To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

User's Manual

μPD78356

16-bit Singlechip Microcontrollers

Instructions

μPD78352A Subseries
μPD78356 Subseries
μPD78366A Subseries
μPD78372 Subseries

Document No. U12117EJ2V0UM00 (2nd edition)
(O.D.No. IEU-1395)
Date Published May 1997 N

Phase-out/Discontinued

[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.

Major Revisions In This Edition

Page	Description
Throughout	Adding the following products μPD78356(A), 78P356(A), 78361A, 78362A, 78P364A, 78363A, 78365A, 78366A, 78368A, 78P368A, 78372(A), 78372(A1), 78372(A2), 78P372(A), 78P372(A1), 78P372(A2)
	Deleting the following products μPD78355A, 78356A, 78P356A, 78362, 78P364, 78365, 78366, 78P368, 78370, 78372, 78P372
	Changing the status of the following products from developing to completed μPD78355, 78356, 78P356
p.329, 336, 341, 346 p.348	Adding the description of the debugging tools at the use of integrated debugger in **APPENDIX A TOOLS**. Adding Section **A.6 Embedded Software**
p.357	Adding **APPENDIX D REVISION HISTORY**

The mark ★ shows major revised points.

Phase-out/Discontinued

[MEMO]

PREFACE

Readers

This manual is intended for engineers of users who understand the functions of the 78K/III Series products and design application systems using the 78K/III Series.
The 78K/III Series covered in the manual includes the following products[Note 1]:

- μPD78352A Subseries : μPD78350, 78350A, 78352A, 78P352
- μPD78356 Subseries : μPD78355, 78356, 78P356, 78356(A), 78P356(A)
- μPD78366A Subseries : μPD78361A, 78362A, 78P364A, 78363A, 78365A, 78366A, 78368A, 78P368A
- μPD78372 Subseries[Note 2]: μPD78372(A), 78372(A1), 78372(A2), 78P372(A), 78P372(A1), 78P372(A2)

Notes 1. In addition to these products, the 78K/III Series provides the μPD78312A, 78322, 78328, and 78334 Subseries products. For details of each subseries, refer to their individual User's Manuals.

2. Other than the above members, the μPD78372 Subseries provides the μPD78P372KL-S of EPROM version which can be used only for experiment of function evaluation. However, this document explains the μPD78P372(A) as a representative product as long as no special differences exist.

Purpose

The purpose of the manual is for the user to understand the instruction functions of the 78K/III Series products.

Organization

The manual consists of the following main parts:

- General description
- Target product list
- CPU architecture
- Addressing
- Instruction set list
- Description of instructions
- Cautions on use

Cautions on use

Cautions regarding the use of the 78K/III Series instructions are collected in CHAPTER 7 CAUTIONS ON USE. Be sure to read this chapter.
Ask NEC or an NEC agent's salesperson for the latest information on the products.

How to read this manual

The manual assumes that the reader has general knowledge of electricity, logical circuits, and microcomputers.

- **If the products are the same in function**
  → This manual describes the μPD78356 as a typical product. To use this manual for other products, replace the μPD78356 with the appropriate product names in reading.

- **If the products differ in function**
  → Functional differences are discussed for each product.

- **To look up the instruction function in detail when you know the mnemonic of the instruction**
  → Use **APPENDIX B** and **APPENDIX C INSTRUCTION INDEX**.

- **To look up an instruction when you do not know its mnemonic, but know the rough function**
  → Look up the mnemonic of the instruction in **CHAPTER 5 INSTRUCTION SET LIST**, then the instruction function in **CHAPTER 6 EXPLANATION OF INSTRUCTIONS**.

- **To understand the instruction functions of the 78K/III Series products in a general way**
  → Read the manual in the sequence of the table of contents.

- **To learn the hardware functions of the 78K/III Series products**
  → Refer to the following user's manuals:

  - μPD78352A User's Manual (IEU-781)
  - μPD78356 User's Manual (U10669E)
  - μPD78362A User's Manual (U10745E)
  - μPD78366A User's Manual (U10205E)
  - μPD78372 User's Manual (U10642E)

Legend

Data representation weight	:	High-order and low-order digits are indicated from left to right.
Active low representation	:	$\overline{\times\times\times}$ (pin or signal name is overlined)
Memory map address	:	Low-order part at upper stage, high-order part at lower stage
Note	:	Explanation of (Note) in the text
Caution	:	Refers to contents that warrant special attention
Remark	:	Supplementary explanation to the text
Number representation	:	Binary number ... ××××B or ××××
		Decimal number ... ××××
		Hexadecimal number ... ××××H

Phase-out/Discontinued

★ **Related documents** The related documents indicated in this publication may include preliminary version. However, preliminary version are not marked as such.

- **Common Documents**

Document Name	Document No.	
	Japanese	English
μPD78356 User's Manual, Instruction	U12117J	IEU-1395
78K/III Series Application Note, Software Fundamental	U12118J	IEA-1272
78K/III Series Application Note, Floating Point operation Programming	U12119J	IEA-1291
μPD78352A Instruction Set	U11955J	–

- **Individual Documents**

μPD78352A Subseries

Document Name	Document No.	
	Japanese	English
μPD78352A Product Letter	IF-6335	IF-2036
μPD78350 Data Sheet	IC-8279	IC-2845
μPD78350A, 78352A Data Sheet	IC-8823	IC-3391
μPD78P352 Data Sheet	IC-8423	IC-2957
μPD78352A User's Manual Hardware	IEU-781	IEU-1327
μPD78352A Special Function Register Table	IEM-5540	IEM-1215

μPD78356 Subseries

Document Name	Document No.	
	Japanese	English
μPD78356 Product Letter	IF-6298	–
μPD78355, 78356 Data Sheet	U10155J	U10155E
μPD78P356 Data Sheet	U10325J	U10325E
μPD78356(A) Data Sheet	U11148J	U11148E
μPD78P356(A) Data Sheet	U11149J	U11149E
μPD78356 User's Manual, Hardware	U10669J	U10669E
μPD78356 Special Function Register Table	IEM-5576	IEM-1214

Caution The contents of the documents listed above are subject to change without prior notice to user's. Be sure to use the latest edition when starting design.

Phase-out/Discontinued

μPD78366A Subseries

Document Name	Document No.	
	Japanese	English
μPD78362A Data Sheet	U10098J	U10098E
μPD78P364A Data Sheet	U10106J	U10106E
μPD78363A, 78365A, 78366A Data Sheet	U11109J	U11109E
μPD78P368A Data Sheet	U11373J	U11373E
μPD78362A User's Manual, Hardware	U10745J	U10745E
μPD78366A User's Manual, Hardware	U10205J	U10205E
μPD78362A Special Function Register Table	U10210J	–
μPD78366A Special Function Register Table	U10107J	–

μPD78372 Subseries

Document Name	Document No.	
	Japanese	English
μPD78372 Product letter	IF-6351	–
μPD78370(A), 78372(A) Data Sheet	U10789J	U10789E
μPD78P372(A) Data Sheet	U12029J	U12029E
μPD78372 User's Manual, Hardware	U10642J	U10642E
μPD78372 Special Function Register Table	U10631J	U10631E

Caution The contents of the documents listed above are subject to change without prior notice to user's. Be sure to use the latest edition when starting design.

CONTENTS

CHAPTER 1 GENERAL DESCRIPTION 1

1.1 78K/III Series Product Development 2

1.2 μPD78352A Subseries Products Overview 3

1.2.1 Features 3
1.2.2 Application fields 3
1.2.3 Ordering information and quality grade 3
1.2.4 Function outline 4
1.2.5 Block diagram 5

1.3 Outline of μPD78356 Subseries Products 6

1.3.1 Features 6
1.3.2 Application fields 6
1.3.3 Ordering information and quality grade 7
1.3.4 Function outline 8
1.3.5 Block diagram 10

1.4 μPD78366 Subseries Products Overview 11

1.4.1 Features 11
1.4.2 Application fields 11
1.4.3 Ordering information and quality grade 12
1.4.4 Function outline (μPD78361A, 78362A, 78P364A) 13
1.4.5 Function outline (μPD78363A, 78365A, 78366A, 78368A, 78P368A) 15
1.4.6 Block diagram (μPD78361A, 78362A, 78P364A) 17
1.4.7 Block diagram (μPD78363A, 78365A, 78366A, 78368A, 78P368A) 18

1.5 μPD78372 Subseries Products Overview 19

1.5.1 Features 19
1.5.2 Application fields 19
1.5.3 Ordering information and quality grade 20
1.5.4 Function outline 21
1.5.5 Block diagram 22

CHAPTER 2 TARGET PRODUCT LIST 23

CHAPTER 3 CPU ARCHITECTURE 29

3.1 Memory Space 29

3.1.1 Vector table area 33
3.1.2 CALLT instruction table area 34
3.1.3 CALLF instruction entry area 34
3.1.4 Internal RAM area 34
3.1.5 Special function register area 39
3.1.6 External memory area 39

3.2 Processor Registers 40

3.2.1 Control registers 41
3.2.2 General-purpose registers 45
3.2.3 Special function registers (SFRs) 47

3.3 Data Memory Addressing 48
3.3.1 General-purpose register addressing 48
3.3.2 Short direct addressing 49
3.3.3 Special function register (SFR) addressing 49
3.4 Interrupt Function 50
3.4.1 Interrupt request types 51
3.4.2 Interrupt processing modes 53
3.4.3 Macro service function 54
3.4.4 Context switching function 55
3.4.5 Interrupt execution rates 56
3.4.6 Control registers 57

CHAPTER 4 ADDRESSING 59
4.1 Instruction Addressing 59
4.1.1 Relative addressing 59
4.1.2 Immediate addressing 60
4.1.3 Table indirect addressing 61
4.1.4 Register addressing 61
4.1.5 Register indirect addressing 62
4.2 Operand Addressing 63
4.2.1 Register addressing 63
4.2.2 Immediate addressing 64
4.2.3 Direct addressing 65
4.2.4 Short direct addressing 65
4.2.5 Special function register (SFR) addressing 67
4.2.6 Short direct memory indirect addressing 68
4.2.7 Register indirect addressing 69
4.2.8 Based addressing 70
4.2.9 Indexed addressing 71
4.2.10 Based indexed addressing 72

CHAPTER 5 INSTRUCTION SET LIST 73
5.1 List of Operations 73
5.1.1 Operand identifier and description 73
5.1.2 Legend 75
5.1.3 Notational symbols in flag operation field 76
5.1.4 Instruction set differences among 78K/III Series products 77
5.1.5 Operations of basic instructions 78
5.2 Instruction Codes 93
5.2.1 Symbols of instruction codes 93
5.2.2 Instruction codes in various memory addressing modes 96
5.2.3 List of instruction codes 97
5.3 Number of Clocks of the Instructions 113
5.3.1 Description of clock columns 113
5.3.2 Numbers of clocks 115

CHAPTER 6 EXPLANATION OF INSTRUCTIONS 137
6.1 8-Bit Data Transfer Instructions 139
6.2 16-Bit Data Transfer Instructions 142
6.3 8-Bit Operation Instructions 145
6.4 16-Bit Operation Instructions 156
6.5 Multiplication and Division Instructions 160
6.6 Signed Multiplication Instruction 165
6.7 Multiplication and Accumulation Instruction 167
6.8 Multiplication and Accumulation Instruction With Saturation Function 170
6.9 Correlation Operation Instruction 173
6.10 Table Shift Instruction 176
6.11 Increment and Decrement Instructions 178
6.12 Shift and Rotate Instructions 183
6.13 BCD Adjustment Instruction 194
6.14 Data Conversion Instruction 197
6.15 Bit Manipulation Instructions 199
6.16 Call and Return Instructions 207
6.17 Stack Handling Instructions 215
6.18 Special Instructions 223
6.19 Unconditional Branch Instruction 226
6.20 Conditional Branch Instructions 228
6.21 Context Switching Instructions 248
6.22 String Instructions 252
6.23 CPU Control Instructions 273

