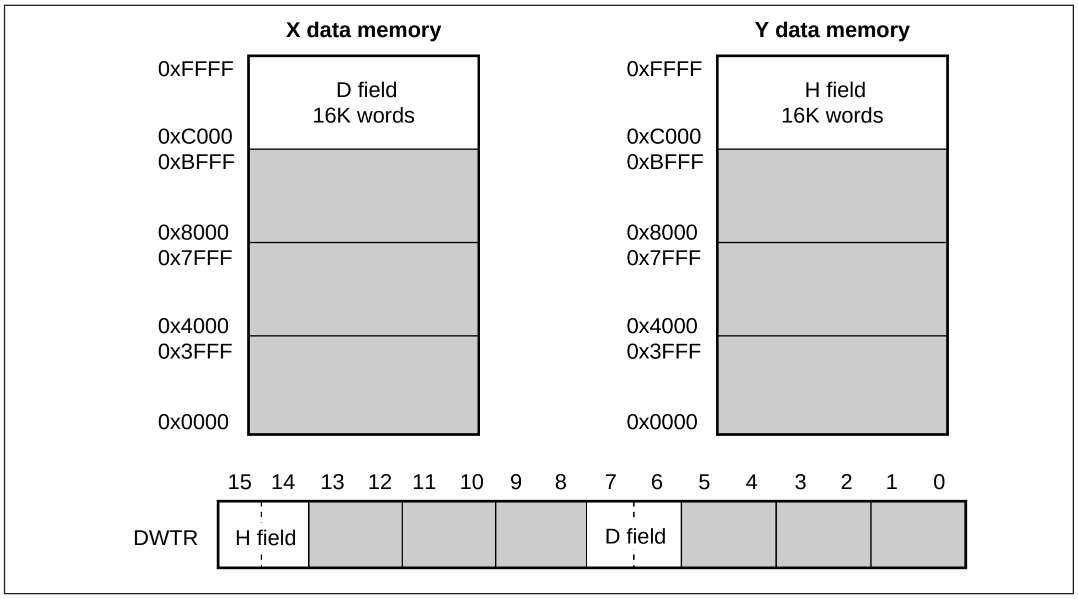
Figure 3-35 (a) shows the image of controlling the DWTR of the μPD77016. Figure 3-35 (b) shows the image of controlling the DWTR of the μPD77015, 77017, 77018, 77018A, and 77019. Table 3-21 shows the relations between the value set to each field of DWTR and the number of wait cycles.

Figure 3-35. Data Memory Control Bank and DWTR Field Configuration

(a) μPD77016



(b) μPD77015, 77017, 77018, 77018A, 77019



Caution

- **With the μPD77016, writing data to bits 9, 8, 1, and 0 is ignored. These bits are undefined when read.**
- **With the μPD77015, 77017, 77018, 77018A, and 77019, writing data to bits 13-8 and 5-0 is ignored. These bits are undefined when read.**

Table 3-21. Set Value of DWTR Field and Number of Wait Cycles

Bit	Wait cycles	Remark
0 0	0	1-cycle access: SRAM etc. with an access time of about 8 ns is connected (at 33 MHz).
0 1	1	2-cycle access: SRAM etc. with an access time of about 35 ns is connected (at 33 MHz).
1 0	3	4-cycle access: SRAM etc. with an access time of about 85 ns is connected (at 33 MHz).
1 1	7	8-cycle access: Mask ROM etc. with an access time of about 150 ns is connected (at 33 MHz)

Caution When DWTR is set, the specified number of wait cycles becomes valid when an instruction immediately after the instruction that has set the data to DWTR is executed.

<2> Overlapping operation by $\overline{\text{WAIT}}$ pin and DWTR

If the hardware control by the $\overline{\text{WAIT}}$ pin and the software control by DWTR are implemented simultaneously, the following operation is performed:

(1) In the external data memory access cycle, the number of wait cycles specified by DWTR is unconditionally inserted.

(2) At this time, the status of the $\overline{\text{WAIT}}$ pin is sampled at the rising edge of the last cycle of all the memory cycles. While the $\overline{\text{WAIT}}$ pin is active (low), wait cycles are continuously inserted.

(e) Bus arbitration

The μPD7701x family is provided with the bus arbitration function to support memory configuration of multiple bus masters. It is considered that the typical multiple bus masters share the memory in the following combination:

- μPD7701x (master) – μPD7701x (slave)
- μPD7701x – host CPU
- μPD7701x – DMA controller

Care must be exercised if the supply voltage of each bus master differs from that of the others.

Figure 3-36 shows a typical example of bus arbitration. The operations are as follows:

<1> When the μPD7701x family executes an instruction to access the common memory, $\overline{\text{BSTB}}$ becomes active (low).

<2> The external device makes $\overline{\text{HOLDRQ}}$ active (low).

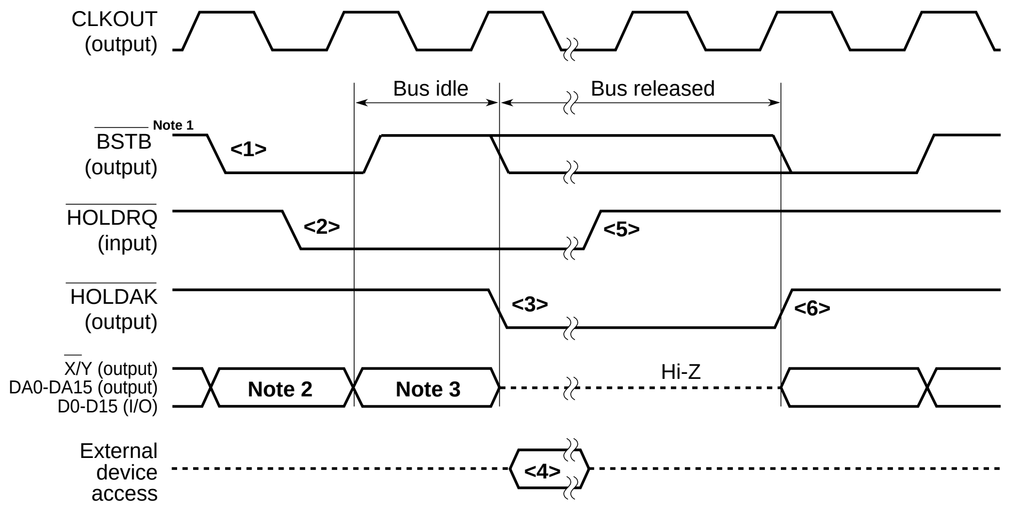
<3> The μPD7701x family makes $\overline{\text{HOLDAK}}$ active after the bus cycle has been completed (after the accesses are completed if accesses are successively generated).

<4> The external device uses the bus.

<5> The external device makes $\overline{\text{HOLDRQ}}$ inactive (high) after completing access.

<6> The μPD7701x family makes $\overline{\text{HOLDAK}}$ inactive (high) to resume bus access.

Figure 3-36. Bus Arbitration Procedure



Note 1. $\overline{\text{BSTB}}$ becomes high when the last access has been completed.
2. External memory access: the last access if successible accesses are generated.
3. Previous value: i.e., not Hi-Z (no external memory access).

(4) Restriction of simultaneous access

As described earlier, the μPD7701x family divides its memory space in various ways. The μPD7701x family has a function to access two memory spaces, X and Y, simultaneously by means of parallel load, etc. This paragraph describes the combination of memory spaces in which access can be made. Table 3-22 shows the combination of memory spaces which can be simultaneously accessed.

Table 3-22. Simultaneous Access to X and Y Memory Spaces

		X memory area via X data bus			
		Internal ROM	Internal RAM	External memory	Peripheral register
Y memory area via Y data bus	Internal ROM	—	OK	—	OK
	Internal RAM	OK	OK	OK	OK
	External memory	—	OK	—	OK
	Peripheral register	OK	OK	OK	—

Remark OK : Can be simultaneously accessed
— : Cannot be simultaneously accessed

3.5.3 Addressing mode

The μPD7701x family is provided with a powerful architecture to realize high-speed, flexible data memory access. The X and Y memory areas are addressed by completely independent but functionally identically addressing units. This subsection describes the architecture and addressing modes implemented.

(1) Function of each part of addressing unit

The functions of the registers and functional blocks shown in Figure 3-31 are as follows:

(a) Data pointers (DP0-DP7)

These eight 16-bit registers are used for indirect addressing. DP0-DP3 are used to specify an address of the X memory space, while DP4-DP7 are used to specify an address of the Y memory space.

The values of DP0-DP7 can be input/output via the main bus.

(b) Index registers (DN0-DN7)

These eight 16-bit registers modify DP0-DP7. After the memory has been accessed, DPn is modified by the value of DNn (n: 0-7, each corresponds respectively). The values of DN0-DN7 can be input/output via the main bus.

The valid number range of this register is given by –32768 (0x8000) to +32767 (0x7FFF).

(c) Modulo registers (DMX, DMY)

These two 16-bit registers specify the ring count range when DP0-DP7 are modified during the ring count operation performed.

The ring count range for DP0-DP3 is specified by DMX. DMY is used to specify that for DP4-DP7.

The values of DMX and DMY can be input/output via the main bus.

The valid number range of this register is given by +1 (0x0001) to +32767 (0x7FFF).

(d) Address ALUs (XAA, YAA: X and Y Address ALUs)

These two 16-bit ALUs are used to modify DP0-DP7.

XAA is used to modify DP0-DP3, while YAA modifies DP4-DP7.

(e) Bit reverse circuits (XBRC, YBRC: X and Y Bit Reverse Circuits)

When a bit reverse access is performed, these circuits output an address that reverses the order of the DP0-DP7 values, so that the highest bit becomes the lowest, and vice versa.

(f) Multiplexer (MUX)

This circuit selects one of several signals for output.

(2) Types of addressing modes

The data memory addressing modes are hierarchically classified below.

There are one type of direct addressing mode and seven types of indirect addressing modes that are implemented by using data pointers (DPs) as the base address indicator.

- Direct addressing
- Indirect addressing
  - * DPn (no change)
  - * DPn++ (post increment)
  - * DPn– – (post decrement)
  - * DPn## (post index addition)
  - * DPn%% (post modulo index addition)
  - * !DPn## (pre-bit reverse and post index addition)
  - * DPn##imm (immediate addition)

(a) Direct addressing

Direct addressing is to directly express an address value and address division (X or Y) in an instruction word. Data of 16 bits is exchanged between a specified address of a specified division (X or Y) and a general-purpose register via X or Y data bus.

For details, refer to “μPD7701x Family User’s Manual Instructions”.

Example 1: Load

```
    R0H = *0x1234:X;
```

16-bit data is loaded to the H part (middle 16 bits) of general-purpose register R0 from address 0x1234 of the X memory.

Example 2: Store

```
    *0x1234:X = R0H;
```

16-bit data is stored to address 0x1234 of the X memory from the H part (middle 16 bits) of general-purpose register R0.

Caution The X and Y memory spaces cannot be accessed simultaneously by means of direct addressing.

(b) Indirect addressing

In all the indirect addressing modes, the DPn register (data pointer) is used. The basic features of indirect addressing are summarized below.

- As the address value, the current value of specified DPn is output in all the modes except the bit reverse index addition mode. In the bit reverse index addition mode, the current value of the specified DPn is reversed and output (refer to Figure 3-37).
- If it is specified to modify DPn, DPn is modified after the data memory has been accessed.
- The modified DPn value, i.e. the new address, is effective from the next instruction onwards.
- DPn alone cannot be modified.
- If an immediate value has been set to DPn, either by an inter-register transfer or an immediate value set instruction, the new address is effective from the next but one following instruction (refer to “μPD7701x Family User’s Manual Instructions”).
- DP0-DP3 are used to access the X memory space, and DP4-DP7 are used to access the Y memory space.

Each indirect addressing mode is described next.

<1> *DPn (no change)

The memory is accessed with the value of DPn. The value of DPn is preserved after the access has been completed.

Example:

```
R1L = *DP0;
```

16-bit data is loaded from the X memory address indicated by the value of DP0 to the L part (lower 16 bits) of R1.

<2> *DPn++ (post increment)

The memory is accessed with the value of DPn. The value of DPn is incremented (+1) after the access has been completed.

Example:

```
R2H = *DP4++;
```

16-bit data is loaded from the Y memory address indicated by the value of DP4 to the H part (middle 16 bits) of R2, and then the value of DP4 is incremented.

<3> *DPn– – (post decrement)

The memory is accessed with the value of DPn. The value of DPn is decremented (–1) after the access has been completed.

Example:

```
R3E = *DP1--;
```

8-bit data (the lower 8 bits of the 16 bits) is loaded from the X memory address indicated by the value of DP1 to the E part (higher 8 bits) of R3, and then the value of DP1 is decremented.

<4> *DPn## (post index addition)

The memory is accessed with the value of DPn. After the access has been completed, the value of DNn is added to DPn. Note that the n-th index register DNn corresponds only to the n-th data pointer DPn (e.g. DN1 to DP1).

The valid number range of DNn is given by –32768 (0x8000) to +32767 (0x7FFF).

Example:

```
R4L = *DP5##;
```

16-bit data is loaded from the Y memory address specified by the value of DP5 to the L part (lower 16 bits) of R4, and then the value of DN5 is added to DP5.

<5> *DPn%% (post modulo index addition)

The memory is accessed with the value of DPn. After the access has been completed, the value of DNn is added to DPn. In addition, modulo adjustment is made by DMX or DMY (DMX is used when n=0-3, and DMY is used when n=4-7). Note that the n-th index register DNn corresponds only to the n-th data pointer DPn (e.g. DN1 to DP1).

The valid number range of DNn is given by –32768 (0x8000) to +32767 (0x7FFF).

For the details of modulo index addition and modulo adjustment, refer to <9> “Modulo index addition and cyclic buffer”.

Example:

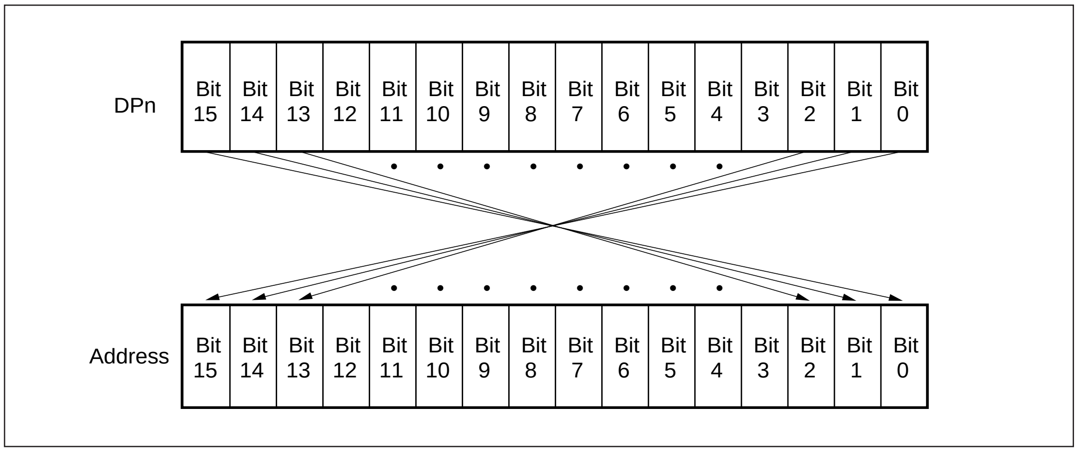
```
R5H = *DP3%%;
```

16-bit data is loaded from the X memory address specified by the value of DP3 to the H part (middle 16 bits) of R5. Then the value of DN3 is added to DP3, and modulo adjustment is made by DMX.

<6> *!DPn## (pre-bit reverse and post index addition)

The memory is accessed by using the value that reverses the order of the DPn values, as shown in Figure 3-37, and the value of DNn is added to DPn after the access has been completed. Note that the value of DNn having the same number as that of DPn must be added to DPn (for example, DN1 to DP1). This function is suitable for applications such as FFT.

Figure 3-37. Reversing Bits of DPn



Example:

```
R6H = *!DP6##;
```

16-bit data is loaded from the Y memory address specified by the reserved bits of DP6 to the H part (middle 16 bits) of R6, and then value of DN6 is added to DP6.

Remark DPn is not modified by bit-reversed access to the address, and the bit-reversed address is not fed back to DPn. After bit-reversed access, the value of DNn is added to the value of DP6 (original DP6 value) before bit reversion.

<7> *DPn##imm (post immediate addition)

The memory is accessed with the value of DPn, and immediate value imm is added to DPn after the access has been completed.

The valid number range of imm is given by –32768 (0x8000) to +32767 (0x7FFF).

Example:

```
R7L = *DP2##100;
```

16-bit data is loaded from the X memory address specified by the value of DP2 to the L part (lower 16 bits) of R7, and then immediate value “100” is added to DP2.

Caution The immediate addition addressing mode cannot be used to access the X and Y memories simultaneously.

<8> Modifying data pointers

Table 3-23 summarizes how the data pointers are modified as a result of accessing the memory in the above addressing modes.

Table 3-23. Modifying Data Pointers

(a) Operation

Example	Operation
DPn	No modification
DPn++	DPn <— DPn + 1
DPn– –	DPn <— DPn – 1
DPn##	DPn <— DPn + DNn (Values of corresponding DN0-DN7 are added to DP0-DP7). Example: DP0 <— DP0 + DN0
DPn%%	(n=0-3) DPn = ((DP$_L$ + DNn) mod (DMX + 1)) + DP$_H$
	(n=4-7) DPn = ((DP$_L$ + DNn) mod (DMY + 1)) + DP$_H$
!DPn##	Reverses bits of DPn and then accesses memory. After memory has been accessed, DPn <— DPn + DNn
DPn##imm	DPn <— DPn + imm

(b) Value range

	Hexadecimal	Decimal
DPn	0x0000 to 0xFFFF	0 to +65535
DNn	0x8000 to 0x7FFF	–32768 to +32767
DMX/DMY	0x0001 to 0x7FFF	1 to +32767
imm	0x8000 to 0x7FFF	–32768 to +32767

<9> Modulo index addition and cyclic buffer

The modulo index addition mode is provided for configuring a cyclic buffer (also called a ring buffer).

• Rule of operation

After the memory has been accessed by using the value of DPn, DPn is modified. At this time, the operation is performed according to the following rules:

(1) Executes operation of $DP_L = DP_L + DNn$.

(2) If $DP_L \leq DMa$ as a result,

$DPn = DP_L + DP_H$ is treated as the operation result.

If not (i.e., when $DP_L > DMa$),

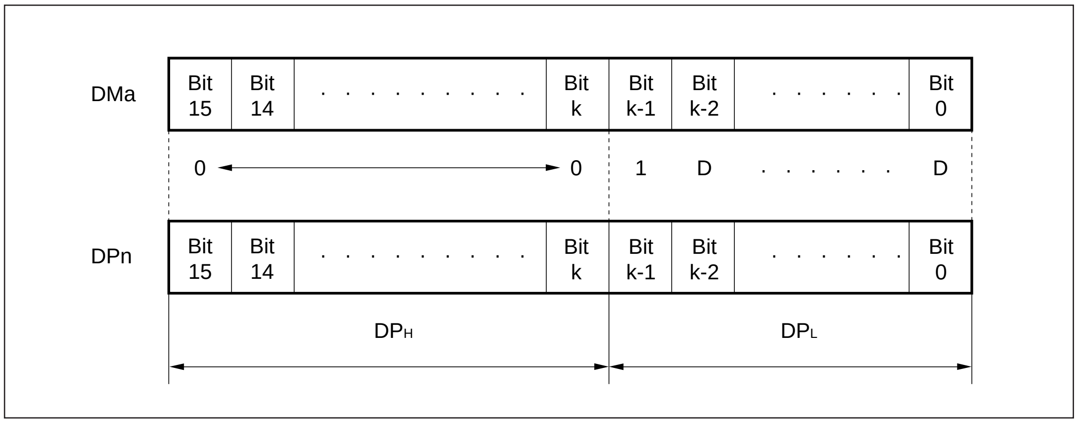
$DPn = (DP_L + DNn) \bmod (DMa + 1) + DP_H$ is treated as the operation result.

Where,

DP_H	:	lower k bits of the initial value of DPn, which is 0, if the value of DMa is in the range of [2^k, $2^{(k-1)}$] (Refer to Figure 3-38.)
DP_L	:	value of lower k bits of DPn in the above case (Refer to Figure 3-38.)
DMa	:	specified DPn corresponding to DMX or DMY

Remark The process (2) above is called modulo adjustment.

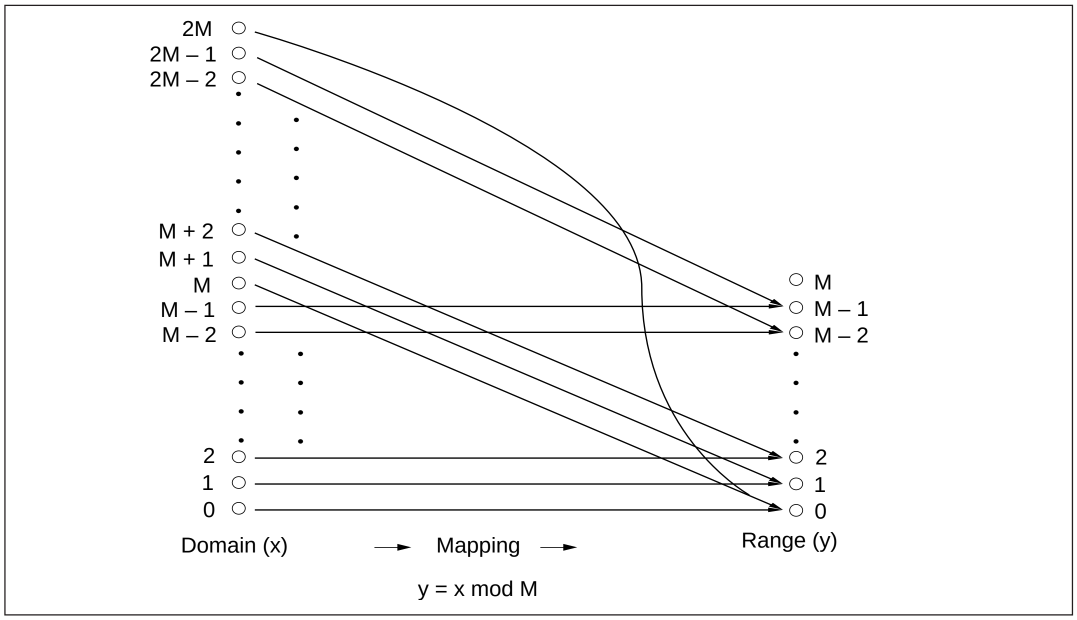
Figure 3-38. Division of DPn



• **Meaning**

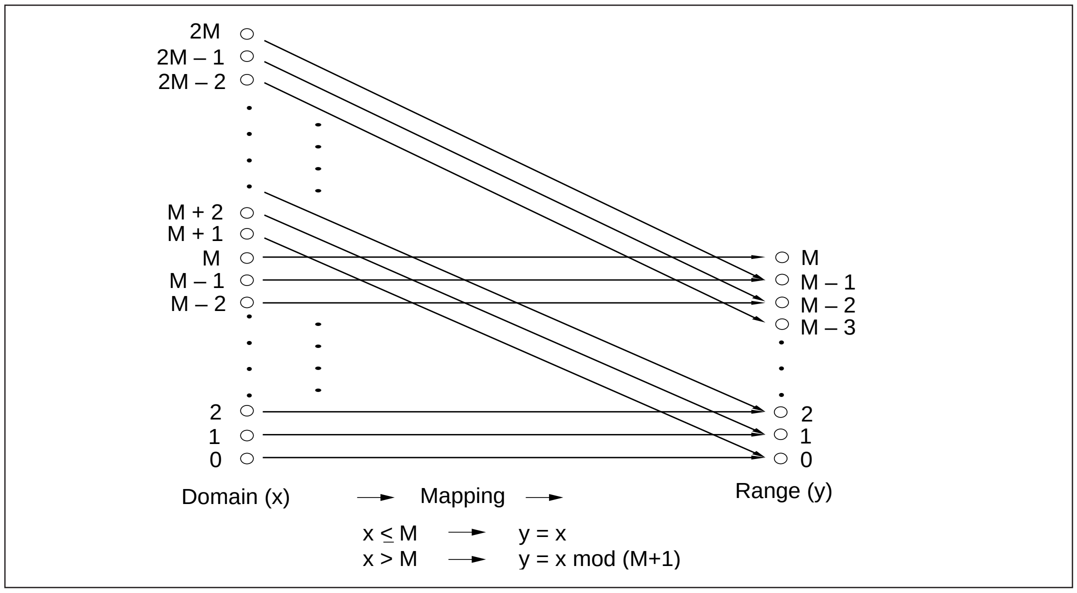
The ordinary modulo operation can be considered as the mapping shown in Figure 3-39.

Figure 3-39. Mapping of Ordinary Modulo Operation



In contrast, modulo adjustment can be considered as the mapping shown in Figure 3-40.

Figure 3-40. Mapping of Modulo Adjustment



The difference between the two in terms of the range is that the usable buffer size in Figure 3-39 is M, while it is M+1 in Figure 3-40, where the value set to DMa is M, because the range in this case corresponding to the size of the buffer. Consequently, the maximum buffer size of 0x8000 can be used, as described below, despite the maximum set value of DMa is 0x7FFF.

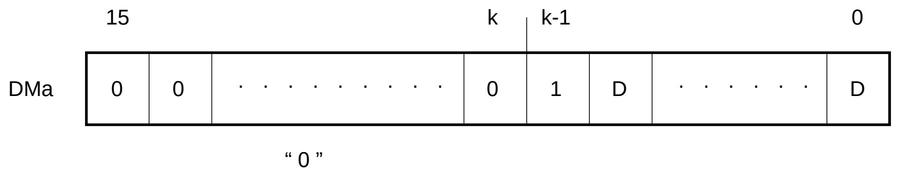
- **Operation range of ring count**

The first address of the range in which a ring count operation is executed in connection with the modulo index addition is determined by the values of the data pointer and modulo register.

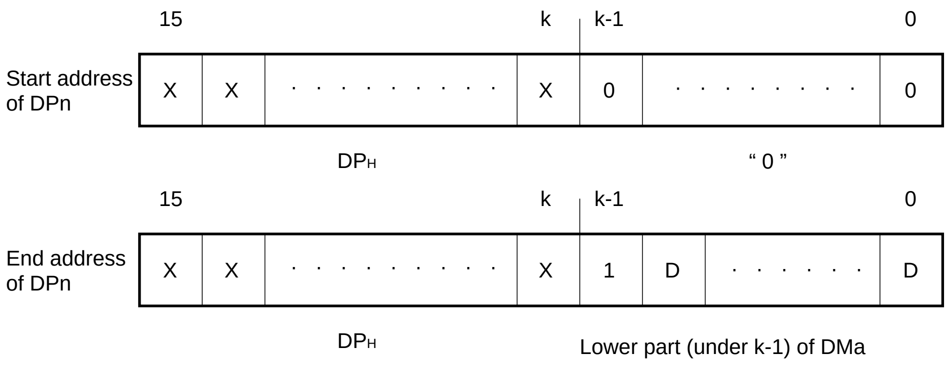
Where the value of the modulo register (DMX/Y) is $2^{k-1} \leq DMX/Y < 2^k$, the starting address of the ring count operation is the 16-bit value whose higher "16–k" bits are the same as those of the data pointer and whose lower k bits are zeros. The end address is the value whose higher "16–k" bits are the same as those of the data pointer and whose lower k bits are the same as those of the modulo register. The higher "16–k" bits of the data pointer always remain unchanged.

Ring Count Operation Range

When DMa is set as follows,



ring count start address and end address are like follows.



- **Restriction**

Observe the following restrictions in executing modulo addressing:

- Keep the range of DMa to [1-0x7FFF].
- Make sure that the absolute value of the value of DNn does not exceed DMa.

Caution **Because 0 cannot be set to DMa, a cyclic buffer with buffer size = 1 cannot be configured.**

- **Example of modulo index addition**

An example of operation process when a cyclic buffer is configured by using modulo index addition is shown below.

Example 1.

```
DMX=0x7;
DN0=1;
DP0=0x0;
```

At this time, the value of DP0 is updated as follows by means of modulo index addition:

```
DP0=0x0
  ↓    0x0+1
DP0=0x1
  ↓    0x1+1
DP0=0x2
  ↓    0x2+1
DP0=0x3
  ↓    0x3+1
DP0=0x4
  ↓    0x4+1
DP0=0x5
  ↓    0x5+1
DP0=0x6
  ↓    0x6+1
DP0=0x7
  ↓    0x7+1=0x8 → 0x8–(0x7+1)=0x0
DP0=0x0
  ↓    0x0+1
DP0=0x1
  ↓    0x1+1
DP0=0x2
  ↓    0x2+1
DP0=0x3
  ↓    0x3+1
DP0=0x4
  :
```

Example 2.

```
DMX=0xA;
DN0=3;
DP0=0x10;

At this time, the value of DP0 is updated as follows by
means of modulo index addition:

DP0=0x10
  ↓    0x10+3
DP0=0x13
  ↓    0x13+3
DP0=0x16
  ↓    0x16+3
DP0=0x19
  ↓    0x19+3=0x1C → 0x1C-(0xA+1)=0x11
DP0=0x11
  ↓    0x11+3
DP0=0x14
  ↓    0x14+3
DP0=0x17
  ↓    0x17+3
DP0=0x1A
  ↓    0x1A+3=0x1D → 0x1D-(0xA+1)=0x12
DP0=0x12
  ⋮
```

3.6 Operation Unit

The general-purpose registers in this unit are source of all operands and destination of all results of arithmetic/logic operations. The general-purpose registers are connected to the

- main bus for inter-register transfers
- X and Y data bus for data exchange with the data memories and peripheral registers

All kinds of arithmetic/logic operations which are part of the following instruction types are carried out in the operation unit:

- trinomial instructions

  all operations which involve 3 input operands, e.g.

```
MADD:  R0 = R0 + R1H * R2H
```

- binomial instructions

  all operations which involve 2 input operands, e.g.

```
ADD:  R0 = R2 + R3
```

- monomial instructions

  all operations which involve 1 input operand, e.g.