CHAPTER 7 CAUTIONS ON USE
7.1 Cautions on CHAPTER 3 CPU ARCHITECTURE 283
7.2 Cautions on CHAPTER 5 INSTRUCTION SET LIST 284
7.3 Cautions on CHAPTER 6 EXPLANATION OF INSTRUCTIONS 284

APPENDIX A TOOLS 285
A.1 78K/III Series Common Tools 285
A.2 μPD78352A Subseries Tools 286
A.3 μPD78356 Subseries Tools 292
A.4 μPD78366A Subseries Tools 298
A.5 μPD78372 Subseries Tools 303
★ A.6 Embedded Software 308

APPENDIX B INSTRUCTION INDEX (MNEMONICS BY FUNCTION) 311

APPENDIX C INSTRUCTION INDEX (MNEMONICS BY ALPHABETICAL ORDER) 313

★ APPENDIX D REVISION HISTORY 315

LIST OF FIGURES

Figure No.	Title	Page
1-1	Configurations of 78K Series and 78K/III Series	1
3-1	Memory Map	30
3-2	Register Configuration	40
3-3	Format of Program Status Word (PSW)	41
3-4	Format of CPU Control Word	44
3-5	Process Bits of General-Purpose Registers	45
3-6	Data Memory Addressing	48
3-7	Handling Interrupt Requests	50
3-8	Process Flow of Maskable Interrupt	52
3-9	Macro Service Process Sequence Example	54
3-10	Context Switching Operation when an Interrupt Request Occurs	55
4-1	Relative Addressing	59
4-2	Immediate Addressing	60
4-3	Table Indirect Addressing	61
4-4	Register Addressing	61
4-5	Register Indirect Addressing	62
4-6	Short Direct Addressing	66
4-7	Special Function Register Addressing	67
4-8	Short Direct Memory Indirect Addressing	68
5-1	8-bit Data that Specifies the Register Pair which Performs Stack Operations	94

LIST OF TABLE

Table No.	Title	Page
3-1	Vector Table Area	33
3-2	Internal RAM Area List	35
3-3	Word Access Operation in Internal RAM Area	35
3-4	External Memory Area List	39
3-5	Configuration of General-Purpose Registers	46
3-6	Interrupt Requests and Processing Modes	50
3-7	Control Register List (μPD78352A Subseries)	57
3-8	Control Register List (μPD78356 Subseries)	57
3-9	Control Register List (μPD78366A Subseries)	57
3-10	Control Register List (μPD78372 Subseries)	57
5-1	Operand Identifier and Description	73
5-2	Absolute Names and Their Corresponding Function Names of an 8-bit Register	74
5-3	Absolute Names and Their Corresponding Function Names of a 16-bit Register	74
5-4	Notational Symbols in Flag Operation Field	76
5-5	Instruction Set Differences among 78K/III Series Products	77
5-6	mod and mem Codes in the Instruction Code Field	96
5-7	Instruction Execution Cycles	132

[MEMO]

CHAPTER 1 GENERAL DESCRIPTION

The 78K Series consists of six series as shown in Figure 1-1.

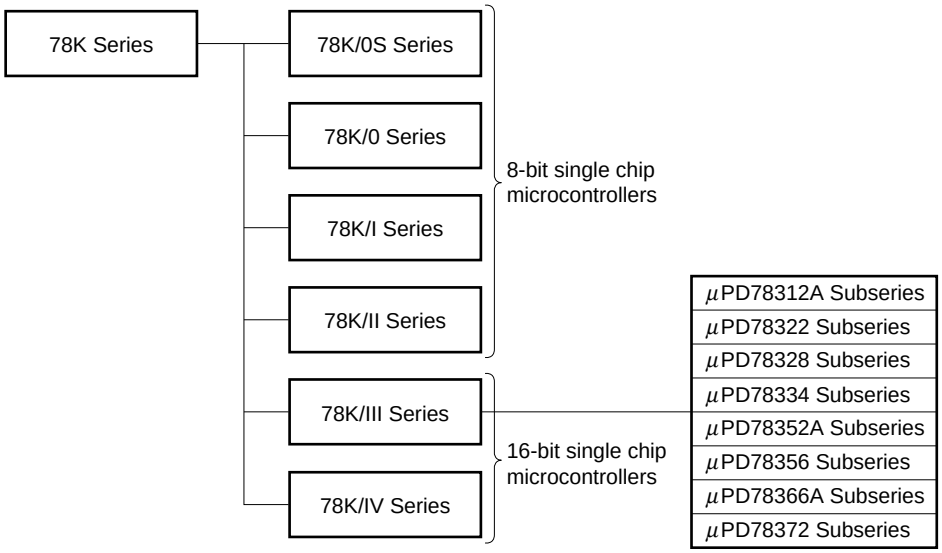
The 78K/III Series, which is one of the six series, provides products each containing a 16-bit CPU.

The on-chip CPU is a high-function CPU which has an instruction set and a high-speed interrupt controller appropriate for control application and comprises a memory space of a maximum of 64K bytes.

The 78K/III Series comprises eight subseries: μPD78312A, 78322, 78328, 78334, 78352A, 78356, 78366A, and 78372, from which the user can select the most appropriate subseries according to the application.

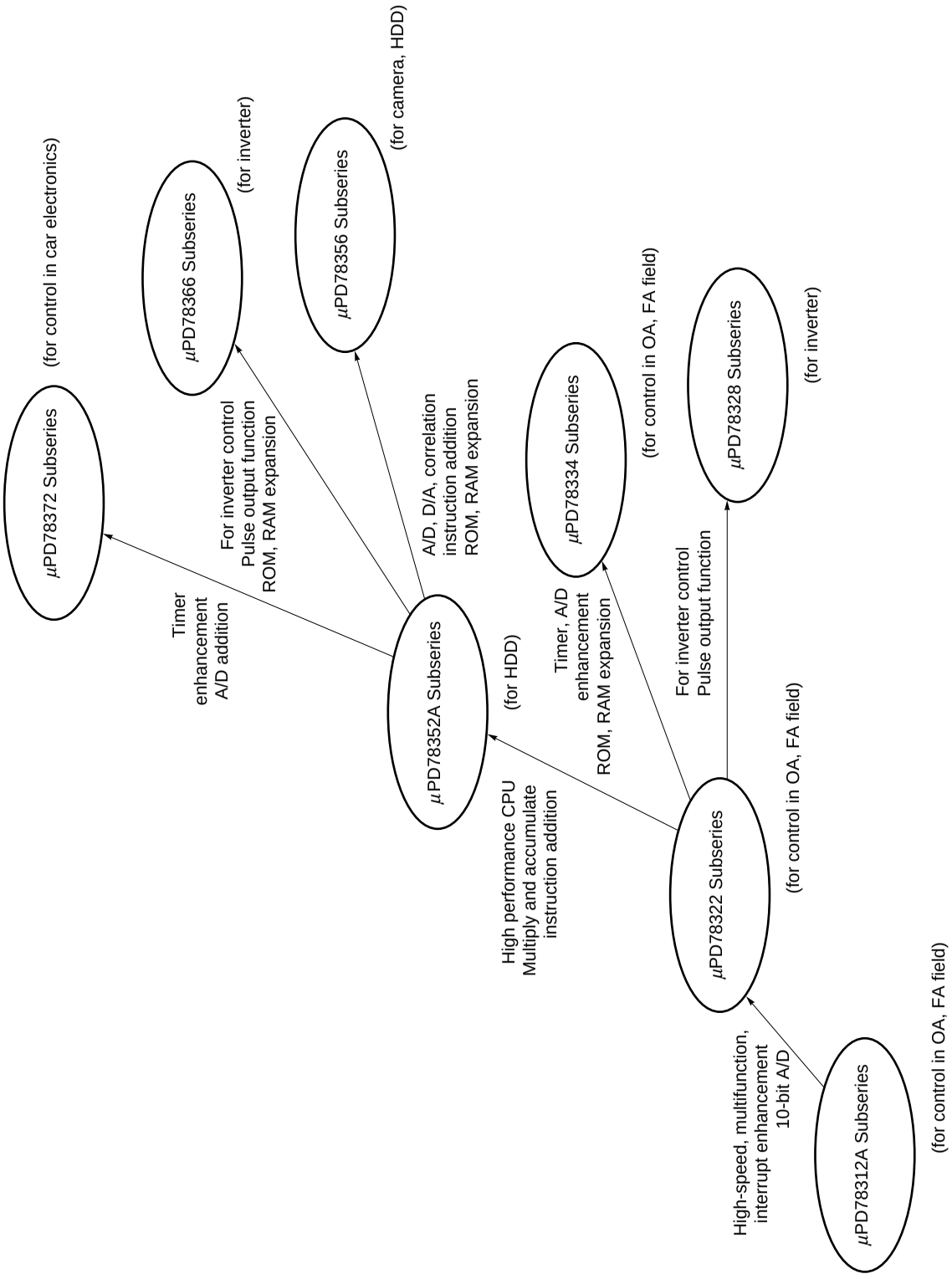
This manual covers only the four subseries of μPD78352A, 78356, 78366A, and 78372, which differ only in peripheral hardware and have the same CPU. Therefore, they have a common instruction set except that the μPD78352A Subseries does not have a multiply and accumulate instruction with a saturation function or a correlation operation instruction.

★ **Figure 1-1. Configurations of 78K Series and 78K/III Series**



Phase-out/Discontinued

1.1 78K/III Series Product Development



1.2 μPD78352A Subseries Products Overview

Applicable products: μPD78350, 78350A, 78352A, 78P352

1.2.1 Features

- ○ Internal 16-bit architecture, external 8-bit data bus
- ○ Pipeline control system and high-speed operation clock for high-speed processing
  Minimum instruction execution time: 160 ns (internal clock: 12.5 MHz, external clock: 25 MHz)
  ...μPD78350
  125 ns (internal clock: 16 MHz, external clock: 32 MHz)
  ...μPD78350A, 78352A, 78P352
- ○ Instruction set of 113 types of instructions appropriate for control application
- ○ Bus cycle wait control can be performed externally (except for μPD78350)
- ○ 8-bit resolution PWM signal output function: Two channels
- ○ Internal high-speed interrupt controller
- ○ Internal memory: 32-Kbyte ROM (μPD78352A)
  None (μPD78350, 78350A)
  32-Kbyte PROM (μPD78P352)
  640-byte RAM

1.2.2 Application fields

- ○ Office automation (OA) field such as HDD and FDD
- ○ Factory automation (FA) field

1.2.3 Ordering information and quality grade

(1) Ordering information

Part number	Package	Internal ROM
μPD78350GC-3BE	64-pin plastic QFP (14 × 14 mm) (resin thickness 2.7 mm)	None
μPD78350AG-22	64-pin plastic QFP (14 × 14 mm) (resin thickness 1.5 mm)	None
μPD78352AG-×××-22	64-pin plastic QFP (14 × 14 mm) (resin thickness 1.5 mm)	Mask ROM
μPD78P352G-22	64-pin plastic QFP (14 × 14 mm) (resin thickness 1.5 mm)	One-time PROM
μPD78P352KK-S**Note**	64-pin ceramic WQFN	EPROM

Note Under development

Remark ××× is a ROM code number.