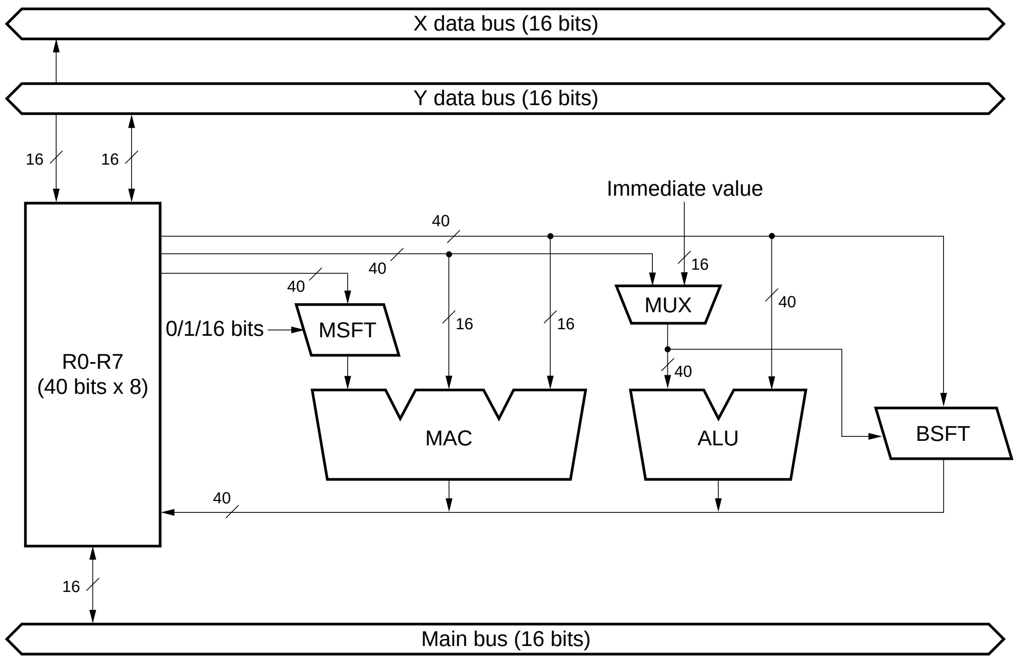
```
NEG:  R0 = –R1
```

The section describes in details the functions and data formats of the general-purpose registers R0-R7, of the multiplier /accumulator MAC and the MAC input shifter MSFT, of the arithmetic/ logic unit ALU, and of the barrel shifter BSFT. For the block diagram of this unit, refer to section 3.6.1 “Block configuration”.

3.6.1 Block configuration

Figure 3-41 is the block diagram of the operation unit.

Figure 3-41. Operation Unit



R0-R7:	General-purpose registers
MAC:	Multiply accumulator
MSFT:	MAC input shifter
ALU:	Arithmetic logical unit
BSFT:	Barrel shifter

3.6.2 General-purpose registers and data formats

The features of the general-purpose registers are as follows:

- 40-bit registers
- Eight registers (R0-R7) available
- Function as input/output parameters of operation instructions
  (only general-purpose registers, in addition to immediate data, can be described as parameters of operation instructions)
- Exchange data with X and Y data memories and peripheral registers (load/store function)
- Transfer data with other registers

(1) Partitioning of the general-purpose registers

Although a general-purpose register consists of 40 bits, the register is divided into three parts, as follows, so that only a specified part of the register can be used to transfer and load/store data or to execute arithmetic operations. In this case, the three parts are exclusive to each other.

- L part : bits 15-0 (lower 16 bits)
- H part : bits 31-16 (middle 16 bits)
- E part : bits 39-32 (higher 8 bits)

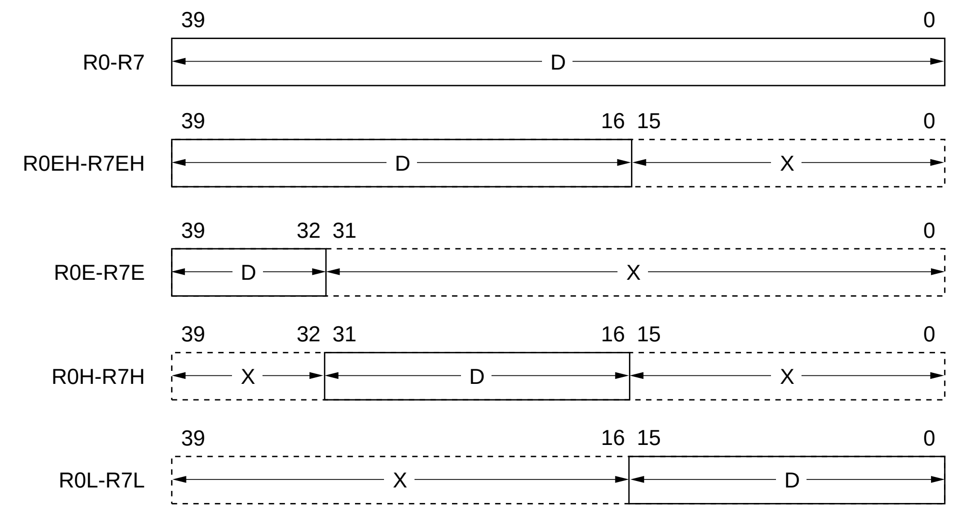
Depending on the type of arithmetic/logic operation respectively data transfer different parts of a general-purpose register are involved, as shown in Table 3-24.

The figure below shows these five formats (except R0HL-R7HL) with assembly names.

Table 3-24. Formats of General-purpose Registers

	R0-R7	R0L-R7L	R0H-R7H	R0E-R7E	R0HL-R7HL	R0EH-R7EH
	40 bits	16 bits	16 bits	8 bits	32 bits	24 bits
MAC multiply/accumulate	X	X	X	–	–	–
MAC exclusive multiply	X	–	X	–	–	–
ALU	X	–	–	–	X	–
BSFT	X	X	–	–	–	–
X/Y bus transfer	X	X	X	X	–	X
inter-register transfer	–	X	–	–	–	–

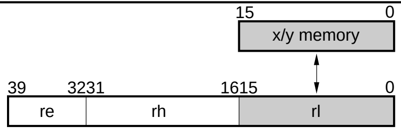
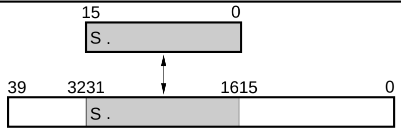
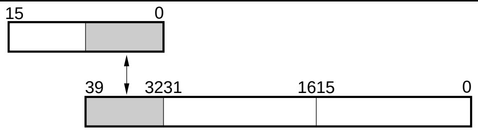
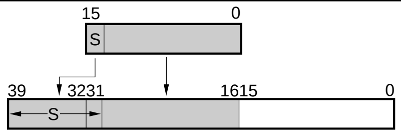
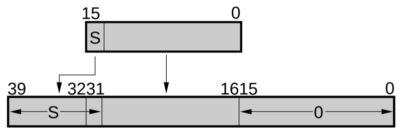
Figure 3-42. Formats of General-purpose Registers



Remarks D: numeric value
X: invalid

Figure 3-43 shows data exchange between general-purpose registers and data memory.

Figure 3-43. Data Exchange between General-purpose Registers and Data Memory

Number format in general register	Operation examples	General-purpose register → Memory		Memory → General-purpose register
16 bit unsigned	Unsigned multiply increment, decrement	rl	 15 0 x/y memory 39 3231 1615 0 re rh rl	rl
16 bit signed fixed point without sign extension	Signed multiply	rh		rh
Extension	Store/restore of 40 bit numbers	re		re
16 bit signed fixed point with sign extension	Signed multiply	–		reh
40 bit signed fixed point	MAC/ALU operations barrel shifter	–		ro

Remarks ▨ Changed

□ Unchanged

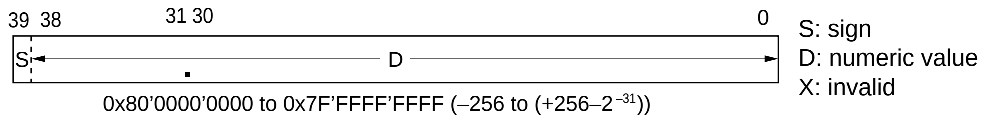
(2) Numeric format

The general-purpose registers of the μPD7701x family can process fixed-point and integer data. The architecture places an emphasis on operations of fixed-point data, however.

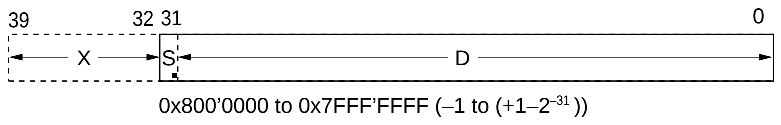
(a) Fixed-point format

The fixed-point format uses the position between bits 31 and 30 as the decimal point. Fixed-point data can be expressed in three ways: in 40-bit, 32-bit, and 16-bit units.

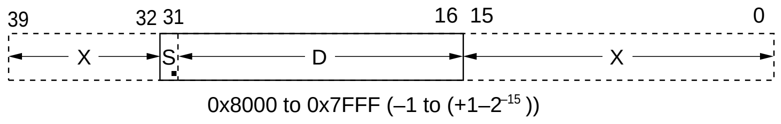
● 40-bit data format (input for addition/subtraction/and/or/xor)



● 32-bit data format (input for exponent instruction)



● 16-bit data format (input for multiplication instruction)

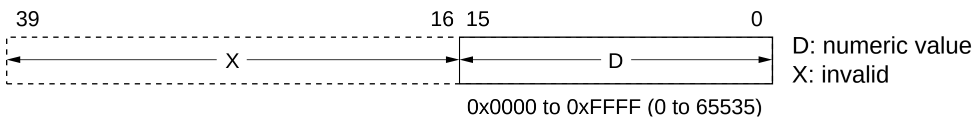


Remark The absolute value of data never exceeds 1 in the 32-bit fixed-point format or 16-bit fixed-point format. As long as an accumulative operation is executed on a general-purpose register with these formats as operands, therefore, the E part functions as an overflow absorbing area (called a head room). This function allows omission of judgment of overflow when 256 accumulative operations are performed even if it is assumed that an overflow of 1LSB (of the E part) occurs as a result of one accumulative operation.

(b) Integer format

The integer format is illustrated below.

● 16-bit data format (input for multiplication and shift instructions)



3.6.3 Operation functions of multiply accumulator (MAC) and MAC input shifter (MSFT)

The multiply accumulator performs the following functions:

- Multiplication

```
MPY:  ro = rh * rh’
```

- Extends multiplication and its result to 40 bits and adds the result to specified general-purpose register

```
MADD:  ro = ro + rh * rh’ (signed-signed multiply)
MSUB:  ro = ro – rh * rh’ (signed-signed multiply)
SUMA:  ro = ro + rh * rl  (signed-unsigned multiply)
UUMA:  ro = ro + rl * rl’ (unsigned-unsigned multiply)
```

- Extends multiplication and its result to 40 bits and adds the result to the result of shifting specified general-purpose register 1/16 bits to the right

```
MAS1:  ro = (ro>>1)  + rh * rh’
MAS16: ro = (ro>>16) + rh * rh’
```

Caution MAC, ALU, and BSFT cannot operate simultaneously.

(1) Multiplication function

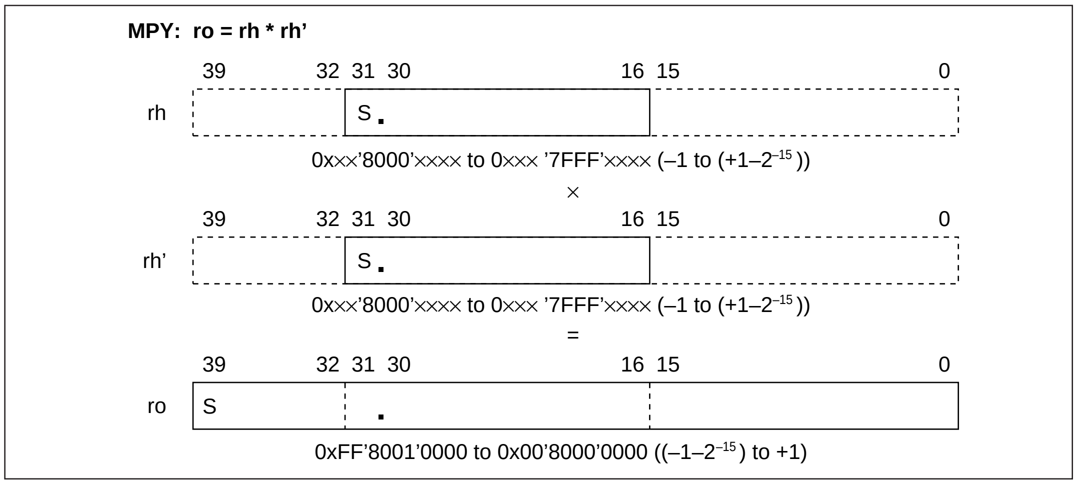
The multiplication function is implemented by the multiply accumulator (MAC). This function can be implemented in the following three ways, depending on the data type to be handled:

- Signed-signed multiply
- Signed-unsigned multiply
- Unsigned-unsigned multiply

(a) Signed-signed multiply

Both of the two operands are of signed 16-bit fixed-point type. Therefore, data is set to the H part of a general-purpose register whose bit 31 indicates the sign. A representation of this operation process is illustrated in Figure 3-44.

Figure 3-44. Signed-Signed Multiply


MPY: ro = rh * rh’
39 32 31 30 16 15 0
rh
S
0x××’8000’×××× to 0x×× ’7FFF’×××× (–1 to (+1–2^–15))
×
rh’
=
ro
0xFF’8001’0000 to 0x00’8000’0000 ((–1–2^–15) to +1)

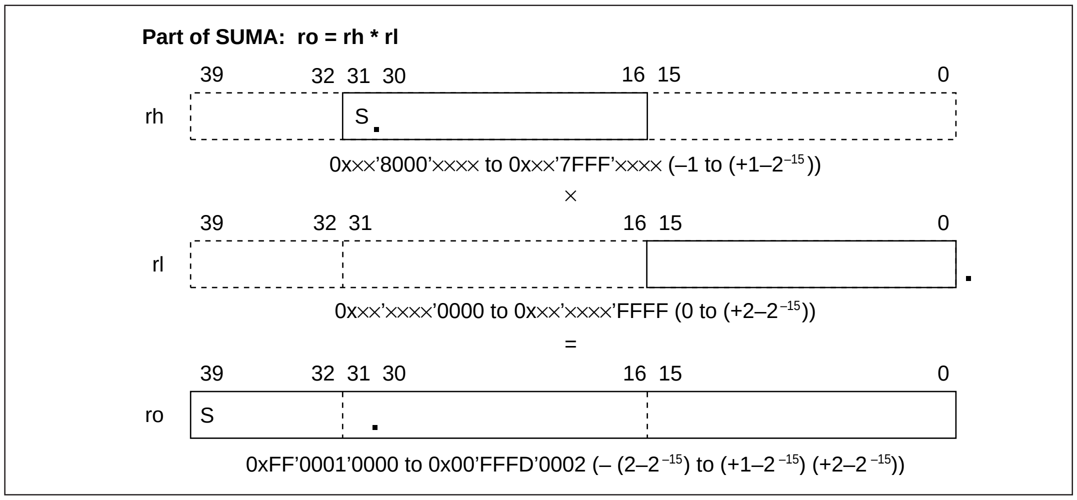
Remarks **1.** If multiplication between 0x8000 (–1) is executed, the result is 0x00’8000’0000. However, because +1 cannot be expressed in the range of the 32-bit fixed-point format, an overflow occurs (extension bit re = 0x00 is different from the sign bit of the 32-bit format in this case). However, the value is accurate when viewed from the point of the 40-bit format (0x00’8000’0000 = +1).

2. Since a multiplication of two 16-bit values produces maximum 31 valid bits, the LSB of the result registers is always 0.

(b) Signed-unsigned multiply

One of the two operands is set to the H part of a general-purpose register in the 16-bit fixed-point type, where bit 31 of the register indicates a sign. The other parameter is set to the L part of a general-purpose register in the integer format. Figure 3-45 shows the image of this operation process.

Figure 3-45. Signed-Unsigned Multiply

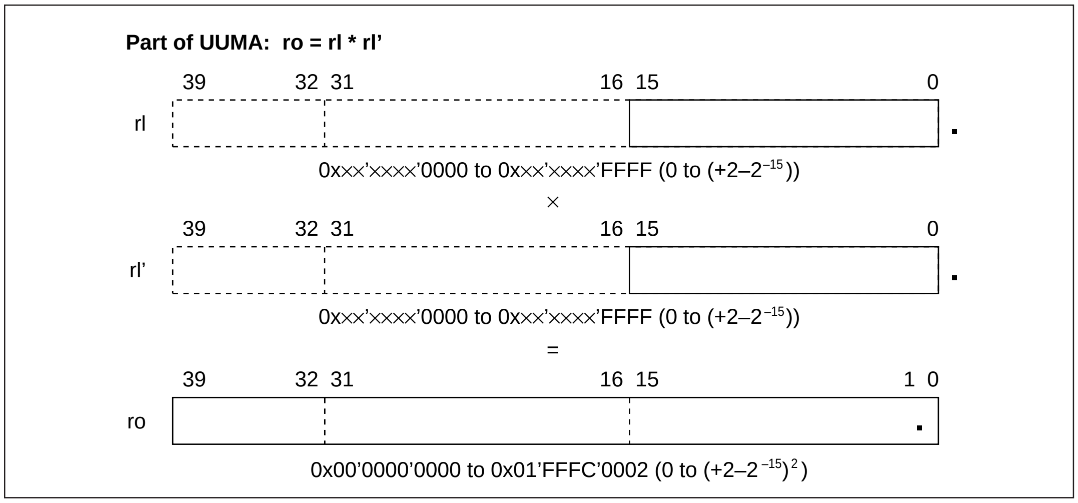

Part of SUMA: ro = rh * rl
39 32 31 30 16 15 0
rh
S
0x××’8000’×××× to 0x××’7FFF’×××× (–1 to (+1–2^–15))
×
39 32 31 16 15 0
rl
0x××’××××’0000 to 0x××’××××’FFFF (0 to (+2–2^–15))
=
ro
0xFF’0001’0000 to 0x00’FFFD’0002 (– (2–2^–15) to (+1–2^–15) (+2–2^–15))

Caution **There is no exclusive instruction that executes this operation. This operation is performed as part of the sign-unsign multiply add instruction.**

(c) Unsigned-unsigned multiply

Both of the two operands are set to the L parts of general-purpose registers in the integer format. Figure 3-46 shows the image of this operation process.

Figure 3-46. Unsigned-Unsigned Multiply


Part of UUMA: ro = rl * rl'
39
32 31
16 15
0
rl
0x××'××××'0000 to 0x××'××××'FFFF (0 to (+2–2^–15))
×
rl'
0x××'××××'0000 to 0x××'××××'FFFF (0 to (+2–2^–15))
=
1 0
ro
0x00'0000'0000 to 0x01'FFFC'0002 (0 to (+2–2^–15)^2)

Caution	**There is no exclusive instruction that executes this operation. This operation is performed as part of the unsign-unsign multiply add instruction.**

(2) Accumulative multiplication function (trinomial operation)

All the trinomial operations executed by the μPD7701x family are accumulative multiplication. The accumulative multiplication can be implemented in the following three ways, depending on the shift command to the register that is used for the accumulative operation (two accumulative operations, accumulative addition and accumulative subtraction, can be executed, however). At this time, the shift processing is executed by the MAC input shifter (MSFT).

- Accumulative multiplication
- 1-bit shift accumulative multiplication
- 16-bit shift accumulative multiplication

The accumulative multiplication can also be classified into the following three types by the data type of the parameters used for the operation:

- Signed-signed multiply
- Signed-unsigned multiply
- Unsigned-unsigned multiply

In all, therefore, the following six types of trinomial operation instructions are available:

- Multiply add (signed-signed multiply and accumulative add)
- Multiply sub (signed-signed multiply and accumulative sub)
- Sign-unsign multiply add (signed-unsigned multipy and accumulative add)
- Unsign-unsign multiply add (unsigned-unsigned multiply and accumulative add)
- 1-bit shift multiply add (signed-signed multiply and accumulative add after 1-bit shift)
- 16-bit shift multiply add (signed-signed multiply and accumulative add after 16-bit shift)

The accumulative multiplication function implements trinomial operations where three parameters are used. Of these, two are the parameters for multiplication and the other is for accumulative operation. General-purpose registers are specified for these parameters. In this case, registers can be specified in duplicate.

Table 3-25 shows these combination.

Table 3-25. Accumulative Multiplication Function

		Signed-signed	Signed-unsigned	Unsigned-unsigned
Multiply/ accumulate	MSFT 0 bit	ro = ro ± rh * rh’ (MADD, MSUB)	ro = ro + rh * rl (SUMA)	ro = ro + rl * rl’ (UUMA)
	MSFT 1 bit	ro = (ro>>1) + rh * rh’ (MAS1)	—	—
	MSFT 16 bit	ro = (ro>>16) + rh * rh’ (MAS16)	—	—
Exclusive multiply (Binomial operation)		ro = rh * rh’ (MPY)	—	—

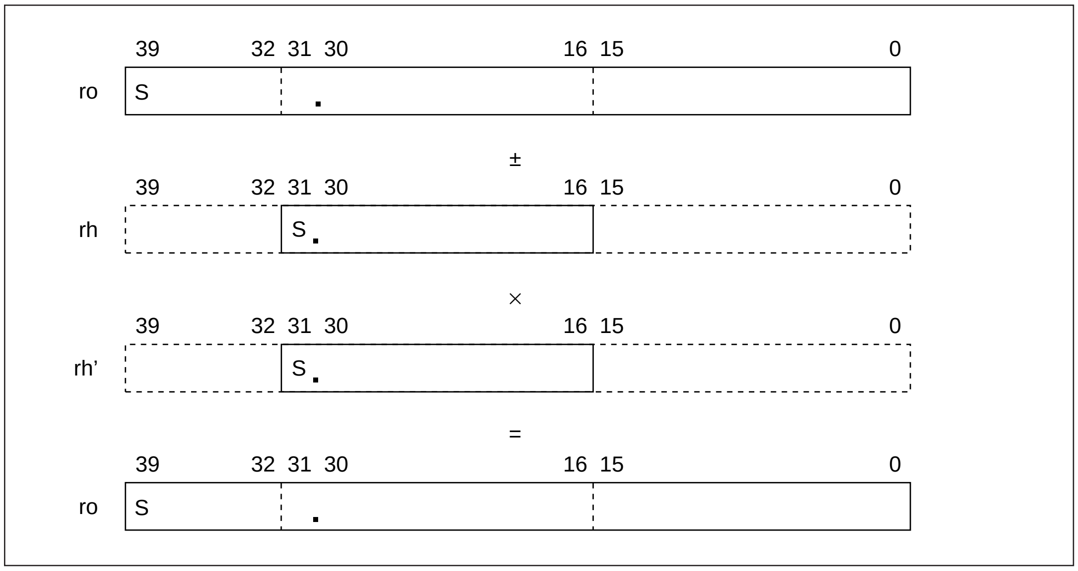
(a) Accumulative multiplication

The multiplier input operands are of (signed) 16-bit fixed point type. The multiplication result is added to respectively subtracted from a 40-bit fixed-point operand. The related instructions are:

```
MADD:  ro = ro + rh * rh’
MSUB:  ro = ro – rh * rh’
```

Figure 3-47 shows the image of this operation.

Figure 3-47. Accumulative Multiplication



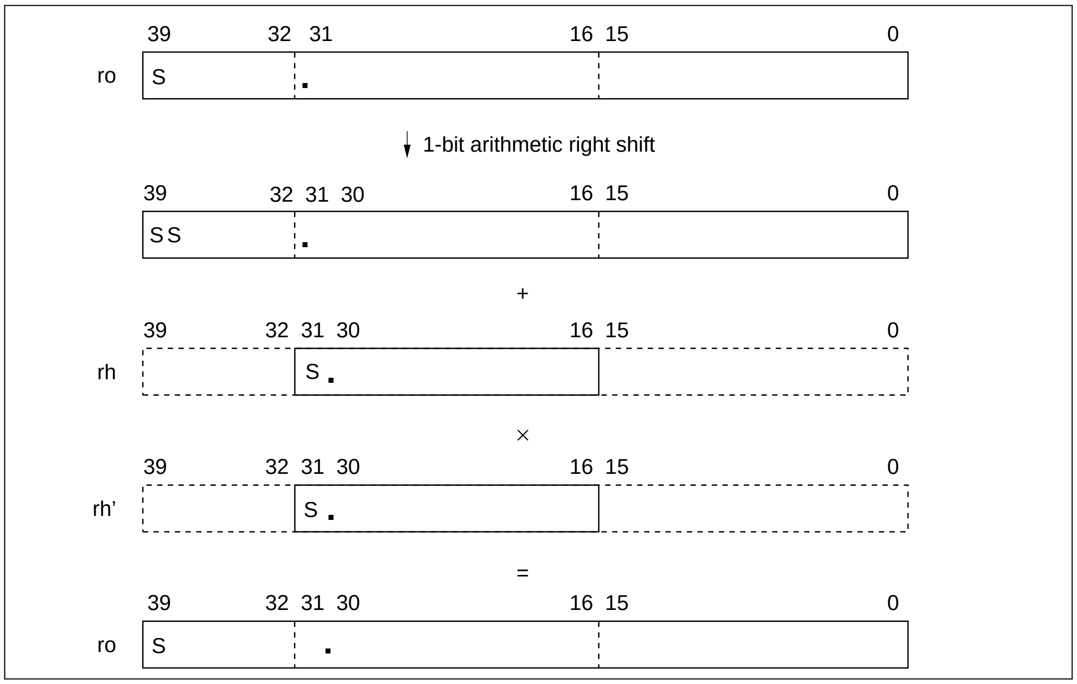
(b) 1-bit shift accumulative multiplication

The multiplier input operands are of (signed) 16-bit fixed point type. The multiplication result is added to a 1 bit right shifted 40-bit fixed-point operand. The related instruction is:

```
  MAS1:  ro = (ro>>1) + rh * rh'
```

Figure 3-48 shows the image of this operation:

Figure 3-48. 1-Bit Shift Accumulative Multiplication



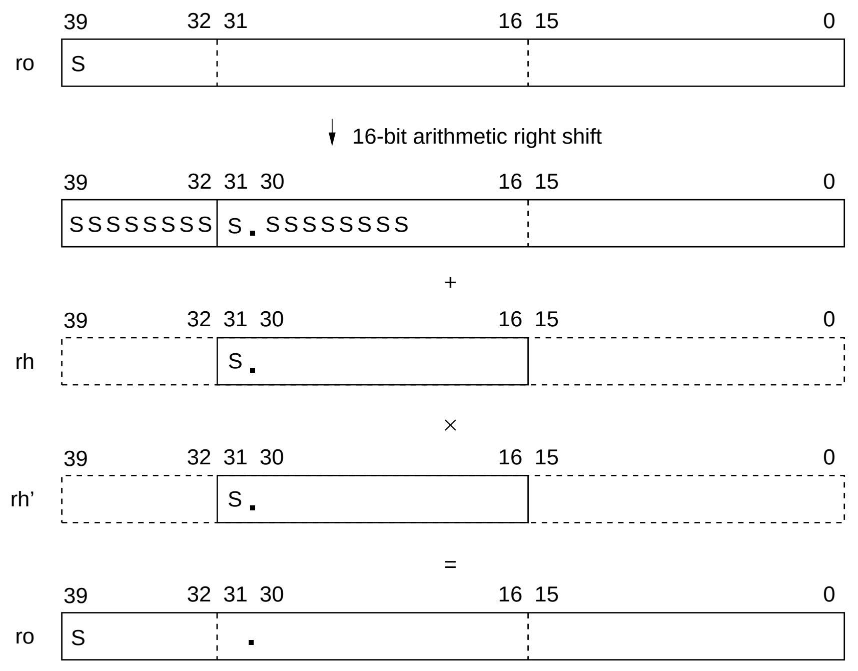
(c) 16-bit shift accumulative multiplication

The multiplier input operands are of (signed) 16-bit fixed point type. The multiplication result is added to a 16 bit right shifted 40-bit fixed-point operand. The related instruction is:

```
   MAS16:  ro = (ro>>16) + rh * rh’
```

Figure 3-49 shows the image of this operation:

Figure 3-49. 16-Bit Shift Accumulative Multiplication



3.6.4 Operation functions of arithmetic and logic unit (ALU)

The arithmetic and logic unit (ALU) executes an arithmetic or logical operation on two or one 40-bit input data, and outputs one 40-bit data.

As both two operands for a binomial operation, general-purpose registers can be specified, or a register can be specified as one of the operands with immediate data specified as the other (immediate data cannot be used with the LT instruction, however). If general-purpose registers are specified as both operands, they can be in duplicate. Any general-purpose register can be specified for a monomial operation. It is also possible to specify any general-purpose register to store the result of the operation.

Caution MAC, ALU, and BSFT cannot operate simultaneously.

(1) Arithmetic operation instruction

(a) Binomial arithmetic operation

The following binomial arithmetic operation instructions are available. For each instruction, refer to “μPD7701x Family User’s Manual Instructions.”

- Multiply instruction (MPY: executed by MAC)
- Add instruction (ADD)
- Immediate add instruction (IADD)
- Subtract instruction (SUB)
- Immediate subtract instruction (ISUB)
- Less-than instruction (LT)

(b) Monomial arithmetic operation

The following monomial arithmetic operation instructions are available. For each instruction, refer to “μPD7701x Family User’s Manual Instructions.”

- Clear instruction (CLR)
- Increment instruction (INC)
- Decrement instruction (DEC)
- Absolute value instruction (ABS)
- Two’s complement instruction (NEG)
- Clip instruction (CLIP)
- Round instruction (RND)
- Exponent instruction (EXP)
- Substitute instruction (PUT) (mainly used for data transfer between general-purpose registers)
- Accumulative addition instruction (ACA)
- Accumulative subtraction instruction (ACS)
- Division instruction (DIV)

(2) Logical operation instruction

(a) Binomial logical operation

The following binomial logical operation instructions are available. For each instruction, refer to “μPD7701x Family User’s Manual Instructions.”

- And instruction (AND)
- Immediate and instruction (IAND)
- Or instruction (OR)
- Immediate or instruction (IOR)
- Exclusive or instruction (XOR)
- Immediate exclusive or instruction (IXOR)

(b) Monomial logical operation

The following monomial logical operation instruction is available. For each instruction, refer to “μPD7701x Family User’s Manual Instructions.”

- One’s complement instruction (NOT)

Caution **The number range of immediate data is 0-0xFFFF (0-65536), and set to bit 15-0. Each operation is executed with 40-bit data that 39-16 are 0 extended to this immediate 16-bit data.**

3.6.5 Operation functions of barrel shifter (BSFT)

The barrel shifter (BSFT) executes shift operations. All the shift operations are binomial operations. The BSFT outputs any shift pattern as 40-bit data in one instruction cycle in response to 40-bit input data.

As both two operands for a binomial operation, general-purpose registers can be specified, or a register can be specified as one of the operands with immediate data specified as the other. If general-purpose registers are specified as both operands, they can be in duplicate. Any general-purpose register can be specified for a monomial operation. It is also possible to specify any general-purpose register to store the result of the operation.

Caution MAC, ALU, and BSFT cannot operate simultaneously.

(1) Shift operation instruction

All the shift operations are binomial operations. The following shift operations instructions are available. For each instruction, refer to "μPD7701x Family User's Manual Instructions."

- Arithmetic right shift instruction (SRA)
- Immediate arithmetic right shift instruction (ISRA)
- Logical right shift instruction (SRL)
- Immediate logical right shift instruction (ISRL)
- Logical left instruction (SLL)
- Immediate logical left shift instruction (ISLL)

Caution The number range of general-purpose register or immediate data as shift value is 0-0x27 (0-39), and set to bit 5-0. The values of bit 15-6 are ignored.

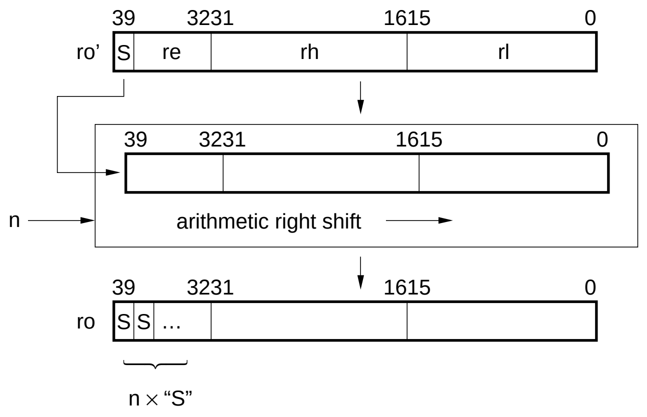
(2) Shift operation function

Figure 3-50 shows each BSFT operations.

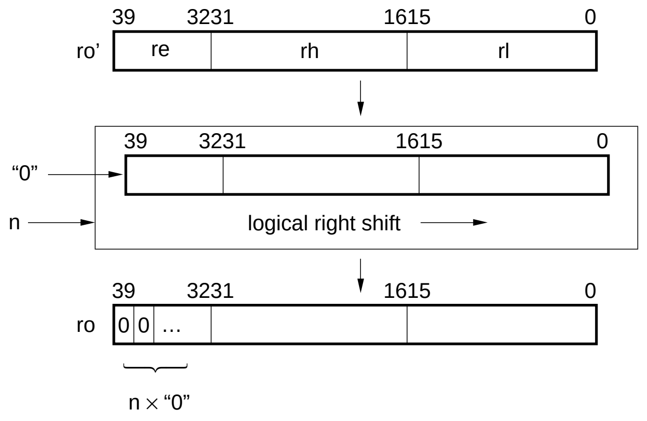
Figure 3-50. Barrel Shifter Operations

Shift number: n = 0 to 39, specified by
- immediate value
- bit 5-0 of rl part of general-purpose register

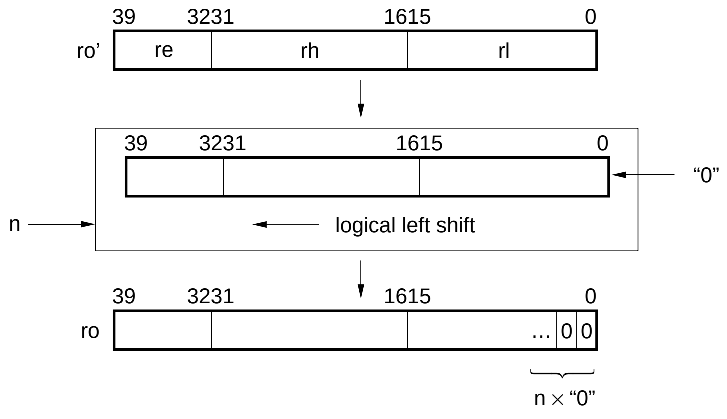
• Arithmetic right shift: ro = ro’ sra n, ro = ro’ sra rl



• Logical right shift: ro = ro’ srl n, ro = ro’ srl rl



• Logical left shift: ro = ro’ sll n, ro = ro’ sll rl



3.7 Peripheral Units

The μPD7701x family is provided with the five peripheral interface functions as listed below.

- Serial interface
- Host interface
- General-purpose I/O port
- Wait control function**Note 1**
- Debug interface**Note 2**

Notes 1. Although the wait control function is not a peripheral function in terms of the general meaning of "peripheral", it is treated in the same manner as a peripheral control function with the μPD7701x family.

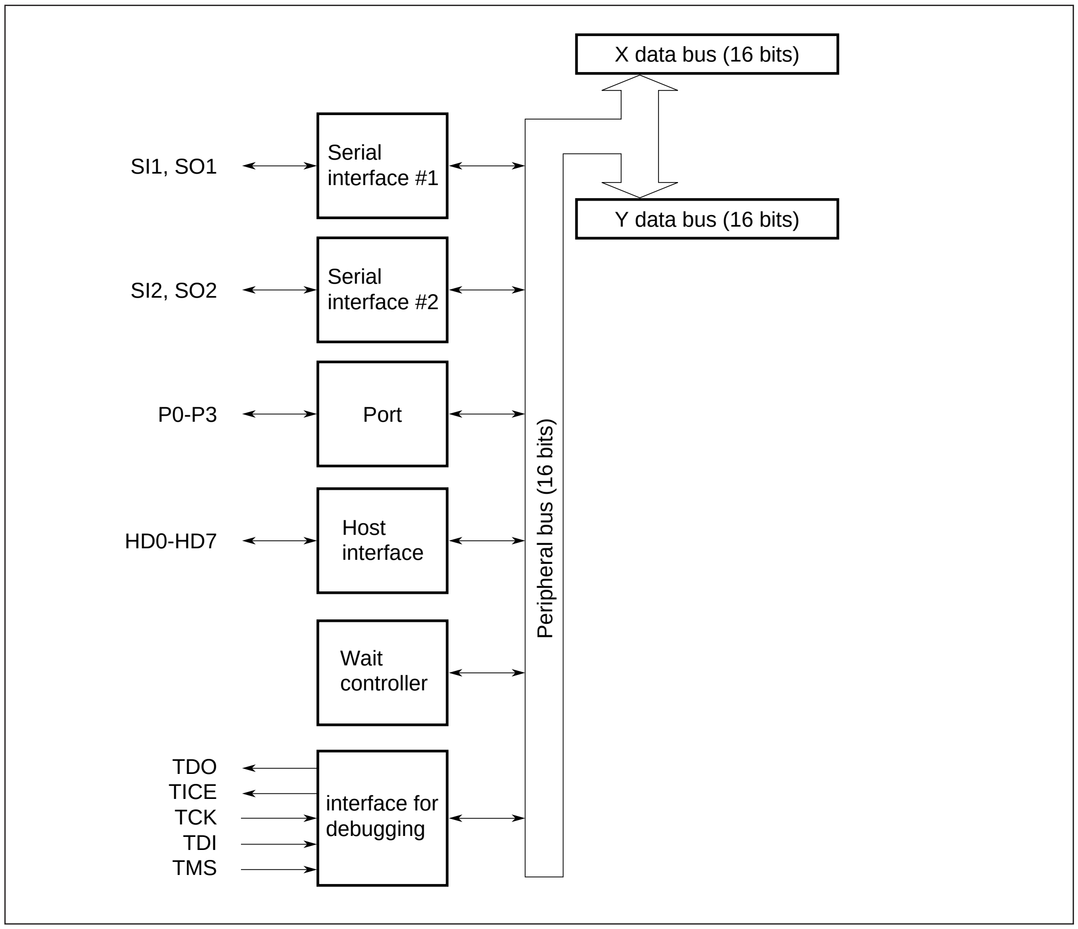
2. The debug interface cannot be used by the user program.

When handling these peripherals from the user application, access the peripheral registers mapped to the internal memory area.

3.7.1 Block configuration

Figure 3-51 shows the block configuration of the peripheral units.

Figure 3-51. Peripheral Units



3.7.2 Peripheral registers

The internal peripheral units can be used by accessing the corresponding peripheral registers mapped in the internal data memory space. Table 3-26 shows the mapping of the peripheral registers in the memory space, and the outline of each register.

Table 3-26. Memory Mapping of Peripheral Registers

X/Y memory address	Register name	Function	Peripheral name	R/W
0x3800	SDT1	Serial data register 1	SIO	R/W
0x3801	SST1	Serial status register 1	SIO	R/W
0x3802	SDT2	Serial data register 2	SIO	R/W
0x3803	SST2	Serial status register 2	SIO	R/W
0x3804	PDT	Port data register	IOP	R/W
0x3805	PCD	Port command register	IOP	R/W
0x3806	HDT	Host data register	HIO	R/W
0x3807	HST	Host status register	HIO	R/W
0x3808	DWTR	Data memory wait cycle register	WTR	R/W
0x3809	IWTR	Instruction memory wait cycle register	WTR	R/W
0x380A-0x383F	Reserved	**Do not access this area**.	—	—

Cautions 1. The register names in this table are not reserved words of the assembler or C language. When using these names with the assembler or C language, the user must explicitly define them.

2. The same register can be accessed, as long as the address is the same, from both the X and Y memory spaces.

3. Even different registers cannot be accessed from both the X and Y memory spaces at the same time.

3.7.3 Serial interface

The μPD7701x family is provided with two channels of serial interfaces, both of which are of the same structure. The main features of these serial interfaces are as follows:

- **Clock supply**

  Separate external clock for serial channels 1 and 2, common clock for serial input and output of one channel

- **Data word format**

  Serial input/output data word length 8 or 16 bits, to specify separately for input and output of each channel

  MSB-first or LSB-first data format, to specify separately for input and output of each channel

- **Internal data bus connection**

  Access of all registers via peripheral bus, connected to X and Y buses

- **Internal handshake**

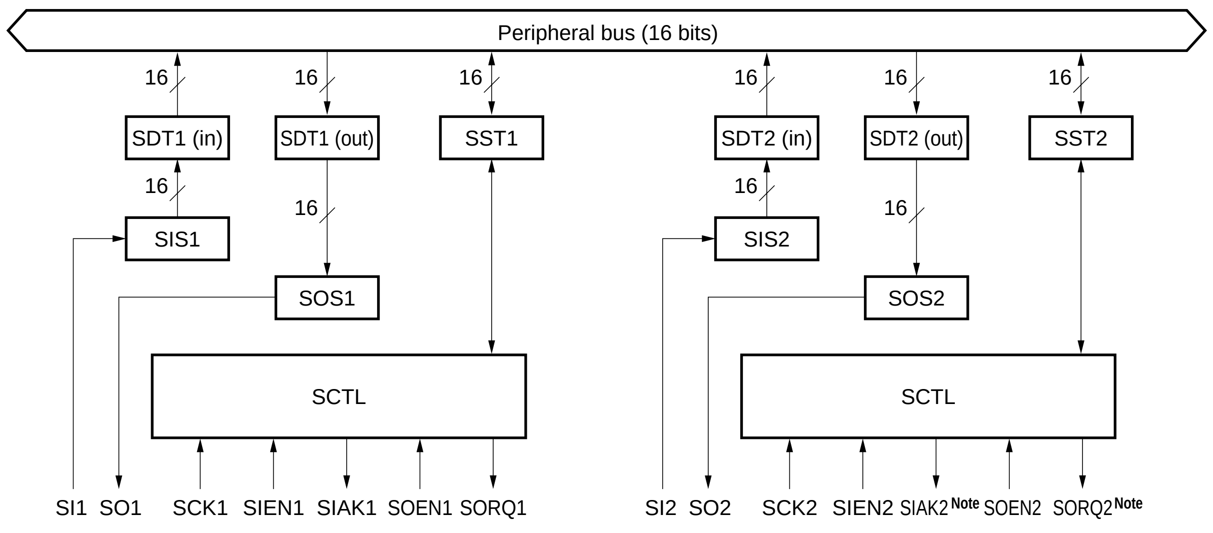
  Internal synchronization by means of polling, wait, or interrupt

- **External handshake**

  External synchronization by means of dedicated status signals

Each serial interface control circuit (SCTL) controls the pins and registers for the serial interface. Figure 3-52 shows the block diagram of the serial interface.

Figure 3-52. Serial Interface



Note Not provided to the μPD77015, 77017, 77018, 77018A, and 77019.

[Operational outline of serial interface]

This section explains the internal logical operations of the serial interface of the μPD7701x family (for detailed timing, refer to Figures 3-54 and 3-55).

To transfer data through serial interface, double buffers are provided for both input and output.

Serial input is performed by the following registers:

- SIS register (serial input shift register) : Inputs serial data from the SI pin 1 bit at a time, and outputs 16-bit parallel data to SDT (in).
- SDT (in) register (serial data input register) : Inputs 16-bit parallel data from the SIS register and outputs 16-bit parallel data to the peripheral bus.

Serial output is performed by the following registers:

- SDT (out) (serial data output register) : Writes 16-bit parallel data from the peripheral bus and outputs 16-bit parallel data to SOS.
- SOS (serial output shift register) : Inputs 16-bit parallel data from SDT (out) and outputs serial data from the SO pin 1 bit at a time.

The serial interface is accessed from an external device by using the 1-bit serial data input pin (SI) and output pin (SO).

In the μPD7701x, serial input/output is performed by using 8-/16-bit parallel input data register SDT (in) and output data register SDT (out). Because data transfer is automatically performed from SIS to SDT (in) and from SDT (out) to SOS, it does not have to be directly controlled by program.

Internal flags are provided to synchronize serial data transfer and to monitor the status of each of the dedicated external pins and registers.

- SIAK (serial input acknowledge) : This is an external pin that monitors the status of SIS.

  SIAK = high level (SIS is empty.) → Input of new serial data can be started.

  SIAK = low level (SIS is not empty.) → Valid data still exists in SIS. New serial data cannot be input.

- SORQ (serial output request) : This is an external pin that monitors the status of SOS.

  SORQ = high level (SOS is not empty.) → Data to be output still exists in SOS. (Data can be output by making SOEN high.)

  SORQ = low level (SOS is empty.) → No data to be output exists in SOS.

- SLEF flag (serial load enable flag) : This flag monitors the status of SDT (in) (this is a flag in the serial status register (SST)).

  SLEF = 1 (SDT (in) is not empty.) → Valid input data exists in SDT (in).

  SLEF = 0 (SDT (in) is empty.) → Input data that can be loaded from SDT (in) does not exist.

- SSEF flag (serial store enable flag): This flag monitors the status of SDT (out) (this is a flag in the serial status register (SST)).

SSEF = 1 (SDT (out) is empty.) → New output data can be stored to SDT (out).

SSEF = 0 (SDT (out) is not empty.) → Valid output data still exists in SDT (out).

Whether data can be actually transferred to the serial input pin (SI) and serial output pin (SO) after appropriate control signals and serial clock have been input is automatically determind by the internal hardware.

- If an attempt is made to output serial data from the SO pin when the SOS register has no data (SORQ = low level), the SO pin goes into a high-impedance state.
- If an attempt is made to input new serial data to the SIS register before the current data of the SIS register is transferred to the SDT (in) register (SIAK = low level), the new data is not written over the current data of the SIS register.

In addition to this hardware control, loading from the SDT (in) register and storing to the SDT (out) register are completely controlled in software. Correctly load or store data by <1> checking SLEF/SSEF or <2> using an interrupt, so that valid data is not written over or that the same data is not loaded or stored two times.

When successively inputting or outputting serial data, keep in mind the following points:

(1) When polling with status flag

Make sure that data transfer is not disrupted by always monitoring the status of the SLEF flag (status flag of SDT (in) register) or SSEF flag (status flag of SDT (out) register).

(2) When using serial input interrupt

If an interrupt occurs, immediately load serial data.

(3) When using serial output interrupt

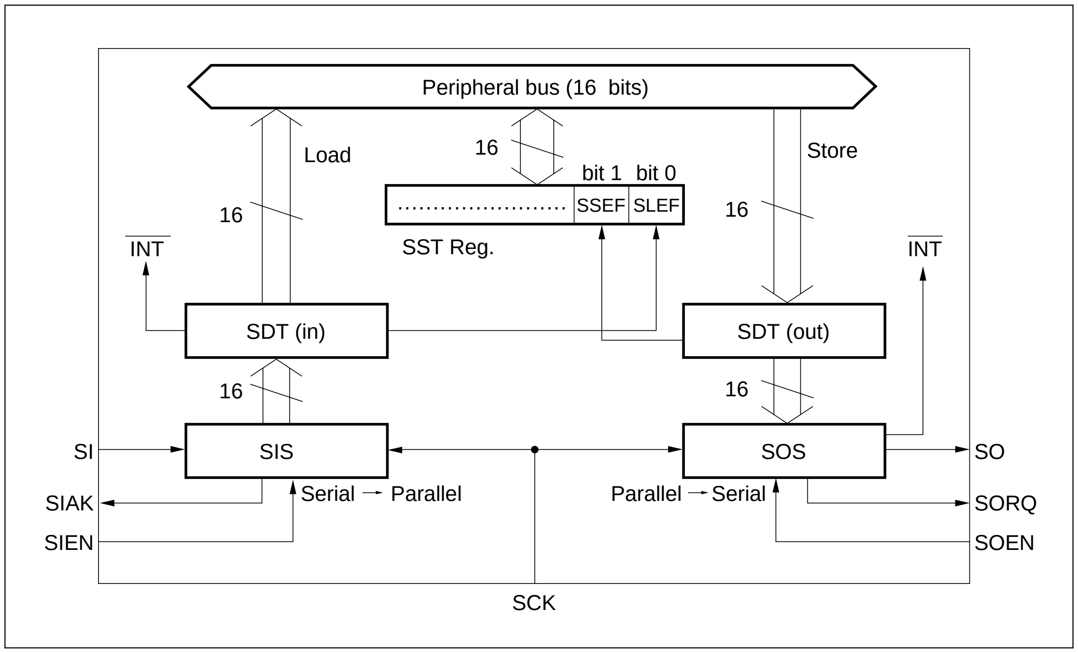
Because the interrupt cannot be used when the first data is transferred (the same applies when a single data is output), do not use an interrupt for transfer. When inputting or outputting the next data and if an interrupt occurs, immediately store the serial data. Before storing the last data, disable the interrupt so that the next interrupt does not occur.

The status of the serial input/output interface and operation block diagram are shown below.

Table 3-27. Status Indicators of Serial Input/output Interfaces

	Register	Status indicator	Status	Comments
Serial output	SIS	SIAK pin	High: empty	Serial input accepted
			Low: not empty	Serial input not accepted
	SDT(in)	SLEF flag	1: not empty	Data can be loaded from SDT(in)
			0: empty	Data cannot be loaded from SDT(in)
Serial output	SOS	SORQ pin	High: not empty	Serial output possible
			Low: empty	Serial output not possible
	SDT(out)	SSEF flag	1: empty	Data can be stored to SDT(out)
			0: not empty	Data cannot be stored to SDT(out)

Figure 3-53. Function Diagram of Serial Interface (1 channel)



(1) Pins of serial interface

The μPD7701x has two sarial interface channels.

The number suffixed to a serial interface pin indicates a channel. All the serial interface signals, except the clock and data signals, are active-high.

Remark The μPD77015, 77017, 77018, 77018A, and 77019 are not provided with the SORQ2 and SIAK2 pins.

(a) SCK1 and SCK2 (serial clock - input)

These are clock input pins for serial data input or output.

Serial data are input and output, and serial interface signals are output and sampled in synchronization with the SCK signal.

(b) SORQ1 and SORQ2 (serial output request - output)

These pins output serial data output request signals.

The output signals change its status at the rising edge of SCK.

When serial data is written to the serial data output register, these pins are asserted active (high level). When SOEN and SORQ are asserted active, serial output is started. These pins are deasserted inactive (low level) after serial output has been started.

These pins are deasserted inactive at hardware reset.

(c) SOEN1 and SOEN2 (serial output enable - input)

These pins input serial data output enable signals.

These signals are sampled at the falling edge of SCK.

They are asserted active (high level) when the external device is ready to input serial output data. When SOEN and SORQ are asserted active, serial output is started.

(d) SO1 and SO2 (serial data output - output)

These pins output serial data.

The status of the output data changes at the rising edge of SCK.

When output is completed, these pins go into a high-impedance state.

(e) SIEN1 and SIEN2 (serial input enable - input)

These pins input serial data input enable signals.

These signals are sampled at the falling edge of SCK.

They are asserted active (high level) when the external device is ready for outputting serial input data. Serial input is started when SIEN and SIAK are asserted active.

(f) SIAK1 and SIAK2 (serial input acknowledge - output)

These pins output serial data input acknowledge signals.

These signals change its status at the rising edge of SCK.

They are asserted active (high level) when serial input is ready. When SIEN and SIAK are asserted active, serial input is started. These signals are deasserted inactive (low level) after serial input has been started.

These pins are deasserted inactive at hardware reset.

(g) SI1 and SI2 (serial data input - input)

These pins input serial data.

The input data is sampled at the falling edge of SCK.

Table 3-38. Pins Status during and after Hardware Reset

Pin	I/O	During reset	After reset
SCK1, 2	I	—	—
SORQ1, 2	O	L	L
SOEN1, 2	I	—	—
SO1, 2	O	Hi-Z	Hi-Z
SIEN1, 2	I	—	—
SIAK1, 2	O	L	L
SI1, 2	O	—	—

(2) Registers of serial interface

The μPD7701x has two serial interface channels. The number suffixed to the registers of the serial interface indicates the channel number.

(a) SDT1 and SDT2 (serial data registers: 0x3800:X/:Y, 0x3802:X/:Y)

A serial data register (SDT) is a 16-bit register that inputs or outputs serial data.

A value can be input to or output from SDT by using a register-to-register transfer instruction.

When 8-bit data is input or output, the serial data is input to or output from the higher 8 bits of SDT.

<1> Serial data output register

This is a 16-bit register that sets serial data to be output.

When a store instruction is executed to SDT, data is input to this register from the peripheral bus.

Output of SO can be selected from the MSB first or LSB first.

When the serial output shift register (SOS) becomes empty. the value of this register is written to SOS.

<2> Serial data input register

This is a 16-bit register that reads serial input data.

When an instruction to load data from SDT is executed, the data of this register is output to the peripheral bus.

Whether the data is output with the MSB first or LSB first can be selected when the data is input.

When the last bit is input to the serial input shift register (SIS), the value of SIS is written to this register.

(b) SST1 and SST2 (serial status register: 0x3801:X/:Y, 0x3803:X/:Y)

The serial status register (SST) is a 16-bit register that indicates the mode setting of serial input/output and status.

This register indicates whether data is transferred with the MSB or LSB first, a bit length (16 or 8 bits), specification of interface with the μPD7701x, overrun, and underrun.

A value can be input to or output from SST by using a register-to-register transfer instruction.

The value of this register is 0x0002 at reset.

Table 3-30 shows the function of each bit of SST.

Table 3-29. Conditions of Serial Input/output Error Flags Settings

Error flag	Set condition	Reset condition
SSER	Store to SDT while SSEF = 0	By hardware reset or by program
SLER	Load from SDT while SLEF = 0	

Changing serial output mode:

The serial output mode (such as data length: 8 or 16 bits and LSB/MSB first) is determined by the setting of SST when data is stored to SDT (out).

Do not change the value of SST when SSEF = 0 (when data exists in SDT (out)).

Change the value of SST when SSEF = 1 (when SDT (out) is empty).

Changing serial input mode:

Do not change the value of SST when serial input is under execution.

If the serial successive input mode is set (SICM =), clear SICM to 0 when SLEF = 1, change the serial input mode (such as data length: 8 or 16 bits and LSB/MSB first), and then set SICM to 1 again.

The new value of SST becomes valid when data is input after the two data input to SDT (in) and SIS have been loaded.

(c) SOS1 and SOS2 (serial output shift registers)

A serial output shift register (SOS) is a 16-bit shift register that outputs and shifts serial data from SO at the rising edge of serial clock SCK.

When the specified number of bits have been output, new data is input from the serial data output register SDT (out).

(d) SIS1 and SIS2 (serial input shift registers)

The serial input shift register (SIS) is a 16-bit shift register that receives and shifts the data input from SI at the falling edge of serial clock SCK.

When the specified number of bits have been input, data is output to serial data input register SDT (in).

Table 3-30. Functions of SST (SST1:0x3801:X/:Y, SST2:0x3803:X/:Y)

Bit	Name	Load/store (L/S)	Bit function
15	SOTF	L/S	Serial output transfer format setting bit • 0: Serial output with MSB first • 1: Serial output with LSB first
14	SITF	L/S	Serial input transfer format setting bit • 0: Serial input with MSB first • 1: Serial input with LSB first
13	SOBL	L/S	Serial output word length setting bit • 0: 16-bit serial output • 1: 8-bit serial output
12	SIBL	L/S	Serial input word length setting bit • 0: 16-bit serial input • 1: 8-bit serial input
11	SSWE	L/S	SDT store wait enable bit • 0: Does not use store wait function. • 1: Uses store wait function. Inserts wait cycles when μPD7701x stores data to SDT(out) with SSEF = 0.
10	SLWE	L/S	SDT load wait enable bit • 0: Does not use load wait function. • 1: Uses load wait function. Inserts wait cycles when μPD7701x loads data from SDT(in) with SLEF = 0.
9	SICM**Note**	L/S	Serial input continuous mode setting flag • 0: Enters single serial input mode after completion of current serial input. • 1: Enters serial input continuous mode to start serial input.
8	SIEF**Note**	L/S	Single serial input enable flag • 1: Starts serial input processing in single serial input mode (only once). SIEF flag set to 1 is automatically reset in next instruction cycle.
7-4	Reserved	—	Reserved bits • Value cannot be set to these bits. • Undefined when read.
3	SSER	L/S	SDT store error flag • 0: No error • 1: Error (Set to 1 when μPD7701x stores data to SDT(out) with SSEF = 0.) • Once set, this flag does not change its status until 0 is written by μPD7701x.
2	SLER	L/S	SDT load error flag • 0: No error • 1: Error (Set to 1 when μPD7701x loads data from SDT(in) with SLEF = 0.) • Once set, this flag does not change its status until 0 is written by μPD7701x.
1	SSEF	L	SDT store enable flag • Set to 1 when contents of SDT(out) is transferred to serial output shift register. • Cleared to 0 when μPD7701x stores data to SDT(out).
0	SLEF	L	SDT load enable flag • Set to 1 when contents of serial input shift register is transferred to SDT(in). • Cleared to 0 when μPD7701x loads data from SDT(in).

Note Table 3-31 shows an example of combination of SICM and SIEF.
When continuous data such as speech data is input, use status 2 (SICM = 1, SIEF = 0).

Remark The SST setting after hardware reset
Initial SST after reset: 0x0002;
• 16 bits input/output word length
• MSB-first for input and output
• No store/load wait function
• No serial input continuous mode
• Serial input not enabled

Table 3-31. Combination of SICM and SIEF Bits

Example of combination	Bit 9 SICM	Bit 8 SIEF	Function
1	0	0	• Status transition mode. This mode is also set when serial input is not performed. • SIAK goes low. Even if this mode is set if SIAK is high, SIAK remains high until serial input is started.
2	1	0	• Continuous serial input mode. • SIAK outputs high level if serial input can be executed. After serial input has been started, SIAK goes low. Serial input is enabled again when SDT (in) is loaded, and SIAK outputs high level. If SDT(in) is empty, when a complete data word has been shifted in, the contents of SIS is transferred immediately (in synchronization with the SCK) to SDT(in) and SIAK goes high. Refer to Figure 3-55 (a).
3	0	1	• Single serial input mode. • SIAK outputs high level if serial input can be executed. After serial input has been started, SIAK goes low. SIAK remains low level even if SDT is loaded. • SIEF flag set to 1 is automatically reset in next instruction cycle. Refer to Figure 3-55 (b).
4	1	1	• The setting of this combination is prohibited.

(3) Timing of serial interface

(a) Serial output timing

Generally, serial output is performed in the following steps. Operations in steps **<1>** through **<6>** without SDT store wait cycles are illustrated in Figure 3-54 (a) and (b) for continuous and non-continuous data, respectively.

<1> The application program executes a store to SDT (serial data register).

<2> Consequently, the SDT store enable flag (SSEF) of the serial status register (SST) is cleared to 0, notifying the application program that no more data must be written to SDT. If the SDT store wait enable bit (SSWE) is set, the SDT store wait function is validated, automatically blocking a write access to SDT.

<3> If the serial output shift register (SOS) is empty, the data set to SDT is transferred to SOS after 3 SCK cycles. The serial output request pin (SORQ) becomes active (high), informing an external device of issuance of a serial output request.

<4> When the external device makes the serial output enable pin (SOEN) active (high level) **(a)**, this pin is sampled at the falling edge of the serial clock pin (SCK) immediately after**Note 1** **(b)**, at the next rising edge of SCK, SORQ becomes low **(c)** and data output to the serial data output pin (SO) is started **(d)**.

<5> After SDT has become empty, SSEF is set to 1, notifying the application program that the next data can be written **(a)**, the SDT store wait function, which has been validated with SSWE = 1, is invalidated. At this time, an interrupt request is generated by SO **(b)**. However, the interrupt is serviced as a valid interrupt or is recorded, depending on the status of the corresponding interrupt enable flag and EI status (refer to 3.4.4 “Interrupt”).

<6> If the next data is not supplied when the output of the last bit data has been completed, SO goes into a high-impedance state at the next rising edge of the SCK**Note 2**.