(2) Quality grade

Standard

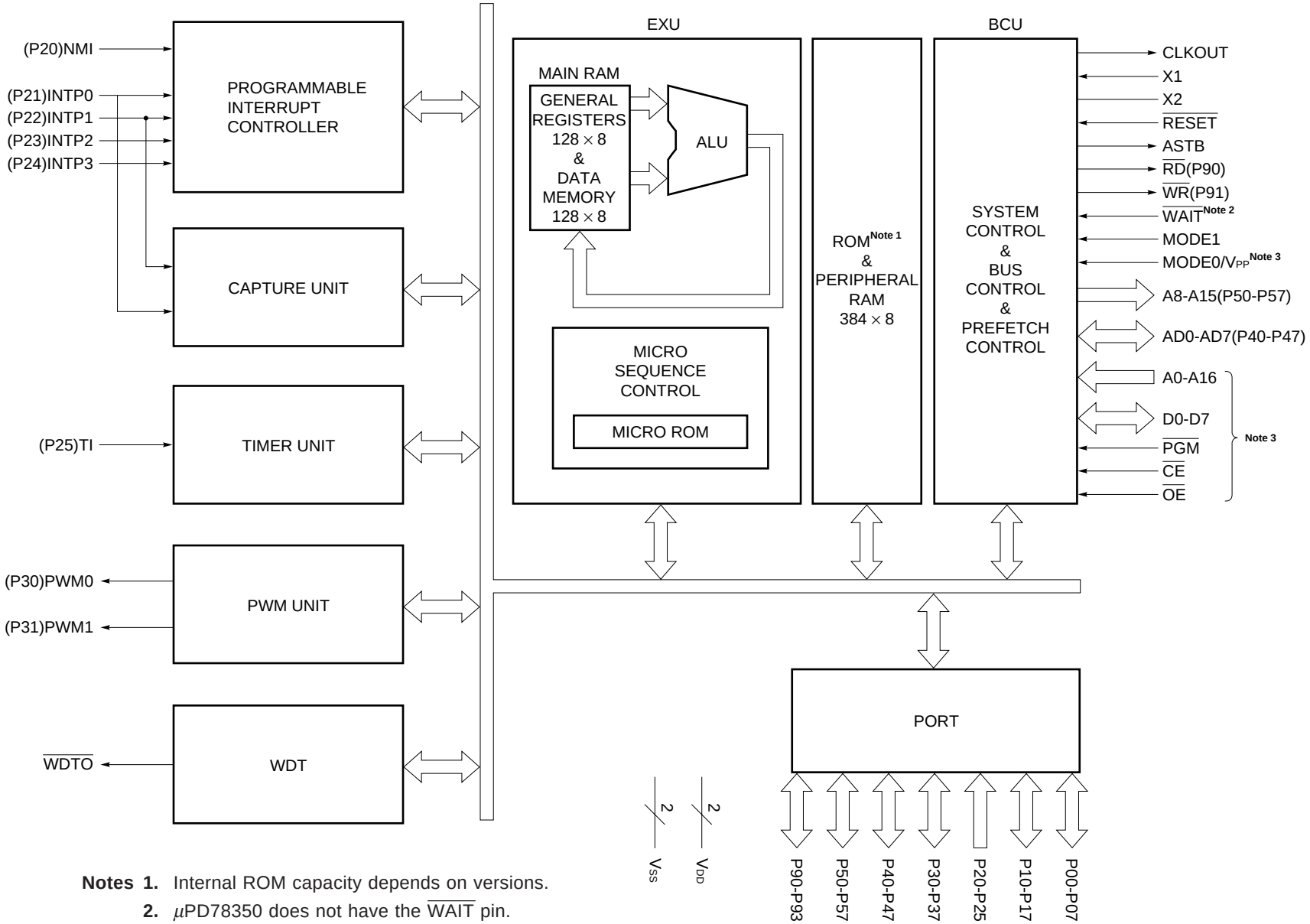
Please refer to "Quality Grades on NEC Semiconductor Devices" (Document No. C11531E) published by NEC Corporation to know the specification of quality grade on the devices and its recommended applications.

1.2.4 Function outline

Parameter \ Part number		μPD78350	μPD78350A	μPD78352A	μPD78P352
Minimum instruction execution time		160 ns [internal clock: 12.5 MHz, external clock: 25.0 MHz]	125 ns [internal clock: 16 MHz, external clock: 32 MHz]		
Internal memory	ROM	–		32 Kbytes	–
	PROM	–		–	32 Kbytes
	RAM	640 bytes			
Memory space		64 Kbytes (external expansion is enabled)			
General-purpose registers		8 bits × 16 registers × 8 banks			
No. of basic instructions		113			
Instruction set		• 16-bit transfer and operations • Multiplication and division (16 bits × 16 bits, 32 bits ÷ 16 bits) • Bit manipulation • String • Multiply and accumulate (16 bits × 16 bits + 32 bits)			
Input/output lines	Input	6			
	I/O	24		44	
Capture/timer unit		• 16-bit timer × 1 16-bit capture register × 2 • 16-bit timer × 1 16-bit compare register × 1 • 16-bit timer × 1 16-bit compare register × 1			
PWM unit		8-bit resolution PWM output: Two channels			
Interrupt function		• External: 5, internal: 4 • Four priority levels can be specified by software • One of three interrupt service modes can be selected (vectored interrupt, macro service, or context switching)			
External wait pin		None	External bus cycle wait control is enabled		
Packages	Without window	64-pin plastic QFP (14 × 14 mm) (resin thickness 2.7 mm)	64-pin plastic QFP (14 × 14 mm) (resin thickness 1.5 mm)		
	With window	–			64-pin ceramic WQFN[Note]
Others		• Watchdog timer is contained • Standby function (HALT mode, STOP mode)			

Note Under development

1.2.5 Block diagram



Notes 1. Internal ROM capacity depends on versions.

2. μPD78350 does not have the $\overline{\text{WAIT}}$ pin.

3. In PROM programming mode of μPD78P352.

1.3 Outline of μPD78356 Subseries Products

Applicable products: μPD78355, 78356, 78P356, 78355A, 78356A, 78P356A

1.3.1 Features

- ○ Internal 16-bit architecture, external 16-bit or 8-bit data bus
- ○ Pipeline control system and high-speed operation clock for high-speed processing
  Minimum instruction execution time: 125 ns (internal clock: 16 MHz, external clock: 32 MHz)
  ...μPD78355, 78356, 78P356
★ 160 ns (internal clock: 12.5 MHz, external clock: 25 MHz)
  ...μPD78356(A), 78P356(A)
- ○ Instruction set of 115 types of instructions appropriate for control application
- ○ Real-time pulse unit provided with various timer/counters
- ○ Ultra high-speed 10-bit resolution A/D converter: 8 channels
  A/D conversion time: About 2 μs at 32-MHz operation for μPD78355, 78356, and 78P356
★ About 2 μs at 25-MHz operation for μPD78356(A), 78P356(A)
- ○ 8-bit resolution D/A converter: Two channels
- ○ 8-, 10-, 12-bit resolution variable PWM signal output function: Two channels
- ○ Three independent channels of serial interface (containing dedicated baud rate generator)
- ○ Internal high-speed interrupt controller
- ○ Internal ECC circuit (μPD78P356 and 78P356(A))
  Internal PROM contents can be made highly reliable
- ○ Internal memory: 48-Kbyte ROM (μPD78356, 78356(A))
  None (μPD78355)
  48-Kbyte PROM (μPD78P356, 78P356(A))
  2-Kbyte RAM

1.3.2 Application fields

- **Standard**
  - ○ For high-speed servo control of HDD, FDD, etc.
  - ○ For auto focus control of camcorder, single-lens reflex camera, etc.
  - ○ For motor control in factory automation (FA) field

★ • **Special**
  - ○ For car electronics controller

1.3.3 Ordering information and quality grade

★ **(1) μPD78355, 78356, 78P356**

- **Ordering information**

Part number	Package	Internal ROM
μPD78355GC-7EA	100-pin plastic QFP (14 × 14 mm)	None
μPD78355GD-5BB	120-pin plastic QFP (28 × 28 mm)	None
μPD78356GC-×××-7EA	100-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD78356GD-×××-5BB	120-pin plastic QFP (28 × 28 mm)	Mask ROM
μPD78P356GC-7EA	100-pin plastic QFP (14 × 14 mm)	One-time PROM
μPD78P356GD-5BB	120-pin plastic QFP (28 × 28 mm)	One-time PROM
μPD78P356KP-S	120-pin ceramic WQFN	EPROM

Remark ××× is a ROM code number.

- **Quality grade**
  Standard

Please refer to "Quality Grades on NEC Semiconductor Devices" (Document No. C11531E) published by NEC Corporation to know the specification of quality grade on the devices and its recommended applications.

★ **(2) μPD78356(A), 78P356(A)**

- **Ordering information**

Part number	Package	Internal ROM
μPD78356GD(A)-×××-5BB	120-pin plastic QFP (28 × 28 mm)	Mask ROM
μPD78P356GD(A)-5BB	120-pin plastic QFP (28 × 28 mm)	One-time PROM

Remark ××× is a ROM code number.

- **Quality grade**
  Special

Please refer to "Quality Grades on NEC Semiconductor Devices" (Document No. C11531E) published by NEC Corporation to know the specification of quality grade on the devices and its recommended applications.

1.3.4 Function outline

(1/2)

★

Parameter \ Part number		μPD78355	μPD78356	μPD78P356	μPD78356(A)	μPD78P356(A)
Minimum instruction execution time		125 ns [internal clock: 16 MHz, external clock: 32 MHz]			160 ns [internal clock: 12.5 MHz external clock: 25 MHz]	
Internal memory	ROM	–	48 Kbytes	–	48 Kbytes	–
	PROM	–	–	48 Kbytes	–	48 Kbytes
	RAM	2 Kbytes				
Memory space		64 Kbytes (external expansion is enabled)				
General-purpose registers		8 bits × 16 registers × 8 banks				
No. of basic instructions		115				
Instruction set		• 16-bit transfer and operations • Multiplication and division (16 bits × 16 bits, 32 bits ÷ 16 bits) • Bit manipulation • String • Multiply and accumulate (16 bits × 16 bits + 32 bits) • Correlation operation				
Input/output lines	Input	9 (eight lines are also used for analog input)				
	I/O	48	67			
Real-time pulse unit		• 16-bit timer × 1 16-bit compare register × 4 16-bit capture/compare register × 3 timer output × 6 • 16-bit timer × 1 16-bit compare register × 2 timer output × 2 • 16-bit timer × 1 16-bit compare register × 2 timer output × 2 • 16-bit timer × 1 16-bit capture/compare register × 2 • 10-bit timer × 1 10-bit compare register × 1 • 16-bit up/down counter × 1 16-bit compare register × 2				
Real-time output port		Pulse output associated with real-time pulse unit: Eight				
PWM unit		8-, 10-, 12-bit resolution variable PWM output: Two channels				
A/D converter		10-bit resolution, eight channels				
D/A converter		8-bit resolution, two channels				
Serial interface		With dedicated baud rate generator UART: One channel Clocked serial interface/SBI: One channel Clocked serial interface (with pin change function): One channel				
Interrupt function		• External: 6, internal: 25 (five are also used for external interrupts) • Four priority levels can be specified by software • One of three interrupt service modes can be selected (vectored interrupt, macro service, or context switching)				