Notes **1.** Before SOEN becomes active, SCK must rise at least three times.

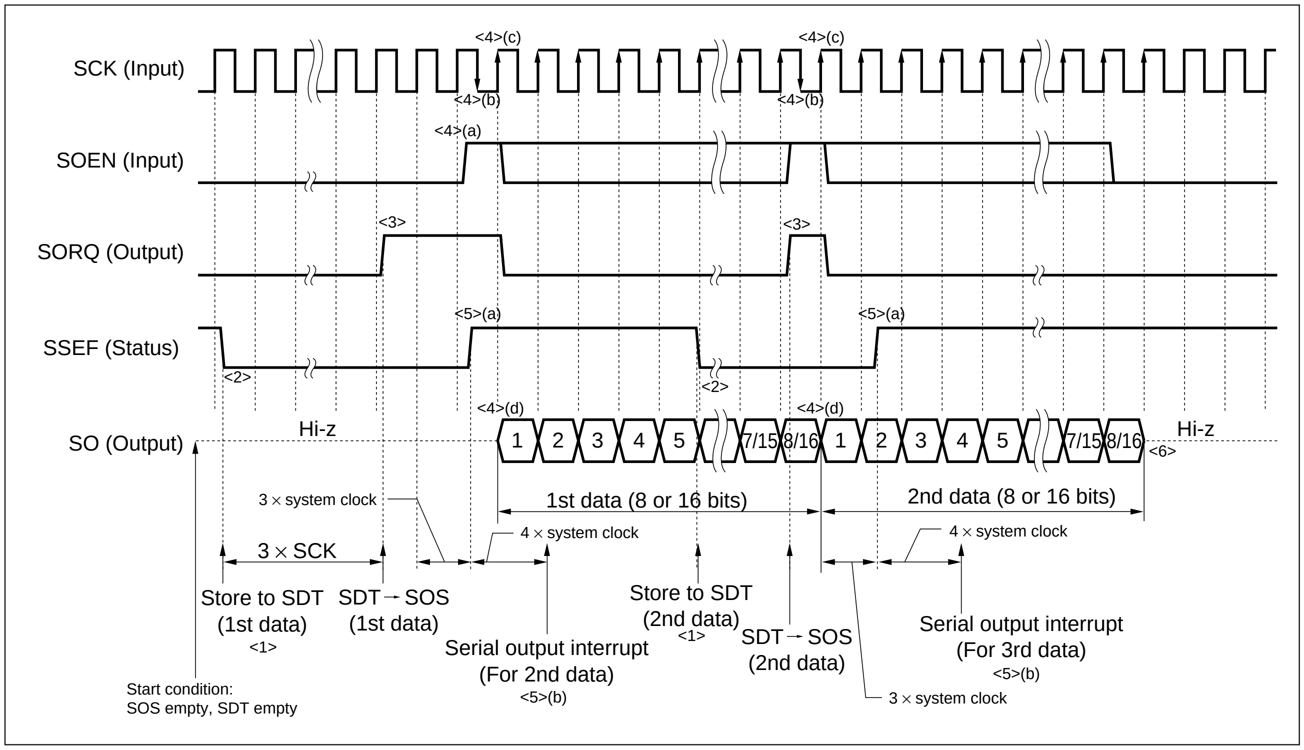
Bear this in mind especially in a system configuration where the clock is used in burst mode for only inputting/outputting data.

2. Under the following conditions, SO does not go into a high-impedance state but successively outputs the next data:
if the next data has been already supplied before the last bit is output, and if SOEN becomes active before falling of SCK in the last bit output cycle and is sampled as valid (refer to Figure 3-54 (a)).

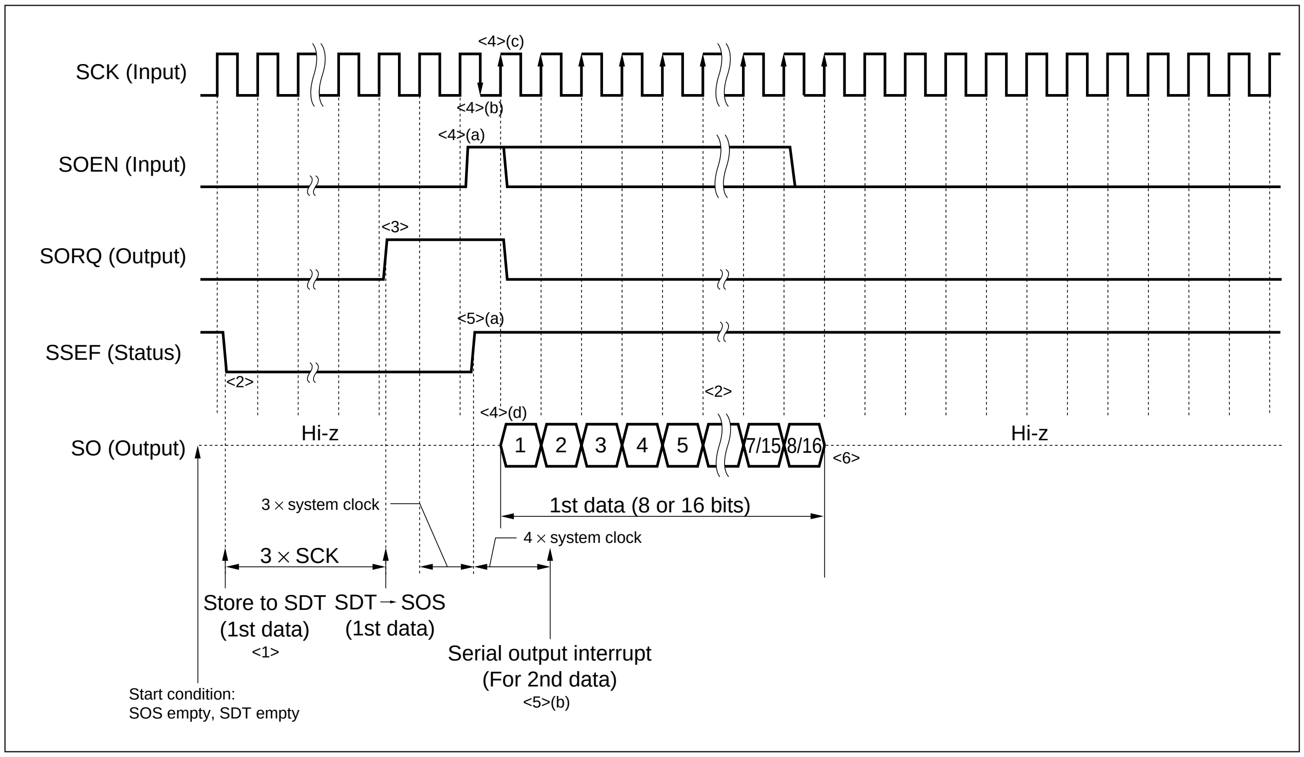
After the last bit has been output, the rising edge of SCK must be supplied at least once.

Figure 3-54. Serial Interface Output timing

(a) Continuous data



(b) Non-continuous data



(b) Timing of serial input

Generally, serial input is performed in the following steps. Operations in steps **<1>** through **<4>** without SDT load wait cycles are illustrated in Figure 3-55 (a) and (b). Figure 3-55 (a) and (b) show operations of steps **<1>** through **<4>** for input mode of continuous and single, respectively.

<1> Serial data input sequence is started when an external device makes the serial input enable pin (SIEN) active (high level) with the serial input enable (SIAK) pin being active (high level).

<2> Changes in SIEN in <1> are sampled at the falling edge of SCK immediately after**Note 1** **(a)**, SIAK goes low at the next rising edge of SCK **(b)**, and inputting data given to the serial data input pin (SI) is started from the falling edge of the same SCK cycle **(c)**. The data is loaded from the SI pin to the serial input shift register (SIS) bit by bit in synchronization with the falling edge of SCK.

<3> SIAK becomes active **(a)** in synchronization with the rising edge of the SCK cycle, in which the last bit of the specified number of bits is loaded, immediately before the loading, informing the external device that the next data can be input. When the last bit has been loaded**Note 2** **(b)** and if the SDT load enable flag (SLEF) is 0, the loaded bit is immediately transferred from SIS to SDT**Note 3**.

After that, SLEF changes to 1, informing the application program that the serial input data word has been completed **(c)**. If the data wait status is set with the SDT load wait enable bit (SLWE) set to 1, the wait function is released. Although an interrupt request is generated by SI **(d)** at this time, the interrupt is serviced as a valid interrupt or is recorded, depending on the statuses of the corresponding interrupt enable flags and the EI bit (refer to 3.4.4 “Interrupt”).

<4> When the application program executes a load from SDT **(a)**, SLEF is cleared to 0, indicating that the input data is empty **(b)**. If SLWE is 1 at this time, the SDT load wait function is validated, automatically blocking further read access to SDT.

Notes 1. Before SIEN becomes active, SCK must rise at least three times. The hardware of the serial input/output block performs a pipeline operation with SCK used as a timing clock.

Bear this in mind especially in a system configuration where the clock is used in burst mode for only inputting/outputting data.

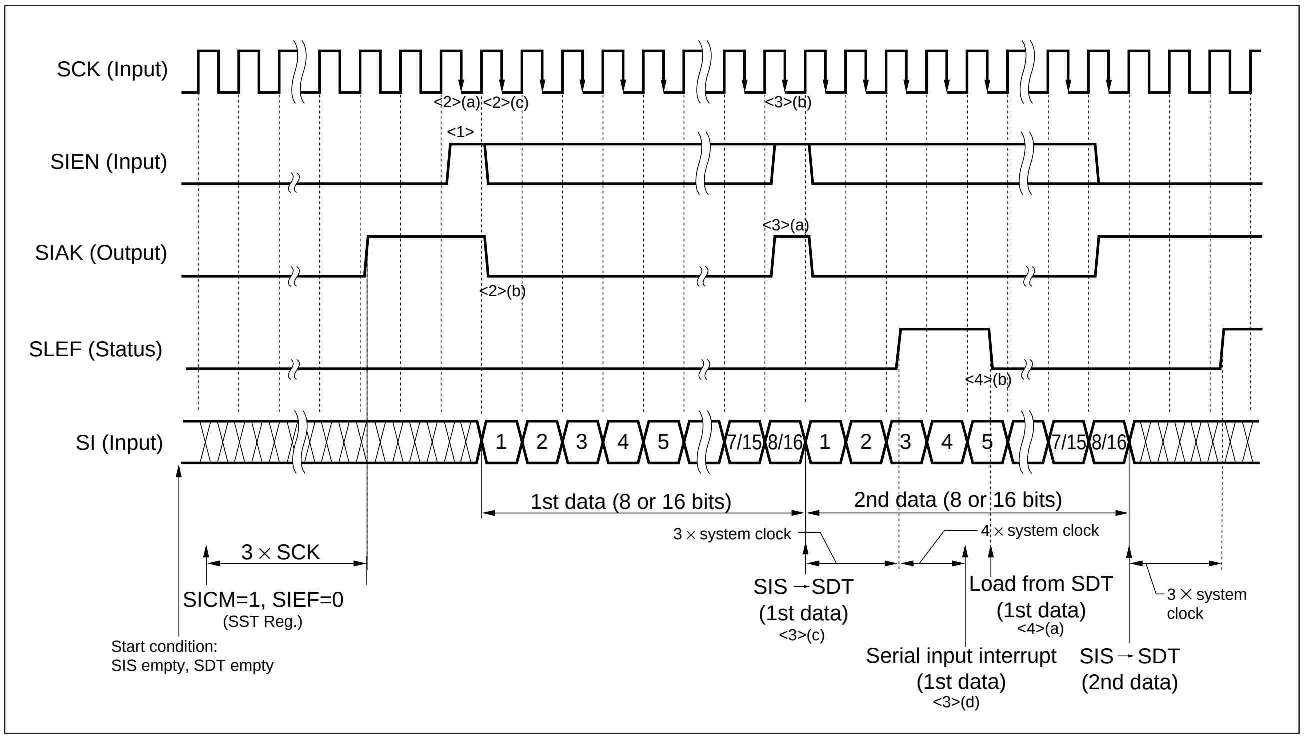
With a system configuration where the clock is successively supplied, exercise care in respect to the first data after reset.

After the last bit has been output, the rising edge of SCK must be supplied at least two times.

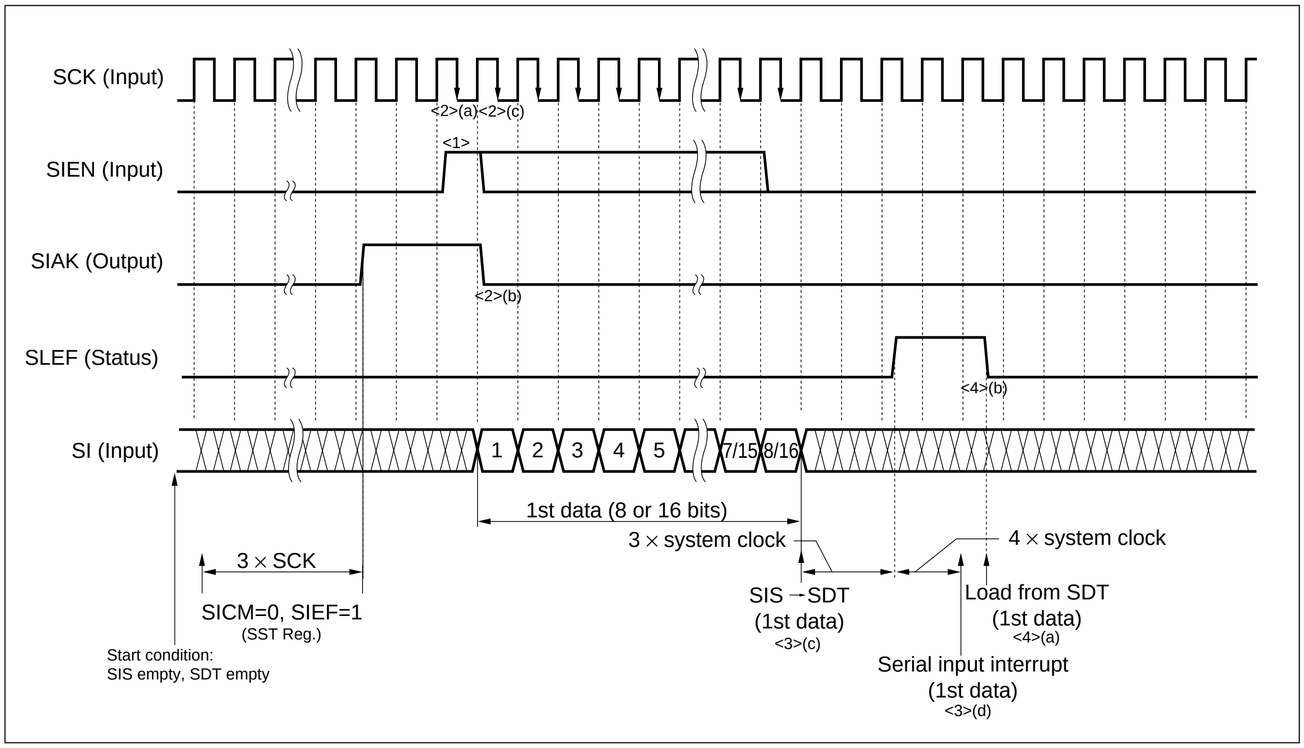
2. If SIEN becomes active and sampled as valid before SCK falls in the last bit input cycle, the next data is loaded from the successive next SCK cycle (refer to Figure 3-55).

3. SDTs are used separately for serial input and serial output.

Figure 3-55. Serial Interface Input timing
(a) SICM = 1, SIEF = 0; Continuous mode



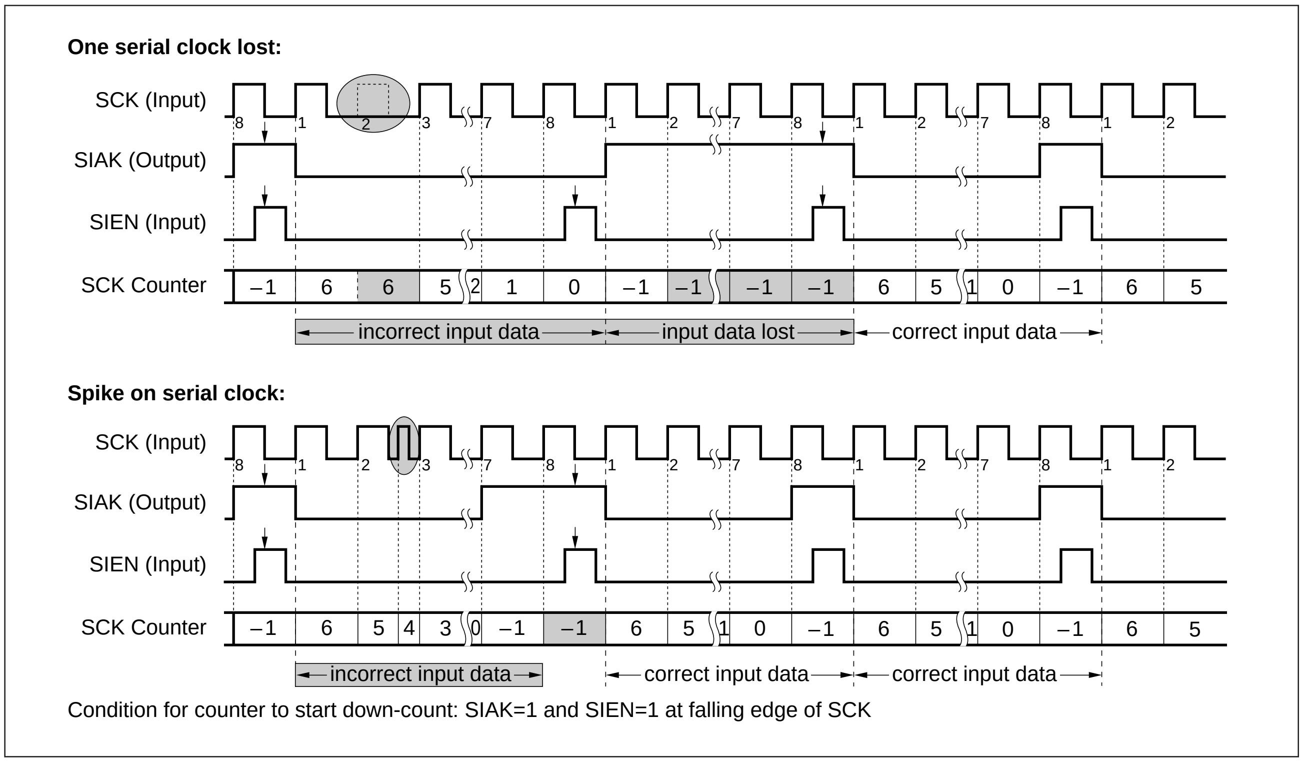
(b) SICM = 0, SIEF = 1; Single mode



(c) I/O timing of non-standard serial clock

Figure 3-56 shows the operation of the serial clock counter which are caused by non-standard serial clock.

Figure 3-56. Serial Interfaces - Operation of the Serial Clock Counter



★ Data can be input/output even when SIEN and SOEN are active. If a bit shift occurs, however, the I/O timing cannot be corrected because a non-standard serial clock is input. By deasserting SIEN and SOEN inactive, this bit shift can be corrected as shown above. In the above example, SIEN is input by counting SCK. However, it is more accurate if SIEN is input depending on the status of SIAK.

(4) Handshake

There are three means to handshake with the serial interface of the μPD7701x family, which can be implemented by application programs:

- Polling
- Wait
- Interrupt

Each format is described next.

(a) Polling

Synchronization of handshaking is established by always monitoring and evaluating the SDT store enable flag (SSEF) and SDT load enable flag (SLEF) of the serial status register (SST). Here is an example of serial output by means of polling:

```
/* Explicitly define SST1 and SO1 because they are not reserved
words. */
      #define SST1 0x3801
      #define SO1  0x3800

/*Disable internal interrupts SO1 and SI1.*/
      R0L = SR            ;
      R0 = R1 | 0x0030    ;
      SR = R0L            ;

      R0L = 0x0           ; Set serial status as follows:
      *SST1:X = R0L       ; • MSB first output
                          ; • MSB first input
                          ; • 16-bit word output
                          ; • 16-bit word input
                          ; • SDT store wait function is not used.
                          ; • SDT load wait function is not used.
                          ; • Serial input is not performed.
                          ; • Clear serial input/output error
                          ;   flag.

      POLL: R0L = *SST1:X ; Judge SSEF and loop to wait until
                          ; store is enabled.
      R0 = R0 & 0x2       ;
      if (R0 == 0)jmp POLL;

      *SO1:X = R1H        ; Data of R1H is output because store is
                            enabled.
```

(b) Wait

Under the following conditions, execution of data exchanges with the SDT(in) and/or SDT(out) registers cause instruction wait cycles:

- when the store wait function is enabled (SSWE = 1) and a store to SDT(out) for serial output is to be executed, while SSEF = 0 (valid data exists in SDT(out)).
- when the load wait function is enabled (SLWE = 1) and a load from SDT(in) for serial input is to be executed, while SLEF = 0 (valid data does not exist in SDT(in)).

The advantage of this format is that describing handshake procedures in the application program is not needed, because the handshake procedure is automatically executed by hardware. Here is an example of serial output by using the wait function:

```
/* Explicitly define SST1 and SO1 because they are not
reserved words. */
   #define SST1 0x3801
   #define SO1  0x3800

/* Disable internal interrupts SO1 and SI1. */
   R0L = SR            ;
   R0 = R0 | 0x0030    ;
   SR = R0L            ;

   R0L = 0x800         ; Set serial interface as follows:
   *SST1:X = R0L       ; • MSB first output
                       ; • MSB first input
                       ; • 16-bit word output
                       ; • 16-bit word input
                       ; • SDT store wait function is used.
                       ; • SDT load wait function is not used.
                       ; • Serial input is not performed.
                       ; • Clear serial input/output error flag.

   *SO1:X = R1H        ; Data of R1H is output as soon as
                       ; SSEF = 1.
```

Caution **When data is written from the application program to SDT, the wait is not released unless SDT is transferred to SOS (i.e., unless all the bits of the previous data of SOS are shifted out to the external device). If internal writing of DSP and external reading do not correspond on a one-to-one basis vis-a-vis SDT, a hang-up may occur.**

During wait, interrupts are delayed (refer to secton 3.4.4 “Interrupt”).

(c) Interrupt

Handshaking is established by interrupts, if data can be stored to SDT (out) and data can be loaded from SDT (in). Therefore, the advantage of this format is that, even while other processing is under execution, serial input/output can be executed independently (asynchronously) of the processing. Here is an example of serial input/output using an interrupt:

```
/* definition of serial I/O register names            */
#define SST1     *0x3801:X
#define SI1      *0x3800:X
#define SO1      *0x3800:X

/* interrupt vector table entries                     */

SegSI1   IMSEG AT 0x220   ; sio#1 input interrupt routine

         R0H = SI1        : load from SDT (in)
         R0 = R0H*R1H     ;
         *DP0++ = R0H     ; save to buffer
         RETI             ; return from interrupt

SegSO1   IMSEG AT 0x224   ; sio#1 output interrupt routine

         R0H = *DP4++     ; read from buffer
         SO1 = R0H        ; save to SDT(out)
         RETI             ; return from interrupt
         NOP              ;

/* disable interrupts to initialize serial input/output */

         R1L = EIR        ; disable interrupts generally
         R1 = R1 | 0x8000 ; EI = 1
         EIR = R1L        ;
         NOP              ; wait 2 cycles until
         NOP              ; EI = 1 effective

         R0L = SR         ; enable SI1 and SO1 interrupts
         R0 = R0 & 0xFFCF ;
         SR = R0L         ;

         R1 = R1 & 0x7FFF ; enable interrupts generally
         EIR = R1L        ;
         FINT             ; discard all pervious interrupts

                          ; initialize SST1
         R0L = 0x0200     ; input/output: MSB-first, 16-bit
                          ; no load/store wait function
         SST1 = R0L       ; serial input continuous mode
         SO1 = R0L        ; dummy store (see below)
```

Cautions **Note the following points when executing serial output interrupt because the interrupt occurs after data has been transferred from the SDT register to the serial output shift register:**

(1) Transfer first dummy data and then forcibly generate an interrupt, or do not use interrupts during the transfer of the first data.

(2) When transferring data in burst mode, first disable interrupts immediately before the instruction which transfers the last word to SDT, then execute the instruction introduced in (1) after the completion of the next burst data preparation, to transfer the next burst data. This is because the first word of data to be burst-transferred may not be completely prepared if an interrupt is generated during the transfer of the last burst data word.

Example:

```
                    ; /* When last word is stored to SDT. */
R0L = SR            ; /* (DI status during interrupt processing) */
R0 = R0 | 0x0020    ; /* SO1 interrupt is disabled. */
SR = R0L            ;
*SO1: X = R0H       ;
```

3.7.4 Host interface

The μPD7701x family is provided with a host interface that transfers data with an external host CPU and DMA controller. The features of this host interface are as follows:

- 8-bit parallel port
- Data range

  Higher 8 or lower 8 bits are selected by address.
- Internal data bus connection

  Connected to X and Y buses.
- Internal 16-bit, external 8-bit configuration

  External device is interfaced through 8-bit data bus.
- Internal handshake

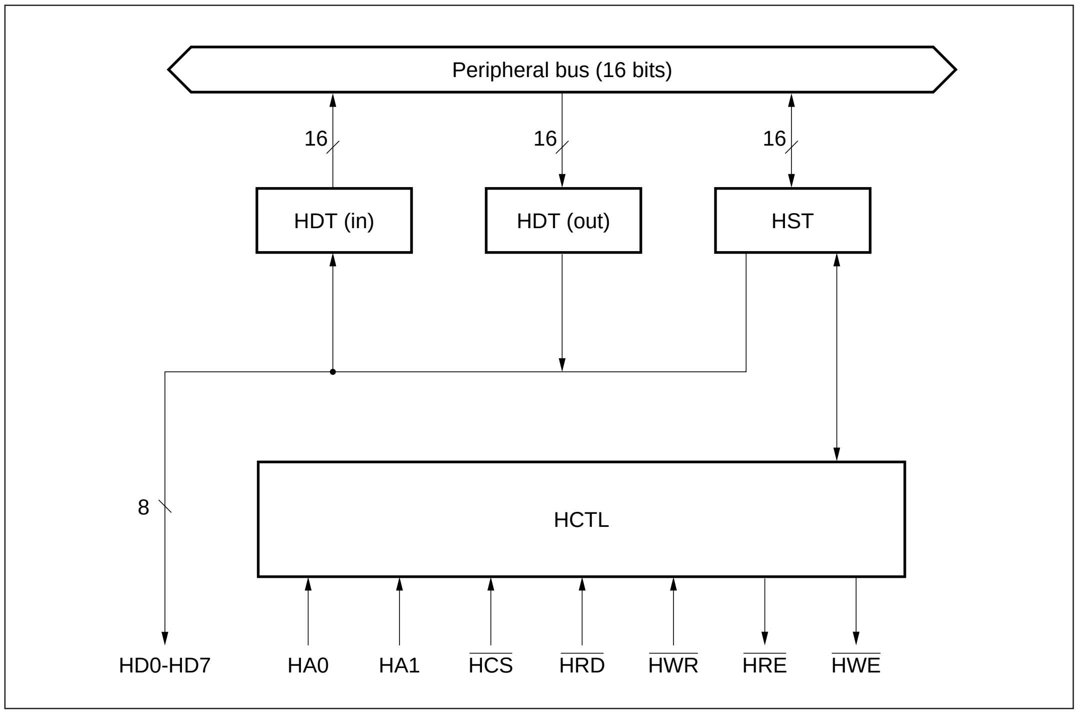
  Handshake by means of polling, wait, or interrupt
- External handshake

  Handshake by means of dedicated status signal.

A host interface control circuit (HCTL) controls the pins and registers.

Figure 3-57 shows the block diagram of the host interface.

Figure 3-57. Host Interface



[Operational outline of host interface]

This section explains the internal logical operation of the host interface of the μPD7701x family (for the detailed timing, refer to Figures 3-59 and 3-60).

One buffer stage is provided for both input and output to transfer data via host interface.

Host input is performed by using the following registers:

- HDT (in) register (host data input register):

  Inputs 8-bit parallel data (higher byte and lower byte) from the HD0 through HD7 pins, and output 16-bit parallel data to the peripheral bus.

Host output is performed by using the following registers:

- HDT (out) register (host data output register):

  Writes 16-bit parallel data from the peripheral bus and outputs 8-bit parallel data (higher byte and lower byte) from the HD0 through HD7 pins.

The host interface can be accessed from the external device by using 8 bits of host data I/O pins. Internally, the interface can be accessed by using the parallel input register HDT (in) and output register HDT (out).

To establish synchronization for host data transfer, the following internal flags are provided to monitor the status of the dedicated external pins and registers.

- $\overline{\text{HWE}}$ (host write enable), HWEF (host write enable flag):

  These are an external pin and a flag (flag of the host status register) that monitor the status of HDT (in).

  $\overline{\text{HWE}}$ = high level, HWEF = 0 (HDT (in) is not empty.) $\rightarrow$

  Valid data still exists in HDT (in). The host cannot write new data to HDT (in).

  The μPD7701x can load data from HDT (in).

  $\overline{\text{HWE}}$ = low level, HWEF = 1 (HDT (in) is empty.) $\rightarrow$

  The host can write new data to HDT (in).

  The μPD7701x cannot load data from HDT (in).

- $\overline{\text{HRE}}$ (host read enable), HREF (host read enable flag):

  These are an external pin and a flag (flag of the host status register) that monitors the status of HDT (out).

  $\overline{\text{HRE}}$ = high level, HREF = 0 (HDT (out) is empty.) $\rightarrow$

  Valid data does not exist in HDT (out).

  The host can read data from HDT (out).

  The μPD7701x can store output data to the HDT (out).