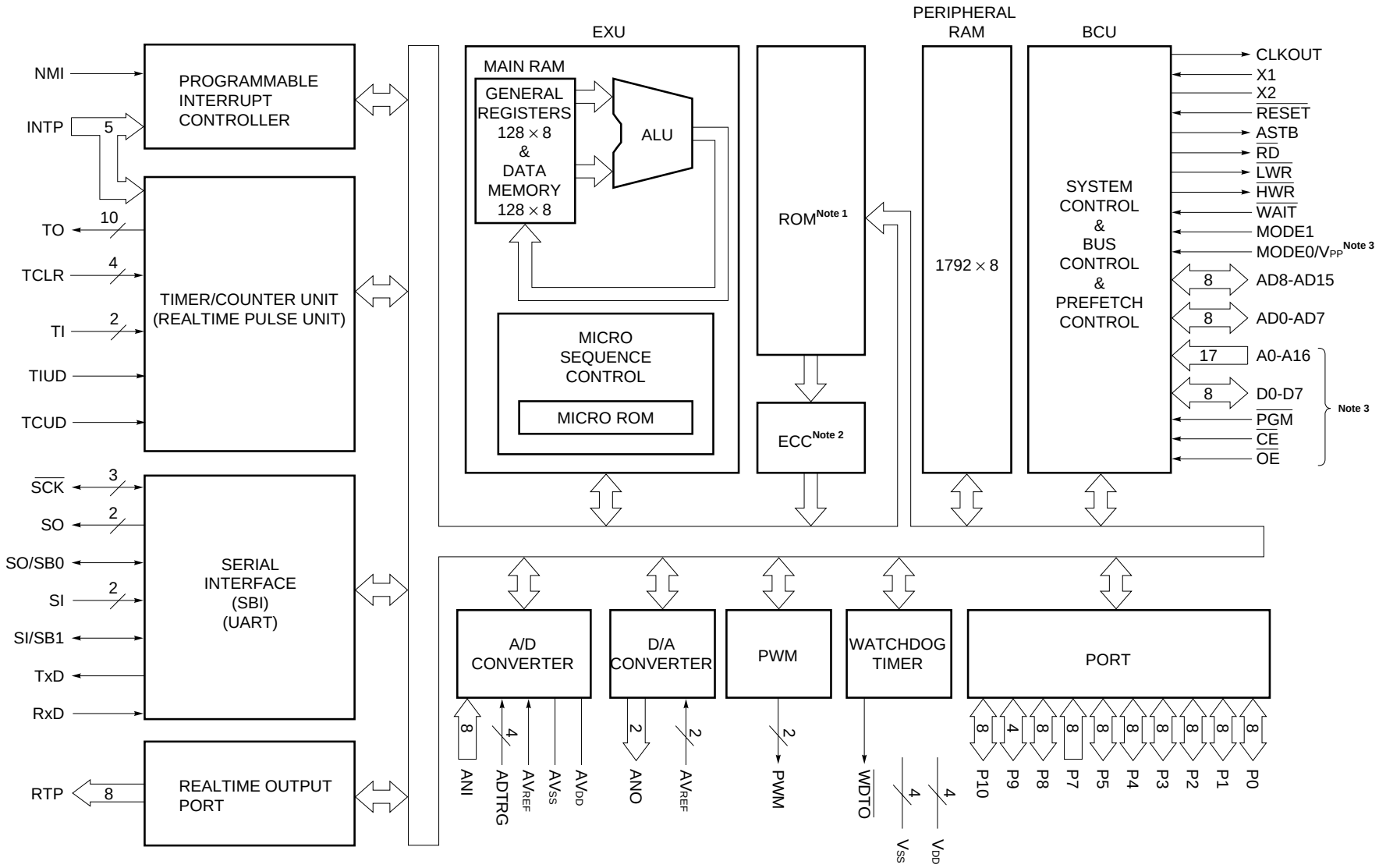
Phase-out/Discontinued

(2/2)

★

Parameter \ Part number		μPD78355	μPD78356	μPD78P356	μPD78356(A)	μPD78P356(A)
Bus sizing function		8-bit or 16-bit external data bus width can be selected				
ECC circuit		None		Available	None	Available
Operating power supply voltage		5 V ± 10 %				
Packages	Without window	• 100-pin plastic QFP (14 × 14 mm) • 120-pin plastic QFP (28 × 28 mm)			• 120-pin plastic QFP (28 × 28 mm)	
	With window	–		120-pin ceramic WQFN	–	
Others		• Watchdog timer is contained • Standby function (HALT mode, STOP mode)				

1.3.5 Block diagram



Notes 1. Internal ROM capacity depends on versions.

2. Only μPD78P356 and μPD78P356(A) contain the ECC circuit.

3. In PROM programming mode of μPD78P356 and 78P356(A).

1.4 μPD78366 Subseries Products Overview

★ **Applicable products: μPD78361A, 78362A, 78P364A, 78363A, 78365A, 78366A, 78368A, 78P368A**

1.4.1 Features

- ○ Internal 16-bit architecture, external 8-bit data bus
- ○ Pipeline control system and high-speed operation clock for high-speed processing
  Minimum instruction execution time: 125 ns (internal clock: 16 MHz, external clock: 8 MHz)
- ○ Internal PLL control circuit (external 8 MHz → internal 16 MHz)
- ○ Instruction set of 115 types of instructions appropriate for control application
- ○ Real-time pulse unit appropriate for inverter control of 10-bit resolution A/D converter: Eight channels
- ○ 8-, 9-, 10-, 12-bit resolution variable PWM signal output function: Two channels
- ○ Two independent channels of serial interface (containing dedicated baud rate generator)
- ○ Internal high-speed interrupt controller
- ○ Internal memory: 24-Kbyte ROM (μPD78362A, 78363A)
  32-Kbyte ROM (μPD78361A, 78366A)
  48-Kbyte ROM (μPD78368A)
  None (μPD78365A)
  48-Kbyte PROM (μPD78P364A, 78P368A)
  768-byte RAM (μPD78362A, 78363A)
  2-Kbyte RAM (μPD78361A, 78365A, 78366A, 78368A, 78P364A, 78P368A)

1.4.2 Application fields

- ○ Inverter air conditioners
- ○ Factory automation (FA) field of robots, automatic machine tools, etc.

1.4.3 Ordering information and quality grade

(1) Ordering information

Part number	Package	Internal ROM
μPD78361ACW-×××[Note]	64-pin plastic shrink DIP (750 mil)	Mask ROM
μPD78362ACW-×××	64-pin plastic shrink DIP (750 mil)	Mask ROM
μPD78P364ACW	64-pin plastic shrink DIP (750 mil)	One-time PROM
μPD78363AGF-×××-3B9	80-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD78365AGF-3B9	80-pin plastic QFP (14 × 20 mm)	None
μPD78366AGF-×××-3B9	80-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD78368AGF-×××-3B9[Note]	80-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD78P368AGF-3B9	80-pin plastic QFP (14 × 20 mm)	One-time PROM
μPD78P368AKL-S	80-pin ceramic WQFN	EPROM

Note Under development

Remark ××× is a ROM code number.

(2) Quality grade

Standard

Please refer to "Quality Grades on NEC Semiconductor Devices" (Document No. C11531E) published by NEC Corporation to know the specification of quality grade on the devices and its recommended applications.

1.4.4 Function outline (μPD78361A, 78362A, 78P364A)

(1/2)

Parameter \ Part number		μPD78361A[Note]	μPD78362A	μPD78P364A
Minimum instruction		125 ns (internal clock: 16 MHz, external clock: 8 MHz)		
Internal memory	ROM	32 Kbytes	24 Kbytes	–
	PROM	–	–	48 Kbytes
	RAM	2 Kbytes	768 bytes	2 Kbytes
ROMless mode		None		
Memory space		64 Kbytes (external expansion disabled)		
General-purpose register		8 bits × 16 registers × 8 banks		
No. of basic instructions		115		
Instruction set		• 16-bit transfer and operations • Multiplication and division (16 bits × 16 bits, 32 bits ÷ 16 bits) • Bit manipulation • String • Multiply and accumulate (16 bits × 16 bits + 32 bits) • Correlation operation		
Input/output lines	Input	14 (eight alternate analog input included)		
	I/O	38		
Real-time pulse unit		• 16-bit timer × 1 10-bit dead time timer × 3 16-bit compare register × 4 Two output modes selectable; { mode 0: 6-channel set/reset output / mode 1: 6-channel buffer output • 16-bit timer × 1 16-bit compare register × 1 • 16-bit timer × 1 16-bit capture register × 1 16-bit capture/compare register × 1 • 16-bit timer × 1 16-bit capture register × 2 16-bit capture/compare register × 1 • 16-bit timer × 1 16-bit compare register × 2 16-bit resolution PWM output: 1 channel		
Real-time output port		4 (buffer output in 4-bit units)		
PWM unit		8-, 9-, 10-, 12-bit resolution variable PWM output: 2 channels		
A/D converter		8-channel 10-bit resolution		
Serial interface		With dedicated baud rate generator 1 UART 1 clocked synchronous serial interface/SBI		

Note Under development

(2/2)

Parameter \ Part number	μPD78361A[Note]	μPD78362A	μPD78P364A
Interrupt function	• External: 6, internal: 14 (Two alternate external) • Four priority levels can be specified by software • One of three interrupt service modes can be selected (vectored interrupt, macro service, or context switching)		
Package	64-pin plastic shrink DIP (750 mil)		
Others	• Watchdog timer incorporated • Standby function (HALT mode, STOP mode) • PLL control circuit incorporated		

Note Under development

1.4.5 Function outline (μPD78363A, 78365A, 78366A, 78368A, 78P368A)

(1/2)

Parameter \ Part number		μPD78363A	μPD78365A	μPD78366A	μPD78368A[Note]	μPD78P368A
Minimum instruction execution time		125 ns (internal clock: 16 MHz, external clock: 8 MHz)				
Internal memory	ROM	24 Kbytes	–	32 Kbytes	48 Kbytes	–
	PROM	–	–	–	–	48 Kbytes
	RAM	768 bytes	2 Kbytes			
ROMless mode		Available	ROMless product	Available		None
Memory space		64 Kbytes (external expansion is enabled)				
General-purpose registers		8 bits × 16 registers × 8 banks				
No. of basic instructions		115				
Instruction set		• 16-bit transfer and operations • Multiplication and division (16 bits × 16 bits, 32 bits ÷ 16 bits) • Bit manipulation • String • Multiply and accumulate (16 bits × 16 bits + 32 bits) • Correlation operation				
Input/output lines	Input	14 (eight lines are also used for analog input)				
	I/O	49	31	49		
Real-time pulse unit		• 16-bit timer × 1 10-bit dead time timer × 3 16-bit compare register × 4 One of two output modes can be selected { Mode 0 Set/reset output: Six channels; Mode 1 Buffer output: Six channels • 16-bit timer × 1 16-bit compare register × 1 • 16-bit timer × 1 16-bit capture register × 1 16-bit capture/compare register × 1 • 16-bit timer × 1 16-bit capture register × 2 16-bit capture/compare register x 1 • 16-bit timer × 1 16-bit compare register × 2 16-bit resolution PWM output: One channel				
Real-time output port		Four (buffer output in 4-bit units)				
PWM unit		8-, 9-, 10-, 12-bit resolution variable PWM output: Two channels				
A/D converter		10-bit resolution, eight channels				
Serial interface		With dedicated baud rate generator UART (with pin change function): One channel Clocked synchronous serial interface/SBI: One channel				

Note Under development

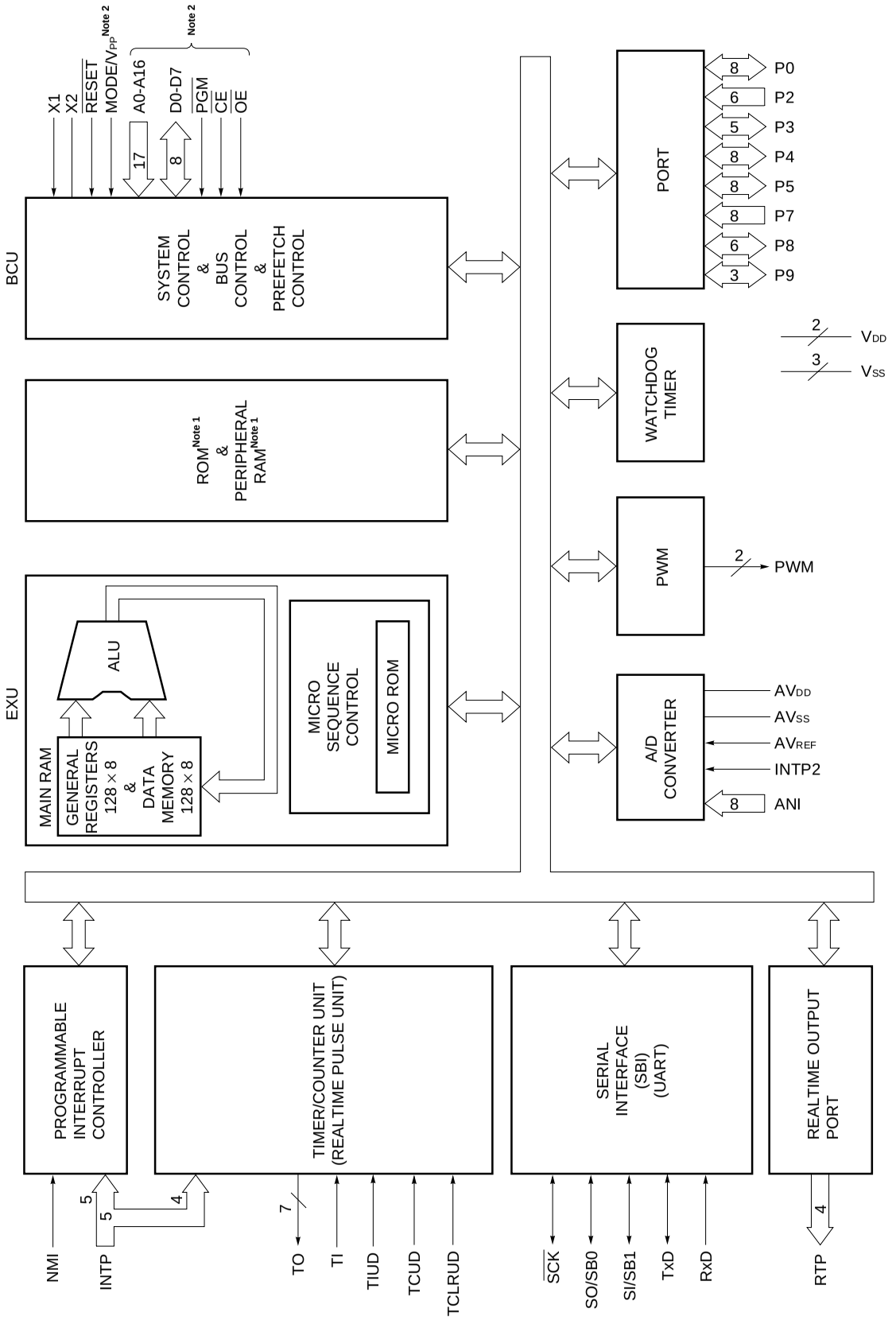
(2/2)

Parameter \ Part number		μPD78363A	μPD78365A	μPD78366A	μPD78368A[Note]	μPD78P368A
Interrupt function		• External: 6, internal: 14 (two are also used for external interrupts) • Four priority levels can be specified by software • One of three interrupt service modes can be selected (vectored interrupt, macro service, or context switching)				
Packages	Without window	80-pin plastic QFP (14 × 20 mm)				
	With window	–				80-pin ceramic WQFN
Others		• Watchdog timer is contained • Standby function (HALT mode, STOP mode) • PLL control circuit is contained				

Note Under development

Phase-out/Discontinued

1.4.6 Block diagram (μPD78361A, 78362A, 78P364A)

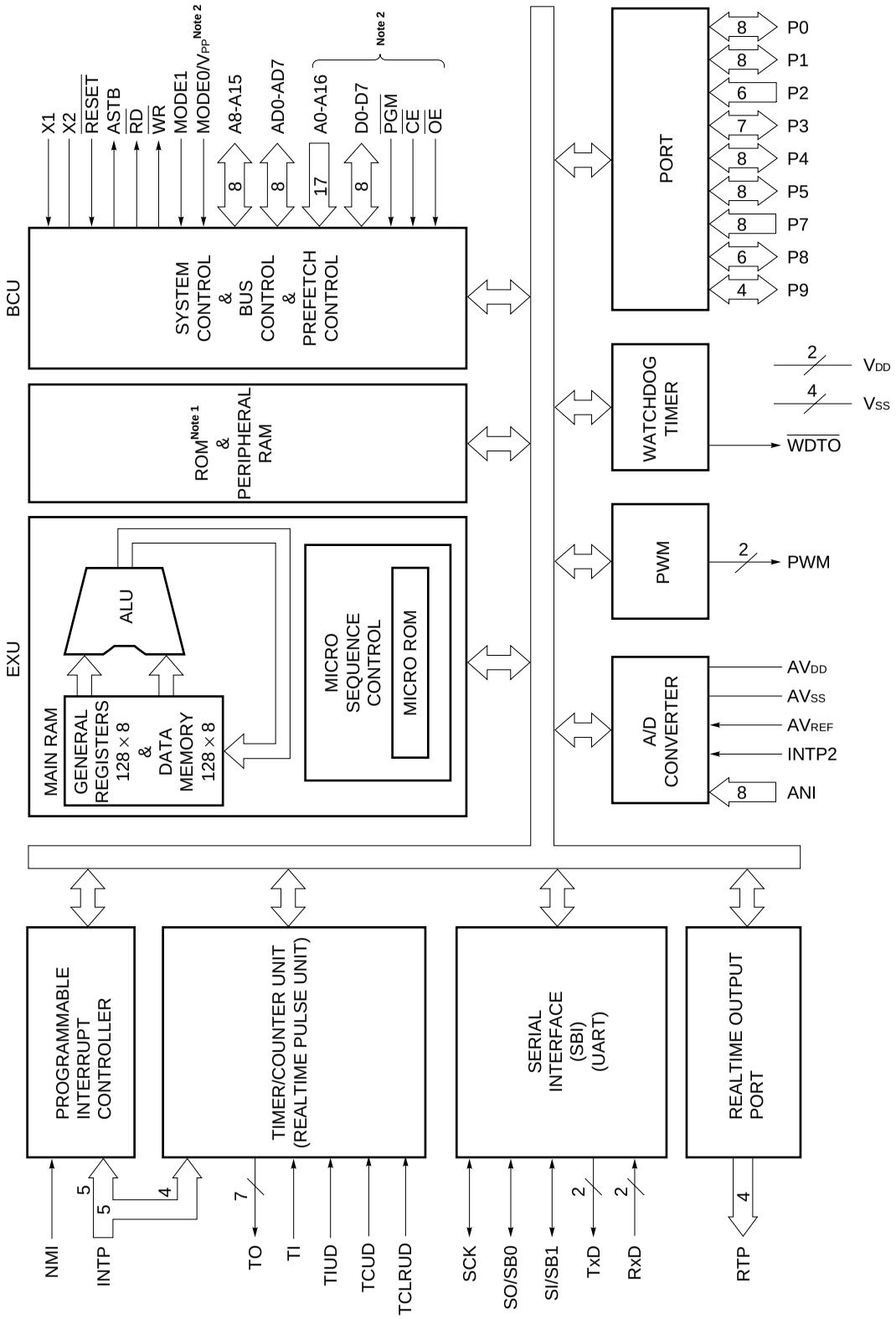


Notes 1. Capacity internal ROM and RAM depends on versions.

2. In PROM programming mode of μPD78P364(A).

Phase-out/Discontinued

1.4.7 Block diagram (μPD78363A, 78365A, 78366A, 78368A, 78P368A)



Notes 1. Capacity of internal ROM and RAM depends on versions.

2. In PROM programming mode of μPD78P368A.

1.5 μPD78372 Subseries Products Overview

★ **Applicable products: μPD78372(A), (A1), (A2), 78P372(A), (A1), (A2)**

1.5.1 Features

- ○ Internal 16-bit architecture, external 16-bit or 8-bit data bus
- ○ Pipeline control system and high-speed operation clock for high-speed processing
- ★ Minimum instruction execution time: 160 ns (internal clock: 12.5 MHz, external clock: 25 MHz) for μPD78372(A) and 78P372(A)
  200 ns (internal clock: 10 MHz external clock: 20 MHz) for μPD78372(A1), 78372(A2), 78P372(A1), 78P372(A2)
- ○ Real-time pulse unit that can output a maximum of 10 timer outputs
- ○ 10-bit resolution A/D converter: 16 channels
- ○ Two independent channels of serial interface (containing dedicated baud rate generator)
- ○ Internal high-speed interrupt controller
- ○ Internal ECC correction circuit (μPD78P372(A), (A1), (A2))
  Internal ROM contents can be made highly reliable
- ○ Internal memory: 24-Kbyte ROM (μPD78372(A), (A1), (A2))
  24-Kbyte PROM (μPD78P372(A), (A1), (A2))
  768-byte RAM
- ★ ○ Supporting QTOP™ microcontroller

Remark QTOP microcontroller is a microcontroller with one-time PROM totally supported by NEC's writing service (writing, marking, screening, and inspection).

1.5.2 Application fields

- ★ ○ For car electronics controller

★ 1.5.3 Ordering information and quality grade

(1) Ordering information

Part number	Package	Internal ROM
μPD78372GC(A)-×××-3B9	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD78372GC(A1)-×××-3B9	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD78372GC(A2)-×××-3B9	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD78372GF(A)-×××-3B9	80-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD78372GF(A1)-×××-3B9	80-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD78372GF(A2)-×××-3B9	80-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD78P372GC(A)-3B9	80-pin plastic QFP (14 × 14 mm)	One-time PROM
μPD78P372GC(A)-×××-3B9	80-pin plastic QFP (14 × 14 mm)	One-time PROM (QTOP microcontroller)
μPD78P372GC(A1)-3B9	80-pin plastic QFP (14 × 14 mm)	One-time PROM
μPD78P372GC(A1)-×××-3B9	80-pin plastic QFP (14 × 14 mm)	One-time PROM (QTOP microcontroller)
μPD78P372GC(A2)-3B9	80-pin plastic QFP (14 × 14 mm)	One-time PROM
μPD78P372GC(A2)-×××-3B9	80-pin plastic QFP (14 × 14 mm)	One-time PROM (QTOP microcontroller)
μPD78P372GF(A)-3B9	80-pin plastic QFP (14 × 20 mm)	One-time PROM
μPD78P372GF(A)-×××-3B9	80-pin plastic QFP (14 × 20 mm)	One-time PROM (QTOP microcontroller)
μPD78P372GF(A1)-3B9	80-pin plastic QFP (14 × 20 mm)	One-time PROM
μPD78P372GF(A1)-×××-3B9	80-pin plastic QFP (14 × 20 mm)	One-time PROM (QTOP microcontroller)
μPD78P372GF(A2)-3B9	80-pin plastic QFP (14 × 20 mm)	One-time PROM
μPD78P372GF(A2)-×××-3B9	80-pin plastic QFP (14 × 20 mm)	One-time PROM (QTOP microcontroller)
μPD78P372KL-S	80-pin ceramic WQFN	EPROM

Remark ××× is a ROM code number.