  $\overline{\text{HRE}}$ = low level, HREF = 1 (HDT (out) is not empty.) $\rightarrow$

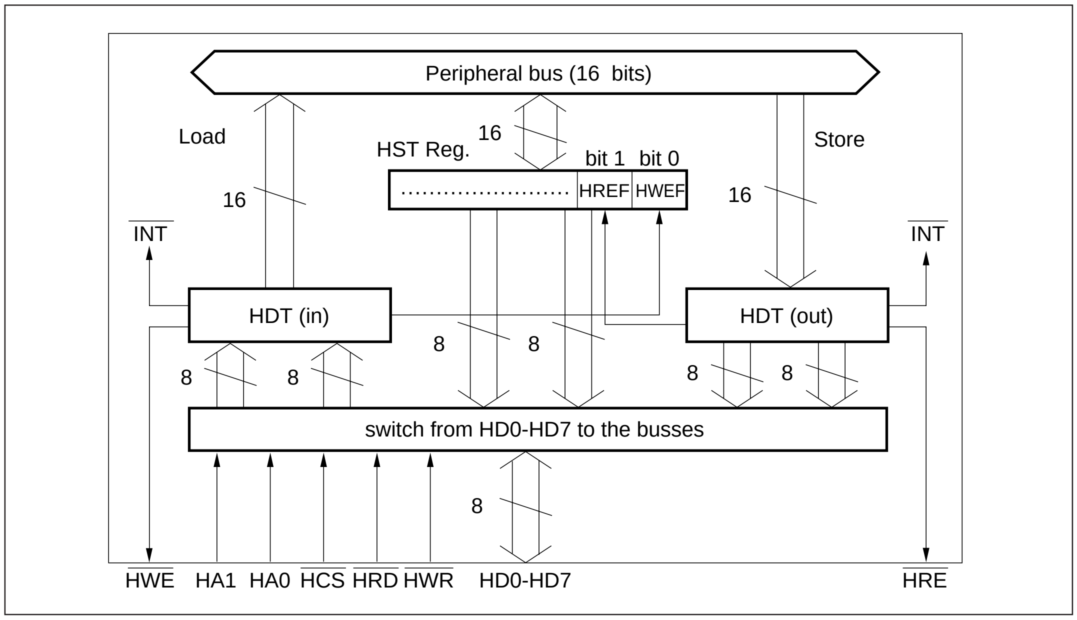
  The host can read new data from HDT (out).

  The μPD7701x can store output data to HDT (out).

Table 3-32. Status Indicators of Host Read/write Interface

Register	Status indicator	Status	Comments
Host write HDT(in)	$\overline{\text{HWE}}$ pin	High: Not empty	Host cannot write to HDT(in)
		Low: Empty	Host can write to HDT(in)
	HWEF flag	1: Empty	Data cannot be loaded from HDT (in)
		0: Not empty	Data can be loaded from HDT(in)
Host read HDT(out)	$\overline{\text{HRE}}$ pin	High: Empty	No new data in HDT(out) to be read by host
		Low: Not empty	Host can read data from HDT(out)
	HREF flag	1: Not empty	Data cannot be stored to HDT(out)
		0: Empty	Data can be stored to HDT(out)

Figure 3-58. Function Diagram of Host Interface



(1) Pins of host interface

All control pins of the host interface are active-low.

(a) $\overline{\text{HCS}}$ (Host Chip Select - input)

This pin inputs a host interface select signal.

This signal is active (low) while the host CPU accesses a register of the host interface.

(b) HA1 and HA0 (Host Addresses 1 and 0 - input)

These pins input an address of the host interface.

They specify a register of the host interface to be accessed.

Do not change the statuses of these pins while the host CPU is accessing a register of the host interface.

(c) $\overline{\text{HRD}}$ (Host Read strobe - input)

This pin inputs the read strobe signal of the host interface.

It becomes active (low) when the host CPU reads the data of a register of the host interface. This signal must not be active concurrently with the $\overline{\text{HWR}}$ signal.

(d) $\overline{\text{HWR}}$ (Host Write strobe - input)

This pin inputs the write strobe signal of the host interface.

It becomes active (low level) when the host CPU writes data to a register of the host interface. This signal must not be active concurrently with the $\overline{\text{HRD}}$ signal.

(e) HD0-HD7 (Host Data 0-7 - input/output)

These pins input or output data to or from the host interface.

Data is input or output when the host CPU accesses a register of the host interface. These pins go into a high-impedance state when $\overline{\text{HCS}}$ is inactive (high).

(f) $\overline{\text{HRE}}$ (Host Read Enable - output)

This pin outputs a signal indicating that HDT is enabled to be read.

It is asserted active (low level) if HDT is enabled to be read and is deasserted inactive (high level) at the falling edge of the $\overline{\text{HRD}}$ pin when the higher byte of the data of HDT is read. This pin remains unchanged even if the lower byte of HDT is accessed.

This pin is deasserted inactive at hardware reset.

(g) $\overline{\text{HWE}}$ (Host Write Enable - output)

This pin outputs a signal indicating that HDT is enabled to be written.

It is asserted active (low level) if HDT is enabled to be written and is deasserted inactive (high level) at the falling edge of the $\overline{\text{HWR}}$ pin when data is written to the higher byte of HDT. This pin remains unchanged even if the data of the lower byte of HDT is accessed.

This pin is deasserted inactive at hardware reset.

Table 3-33. Pins Status during and after Hardware Reset

Pin	I/O	During reset	Initial after reset
$\overline{\text{HCS}}$	I	—	—
HA0, HA1	I	—	—
$\overline{\text{HRD}}$	I	—	—
$\overline{\text{HWR}}$	I	—	—
★ HD0-HD7	I/O	Hi-Z (when $\overline{\text{HCS}}$ pin is inactive)	
$\overline{\text{HRE}}$	O	H	H
$\overline{\text{HWE}}$	O	H	H

(2) Registers of host interface

(a) Host data register (HDT-0x3806:X/:Y)

This 16-bit register is used to input or output data to or from the host interface. Data can be stored to and loaded from HDT by use of load/store instructions.

<1> Host data output register

This 16-bit register sets data to be output from the host interface.

When a store to HDT is executed, data is input to this register through the peripheral bus.

When data is read by an external device, the higher or lower 8 bits are specified by HA0.

<2> Host data input register

This 16-bit register sets the data to be input from the host interface.

When a load from HDT is executed, the data of this register is output to the peripheral bus.

When data is written by an external device, the higher or lower 8 bits are specified by HA0.

(b) Host interface status register (HST-0x3807:X/:Y)

Host interface status register HST is a 16-bit register that indicates the mode setting and status of the host interface.

It indicates the specification, and write or read error between the host CPU and host interface and between the host interface and μPD7701x.

Data can be input to or output from HST by using a load/store instruction.

When the value of this register is read by the external device, the higher 8 bits or lower 8 bits are specified by HA0.

The value of HST is set to 0x0301 at reset.

Table 3-34 shows the function of each bit of HST, and Table 3-35 shows the set condition of the host I/O error flags.

Table 3-34. Function of HST (0x3807:X/:Y)

Bit	Name	R/W from host	Load/Store from μPD7701×	Bit function
15-11	Reserved	—	_	Reserved bits • No value can be set to these bits. • These bits are undefined when read.
10	HAWE	R	Load/Store	HDT access wait enable bit • 0: Wait is not used • 1: Wait is used Wait cycles are inserted if the μPD7701x attempts to store data to HDT (out) while HREF=1, or to load data from HDT (in) while HWE=1.
9	HREM	R	Load/Store	HRE mask bit • 0: Does not mask. $\overline{\text{HRE}}$ changes according to the HREF status (refer to below). • 1: Masks $\overline{\text{HRE}}$ becomes inactive (high level).
8	HWEM	R	Load/Store	HWE mask bit • 0: Does not mask. $\overline{\text{HWE}}$ changes according to the HWEF status (refer to below). • 1: Masks $\overline{\text{HWE}}$ becomes inactive (high level).
7	UF1	R	Load/Store	User’s flag
6	UF0	R	Load/Store	User’s flag
5	HRER	R	Load/Store	Host read error flag • 0: No error • 1: Error Set to 1 when host CPU reads HDT when HREF is 0. • Once set to 1, it does not change until 0 is written by program.
4	HWER	R	Load/Store	Host write error flag • 0: No error • 1: Error Set to 1 when host CPU writes HDT when HWER is 0. • Once set to 1, it does not change until 0 is written by program.
3	HSER	R	Load/Store	HDT store error flag • 0: No error • 1: Error Set to 1 when μPD7701x store to HDT when HREF is 1. • Once set to 1, it does not change until 0 is written by program.
2	HLER	R	Load/Store	HDT load error flag • 0: No error • 1: Error Set to 1 when μPD7701x loads from HDT when HWEF is 1. • Once set to 1, it does not change until 0 is written by program.
1	HREF	R	Load	Host read enable flag • 0: Read disabled • 1: Read enabled Set to 1 when the μPD7701x stores data to HDT. Cleared to 0 when host CPU reads higher byte of HDT. • Ignored when written.
0	HWEF	R	Load	Host write enable flag • 0: Write disabled • 1: Write enabled Set to 1 when the μPD7701x loads data to HDT. Cleared to 0 when host CPU writes higher byte of host CPU. • Ignored when written.

Remark The HST setting after hardware reset:
initial HST after reset: 0x0301:
- no wait function
- HRE/HWE mask: Masked
- host write enabled
- host read disabled

Table 3-35. Conditions of Host Input/Output Error Flags Settings

Error flag	Set condition	Reset condition
HRER	Host read while HREF=0	by hardware reset or by program
HWER	Host write while HWEF=0	
HSER	Store to HDT while HREF=1	
HLER	Load from HDT while HWEF=1	

(3) Registers of host interface when viewed from host

The host CPU specifies the higher or lower bytes of either the host status register HST or host data register HDT by use of the HA0 and HA1 inputs. Table 3-36 shows the registers of the host interface when they are accessed by an external device.

Table 3-36. Selecting Host Interface Registers

$\overline{\text{HCS}}$	$\overline{\text{HRD}}$	$\overline{\text{HWR}}$	HA1	HA0	Register subject to transfer	Byte
0	0	0	x	x	Disabled	—
0	0	1	0	0	HDT (output)	Lower 8 bits
0	0	1	0	1	HDT (output)	Higher 8 bits
0	0	1	1	0	HST	Lower 8 bits
0	0	1	1	1	HST	Higher 8 bits
0	1	0	0	0	HDT (input)	Lower 8 bits
0	1	0	0	1	HDT (input)	Higher 8 bits
0	1	0	1	x	Disabled	—
0	1	1	x	x	No register	—
1	x	x	x	x	No register	—

(4) Timing of host interface

(a) Host read operation (μPD7701x —> host)

Data is transferred from the μPD7701x to the host in the following steps. Figure 3-59 shows reading operations of 16-bit data to HDT without wait cycles.

<1> The application program of the μPD7701x stores data to the host data register (HDT) **(a)**, **(b)**.

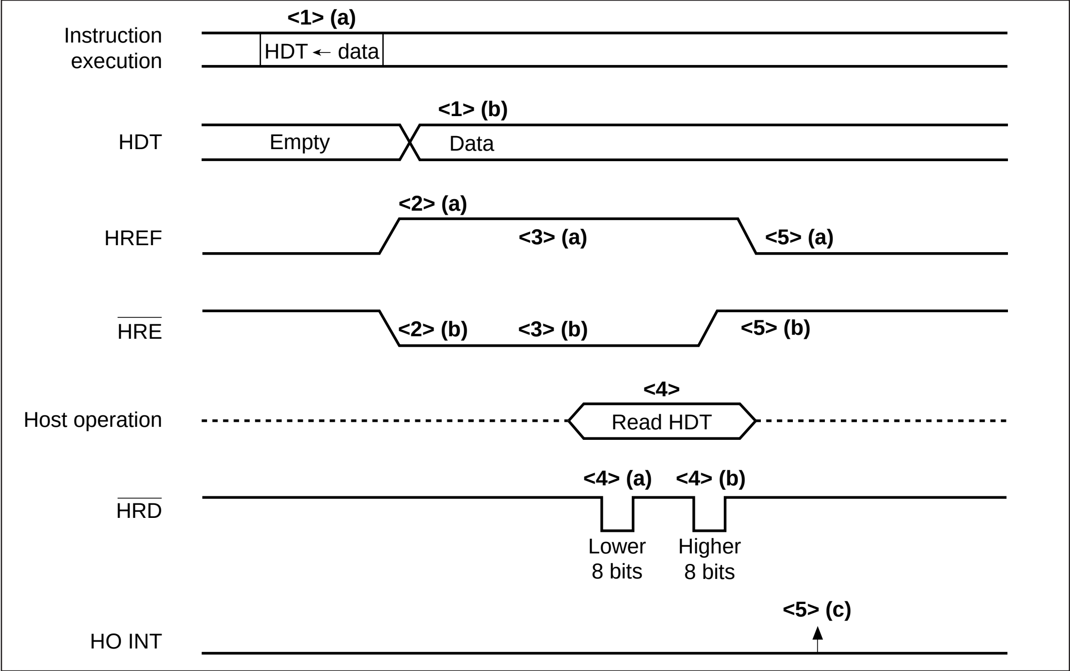
<2> Consequently, the host read enable flag (HREF) of the host interface status register (HST) is set to 1 **(a)**. If the $\overline{\text{HRE}}$ mask bit (HREM) of HST is 0, the $\overline{\text{HRE}}$ pin becomes active (low), and is output to external devices as a hardware signal **(b)**.

<3> The host can recognize that data is present in HDT by any of the following methods:
(1) Reads HST and detects HREF = 1 by software **(a)**, or
(2) Detects the low level of the $\overline{\text{HRE}}$ pin **(b)**.

<4> The host reads HDT. If 16-bit data is transferred at this time, the lower 8 bits **(a)** and then the higher 8 bits **(b)** must be read in this order. If 8-bit data is transferred, the higher 8 bits are always read (refer to the logic of HREF and $\overline{\text{HRE}}$).

<5> HREF of HST is cleared to 0 after step **<4> (b)**, and the $\overline{\text{HRE}}$ pin becomes inactive (high) in step **<4> (b)**. At this time, an interrupt request is generated by HO **(c)**. This interrupt is processed as a valid interrupt or is recorded depending on the status of the corresponding interrupt enable flag and EI status (refer to 3.4.4 “Interrupt”).

Figure 3-59. Host Read Sequence (μPD7701x —> host): HDT read without wait



(b) Host write operation (μPD7701x <— host)

Data is transferred from the host to the μPD7701x in the following steps. Figure 3-60 shows examples of writing HDT without wait cycles when 16-bit data is transferred.

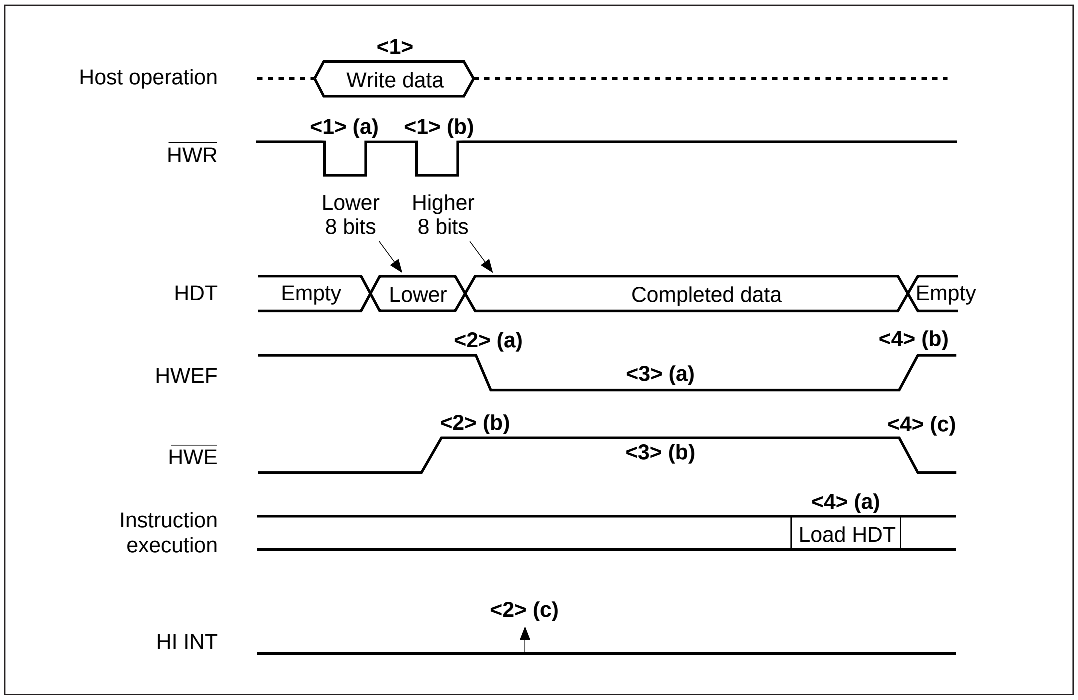
<1> The host writes data to the HDT of the μPD7701x. If 16-bit data is transferred at this time, the lower 8 bits **(a)** and the higher 8 bits **(b)** are written in this order; if 8-bit data is transferred, data is always written to the higher 8 bits of HDT (refer to the logic of HWEF and $\overline{\text{HWE}}$).

<2> Consequently, HWEF of HST is cleared to 0, informing the application program of the μPD7701x that data has been written to HDT **(a)**. The $\overline{\text{HWE}}$ pin becomes inactive (high level in step **(b)**), informing an external device that HDT is busy **(b)**. An interrupt request is also generated by HI **(c)**. Whether this interrupt is processed as a valid interrupt or is recorded depends on the status of the corresponding interrupt request flag or EI status (refer to 3.4.4 "Interrupt").

<3> The application program of the μPD7701x can recognize that HDT is ready with the data from the host by either of the following methods:
(1) by detecting 0 of HWEF of HST **(a)**, or
(2) by waiting for interrupt caused by HI **(b)**.

<4> Consequently, the application program loads from HDT **(a)**. As a result of the load, HWEF is set to 1 **(b)**. At the same time, the $\overline{\text{HWE}}$ pin becomes active (low) **(c)**, and the external circuit recognizes that HDT is enabled to write.

Figure 3-60. Host Write Sequence (μPD7701x <— host): HDT write without wait



(5) Handshake

Handshaking between the μPD7701x and host can be established by:

- Polling
- Wait
- Interrupt

Each mode is described next.

(a) Polling

Synchronization of handshaking is established by always monitoring and evaluating the host read enable flag (HREF) and host write enable flag (HWEF) of the host interface status register (HST). Here is an example of host read (μPD7701x —> host) by means of polling:

```
/* Explicitly define HST and HDO because they are not reserved
words. */
     #define HST 0x3807
     #define HDO 0x3806

/*Disable internal interrupts HO and HI.*/
     R0L = SR             ;
     R0 = R0 | 0x0300     ;
     SR = R0L             ;

     R0L = 0x0            ; Set host status as follows:
     *HST:X = R0L         ; • Does not use HDT access wait
                          ;   function.
                          ; • Does not mask HRE function.
                          ; • Does not mask HWE function.
                          ; • Clears all user flags.
                          ; • Clears all error flags.

POLL: R0L = *HST:X        ; Judges HREF and loops to wait
                          ; until host reads HDT.
      R0=R0 & 0x2         ;
      if (R0! = 0)jmp POLL ;

      *HDO:X = R1H        ; Data of R1H is output because HDT
                          ; has become empty.
```

(b) Wait

Under the following conditions, execution of data exchanges with the HDT(in) and/or HDT(out) registers cause instruction wait cycles:

- when the load/store wait function is enabled (HAWE=1) and a store to HDT(out) is to be executed, while HREF=1 (valid data exists in HDT(out))
- when the load/store wait function is enabled (HAWE=1) and a load from HDT(in) is to be executed, while HWEF=1 (valid data does not exists in HDT(in))

Therefore, the advantage of this format is that writing handshake procedures is not required in application program, because the handshake procedure is automatically executed by hardware. Here is an example of host read by using the wait function:

```
/*Explicitly define HST and HDO because they are not reserved
words.*/
   #define HST 0x3807
   #define HDO 0x3806

/Disable internal interrupts HO and HI.*/
   R0L = SR              ;
   R0 = R0 | 0x0300      ;
   SR = R0L              ;

   R0L = 0x0400          ; Set host status as follows:
   *HST:X = R0L          ; • Uses HDT access wait function.
                         ; • Does not mask HRE function.
                         ; • Does not mask HWE function.
                         ; • Clears all user flags.
                         ; • Clears all error flags.

   *HDO:X = R1H          ; Outputs data of R1H.  If HDT is
                         ; busy, wait cycle is inserted.
```

Caution **When data is written from the application program to HDT, the wait is not released unless HDT is read by the external device. If internal writing of DSP and external reading do not correspond on a one-to-one basis vis-a-vis HDT, a hang-up may occur.**

During wait, interrupts are delayed (refer to section 3.4.4 “Interrupt”).

(c) Interrupts

Handshaking can be established by generating an interrupt, if data can be stored to HDT (out) or loaded from HDT (in) by the μPD7701x. Therefore, the advantage of this format is that host input/output can be executed independently (asynchronously) of the other processing even while other processing is under execution. Here is an example of host input/output using an interrupt:

★

```
/ * Define host I/O */
   #define HST * 0x3807: X
   #define HDO * 0x3806: X
   #define HDI * 0x3806: X

/* Entry of interrupt vector table */
SegHi IMSEG AT 0x230  ; hio input interrupt routine

   R0H = HDI          ; Read from HDT (in)
   *DP0++ = R0H       ; Saves to buffer
   RETI               ; Returns from interrupt
   NOP                ;

SegHo IMSEG AT 0x234  ; hio output interrupt routine

   R0H = *dp4++       ; Reads from buffer
   HDO = R0H          ; Writes to HDT (out)
   RETI               ; Returns from interrupt
   NOP                ;

/* Disables interrupts to initialize host I/O */
   R1L = EIR          ; Disables all interrupts
   R1 = R1 | 0x8000   ; EI = 1
   EIR = R1L          ;
   NOP                ; Two wait cycles are necessary until EI = 1
   NOP                ;  becomes valid

   R0L = SR           ; Enables HI and HO interrupts
   R0 = R0&0xFCFF     ;
   SR = R0L           ;

   R1 = R1&0x7FFF     ; Enables all interrupts
   EIR = R1L          ;
   FINT               ; Discards previous interrupt

                      ; Initializes HDT
   R0L = 0x0          ; Without HDT access wait function
   HST = R0L          ; No HRE, HWE mask.  Clears user flag
   HD0 = R0L          ; Dummy store (Refer to “Caution” below)
```

★ **Caution** **Because the host output interrupt occurs at the rising edge of the $\overline{\text{HRD}}$ pin when the higher byte of the HDT register is accessed, the following points must be noted.**

(1) Transfer the first data by forcibly generating an interrupt by transferring dummy data or by transferring data without using an interrupt.

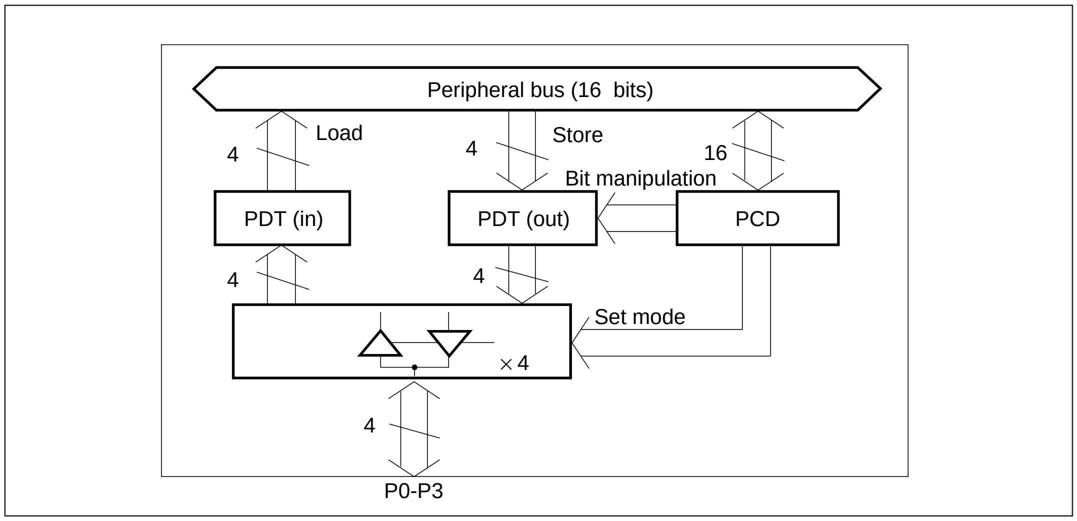
(2) If data is transferred in the burst mode, the chances are that the first data for the next burst transfer is not generated if an interrupt occurs at the last word of the burst data. Therefore, disable the interrupt by the instruction immediately before the one that transfers the last word to HDT, execute the same instruction as (1) after generation of the next burst data has been completed, and transfer the next burst data.

3.7.5 General-purpose input/output port

The μPD7701x is provided with a 4-pin input/output port. The following are the features of this port.

- Set in the input mode at hardware reset, and the values input to P0 and P1 after reset determine the boot mode.
- Each pin can be set in the input or output mode by the application program.
- The output value of the pin set in the output mode can be controlled independently.

Figure 3-61. General-purpose Input/Output Port



(1) Usage of the general-purpose port

There are three methods for using general-purpose port.

(a) Mode change (input —> output, output —> input)

The port command register PCD is used to set the port pins P0-P3 in the input or output mode.

$\boxed{\text{store to PCD}}$ ······ • set the mode of each bit (input or output)

(b) Input data (P0-P3 —> μPD7701x)

Input data is loaded from the port data register PDT(in).

$\boxed{\text{load from PDT}}$ ······ • input the 4-bit data

(c) Output data (P0-P3 <— μPD7701x)

There are two methods for setting an output port pin to a defined status.

(i) using PDT

Output data is stored to the port data register PDT(out).

$\boxed{\text{store to PDT}}$ ······· • output the 4-bit data

(ii) using PCD

The status of a single output port pin can be manipulated by the port command register PCD. Mode setting and bit manipulation can be set concurrently.

$\boxed{\text{store to PCD}}$ ······ • set the mode of each bit (input or output)
• manipulate the 1-bit output data (low or high)

(2) Pins of port interface

(a) P0-P3 (general-purpose input/output port)

These are general-purpose input/output pins and have the following functions:

• The output pin status is changed in synchronization with the rising edge of CLKOUT.

• The input pin is sampled in synchronization with the rising edge of CLKOUT.

(3) Port-related registers

(a) Port data register (PDT-0x3804:X/:Y)

This 16-bit register transfers data by using the general-purpose input/output port. To input data from the general-purpose input/output port, a load from PDT is performed. To output data, the data is stored to PDT whose value is then set to P0-P3. These pins correspond to the bit 0 to bit 3 of PDT. Data can be exchanged with the PDT register by use of load/store instructions.

When a load from PDT is executed, the data of this register is output to the peripheral bus. In input mode, the bit n of the PDT is set to 1 when high is input to the Pn pin, cleared to 0 for a low level input; if the Pn’ is an output pin, the value of bit n’ is undefined when load from PDT is performed, where n and n' are suffixes for correspondence indication and different numbers each other.

When a store to PDT is executed, the data is input to this register from the peripheral bus. In output mode, the Pn’ pin outputs high when the bit n’ of the PDT is set to 1, and outputs a low for 0; if the Pn pin is an input pin, the bit n value does not affect the port pin.

(b) Port command register (PCD-0x3805:X/:Y)

This 16-bit register specifies the input or output direction of the general-purpose input/output port, and bit manipulation of the output pins.

Data can be exchanged with the PCD register by use of load/store instructions. Note that not all of the PCD register bits can be loaded to a general-purpose registers (refer to the following table).

The value of PCD is cleared to 0 at reset.

Table 3-37 shows the function of each bit of PCD.

Table 3-37. Port Command Register (PCD - 0x3805:X/:Y)

Bit	Name	Category	Load/Store (L/S)	Bit function
15	BE	Bit manipulation	S	Bit manipulation enable bit • 0: Does not manipulate bit. • 1: Manipulates bit Manipulation method is specified by B1, B0, and PSR. • Undefined when read.
14	PSR	Bit manipulation	S	Port set/reset specification bit • 0: Reset (low level) • 1: Set (high level) • Manipulation port is specified by B1 and B0. • Valid when BE = 1. • Undefined when read.
13	ME	Mode setting	S	Mode setting enable bit • 0: Does not set mode. • 1: Sets mode. Contents to be set are specified by IO and M3-M0. • Undefined when read.
12	IO	Mode setting	S	Input/output specification bit • 0: Specifies input mode. • 1: Specifies output mode. • Port to be set is specified by M3-M0. • Valid when ME = 1. • Undefined when read.
11,10	Reserved	—	—	Reserved bits • No value can be set to these bits. • Undefined when read.
9, 8	B1, B0	Bit manipulation	S	Bit manipulation port specification bits • B1, B0 = 00:P0 01:P1 10:P2 11:P3 • Set/reset is specified by PSR. • Valid when BE = 1. • Undefined when read.
7-4	Reserved	—	—	Reserved bits • No value can be set to these bits. • Undefined when read.

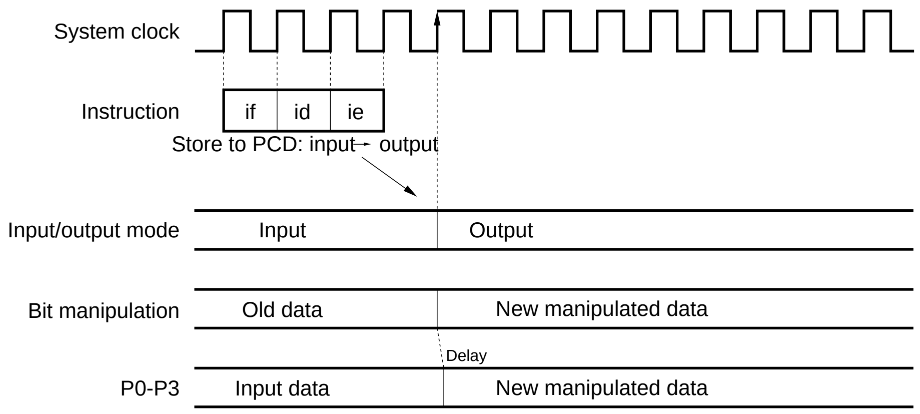
3-0	M3-M0	Mode setting	S	Mode setting port specification bits
				M3 = 0: P3 unselected, 1: P3 selected
				M2 = 0: P2 unselected, 1: P2 selected
				M1 = 0: P1 unselected, 1: P1 selected
				M0 = 0: P0 unselected, 1: P0 selected
				• Selection can be specified independently.