(2) Quality grade

Special

Caution No quality grades apply to the μPD78P372KL-S. Use this product only for the function evaluation.

Please refer to "Quality Grades on NEC Semiconductor Devices" (Document No. C11531E) published by NEC Corporation to know the specification of quality grade on the devices and its recommended applications.

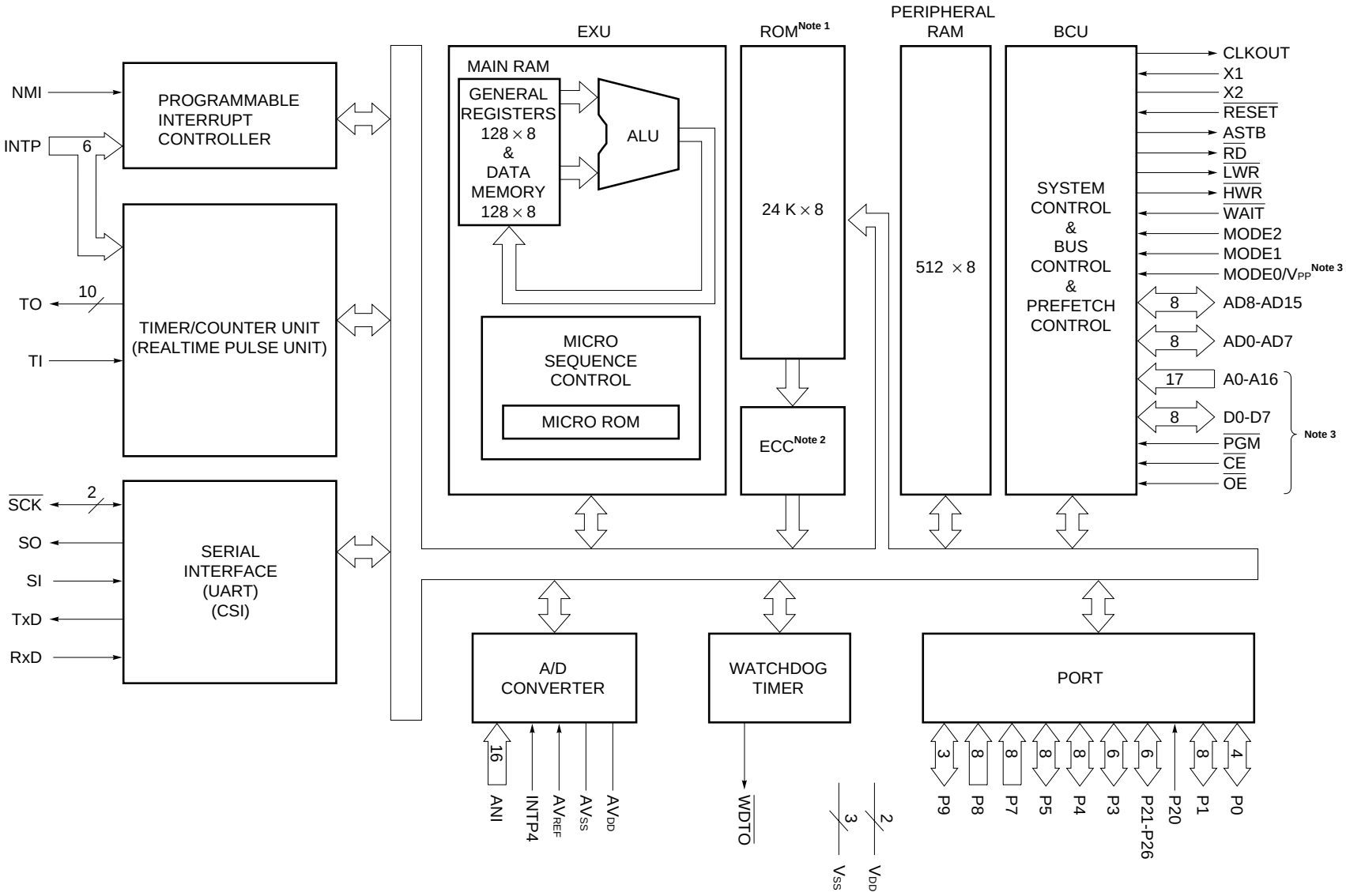
1.5.4 Function outline

★

Parameter \ Part number		μPD78372(A), (A1), (A2)	μPD78P372(A), (A1), (A2)
Minimum instruction execution time		160 ns (internal clock: 12.5 MHz, external clock: 25 MHz) for μPD78372(A) and 78P372(A) 200 ns (internal clock: 10 MHz, external clock: 20 MHz) for μPD78372(A1), (A2) and 78P372(A1), (A2)	
Internal memory	ROM	24 Kbytes	–
	PROM	–	24 Kbytes
	RAM	768 bytes	
Memory space		64 Kbytes (external expansion is enabled)	
General-purpose registers		8 bits × 16 registers × 8 banks	
No. of basic instructions		115	
Instruction set		• 16-bit transfer and operations • Multiplication and division (16 bits × 16 bits, 32 bits ÷ 16 bits) • Bit manipulation • String • Multiply and accumulate (16 bits × 16 bits + 32 bits) • Correlation operation	
Input/output lines	Input	17 (16 lines are also used for analog input)	
	I/O	43	
Real-time pulse unit		• 18/16-bit timer counter × 1 18/16-bit capture/compare register × 6 timer output × 6 • 16-bit timer/event counter × 1 16-bit compare register × 4 timer output × 4	
A/D converter		10-bit resolution, 16 channels	
Serial interface		With dedicated baud rate generator UART: One channel Clocked serial interface: One channel	
Interrupt function		• External: 11, internal: 18 (six are also used for external interrupts) • Four priority levels can be specified by software • One of three interrupt service modes can be selected (vectored interrupt, macro service, or context switching)	
Bus sizing function		8-bit or 16-bit external data bus width can be selected	
ECC circuit		None	Contained
Packages	Without window	• 80-pin plastic QFP (14 × 14 mm) • 80-pin plastic QFP (14 × 20 mm)	
	With window	–	80-pin ceramic WQFN[Note]
Others		• Watchdog timer is contained • Standby function (HALT mode, STOP mode, standby function invalidation mode)	

Note This is the μPD78P372KL-S which can be used as only function evaluation.

1.5.5 Block diagram



Notes 1. μPD78P372(A), (A1), (A2) contains 24-Kbyte PROM.

2. Only μPD78P372(A), (A1), (A2) contains the ECC circuit.

3. In PROM programming mode of μPD78P372(A), (A1), (A2).

CHAPTER 2 TARGET PRODUCT LIST

This chapter lists the functions of the target products.
For details, refer to the appropriate user's manual.

Phase-out/Discontinued

(1/5)

Subseries name			μPD78352A			
Parameter \ Part number			μPD78350	μPD78350A	μPD78352A	μPD78P352
No. of basic functions			113			
Minimum instruction execution time			160 ns [internal clock: 12.5 MHz, external clock: 25.0 MHz]	125 ns [internal clock: 16 MHz, external clock: 32 MHz]		
Internal memory	ROM		–		32K bytes	–
	PROM		–		–	32K bytes
	RAM		640 bytes			
Memory space			Programs, data: 64 Kbytes			
Memory expansion function			External space of a maximum of 64 Kbytes can be expanded			
General-purpose registers			8 bits × 16 registers × 8 banks			
Input/output lines	Total		30		50	
		Input	6			
		I/O	24		44	
Capture/timer unit			• 16-bit timer × 3			
	Auxiliary registers		• 16-bit compare register × 2 • 16-bit capture register × 2			
	Pulse output		None			
PWM unit			8-bit resolution PWM output × 2			
Watchdog timer			Contained			
Interrupt function			External: 5, internal: 4			
Standby function			HALT mode, STOP mode			
Packages	Without window		64-pin plastic QFP (14 × 14 mm) (resin thickness 2.7 mm)	64-pin plastic QFP (14 × 14 mm) (resin thickness 1.5 mm)		
	With window		–			64-pin ceramic WQFN[Note]

Note Under development

Phase-out/Discontinued

(2/5)

Subseries name			μPD78356				
★ Parameter \ Part number			μPD78355	μPD78356	μPD78P356	μPD78356(A)	μPD78P356(A)
No. of basic functions			115				
Minimum instruction execution time			125 ns [internal clock: 16 MHz, external clock: 32 MHz]			160 ns [internal clock: 12.5 MHz external clock: 25 MHz]	
Internal memory	ROM		–	48 Kbytes	–	48 Kbytes	–
	PROM		–	–	48 Kbytes	–	48 Kbytes
	RAM		2 Kbytes				
Memory space			Programs, data: 64 Kbytes				
Memory expansion function			External space of a maximum of 64 Kbytes can be expanded				
General-purpose registers			8 bits × 16 registers × 8 banks				
Input/output lines	Total		57	76			
		Input	9 (eight lines are also used for analog input)				
		I/O	48	67			
Real-time pulse unit			• 16-bit timer × 5 • 10-bit timer × 1				
	Auxiliary registers		• 16-bit compare register × 10 • 10-bit compare register × 1 • 16-bit capture/compare register × 5				
	Pulse output		10				
Real-time output port			8				
PWM unit			8-, 10-, 12-bit resolution variable PWM output × 2				
A/D converter			10-bit resolution × 8				
D/A converter			8-bit resolution × 2				
Serial interface			• With dedicated baud rate generator • UART × 1 • CSI (3-wire serial I/O, SBI) × 1 • CSI (3-wire serial I/O) (with pin change function) × 1				
Watchdog timer			Contained				
Interrupt sources			External: 6, internal: 25 (five are also used for external interrupts)				
Standby function			HALT mode, STOP mode				
Bus sizing function			8-bit or 16-bit external data bus width can be selected				
ECC circuit			None		Available	None	Available
Operating power supply voltage			5 V ± 10 %				
Packages	Without window		100-pin plastic QFP (14 × 14 mm) 120-pin plastic QFP (28 × 28 mm)				120-pin plastic QFP (28 × 28 mm)
	With window		–		120-pin ceramic WQFN	–	

Phase-out/Discontinued

(3/5)

Subseries name			μPD78366A		
★ Parameter \ Part number			μPD78361A[Note]	μPD78362A	μPD78P364A
No. of basic functions			115		
Minimum instruction execution time			125 ns (internal clock: 16 MHz, external clock: 8 MHz at operation)		
Internal memory	ROM		32 Kbytes	24 Kbytes	–
	PROM		–	–	48 Kbytes
	RAM		2 Kbytes	768 bytes	2 Kbytes
ROMless mode			None		
Memory space			Programs, data: 64 Kbytes		
Memory expansion function			None		
General-purpose register			8 bits × 16 registers × 8 banks		
Input/output lines	Total		52		
		Input	14 (8 alternate analog input)		
		I/O	38		
Real-time pulse unit			• 16-bit timer × 5 • 10-bit timer × 3		
	Auxiliary registers		• 16-bit compare register × 7 • 16-bit capture register × 3 • 16-bit capture/compare registers × 2		
	Pulse output		7		
Real-time output port			4		
PWM unit			8-, 9-, 10-, 12-bit resolution variable PWM output × 2		
A/D converter			10-bit resolution × 8		
Serial interface			• With dedicated baud rate generator • UART × 1 • CSI (3-wire serial I/O, SBI) × 1		
Watchdog timer			Available		
Interrupt sources			External: 6, internal: 14 (alternate external: 2)		
Standby function			HALT mode, STOP mode		
PLL control circuit			Available (external 8 MHz to internal 16 MHz)		
Package			64-pin plastic shrink DIP (750 mil)		