		Mode status	L	Input/output mode status bits
				M3 = 0: P3 input mode, 1: P3 output mode
				M2 = 0: P2 input mode, 1: P2 output mode
				M1 = 0: P1 input mode, 1: P1 output mode
				M0 = 0: P0 input mode, 1: P0 output mode

(4) Timing of port interface

The general-purpose I/O port is not assumed to be used synchronously, but is synchronized with the rising edge of CLKOUT during data input/output.

(a) Mode change from input to output



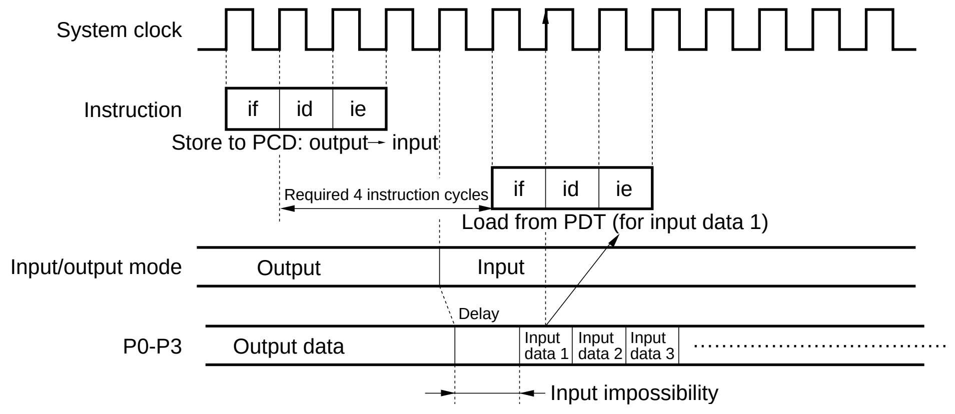
The mode of each pin is changed from input to output two system clocks after the execution cycle of the instruction that stores data to the PCD register.

Example program:

```
#define PCD=0x3805
#define PDT=0x3804

R1L=0x0000    ;
*PDT:x=R1L    ;     initialize PDT
R0L=0x3001    ;
*PCD:x=R0L    :     P0 —> output port
```

Caution **Because the PDT register is undefined after hardware reset, write data to the PDT register before storing data to the PCD register.**

(b) Mode change from output to input

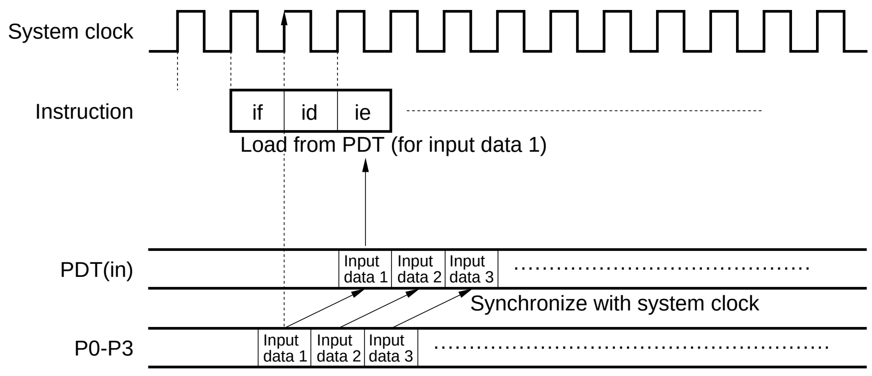


The mode of each pin changes from output to input after two system clocks since execution cycle of store to PCD register, but the μPD7701x inhibit the pin's data from being input during two system clocks after then. Therefore it is required that minimum 4 system clocks between store to PCD register and load from PDT register.

Example program:

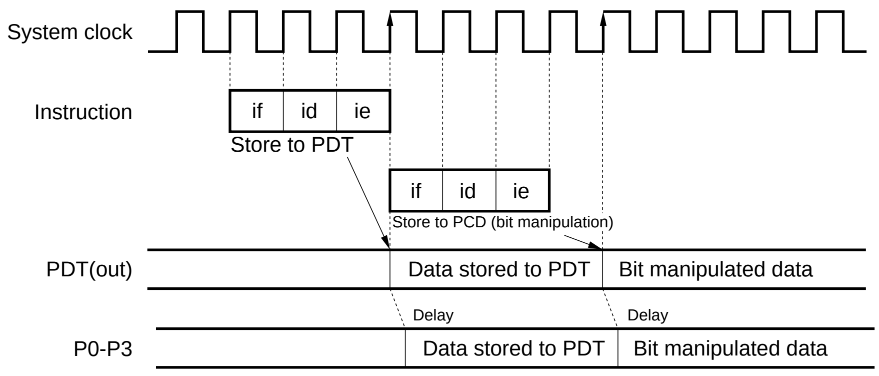
```
#define PCD=0x3805
#define PDT=0x3804

R0L=0x200f    ;
*PCD:x=R0L    ; P0-P3 output —> input
<required minimum 4 system clocks between instructions>
R1L=*PDT:x    ; load from PDT
```

(c) Timing of input ports



The pin’s input data is loaded after synchronized with the rising edge of two system clocks.

(d) Timing of output ports



(In case of store to PDT register)

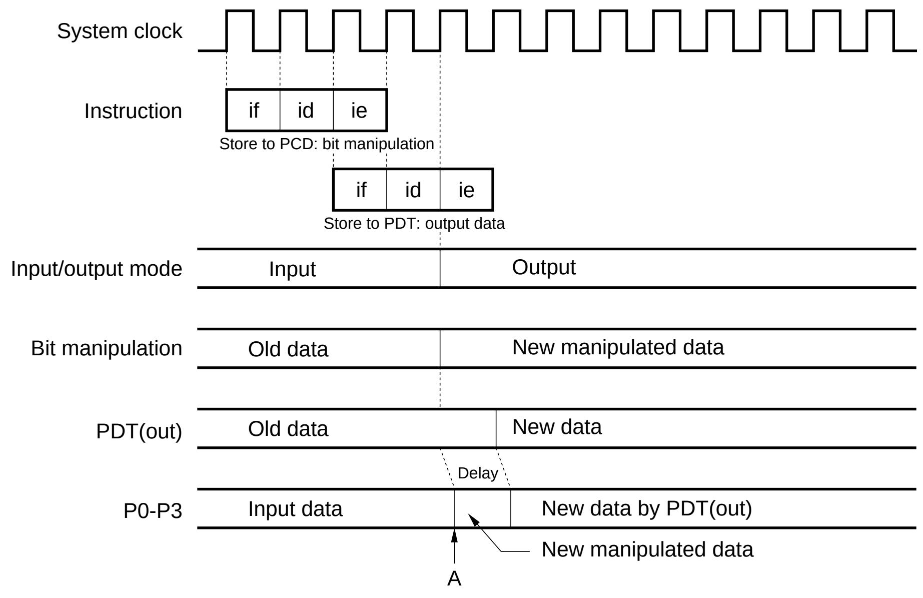
The output data is output after one system clock since execution cycle of store to PDT register.

(In case of store to PCD register)

The 1-bit manipulated data is output after two system clocks since execution cycle of store to PCD register.

Caution **If bit manipulation by the PCD register and data output by storing to the PDT register are executed at the same time, the data of the PDT register takes precedence over bit manipulation by the PCD register.**

(e) Output port setting (by use of PCD and PDT registers)



The manipulated data is output after two system clocks since execution cycle of store to PCD register. Next, when the output data is output after one system clock since execution cycle of store to PDT register, spike noise may occur at point A. Therefore it is required that minimum 1 system clock between store to PCD register and store to PDT register.

Example program:

```
#define PCD=0x3805
#define PDT=0x3804

R0L=0xf00f    ;
*PCD:x=R0L    ;P0-P3 input –> output, P0 –> high
R1L=0x0000    ;<required minimum 1 system clock
              ;between instructions>
*PDT:x=R1L    ;P0-P3 –> low
```

Cautions 1. If at least one system clock is not inserted between the instruction that stores data to the PCD register and the instruction that stores data to the PDT register, a spike may be generated at point A.

2. Because the value of the PDT register is undefined after hardware reset, set the PDT register before storing data to the PCD register.

(4) Example of port programming

Here is an example of a program using the general-purpose input/output port. In this example, the following is executed:

- P0 and P1 are set in the output mode.
- P2 and P3 are set in the input mode.
- P0 outputs a low level, and P1 outputs a high level.

Example of programming general-purpose input/output port

```
#define     PDT    0x3804
#define     PCD    0x3805
#define     All_In_mode  0x200F
#define     P0_Out_mode  0x3001
#define     P1_Out_mode  0x3002
#define     Out_P0_Low   0x8000
#define     Out_P1_High  0xC100

R0L = All_In_mode               ; P3-P0 input pins
*PCD:x = R0L;
R0L = P0_Out_mode+Out_P0_Low    ; P0 output pin (low level)
*PCD:x = R0L;
R0L = P1_Out_mode+Out_P1_High   ; P1 output pin (high level)
*PCD:x = R0L;
```

3.7.6 Wait controller

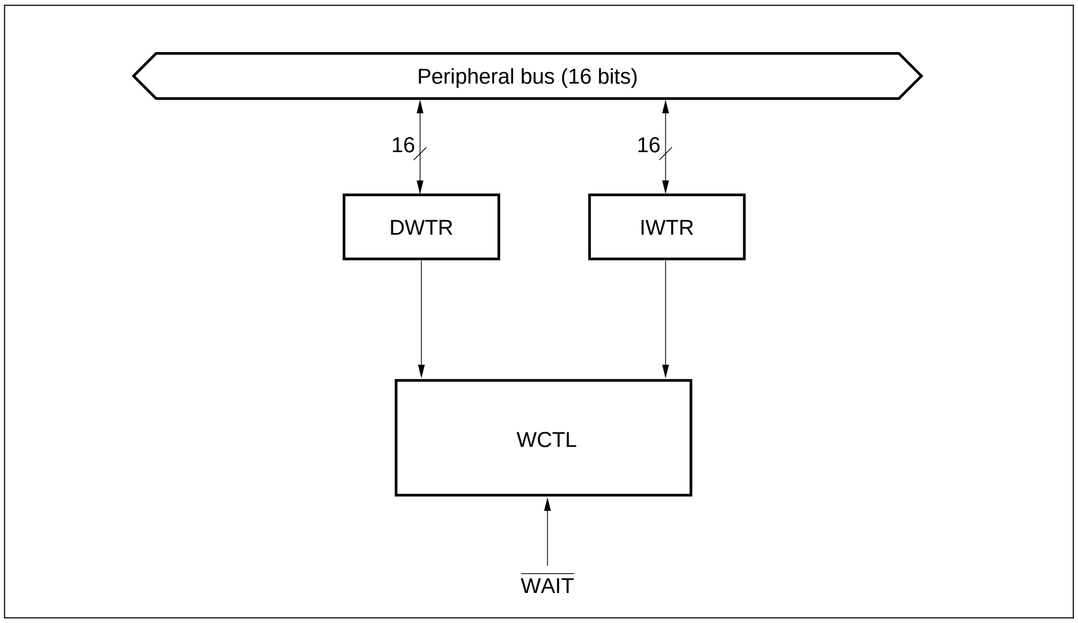
When the external memory area is accessed, the number of wait cycles to be inserted can be specified in advance by using a register.

The main features of the wait controller are as follows:

- Independently controls the data memory space and instruction memory space.
- Four types of wait cycles (0, 1, 3, and 7 wait cycles).
- Can be used with the $\overline{\text{WAIT}}$ pin (data memory space).

Figure 3-62 shows the block diagram of the wait controller.

Figure 3-62. Wait Controller



(1) Data memory wait cycle register (DWTR-0x3808:X/:Y)

Refer to paragraph (d) “Wait controller” in section 3.5.2 (3).

(2) Instruction memory wait cycle register (IWTR-0x3809:X/:Y)

Refer to paragraphs (d) “External instruction memory interface” and (f) “Wait function for instruction memory access” in section 3.4.2 (1).

3.7.7 Debug interface (JTAG)

The *μ*PD7701x family is provided with the following functions conforming to JTAG interface:

- JTAG port
- Boundary scan test function
- Debug function (In-Circuit Emulation function)

(1) JTAG port

Joint Test Action Group (JTAG) is an organization founded to promote standardization of boundary scan, a technique to facilitate testing of printed wiring boards that are mounted in electronic systems, and a standardization plan by this organization is recommended as "IEEE1149.1."

A device conforming to JTAG has an access port for testing, and the device can be tested independently of the internal logic.

The *μ*PD7701x family is provided with a register and a control circuit for In-Circuit Emulation, in addition to the instruction register, bypass register, and boundary scan register, which are specified to be essential by the above recommendation. For the details of JTAG, refer to "IEEE1149.1."

[Debug pins (TAP: test access port)]

Four pins and In-Circuit Emulation pin (TICE) conforming to the recommendation are provided.

- TCK (input) .... Test clock input pin.
  Input 0 when not used (conforms to recommendation).
- TMS (input)........... Test mode select input.
  Sampled at the rising edge of TCK. Internally pulled up.
- TDI (input) .... Test data input.
  Sampled at the rising edge of TCK. Internally pulled up.
- TDO (output) .... Test data output.
  Changes output in synchronization with the falling edge of TCK.
- TICE (output)........ Output to organize the break mode of In-Circuit Emulation.

Caution Do not stop TCK while it is high.

(2) Boundary scan test function

The boundary scan test method allows testing of the board level and chip level of the target system in a consistent test phase. This is why this method has been widely employed for automatic systems at many production sites.

The μPD7701x family has boundary scan functions as described below.

(a) Test instruction register

This 8-bit register is used to select test parameters and a test data register. Table 3-38 lists the supported instructions.

Table 3-38. Test Instructions

Bit 76543210	Instruction
00000000	EXTEST instruction
00000001	SAMPLE/PRELOAD instruction
11111111	BYPASS instruction

Caution The operation is undefined if data other than above is input.

(b) Test bypass register

This register outputs the data input from TDI to TDO. Refer to BYPASS instruction in Table 3-38.

(3) Debug function (in-circuit emulator function)

The μPD7701x family is provided with debug monitoring functions using JTAG with a run-time program. These functions have the following features:

- **Break function**
  - Break by fetch of specified instruction address
  - Break by reading/writing specified data memory address

- **Non-break monitor function**
  - References or changes the contents of a register or memory during program execution

Cautions 1. Detailed operations of the debug function are not made public to users.

2. The debug function is used by the hardware debugger for the μPD7701x family (IE-77016). Figure 3-64 shows the JTAG pin connections when the IE-77016 is used. When the IE-77016 is not required, connect these pins according to 2.4 "Handling of Unused Pins."

Figure 3-63. Appearance of JTAG Pins

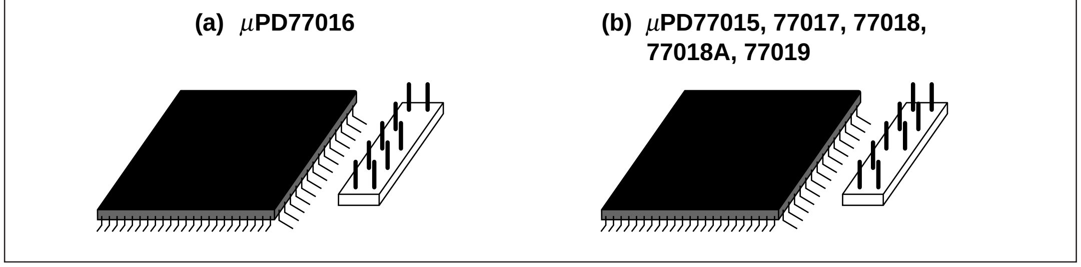
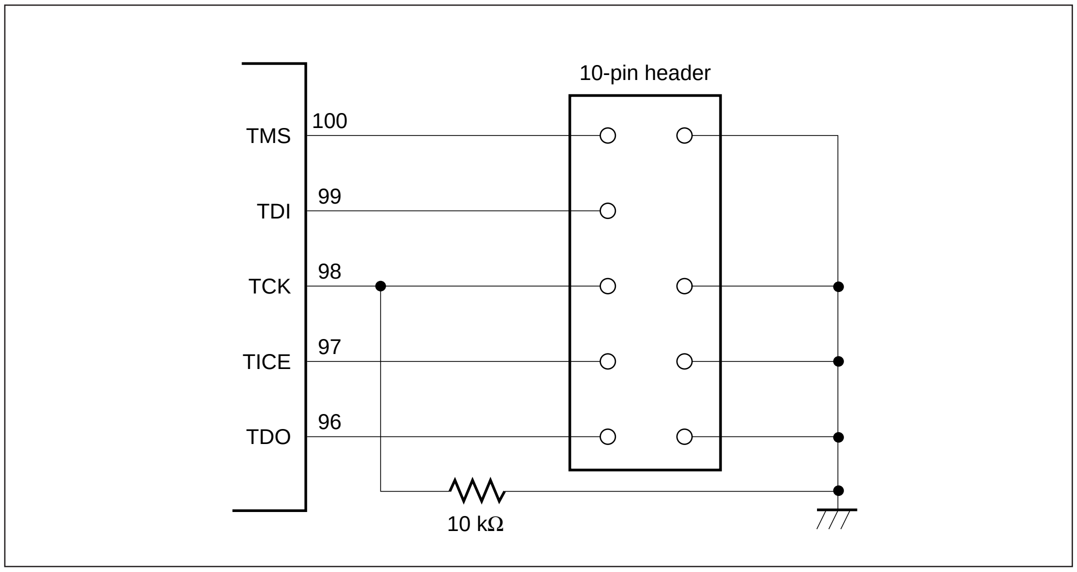
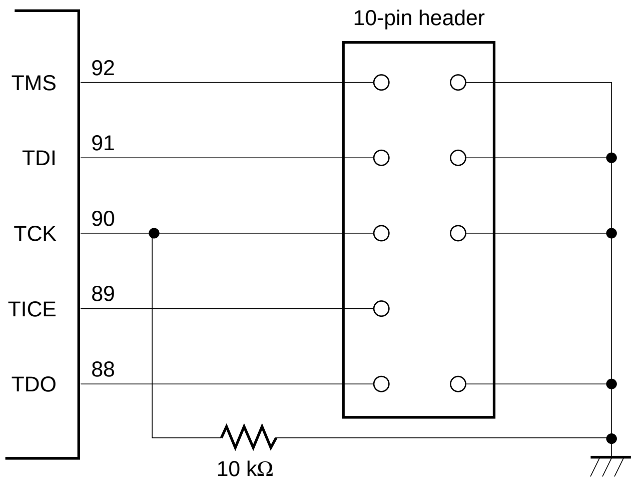


Figure 3-64. The JTAG Pin Processing

(a) μPD77016



(b) μPD77015, 77017, 77018, 77018A, 77019



Chapter 4

Boot Function

This chapter explains the boot function of the μPD7701x family. First, the functional outline is explained, and then the types of the boot function (modes), booting at reset, a boot subroutine, and the time required for booting are described in that order.

4.1 General

The μPD7701x family is provided with a program to be booted up to the internal instruction RAM. This program is stored in its internal ROM at addresses 0-0xFF of the instruction space.

This program provides users with several subroutine services, including a reset boot function and a reboot function. The reset boot function enables execution of booting immediately after the hardware is reset and the program counter is cleared to “0”. The reboot function allows users to rewrite program data in the RAM area of the instruction space from the application program. The subroutine entry points for using these functions are made public to users.

This chapter describes boot modes (by classifying and comparing modes by aspect), the boot functions in each mode, boot parameters, and how to call boot subroutines.

The following registers are used to execute booting:

[Registers whose contents are affected by boot execution]

- R7
- DP3
- DP7
- HDT (host data register)

[Registers that are set before booting]

- IWTR (instruction memory wait cycle register)
- DWTR (data memory wait cycle register)
- HST (host interface status register)

Caution **All the above registers are not always used depending on the selected boot mode. For details, refer to the description of each boot mode.**

4.2 Boot Modes

4.2.1 Classification of boot modes

The boot modes can be classified according to the following three attributes:

- Boot starting format (reset boot vs. reboot)
- Boot source (self-boot vs. host boot)
- Transfer word size (word boot vs. byte boot)

(1) Classification by boot starting format

Booting can be classified into the following two types according to the starting format: (a) Reset boot and (b) Reboot.

(a) Reset boot

Booting by resetting the μPD7701x hardware is called reset boot. When the hardware is reset, the PC (program counter) is cleared to “0” indicating address 0 of the instruction memory. Because this address is allocated as the reset boot entry point provided in the internal boot ROM area, the μPD7701x automatically executes booting. With the reset boot, the instruction memory area to be booted is limited to the internal instruction RAM (starting from address 0x200).

(b) Reboot

Booting by calling a boot servicing subroutine from an application program is called reboot. Some subroutine entry points for booting are provided in the boot ROM area and made public to users. By accessing these entry points from an application program, any part of the instruction RAM can be rewritten at any time.

The reboot function is mainly used to load program data to the external instruction memory or to rewrite the contents of the external instruction memory with the μPD77016.

(2) Classification by boot source

Booting consists in rewriting instructions in a certain format. The boot function can be classified into the following two types depending on from where the op code data to be rewritten and the parameters for rewriting are obtained: (a) Self-boot and (b) Host boot

(a) Self-boot

Self-booting transfers program code data from the data memory to the instruction memory. Boot parameters are set in the data memory (Y memory: from address 0x4000) in the case of reset boot, and in the registers in the case of reboot.

To configure a stand-alone system with the μPD77016 , store boot parameters and program data in an external ROM, and execute reset boot in the self-boot mode.

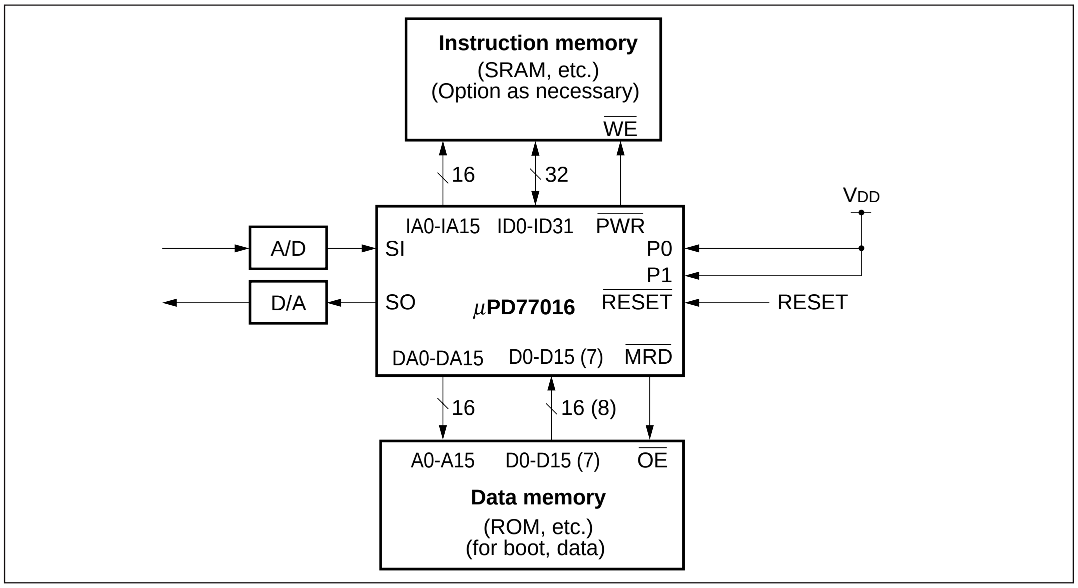
To configure a stand-alone system with a ROM version product (μPD77015, 77017, 77018, 77018A, or 77019), store boot parameters and program data in the ROM area, and execute reset boot in the self-boot mode. The boot parameters are always stored in the internal Y ROM, whereas the program data can be stored either in the internal or external X or Y ROM.

The μPD77019-013 cannot execute self-boot from the internal data ROM because its internal data ROM is masked. However, because the boot parameters for self-boot are set in advance, self-boot can be executed only from the external Y data memory.

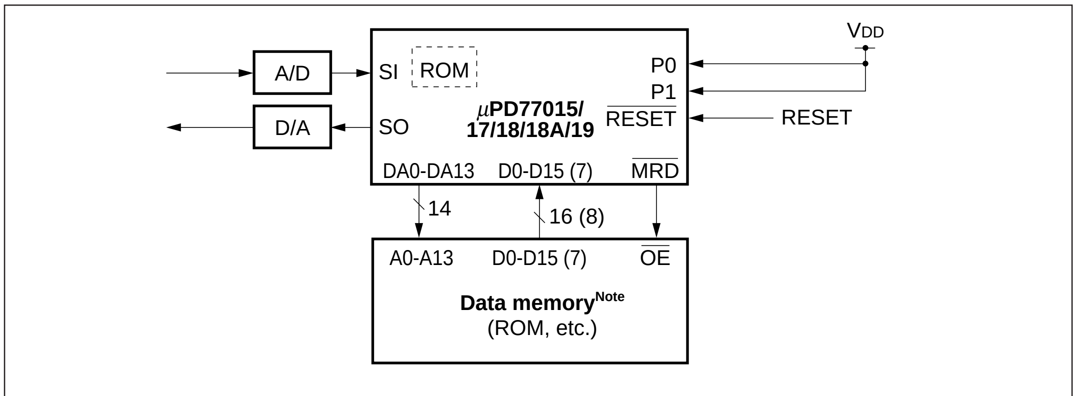
Figure 4-1 shows a configuration example of a system in the self-boot mode.

Figure 4-1. Example of Self-boot System Configuration

(a) μPD77016



(b) μPD77015, 77017, 77018, 77018A, 77019



Note Use this as an option as necessary. This memory is essential for the μPD77019-013.

(b) Host boot

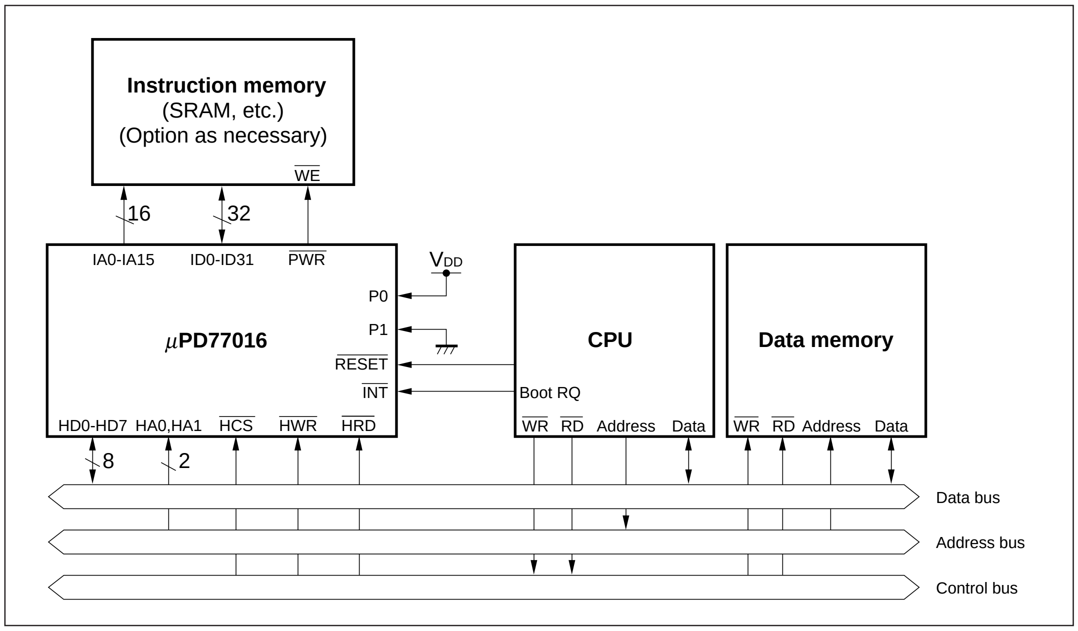
When host boot is executed, the μPD7701x reads program code data through the host interface and transfers this data to the instruction memory. In the case of reset boot, the boot parameters are read through the host interface prior to the program code data read. In the case of reboot the boot parameters have to be set to some registers by the application program.

Host boot can be used in a system where a CPU is connected to the host interface and a program is downloaded from that CPU to the μPD7701x.

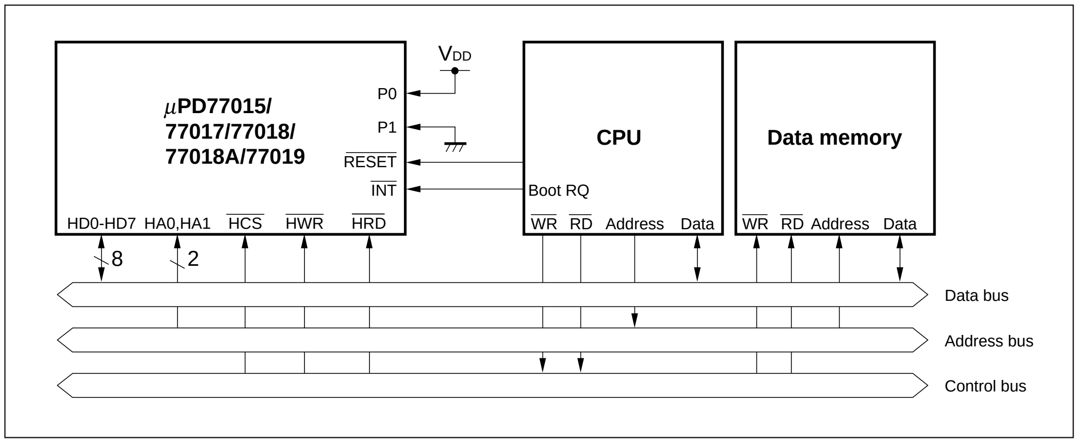
Figure 4-2 shows an example of host boot system configuration.

Figure 4-2. Configuration Example of Host Boot System

(a) μPD77016



(b) μPD77015, 77017, 77018, 77018A, 77019



(3) Classification by transfer word size

This classification is meaningful for self-boot only. The parameter for self-boot can specify the size for data memory reading as follows:

- 16-bit word/1 address (word boot): Refer to Figure 4-3.
- 8-bit byte/1 address (byte boot): Refer to Figure 4-4.

In the case of word boot, therefore, two data memory addresses correspond to one instruction step; in the case of byte boot, four data memory addresses correspond to one instruction step.

Normally, for self-boot at reset, the boot parameters and program code data are fixed in ROM. With the μPD77016, if the ROM in which the parameters and program code data are stored is configured of 8 bits (1 byte)/1 address, a cost reduction can be achieved when a small-scale system is organized.

Figure 4-3. Illustration of Word Boot

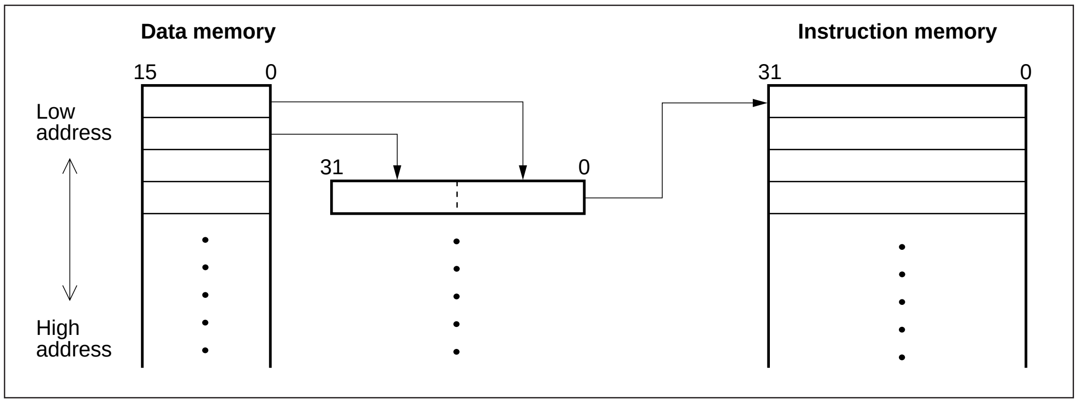
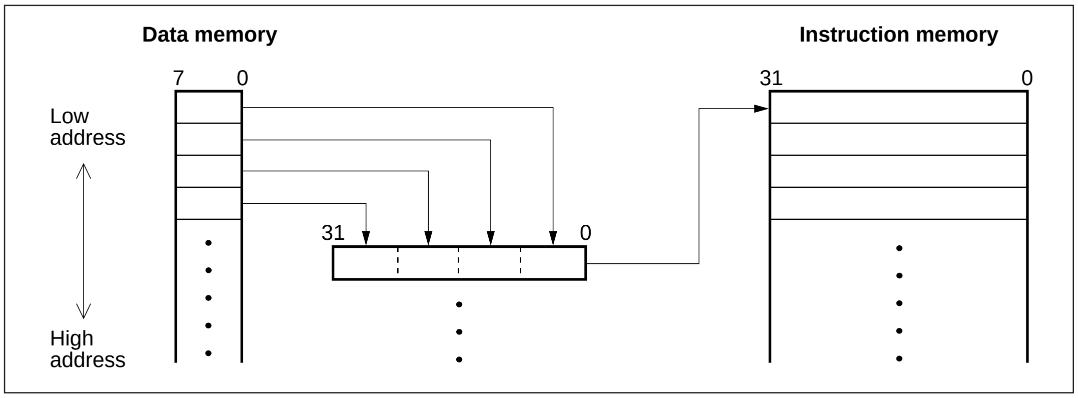


Figure 4-4. Illustration of Byte Boot



4.3 Boot at Reset

The μPD7701x family executes the boot program located at address 0 after hardware reset is input. The boot program first reads general-purpose port pins P0 and P1 and determines the boot mode (self-boot or host boot) depending on the bit pattern. Table 4-1 shows the relation between the bit pattern of P0 and P1 at reset and the boot mode.

Table 4-1. P0 and P1 Reset Values and Boot Modes

P1	P0	Reaction (boot mode)
0	0	Does not execute boot. Branches to address 0x200.Note
0	1	Executes host boot, and then branches to address 0x200.
1	1	Executes self-boot, and then branches to address 0x200.
1	0	Setting prohibited

Note This setting is used by the DSP when it must be reset, to execute reset boot once on power ON and subsequently to return from a power-down mode.

When reset boot is executed, no parameter to specify the load starting address of the instruction memory is provided. The instruction memory will always be loaded from the fixed address 0x200 (starting address of internal RAM area)

Caution **With the μPD77016, only the internal instruction memory is subject to reset boot. Use reboot for booting up the external instruction memory.**

4.3.1 Self-boot operation

(1) Parameters for self-booting

The following parameters are first read from address 0x4000 of the Y memory:

- Memory space command
- Word boot or byte boot command

Table 4-2 shows the contents of the parameters.

Table 4-2. Parameters for Self-booting (0x4000: Y)

Bit no.	Value	Meaning
0	0	Y memory boot. Reads program codes from Y memory space.
	1	X memory boot. Reads program codes from X memory space. Note
1	0	Word boot. Reads data memory in 16-bit units. Therefore, one instruction memory address corresponds to two addresses of data memory. Refer to Figure 4-3.
	1	Byte boot. Reads data memory in 8-bit units. Therefore, one instruction memory address corresponds to four data memory addresses. Refer to Figure 4-4.
2-7	Any	In byte boot mode
2-15	Any	In word boot mode

Note All boot parameters are read from the Y memory space even when X memory boot is specified. At this time, seven wait cycles are set as the data memory wait cycles. The parameter addresses are as follows:

- 0x4000: Y-0x4004: Y (in word boot mode)
- 0x4000: Y-0x4009: Y (in byte boot mode)

Caution The registers DWTR, IWTR, R7, DP3 and DP7 are changed by the boot routine.

(a) Parameters for word boot

Table 4-3 shows the memory map of the parameters for word boot.

Table 4-3. Memory Map of Parameters for Word Boot

Address	Memory value
0x4000: Y	16/8 bits, X/Y
0x4001: Y	Value set to DWTR
0x4002: Y	Value set to IWTR
0x4003: Y	Starting address of data memory that stores program to be read
0x4004: Y	Number of steps of program Note

Note This value is calculated with 32 bits = 1 word, and does not indicate the number of words when the program is located in the external data memory.

(b) Parameters for byte boot

Table 4-4 shows the memory map of the parameters for byte boot.

Table 4-4. Memory Map of Parameters for Byte Boot

Address	Memory value
0x4000: Y	16/8 bits, X/Y
0x4001: Y	—
0x4002: Y	Value set to DWTR (lower byte)
0x4003: Y	Value set to DWTR (higher byte)
0x4004: Y	Value set to IWTR (lower byte)
0x4005: Y	Value set to IWTR (reserved)
0x4006: Y	Starting address of data memory storing program to be read (lower byte)
0x4007: Y	Starting address of data memory storing program to be read (higher byte)
0x4008: Y	Number of steps of program**Note** (lower byte)
0x4009: Y	Number of steps of program**Note** (higher byte)

Note This value is calculated with 32 bits = 1 word, and does not indicate the number of words when the program is located in the external data memory.

★ **(2) Parameters for self-boot of μPD77019-013**

With the μPD77019-013, the following information is defined in advance as boot parameters.

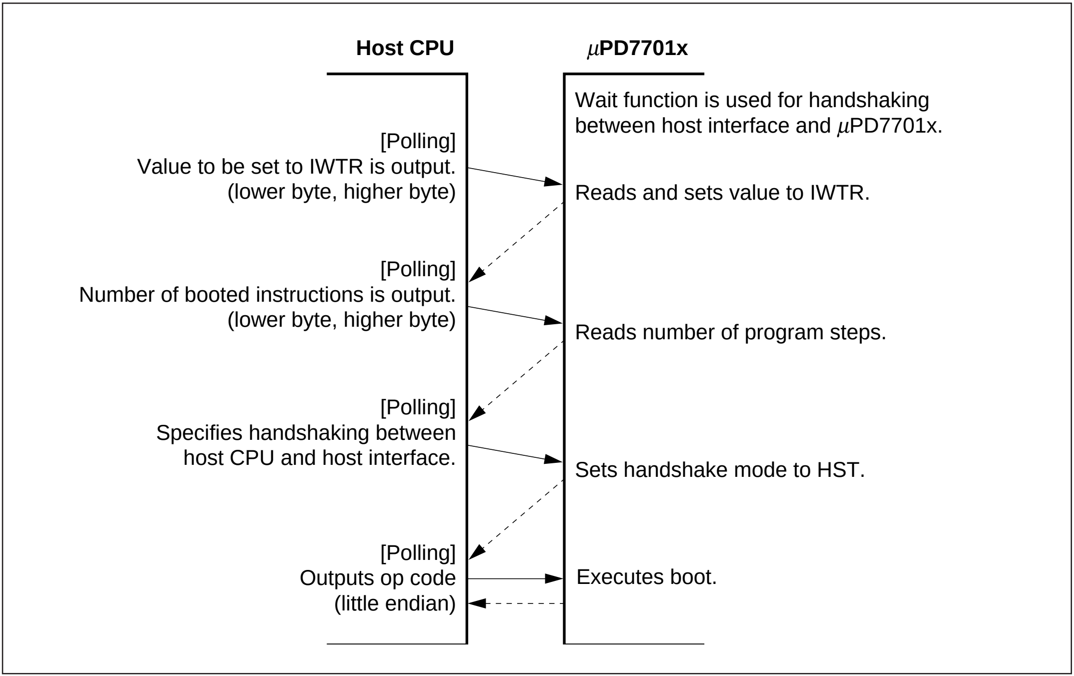
0x4000: Y	0 (Y memory boot/word boot)
0x4001: Y	0xC0C0 (set value of DWTR: 7 wait)
0x4002: Y	0 (set value of IWTR: 0 wait)
0x4003: Y	0xC000 (first address of boot code storage destination)
0x4004: Y	0x1000 (number of steps of program: 4K words)

To execute self-boot, connect 8K words or more of 16-bit PROM (that can be accessed with 7 wait cycles) from address 0xC000 of the external Y data area, and store the codes for boot in that PROM.

4.3.2 Host boot operation

When the host boot mode is executed, boot parameters and op codes are obtained through the host interface. Figure 4-5 shows a conceptual drawing of the host boot procedure.

Figure 4-5. Host Boot Procedure


Host CPU
μPD7701x
Wait function is used for handshaking between host interface and μPD7701x.
[Polling] Value to be set to IWTR is output. (lower byte, higher byte)
Reads and sets value to IWTR.
[Polling] Number of booted instructions is output. (lower byte, higher byte)
Reads number of program steps.
[Polling] Specifies handshaking between host CPU and host interface.
Sets handshake mode to HST.
[Polling] Outputs op code (little endian)
Executes boot.

(1) Setting of host interface

Prior to host boot operations, initial settings for the host interface are performed as follows. However, these settings, except the settings of HAWE bit and transfer size, are overwritten by the HST setting parameters to be sent later.

HST=0x0401

- HAWE = 1 : Uses wait function.
- HREM = 0 : Does not mask $\overline{\text{HRE}}$.
- HWEM = 0 : Does not mask $\overline{\text{HWE}}$.
- Transfer size = 16-bit mode : Host writes the specified parameters to HDT starting with the lower 8 bits and then the higher 8 bits (this is not the set value of HST but a host boot rule).

Caution The value of the HST register is changed when the HST setting parameters are set in the boot process.

(2) Parameters for host boot

If reset boot is executed, the following parameters are used for host boot:

• IWTR set value	: Determines the number of wait cycles of the instruction memory. For the meaning of the set value, refer to the paragraph (f) "Wait function of external instruction memory" in section 3.4.2 (1).
• Number of booted instructions	: Indicates the number of instruction steps of the program to be booted (number of instructions to be booted). The number of data actually transferred is two times the number of instruction steps.
• HST setting value	: Data to be set to HST. All the bits of HST, except HAWE (bit 10), are set. HAWE is set to "1", regardless of the value set to HST.
• Instruction code	: The lower 16 bits (bits 15-0) and the higher 16 bits (bits 31-16) of a 32-bit op code are transferred in this order. If the host interface is set to 8-bit width, therefore, bits 7-0, 15-8, 23-16, and 31-24 are transferred in that order.

Caution When the μPD77015, 77017, 77018, 77018A, or 77019 is used, dummy data for IWTR set values must be transferred though IWTR set values are not specified.

The above parameters are transferred from the host in the following sequence:

1st transfer	: Lower 8 bits of IWTR set value[Note 1]
2nd transfer	: Higher 8 bits of IWTR set value[Note 1]
polling	: Wait for μPD7701x loaded data from HDT (in)
3rd transfer	: Lower 8 bits of number of booted instructions
4th transfer	: Higher 8 bits of number of booted instructions
polling	: Wait for μPD7701x loaded data from HDT (in)
5th transfer	: Lower 8 bits of HST set value
6th transfer	: Higher 8 bits of HST set value
polling	: Wait for μPD7701x loaded data from HDT (in)
7th transfer	: 1st op code (bits 7-0)
8th transfer	: 1st op code (bits 15-8)
polling	: Wait for μPD7701x loaded data from HDT (in)
9th transfer	: 1st op code (bits 23-16)
10th transfer	: 1st op code (bits 31-24)
polling	: Wait for μPD7701x loaded data from HDT (in)
11th transfer	: 2nd op code (bits 7-0)
:	:
(4n+6)th transfer	: nth op code (bits 31-24)[Note 2]
polling	: Wait for μPD7701x loaded data from HDT (in)

Notes 1. Dummy data are transferred when the μPD77015, 77017, 77018, 77018A, or 77019 is used.

2. The total number of transferred bytes is 4n + 6, where the number of transferred op codes is n.

4.4 Boot Subroutine (reboot)

Booting by using the boot subroutine to rewrite program data in the instruction memory is called reboot. Usually, the instruction memory cannot be rewritten from an application program. However, by using the reboot function (or by calling a boot subroutine), new instructions can be written to the instruction memory. Some boot subroutines are provided in the boot ROM and their entry points are made public to users as shown in Table 4-5. To execute reboot, set specified parameters to registers, then execute the CALL instruction jumping to a reboot entry address.

The registers and pins not related to reboot retain the status when the boot subroutine is called during and after reboot, and are not initialized by reboot.

Table 4-5. Boot Subroutine Entry Points

Reboot mode			Entry point address
Self-boot	X memory	Word reboot	0x2
		Byte reboot	0x4
	Y memory	Word reboot	0x1
		Byte reboot	0x3
Host boot	Host reboot		0x5

Cautions 1. Bear in mind the following points when executing reboot:

- **The register values are not preserved.**
- **One level of the program stack is used (at entry).**
- **One level of the loop stack is used.**
- **Disable all interrupts throughout the reboot period. (If an interrupt is acknowledged during reboot, normal operation cannot be guaranteed).**
- **After reboot completion, execution returns to the instruction next to the CALL instruction which called the reboot subroutine.**

2. The registers DWTR, IWTR, R7, DP3, DP7, HST and HDT are changed by the boot routine.

4.4.1 Parameters of X memory word or byte reboot

The reboot type that locates op codes in the X memory is called X reboot. X reboot can be classified into two modes:

X memory word reboot that locates a 16-bit word per one data memory address, and

X memory byte reboot that locates an 8-bit byte per one data memory address.

In both modes, the following parameters are set to specified registers, and the entry points shown in Table 4-5 are called.

- R7L : Number of instruction steps to be rebooted
- DP3: Starting address of X memory storing op code
- DP7: Starting address of instruction memory to be loaded

Cautions 1. The values in the parameter registers are not preserved.
2. Set IWTR and DWTR as necessary.

4.4.2 Parameters of Y memory word or byte reboot

The reboot type that locates op codes in the Y memory is called Y reboot. Y reboot can be classified into two modes:

Y memory word reboot that locates a 16-bit word per one data memory address, and

Y memory byte reboot that locates an 8-bit byte per one data memory address.

In both modes, the following parameters are set to specified registers, and the entry points shown in Table 4-5 are called.

- R7L : Number of instruction steps to be rebooted
- DP3: Starting address of instruction memory to be loaded
- DP7: Starting address of Y memory storing op code

Cautions 1. The values in the parameter registers are not preserved.
2. Set IWTR and DWTR as necessary.

4.4.3 Parameters for host reboot

To reboot from the host interface, the following parameters are set to specified registers, and the entry points shown in Table 4-5 are called.

- R7L : Number of instruction steps to be rebooted
- DP3: Starting address of instruction memory to be loaded

Cautions 1. The values in the parameter registers are not preserved.
2. Set HST and IWTR as necessary. However, be sure to set HAWE of HST to "1" (to use wait) before reading the reboot routine.
3. HDT must be empty (no data should remain before read) when reboot is started.

4.5 Boot Time

Table 4-6 shows the time required for booting.

Table 4-6. Boot Time

Boot mode		Boot time (unit : number of cycles)
Boot	Self-boot & Word boot	Min. 49 + 3D + (4 + 2D + I) $\times$ W Max. 51 + 3D + (4 + 2D + I) $\times$ W
	Self-boot & Byte boot	Min. 70 + 6D + (10 + 4D + I) $\times$ W Max. 72 + 6D + (10 + 4D + I) $\times$ W
	Host boot	29 + 4D + (4 + I) $\times$ W or longer (depending on access speed of host CPU)
Reboot	Self-reboot & Word boot	6 + (4 + 2D + I) $\times$ W
	Self-reboot & Byte boot	6 + (10 + 4D + I) $\times$ W
	Reboot	6 + (4 + I) $\times$ W or longer (depending on access speed of host CPU)

Remarks W : number of instruction words booted
D : data memory wait cycles
I : instruction memory wait cycles

Chapter 5

Development Tools

This chapter introduces the development tools for the μPD7701x family.

Caution **This chapter only introduces currently available development tools. For details, refer to the manual of each tool.**

5.1 Software Tools

The following type of software tool is available:

- Windows^{TM}-based development environments

5.1.1 Integrated development environment work bench (WB77016)

WB77016 is a development environment that unifies Relocatable Assembler, Linker, Editor, and Make Utility. It allows an efficient flow of operations, from program editing to the creation of object programs and software simulation startup.

★ 5.1.2 Software simulator (SM77016, SM77016-H)

SM77016 and SM77016-H simulate the operation of the μPD7701x Family. The focus of the simulation are the program control unit, external memory, instruction memory, host interface, serial interface, and the I/O ports.

The SM77016-H can execute simulation at high speeds by coding the program as a 32 bit Windows95 compatible program.

5.1.3 C compiler (InterTools^{TM} 77016)

This product was developed by TASKING, Inc., a US company. μPD7701x Family software applications can be created through a high-performance C cross-compiler that compiles with the ANSI standards. When debugging, use the software simulator.

5.1.4 System software for in-circuit emulator (ID77016)

The objective of this software package is the control of the IE-77016-PC. The IE-77016 can be manipulated through a user interface that is the same as the software simulator interface.

5.2 Hardware Tools

5.2.1 In-circuit emulator

(1) In-circuit emulator : IE-77016-PC

Host machine	Code
IBM PC/AT™	IE-77016-PC

Remark When using the IE-77016-PC, system software is necessary in addition to the above basic system. Refer to 5.1.4 “system software for in-circuit emulator (ID77016)”.

5.2.2 Options for in-circuit emulators

The following options are available for the IE-77016-PC. Use them as necessary.

	Code	Corresponding host machine
Emulation board for in-circuit emulator	IE-77016-CM-EM6	IBM PC/AT
μPD77017 evaluation board	EB-77017	
★ Adapter for EB-77017 board	TGC-100SDW (Tokyo Eletech Corp.)Note	

Note Consult NEC when you purchase this product.

(1) Emulation board for in-circuit emulator

This emulation board which is mounted in the IE-77016-PC, emulates μPD7701x operations. This board is provided with external instruction memory (32K words × 32 bits) and external X and Y data memories (16K words × 16 bits each) in addition to one μPD77016.

(2) μPD77017 evaluation board

This evaluation board is used to emulate the μPD77015/77017 by using the μPD77016.

One side of this board is connected to the IE-77016-PC interface cable, and the other side is equipped with the adapter to be connected to the target system board (EB-77017).

This board also serves as a level converter between the IE-77016-PC interface cable (5-V interface) and the μPD77015/77017 debugging pins (3-V interface) on the target system where the μPD77015/77017 has been mounted.

This board is provided with an external instruction memory (32K words x 32 bits), external X and Y data memories (32K words × 16 bits each), and a level conversion circuit (from 5 V to 3 V), in addition to one μPD77016.

To emulate the μPD77018, 77018A, and 77019, consult NEC.

★

(3) Adapter for EB-77017 board

This adapter is mounted on the EB-77017 of the evaluation board to connect the target system of the μPD77015/77017.

It is provided with 0.5-mm pitch, 100-pin TQFP package pins equivalent to the μPD77015/77017 pins (for target system connection) and a connector (for EB-77017 connection).

Appendix A

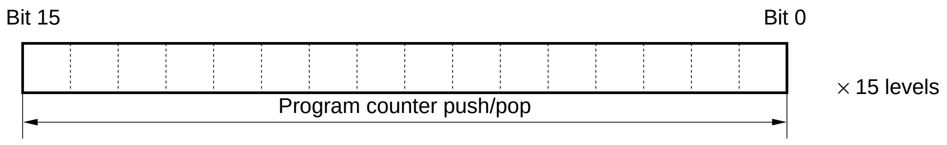
Device Summary

This appendix summarizes the functions of the μPD7701x family described in this manual. Use this appendix when developing your system after having gained a general understanding of the features of this device.

A.1 Register List

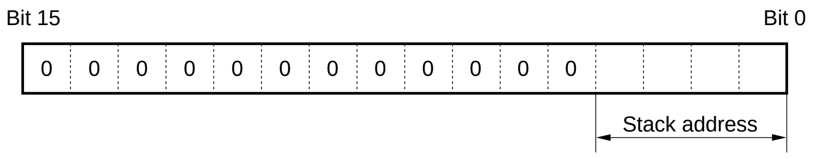
A.1.1 CPU registers

(1) Stack (STK)



Refer to section 3.4.2 “Program execution control block”.

(2) Stack pointer (SP)



Refer to section 3.4.2 “Program execution control block”.

(3) Status register (SR)

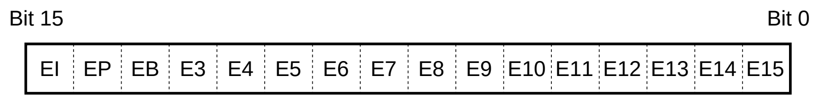
Interrupt enable flag				Reserved		Interrupt enable flag for each cause									
EI	EP	EB	LF			On-chip I/O device						External interrupt master			
						ho	hi	so2	si2	so1	si1	int4	int3	int2	int1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

interrupt enable flag = 1: interrupt disabled

interrupt enable flag = 0: interrupt enabled

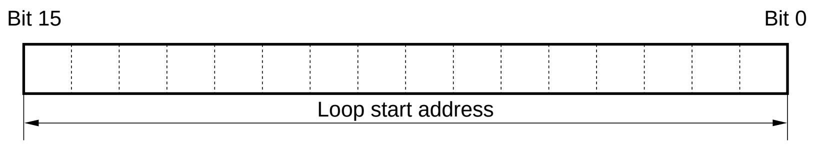
Refer to section 3.4.4 “Interrupt”.

(4) Interrupt enable flag stack register (EIR)



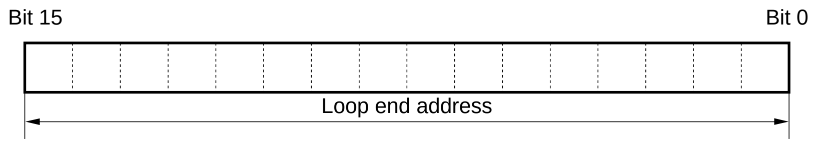
Refer to section 3.4.4 “Interrupt”.

(5) Loop start address register (LSA)



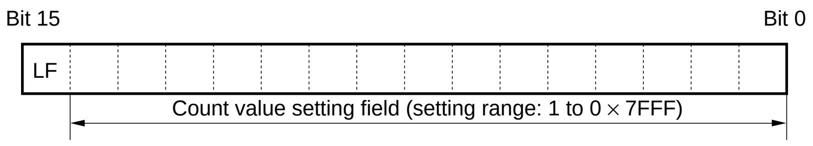
Refer to section 3.4.3 “Flow control block”.

(6) Loop end address register (LEA)



Refer to section 3.4.3 “Flow control block”.

(7) Loop counter (LC)

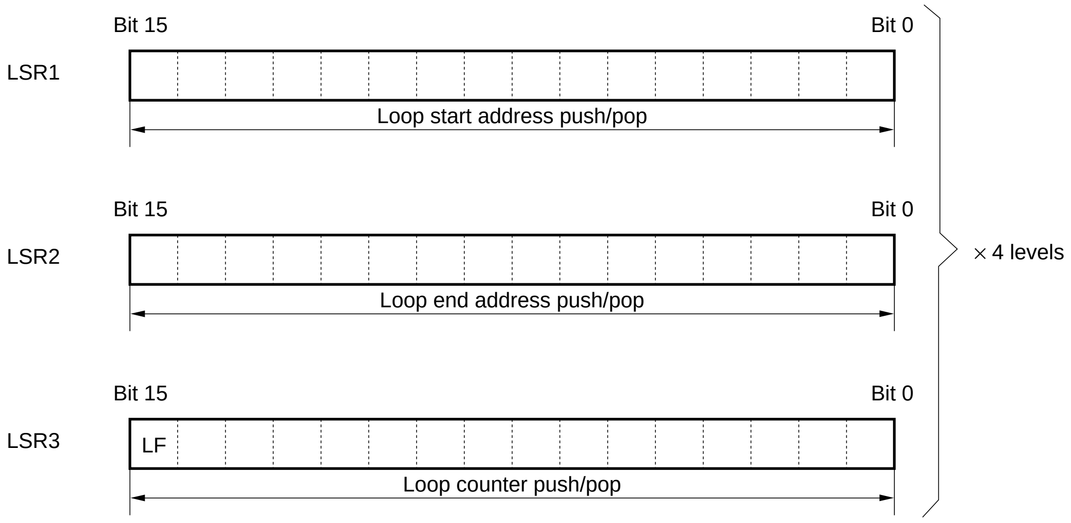


LF=0: loop in progress

LF=1: end of loop (not in progress)

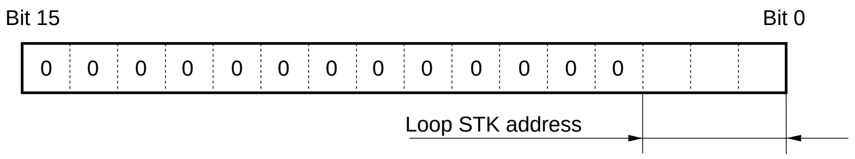
Refer to section 3.4.3 “Flow control block”.

(8) Loop stack (LSTK)



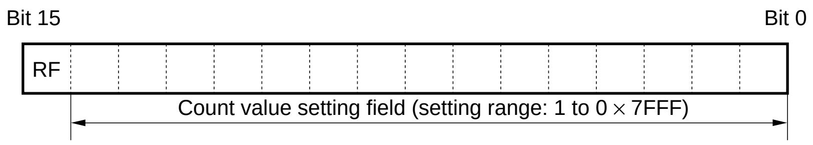
Refer to section 3.4.3 “Flow control block”.

(9) Loop stack pointer (LSP)



Refer to section 3.4.3 “Flow control block”.

(10) Repeat counter (RC)

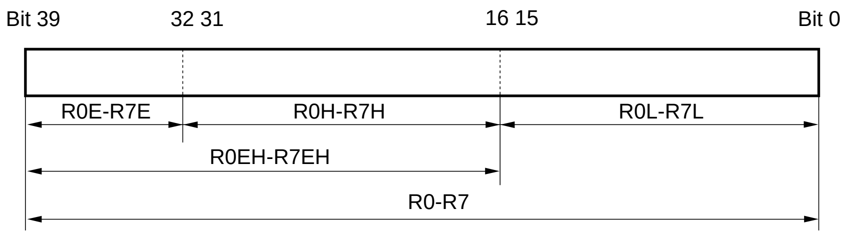


RF=0: Repeat in progress

RF=1: End of repeat (not in progress)

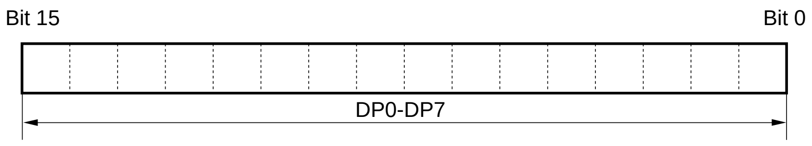
Refer to section 3.4.3 “Flow control block”.

(11) General-purpose registers (R0 to R7)



Refer to section 3.6 “Operation Unit”.

(12) Data pointers (DP0-DP7)

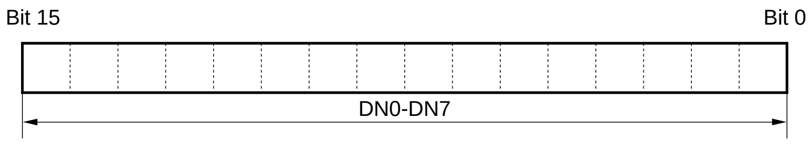


DP0-DP3: Address of X-memory space

DP4-DP7: Address of Y-memory space

Refer to section 3.6 “Operation Unit”.

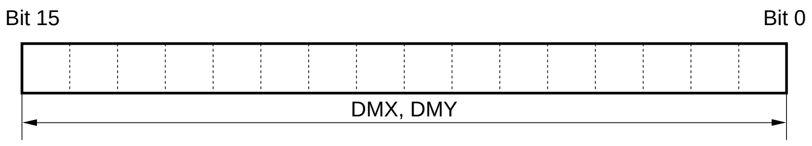
(13) Index registers (DN0-DN7)



DN0-DN7: Modify DP0-DP7

Refer to section 3.5 “Data Addressing Unit”.

(14) Modulo registers (DMX, DMY)



DMX: The ring count range for DP0-DP3 is specified

DMY: The ring count range for DP4-DP7 is specified

Refer to section 3.5 “Data Addressing Unit”.