Note Under development

Phase-out/Discontinued

(4/5)

Subseries name			μPD78366A				
★ Parameter \ Part number			μPD78363A	μPD78365A	μPD78366A	μPD78368A[Note]	μPD78P368A
No. of basic functions			115				
Minimum instruction execution time			125 ns (internal clock: 16 MHz, external clock: 8 MHz)				
Internal memory		ROM	24 Kbytes	–	32 Kbytes	48 Kbytes	–
		PROM	–	–	–	–	48 Kbytes
		RAM	768 bytes	2 Kbytes			
ROMless mode			Available	ROMless product	Available		None
Memory space			Programs, data: 64 Kbytes				
Memory expansion function			External space of a maximum of 64 Kbytes can be expanded				
General-purpose registers			8 bits × 16 registers × 8 banks				
Input/output lines		Total	63	45	63		
		Input	14 (eight lines are also used for analog input)				
		I/O	49	31	49		
Real-time pulse unit			• 16-bit timer × 5 • 10-bit timer × 3				
	Auxiliary registers		• 16-bit compare register × 7 • 16-bit capture register × 3 • 16-bit capture/compare register × 2				
	Pulse output		7				
Real-time output port			4				
PWM unit			8-, 9-, 10-, 12-bit resolution variable PWM output × 2				
A/D converter			10-bit resolution × 8				
Serial interface			• With dedicated baud rate generator • UART (with pin change function) × 1 • CSI (3-wire serial I/O, SBI) × 1				
Watchdog timer			Available				
Interrupt sources			External: 6, internal: 14 (two are also used for external interrupts)				
Standby function			HALT mode, STOP mode				
PLL control circuit			Available (external 8 MHz → internal 16 MHz)				
Packages	Without window		80-pin plastic QFP (14 × 20 mm)				
	With window		–				80-pin ceramic WQFN

Note Under development

Phase-out/Discontinued

(5/5)

Subseries name			μPD78372	
★ Part number / Parameter			μPD78372(A), (A1), (A2)	μPD78P372(A), (A1), (A2)
No. of basic functions			115	
Minimum instruction execution time			• 160 ns (internal clock: 12.5 MHz, external clock: 25 MHz) for μPD78372(A) and 78P372(A) • 200 ns (internal clock: 10 MHz, external clock: 20 MHz) for μPD78372(A1), (A2) and 78P372(A1), (A2)	
Internal memory	ROM		24 Kbytes	–
	PROM		–	24 Kbytes
	RAM		768 bytes	
Memory space			Programs, data: 64 Kbytes	
Memory expansion function			External space of a maximum of 64 Kbytes can be expanded	
General-purpose registers			8 bits × 16 registers × 8 banks	
Input/output lines	Total		60	
		Input	17 (16 lines are also used for analog input)	
		I/O	43	
Real-time pulse unit			• 18/16-bit timer × 1 • 16-bit timer × 1	
	Auxiliary registers		• 18/16-bit capture/compare register × 6 • 16-bit compare register × 4	
	Pulse output		10	
A/D converter			10-bit resolution × 16	
Serial interface			• With dedicated baud rate generator • UART × 1 • CSI (3-wire serial I/O) × 1	
Watchdog timer			Available	
Interrupt sources			External: 11, internal: 18 (six are also used for external interrupts)	
Standby function			HALT mode, STOP mode, standby function invalidation mode	
Bus sizing function			8-bit or 16-bit external data bus width can be selected	
ECC circuit			None	Available
Packages	Without window		• 80-pin plastic QFP (14 × 14 mm) • 80-pin plastic QFP (14 × 20 mm)	
	With window		–	80-pin ceramic WQFN**Note**

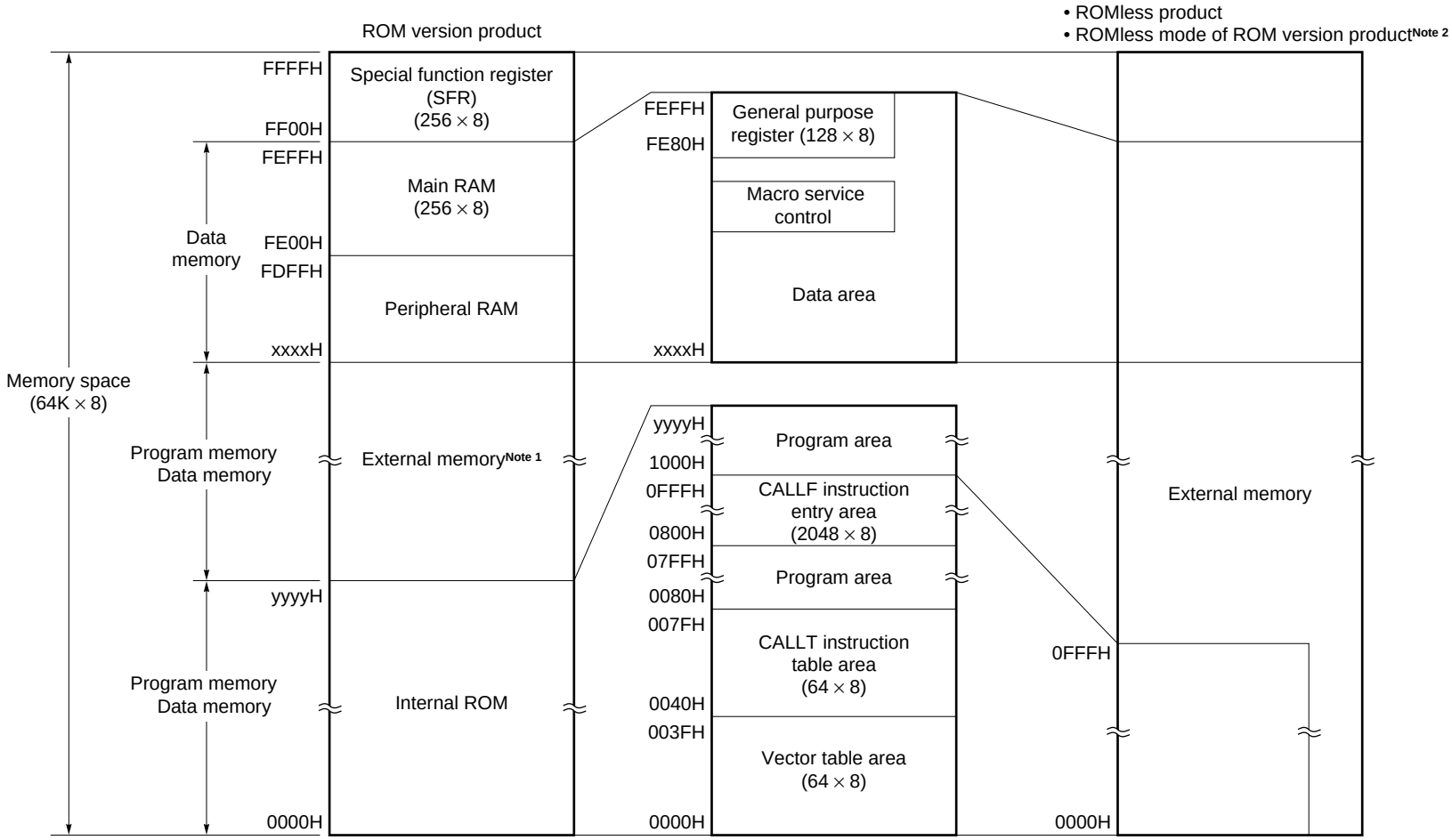
Note This is the μPD78P372KL-S which can be used as only function evaluation.

CHAPTER 3 CPU ARCHITECTURE

3.1 Memory Space

The μPD78356 can access a memory space of a maximum of 64 Kbytes. However, memory mapping varies from one product to another depending on the on-chip memory capacity and the pin state. For details on the address area of the memory map, refer to the appropriate user's manual.

Figure 3-1. Memory Map



Notes 1. Access in external memory expansion mode (except for μPD78361A, 78362A, and 78P364A).

2. The μPD78361A, 78362A, 78P364A, 78P368A does not provide the ROMless mode.

Caution To make a word access to the main RAM area (FE00H-FEFFH) (containing stack handling), only even addresses can be specified in operands.

(1) Memory space of μPD78352A Subseries

Part number	Program memory		Data memory
	Internal ROM	External memory[Note]	Internal RAM
μPD78350 μPD78350A (MODE0, 1=HL)	–	64640 bytes (0000H-FC7FH)	640 bytes (FC80H-FEFFH)
μPD78352A (MODE0, 1=HL)			
μPD78352A μPD78P352 (MODE0, 1=LL)	32768 bytes (0000H-7FFFH)	31872 bytes (8000H-FC7FH)	

Note Access in external memory expansion mode.

(2) Memory space of μPD78356 Subseries

Part number	Program memory		Data memory
	Internal ROM	External memory[Note]	Internal RAM
μPD78355 (MODE0, 1=HL/HH)	–	63232 bytes (0000H-F6FFH)	2048 bytes (F700H-FEFFH)
μPD78356 μPD78356(A) (MODE0, 1=HL/HH)			
μPD78P356 μPD78P356(A) (MODE0, 1=HH)			
μPD78356 μPD78P356 μPD78356(A) μPD78P356(A) (MODE0, 1=LL)	49152 bytes (0000H-BFFFH)	14080 bytes (C000H-F6FFH)	

Note Access in external memory expansion mode.

Remark The μPD78356 Subseries enables the user to change the internal memory capacity by setting a memory expansion register (MM). (See the following table.)

MM6	MM5	Internal ROM
0	0	49152 bytes (0000H-BFFFH)
0	1	32768 bytes (0000H-7FFFH)
1	0	24576 bytes (0000H-5FFFH)
1	1	16384 bytes (0000H-3FFFH)

MM3	Internal RAM
0	2048 bytes (F700H-FEFFH)
1	1024 bytes (FB00H-FEFFH)

★ **(3) Memory space of μPD78366 Subseries**

Part number	Program memory		Data memory
	Internal ROM	External memory[Note 1]	Internal RAM
μPD78361A[Note 2] (MODE=L)	32768 bytes (0000H-7FFFH)	–	2048 bytes (F700H-FEFFH)
μPD78362A[Note 2] (MODE=L)	24576 bytes (0000H-5FFFH)		768 bytes (FC00H-FEFFH)
μPD78P364A[Note 2] (MODE=L)	49152 bytes (0000H-BFFFH)		2048 byte (F700H-FEFFH)
μPD78363A (MODE0, 1=HH)	–	64512 bytes (0000H-FBFFH)	768 bytes (FC00H-FEFFH)
μPD78365A (MODE0, 1=HH)		63232 bytes (0000H-F6FFH)	2048 bytes (F700H-FEFFH)
μPD78366A (MODE0, 1=HH)			
μPD78368A (MODE0, 1=HH)			
μPD78363A (MODE0, 1=LL)	24576 bytes (0000H-5FFFH)	39936 bytes (6000H-FBFFH)	768 bytes (FC00H-FEFFH)
μPD78366A (MODE0, 1=LL)	32768 bytes (0000H-7FFFH)	30464 bytes (8000H-F6FFH)	2048 bytes (F700H-FEFFH)
μPD78368A (MODE0, 1=LL)	49152 bytes (0000H-BFFFH)	14080 bytes (C000H-F6FFH)	
μPD78P368A[Note 2] (MODE0, 1=LL)			

Notes 1. Access in external memory expansion mode (except for μPD78361A, 78362A, and 78P364A).