(15) Error status register (ESR)

Bit 15															Bit 0
-	-	-	-	-	-	-	-	-	-	-	-	ovf	ste	lse	bac

ovf:	Overflow error flag
ste:	Stack error flag
lse:	Loop stack error flag
bac:	Bus access error flag
	0: No error
	1: Error

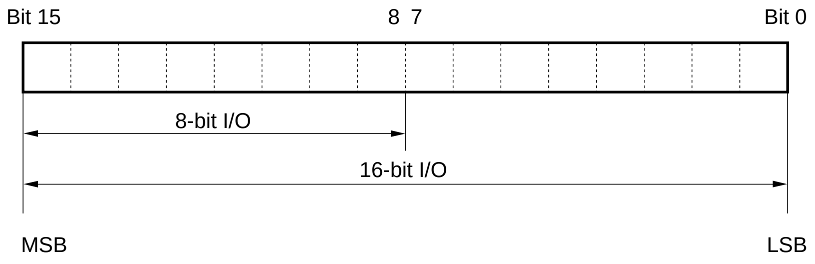
Refer to section 3.4.5 “Error status register”.

A.1.2 Peripheral registers

(1) Peripheral register map

X/Y memory address	Register name	Function	Peripheral name	Load/Store (L/S)
0x3800	SDT1	Serial data register 1	Serial IO	L/S
0x3801	SST1	Serial status register 1	Serial IO	L/S
0x3802	SDT2	Serial data register 2	Serial IO	L/S
0x3803	SST2	Serial status register 2	Serial IO	L/S
0x3804	PDT	Port data register	IO Port	L/S
0x3805	PCD	Port command register	IO Port	L/S
0x3806	HDT	Host data register	Host IO	L/S
0x3807	HST	Host status register	Host IO	L/S
0x3808	DWTR	Data memory wait cycle register	Wait Reg	L/S
0x3809	IWTR	Instruction memory wait cycle register	Wait Reg	L/S
0x380A-0x383F	Reserved	**Do not access this area**.	—	—

(2) Serial data registers (SDT1: 0x3800:X/:Y, SDT2: 0x3802:X/:Y)



Refer to section 3.7.3 “Serial interface”.

(3) Serial status registers (SST1: 0x3801:X/:Y, SST2: 0x3803:X/:Y)

Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SO TF	SI TF	SO BL	SI BL	SS WE	SL WE	SI CM	SI EF	Reserved				SS ER	SL ER	SS EF	SL EF

Bit	Name	Load/store L/S	Bit function
15	SOTF	L/S	Serial output transfer format setting bit • 0: Serial output with MSB first • 1: Serial output with LSB first
14	SITF	L/S	Serial input transfer format setting bit • 0: Serial input with MSB first • 1: Serial input with LSB first
13	SOBL	L/S	Serial output word length setting bit • 0: 16-bit serial output • 1: 8-bit serial output
12	SIBL	L/S	Serial input word length setting bit • 0: 16-bit serial input • 1: 8-bit serial input
11	SSWE	L/S	SDT store wait enable bit • 0: Does not use store wait function. • 1: Uses store wait function. Inserts wait cycles when μPD7701x stores to SDT (out) with SSEF=0.
10	SLWE	L/S	SDT load wait enable bit • 0: Does not use load wait function. • 1: Uses load wait function. Inserts wait cycles when μPD7701x loads from SDT(in) with SLEF=0.
9	SICM**Note**	L/S	Serial input continuous mode setting flag • 0: Enters single serial input mode after completion of current serial input. • 1: Enters continuous serial input mode to start serial input.
8	SIEF**Note**	L/S	Single serial input enable flag • 1: Starts serial input processing in single serial input mode (only once). The SIEF flag that is set to 1 is automatically reset in the next instruction cycle.
7-4	Reserved	—	Reserved bits • Value cannot be set to these bits. • Undefined when read.
3	SSER	L/S	SDT store error flag • 0: No error • 1: Error (Set to 1 when μPD7701x stores data to SDT(out) with SSEF=0.) • Once set, this flag does not change its status until 0 is written by μPD7701x.
2	SLER	L/S	SDT load error flag • 0: No error • 1: Error (Set to 1 when μPD7701x loads data from SDT(in) with SLEF=0.) • Once set, this flag does not change its status until 0 is written by μPD7701x.
1	SSEF	L	SDT store enable flag • Set to 1 when contents of SDT(out) is transferred to serial output shift register. • Cleared to 0 when μPD7701x stores data to SDT(out).
0	SLEF	L	SDT load enable flag • Set to 1 when contents the shift register for serial input is transferred to STD (in). • Cleared to 0 when μPD7701x loads data from SDT(in).

Note Following table shows an example of combination of SICM and SIEF.
When continuous data such as speech data is input, use status 2 (SICM=1, SIEF=0).

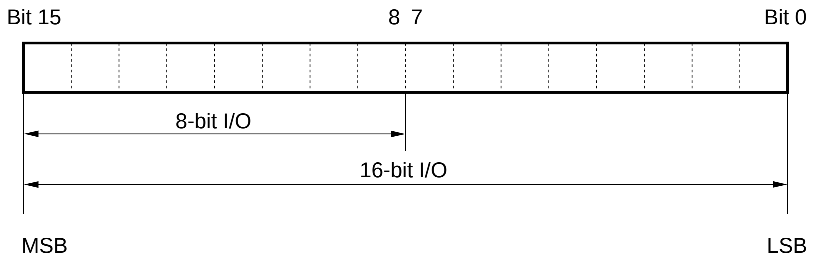
Remarks 1. Combination of SICM and SIEF Bits

Example of combination	Bit 9 SICM	Bit 8 SIEF	Function
1	0	0	• Status transition mode.
2	1	0	• Continuous serial input mode.
3	0	1	• Single serial input mode.
4	1	1	• The setting of this combination is prohibited.

2. Setting of SST after hardware reset: 0x0002.

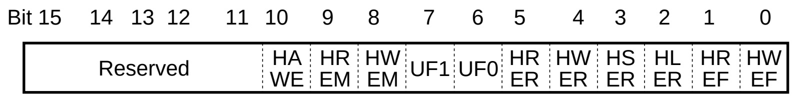
Refer to section 3.7.3 “Serial interface”.

(4) Host data register (HDT: 0x3806:X/:Y)



Refer to section 3.7.4 “Host interface”.

(5) Host status register (HST: 0x3807:X/:Y)



Bit	Name	R/W from host	Load/store (L/S) from μPD7701x	Bit function
15-11	Reserved	—	—	Reserved bits • No value can be set to these bits. • These bits are undefined when read.
10	HAWE	R	L/S	HDT access wait enable bit • 0: Wait is not used • 1: Wait is used Wait cycles are inserted if the μPD7701x attempts to store data to HDT(out) while HREF=1, or to load data from HDT(in) while HWEF=1.
9	HREM	R	L/S	HRE mask bit • 0: Does not mask. HRE changes according to the HREF status (refer to below). • 1: Masks. $\overline{\text{HRE}}$ becomes inactive (high level).
8	HWEM	R	L/S	HWE mask bit • 0: Does not mask. $\overline{\text{HWE}}$ changes according to the HWEF status (refer to below). • 1: Masks $\overline{\text{HWE}}$ becomes inactive (high level).
7	UF1	R	L/S	User’s flag
6	UF0	R	L/S	User’s flag
5	HRER	R	L/S	Host read error flag • 0: No error • 1: Error Set to 1 when host CPU reads HDT when HREF is 0. • Once set to 1, it does not change until 0 is written by program.
4	HWER	R	L/S	Host write error flag • 0: No error • 1: Error Set to 1 when host CPU writes HDT when HWEF is 0. • Once set to 1, it does not change until 0 is written by program.
3	HSER	R	L/S	HDT store error flag • 0: No error • 1: Error Set to 1 when μPD7701x stores to HDT when HREF is 1. • Once set to 1, it does not change until 0 is written by program.
2	HLER	R	L/S	HDT load error flag • 0: No error • 1: Error Set to 1 when μPD7701x loads from HDT when HWEF is 1. • Once set to 1, it does not change until 0 is written by program.
1	HREF	R	L	Host read enable flag • 0: Read disabled • 1: Read enabled Set to 1 when the μPD7701x stores data to HDT. Cleared to 0 when host CPU reads higher byte of HDT. • Ignored when written.
0	HWEF	R	L	Host write enable flag • 0: Write disabled • 1: Write enabled Set to 1 when the μPD7701x loads data from HDT. Cleared to 0 when host CPU writes higher byte of HDT. • Ignored when written.

Remark   The HST setting after hardware reset: 0x0301
- No wait function
- HRE/HWE mask: masked
- Host write enabled
- Host read disabled

Host I/O error flag setting condition

Error flag name	Cause	Releasing condition
HRER	Host read when HREF = 0	Reset by hardware reset or program
HWER	Host write when HWEF = 0	
HSER	Store to HDT when HREF = 1	
HLER	Load from HDT when HWEF = 1	

(6) Port data register (PDT: 0x3804:X/:Y)

Bit 15															Bit 0
-	-	-	-	-	-	-	-	-	-	-	-	P3	P2	P1	P0

0: Low level

1: High level

Refer to section 3.7.5 “General-purpose input/output port”.

(7) Port command register (PCD:0x3805:X/:Y)

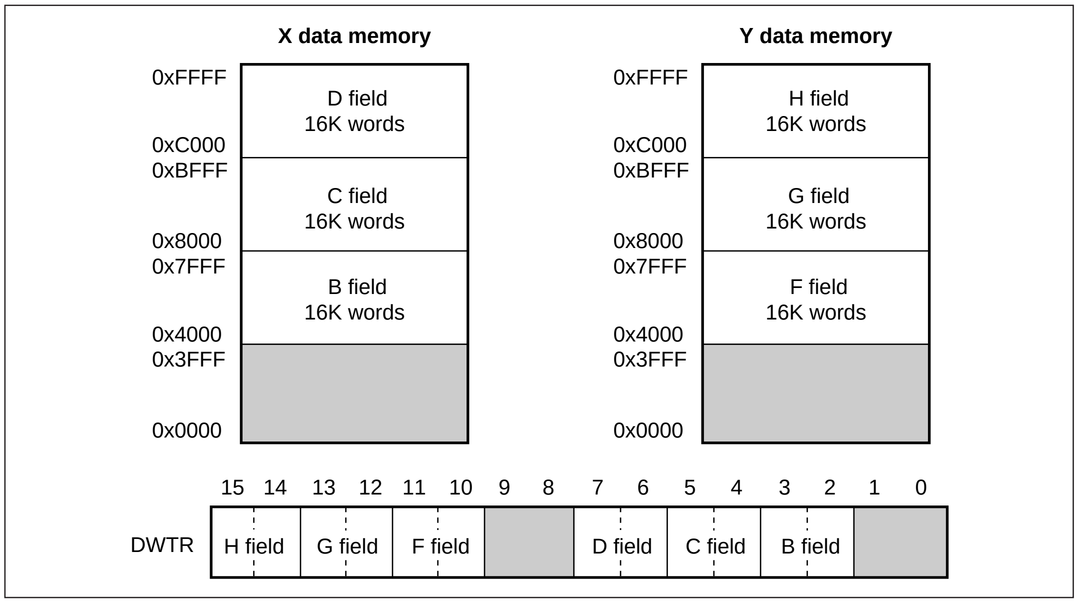
Bit	Name	Category	Load/store (L/S)	Bit function
15	BE	Bit manipulation	S	Bit manipulation enable bit • 0: Does not manipulate bit. • 1: Manipulates bit Manipulation method is specified by B1, B0, and PSR. • Undefined when read.
14	PSR	Bit manipulation	S	Port set/reset specification bit • 0: Reset (low level) • 1: Set (high level) • Manipulation port is specified by B1 and B0. • Valid when BE = 1. • Undefined when read.
13	ME	Mode setting	S	Mode setting enable bit • 0: Does not set mode. • 1: Sets mode. Contents to be set are specified by IO and M3-M0. • Undefined when read.
12	IO	Mode setting	S	Input/output specification bit • 0: Specifies input mode. • 1: Specifies output mode. • Port to be set is specified by M3-M0. • Valid when ME = 1. • Undefined when read.
11, 10	Reserved	—	—	Reserved bits • No value can be set to these bits. • Undefined when read.
9, 8	B1, B0	Bit manipulation	S	Bit manipulation port specification bits • B1, B0 = 00: P0 01: P1 10: P2 11: P3 • Set/reset is specified by PSR. • Valid when BE = 1. • Undefined when read.
7-4	Reserved	—	—	Reserved bits • No value can be set to these bits. • Undefined when read.

3-0	M3-M0	Mode setting	S	Mode setting port specification bits
				M3 = 0: P3 unselected, 1: P3 selected
				M2 = 0: P2 unselected, 1: P2 selected
				M1 = 0: P1 unselected, 1: P1 selected
				M0 = 0: P0 unselected, 1: P0 selected
				• Selection can be specified independently.

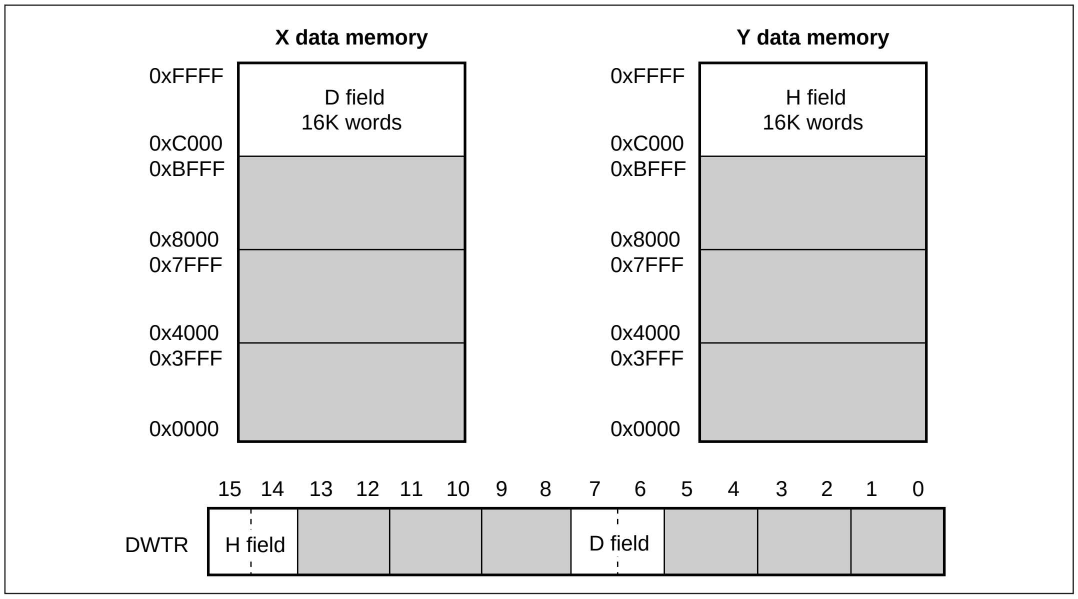
		Mode status	L	Input/output mode status bits
				M3 = 0: P3 input mode, 1: P3 output mode
				M2 = 0: P2 input mode, 1: P2 output mode
				M1 = 0: P1 input mode, 1: P1 output mode
				M0 = 0: P0 input mode, 1: P0 output mode

(8) Data Wait Cycle Register (DWTR)

(a) μPD77016



(b) μPD77015, 77017, 77018, 77018A, 77019



Caution

- **With the μPD77016, writing data to bits 9, 8, 1, and 0 is ignored. These bits are undefined when read.**
- **With the μPD77015, 77017, 77018, 77018A, and 77019 writing data to bits 13-8 and 5-0 is ignored. These bits are undefined when read.**

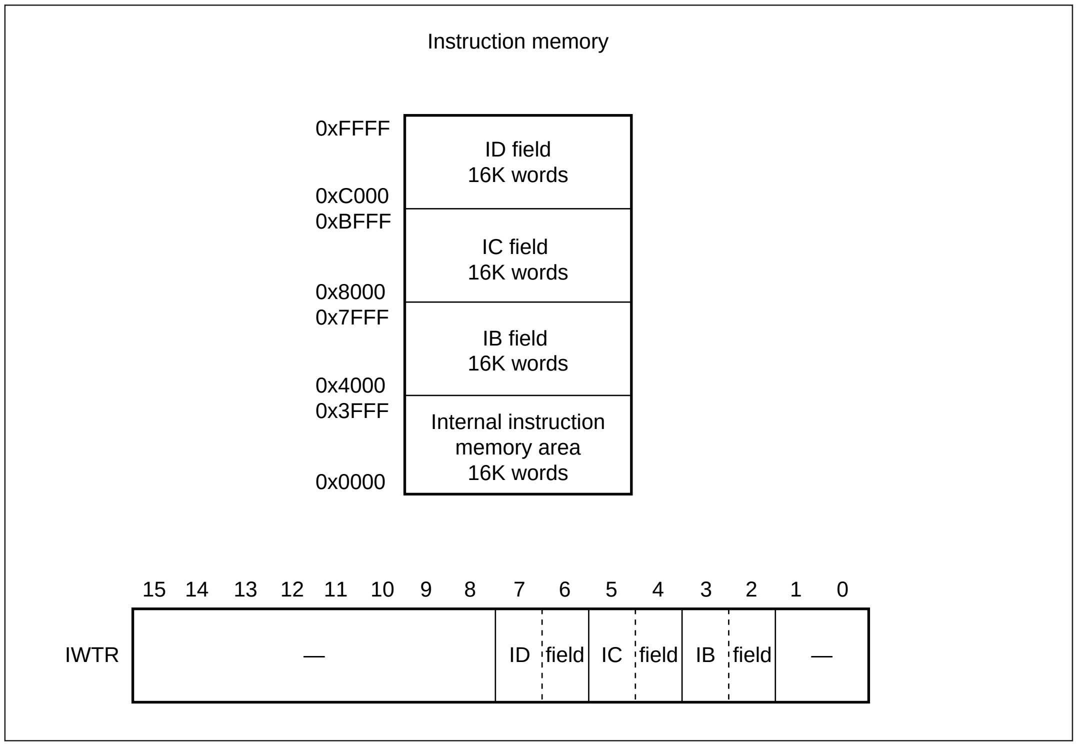
(c) Set Value of Field (Number of Wait Cycles)

Bit		Wait cycles	Remark
0	0	0	1-cycle access: SRAM etc. with an access time of about 8 ns is connected (at 33 MHz).
0	1	1	2-cycle access: SRAM etc. with an access time of about 35 ns is connected (at 33 MHz).
1	0	3	4-cycle access: SRAM etc. with an access time of about 85 ns is connected (at 33 MHz).
1	1	7	8-cycle access: Mask ROM etc. with an access time of about 150 ns is connected (at 33 MHz)

Caution When DWTR is set, the specified number of wait cycles becomes valid when an instruction immediately after the instruction that has set the data to DWTR.

Refer to section 3.5.2 “Data memory space”.

(9) Instruction Wait Cycle Register (IWTR)



Bits		No. of wait cycles	Remarks
0	0	0	1 cycle access: Connects SRAM with access time of 8 ns (at 33 MHz)
0	1	1	2 cycle access: Connects SRAM with access time of 35 ns (at 33 MHz)
1	0	3	4 cycle access: Connects SRAM with access time of 85 ns (at 33 MHz)
1	1	7	8 cycle access: Connects mask ROM with access time of 150 ns (at 33 MHz)

Cautions 1. Data written to bits 15 through 8, 1, and 0 are ignored. The contents of bits 15 through 8, 1, and 0 are undefined when they are read.

2. With the μPD77015, 77017, 77018, 77018A, and 77019, data written to IWTR is ignored, and undefined data is read from IWTR.

Refer to section 3.4.2 “Program execution control block”.

A.2 Interrupt Vector Table

Vector	Internal/external	Interrupt cause
0x200	Internal	Reset
0x204	—	Reserved
0x208	—	Reserved
0x20C	—	Reserved
0x210	External	INT1
0x214	External	INT2
0x218	External	INT3
0x21C	External	INT4
0x220	Internal	SI1 input
0x224	Internal	SO1 output
0x228	Internal	SI2 input
0x22C	internal	SO2 output
0x230	Internal	HI input
0x234	Internal	HO output
0x238	—	Reserved
0x23C	—	Reserved

★ A.3 CPU Registers to Be Initialized and Initial Values

Register name	Initial value	Remark
SR	0xF000	All the interrupts of the respective causes are enabled, and the interrupts are disabled at both the current and past levels. Also indicates that a loop instruction is not under execution.
PC	0	Address 0 is a boot area and execution branches to address 0x200 after boot processing. Therefore, the reset entry as the user area is at address 0x200.
SP	0	–
LC	0b1xxx xxxx xxxx xxxx	Indicates that a loop instruction is not under execution. The count value itself is undefined.
LSP	0	–
RC	0b1xxx xxxx xxxx xxxx	Indicates that a repeat instruction is not under execution. The count value itself is undefined.
EIR	0xFFFF	Indicates the interrupts are disabled at both the current and past levels.
ESR	0	–

★ A.4 Memory-Mapped Registers to Be Initialized and Initial Values

Register name	Initial value	Remark
SST1, SST2	0x0002	Serial interface is initialized as follows: • MSB first for both input and output • 16 bits long for both input and output • Does not use wait function for loading/storing SDT • Status transition mode • Clears error flag of SDT loading/storing • Enables data storing to SDT • No data is loaded from SDT
PCD	0x0000	I/O port is initialized as follows: • Bits are not manipulated • Mode is not set
HST	0x0301	Host interface is initialized as follows: • Does not use wait function for HDT access • Disables both HRE and HWE functions • Clears both UF0 and UF1 to zero • Clears error flag for host read/write • Clears error flag for HDT load/store • Disables reading from host • Enables writing from host

★ A.5 Pins to Be Initialized and Initial Status

Pin name	Initial value
$\overline{X}$/Y	High-level output[Note 1]
DA0-DA15	Low-level output[Note 1]
D0-D15	High impedance
IA0-IA15[Note 2]	Low-level output (high impedance during reset)
ID0-ID31[Note 2]	High impedance
$\overline{\text{PWR}}$[Note 2]	High-level output (high impedance during reset)
$\overline{\text{MRD}}$, $\overline{\text{MWR}}$, $\overline{\text{BSTB}}$	High-level output[Note 1]
SORQ1, SORQ2, SIAK1, SIAK2	Low-level output
SO1, SO2	High impedance
$\overline{\text{HRE}}$, $\overline{\text{HWE}}$	High-level output
P0-P3	Input status
TICE	Low-level output

Notes 1. These pins go into a high-impedance state when the bus is released ($\overline{\text{HOLDAK}}$ = 0). The bus is released even during reset by clearing $\overline{\text{HOLDRQ}}$ to 0.

2. μPD77016 only

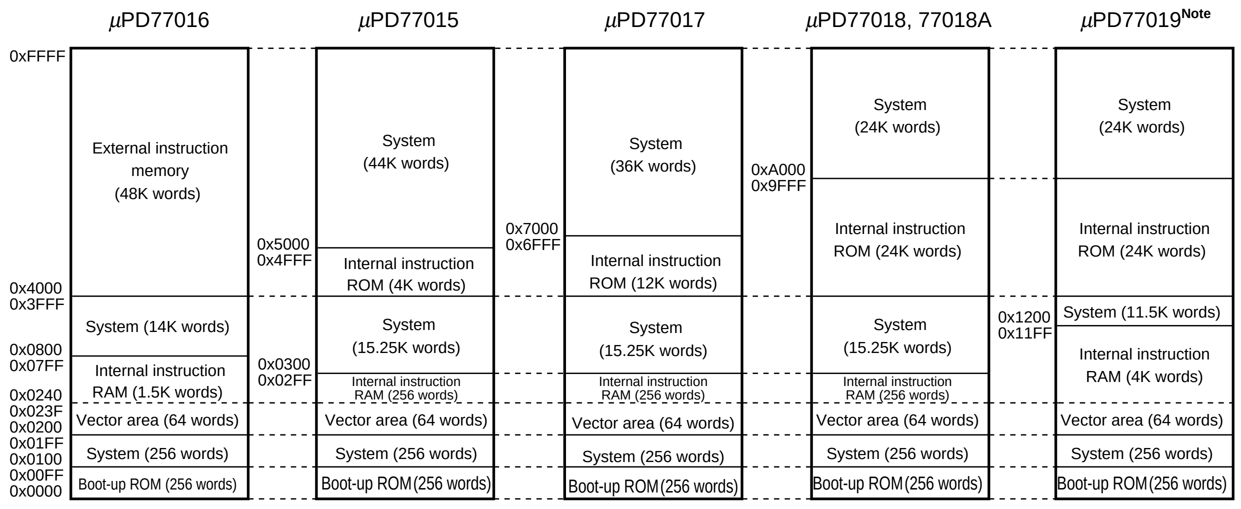
★ A.6 Status of Output Pins during Reset to Release STOP Mode

Pin name	Initialized status	Status during reset to release STOP mode
CLKOUT	System clock Note	Undefined Note
$\overline{X}$/Y	Low level	Undefined
DA0-DA13	0x0000	
D0-D15	High impedance	
$\overline{MRD}$	High level	
$\overline{MWR}$		
$\overline{BSTB}$		
$\overline{HOLDAK}$		
HD0-HD7	High impedance	
$\overline{HRE}$	High level	
$\overline{HWE}$		
SO1, SO2	High impedance	
SIAK1	Low level	
SORQ1		
P0-P3	Input status	

Note If CLKOUT is fixed to low level by mask option, CLKOUT is low in the initialized status and the status during reset to release the STOP mode.

A.7 Memory Map

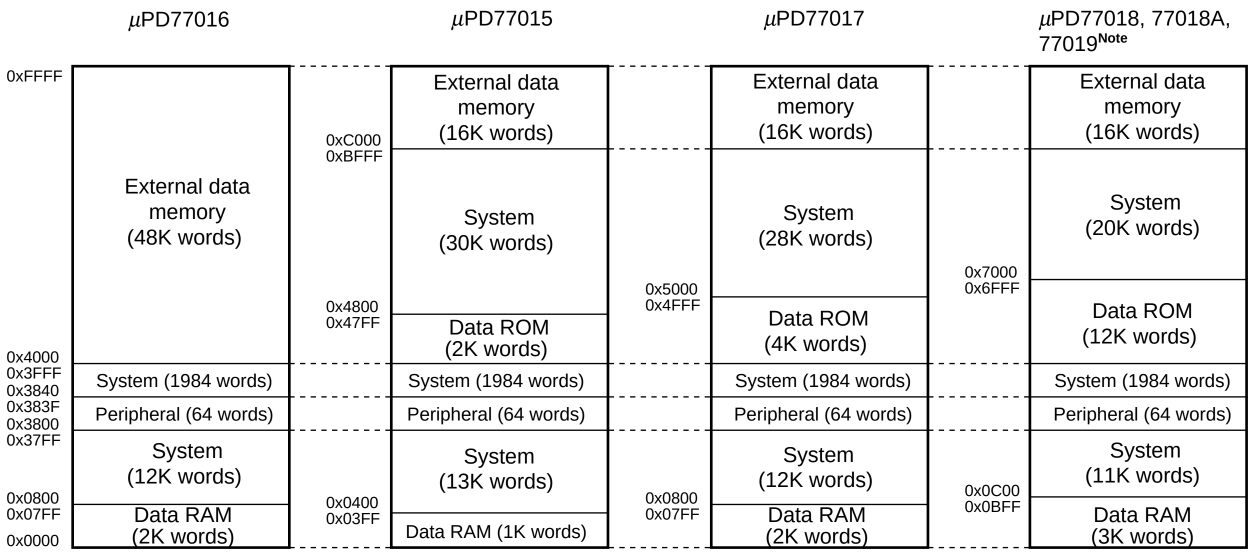
★ A.7.1 Instruction memory map



Caution **No program or data must be stored to the addresses reserved for the system, nor must these addresses be accessed. If any of these addresses is accessed, normal operation of the μPD7701x family is not guaranteed.**

Note The μPD77019-013 does not have the internal ROM of the μPD77019.

★ A.7.2 Data memory map (X/Y)



Caution **No program or data must be stored to the addresses reserved for the system, nor must these addresses be accessed. If any of these addresses is accessed, normal operation of the μPD7701x family is not guaranteed.**

Note The μPD77019-013 does not have the internal ROM of the μPD77019.

[MEMO]

Appendix B

Ordering Information

★ B.1 Ordering Information

Part Number	Package
μPD77016GM-KMD	160-pin plastic QFP (fine pitch) (24 × 24 mm)
μPD77015GC-xxx-9EU	100-pin plastic QFP (fine pitch) (14 × 14 mm)
μPD77017GC-xxx-9EU	100-pin plastic QFP (fine pitch) (14 × 14 mm)
μPD77018GC-xxx-9EU	100-pin plastic QFP (fine pitch) (14 × 14 mm)
μPD77018AGC-xxx-9EU	100-pin plastic QFP (fine pitch) (14 × 14 mm)
μPD77018AS9-xxx-YJC	116-pin plastic BGA (fine pitch) (12 × 12 mm)
μPD77019GC-xxx-9EU	100-pin plastic QFP (fine pitch) (14 × 14 mm)
μPD77019GC-013-9EU	100-pin plastic QFP (fine pitch) (14 × 14 mm)

Remark xxx indicates code suffix.

B.2 Mask Option

The μPD77015, 77017, 77018, 77018A, and 77019 are provided with a PLL and have the following mask option functions.

B.2.1 Disabling CLKOUT output

An internal system clock is generated from an external clock input to the X1 pin to serve as the internal basic timing of the device. This internal system clock is also output from the CLKOUT pin, but this output can be disabled by mask option.

B.2.2 Clock multiple

The external clock is multiplied by the PLL. The multiple can be set by mask option. Available frequency ratios of the external clock to the internal system clock are as follows. However, the multiplication rate of the mask option of the μPD77019-013 is fixed to 4 times.

- 1 (external): 1 (internal)
- 1 (external): 2 (internal)
- 1 (external): 3 (internal) (μPD77018A and 77019 only)
- 1 (external): 4 (internal)
- 1 (external): 8 (internal)

B.3 Mask ROM Ordering Format

For how to place your order for a mask ROM, refer to “WB77016 User’s Manual”.

Appendix C
Index

C.1 Key Words

C.2 Acronyms, etc.

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: 02-719-5951

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 98.2