2. The μPD78361A, 78362A, 78P364A, 78P368A does not provide the ROMless mode.

Remark The μPD78P364A and 78P368A Subseries enables the user to change the internal memory capacity by setting a memory expansion register (MM). (See the following table.)

MM6	MM5	Internal ROM	Internal RAM
0	0	49152 bytes (0000H-BFFFH)	2048 bytes (F700H-FEFFH)
0	1	32768 bytes (0000H-7FFFH)	
1	0		1024 bytes (FB00H-FEFFH)
1	1	24576 bytes (0000H-5FFFH)	768 bytes (FC00H-FEFFH)

★ **(4) Memory space of μPD78372 Subseries**

Part number	Program memory		Data memory
	Internal ROM	External memory[Note]	Internal RAM
μPD78372(A), (A1), (A2) (MODE0, 1=HL/HH)	–	64512 bytes (0000H-FBFFH)	768 bytes (FC00H-FEFFH)
μPD78P372(A), (A1), (A2) (MODE0, 1=HH)			
μPD78372(A), (A1), (A2) μPD78P372(A), (A1), (A2) (MODE0, 1=LL)	24576 bytes (0000H-5FFFH)	39936 bytes (6000H-FBFFH)	

Note Access in external memory expansion mode.

3.1.1 Vector table area

Interrupt branch addresses according to peripheral hardware interrupt requests, reset input, external interrupt requests, and break instructions are stored in the 64-byte area of 0000H-003FH.

When an interrupt request occurs, the corresponding vector table contents are set in the program counter (PC) for causing the program flow to branch. At this time, the contents of the even address are set in the low-order eight bits of the program counter and the contents of the odd address in the high-order eight bits.

If the TPF bit of a CPU control word (CCW) is set to 1, the 8002H-803FH area can be used as the vector table area in place of 0002H-003FH.

★ **Caution The μPD78361A, 78362A, and 78P364A do not contain the CPU control word.**

Table 3-1. Vector Table Area

Vector table address		Interrupt source
TPF=0	TPF=1	
0000H		$\overline{\text{RESET}}$ pin input
0002H	8002H	NMI pin input
0004H	8004H	Watchdog timer
0006H	8006H	Vary from one product to another
• • •		
003AH	803AH	
003CH		OPE code trap
003EH		BRK instruction

3.1.2 CALLT instruction table area

Call addresses of 1-byte call instruction (CALLT) can be stored as 32 table entries in the 64-byte area of 0040H-007FH.

If the TPF bit of the CPU control word (CCW) is set to 1, the 8040H-807FH area can be used as the CALLT instruction table in place of 0040H-007FH.

★ **Caution The μPD78361A, 78362A, and 78P364A do not contain the CPU control word.**

3.1.3 CALLF instruction entry area

In the 0800H-0FFFH area, a direct subroutine call can be made by a 2-byte call instruction (CALLF).

3.1.4 Internal RAM area

The internal RAM area of the μPD78356 consists of the following two areas:

- Internal RAM area
  - Peripheral RAM: Addresses vary from one product to another
  - Main RAM : FE00H-FEFFH (256 bytes)

The main RAM area can be accessed at high speed. A group of general-purpose registers consisting of eight register banks and macro service control words for controlling the macro service function are mapped in the main RAM area.

- General-purpose register group: FE80H-FEFFH (128 bytes)
- Macro service control words: Addresses vary from one product to another

★ **Table 3-2. Internal RAM Area List**

Part number	Internal RAM	Peripheral RAM	Main RAM
μPD78350 μPD78350A μPD78352A μPD78P352	640 bytes (FC80H-FEFFH)	384 bytes (FC80H-FDFFH)	256 bytes (FE00H-FEFFH)
μPD78355 μPD78356 μPD78P356 μPD78356(A) μPD78P356(A)	2048 bytes (F700H-FEFFH)	1792 bytes (F700H-FDFFH)	
μPD78362A μPD78363A	768 bytes (FC00H-FEFFH)	512 bytes (FC00H-FDFFH)	
μPD78361A μPD78P364A μPD78365A μPD78366A μPD78368A μPD78P368A	2048 bytes (F700H-FEFFH)	1792 bytes (F700H-FDFFH)	
μPD78372(A), (A1), (A2) μPD78P372(A), (A1), (A2)	768 bytes (FC00H-FEFFH)	512 bytes (FC00H-FDFFH)	

Cautions 1. To make a word access to the main RAM area (FE00H-FEFFH) (containing stack handling), the access operation varies depending on whether the reference address is even or odd. (See Table 3-3.) Therefore, if an access to an even address and an access to an odd address are mixed, an error occurs. Set only even reference addresses. (See Examples 1 and 2.)
To execute a 16-bit data transfer instruction, specify even addresses in the operands. If an odd address is specified, an error occurs in the assembler package (RA78K3).

2. Do not make a word access across the peripheral RAM area and the main RAM area. (See Example 3.)

Table 3-3. Word Access Operation in Internal RAM Area

Access area \ Reference address (n)	Even	Odd
Main RAM	○	×
Peripheral RAM	○	○

Remark ○: Addresses n and n+1 are accessed
×: Addresses n and n–1 are accessed

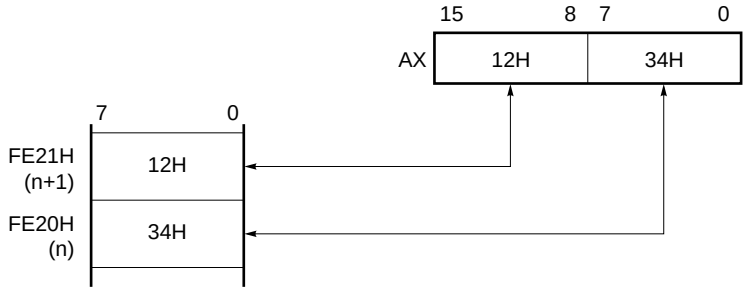
Word access examples in the internal RAM area are given in Examples 1 to 5.

Example 1: To write/read word data into/from even address (FE20H) in main RAM area

When word data is written into an even address (n) in the main RAM area, the low-order eight bits of the word data are written into the even address n and the high-order eight bits are written into the odd address n+1.

When word data is read from an even address (n) in the main RAM area, word data is read from addresses n and n+1.

```
MOVW AX, #1234H
MOVW 0FE20H, AX  ; Write word data into FE20H
MOVW AX, 0FE20H  ; Read word data from FE20H
```



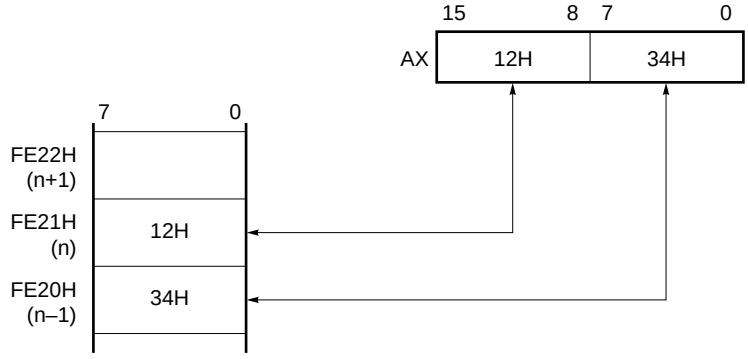
n: Reference address

2: To write/read word data into/from odd address (FE21H) in main RAM area

When word data is written into an odd address (n) in the main RAM area, the high-order eight bits of the word data are written into the odd address n and the low-order eight bits are written into the even address n-1.

When word data is read from an odd address (n) in the main RAM area, word data is read from addresses n and n-1.

```
MOVW AX, #1234H
MOVW DE, #0FE21H
MOVW [DE], AX      ; Write word data into FE21H
MOVW AX, [DE]      ; Read word data from FE21H
```



n: Reference address

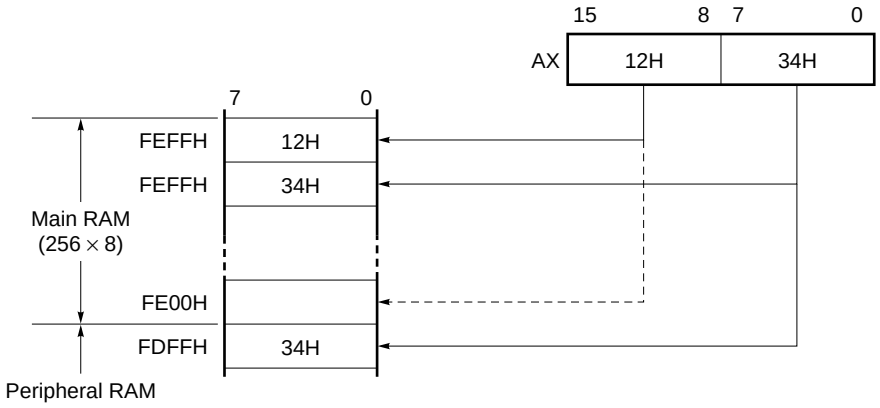
3: To write/read word data across peripheral RAM area and main RAM area

When word data is written across the peripheral RAM area and the main RAM area, the data will be written into the address 256 bytes distant, causing an error to occur.

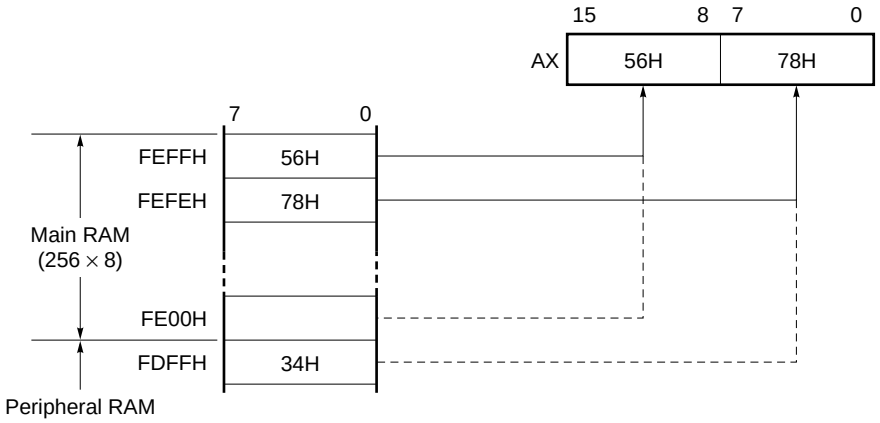
When word data is read from the end address of the peripheral RAM (FDFFH), word data will be read from FEFEH and FEFFH 256 bytes distant.

```
MOVW AX, #1234H
MOVW DE, #0FDFFH
MOVW [DE], AX       ; Write word data into peripheral RAM (FDFFH)
      :
MOVW DE, #0FDFFH
MOVW AX, [DE]       ; Read word data from peripheral RAM (FDFFH)
```

To write data:



To read data:

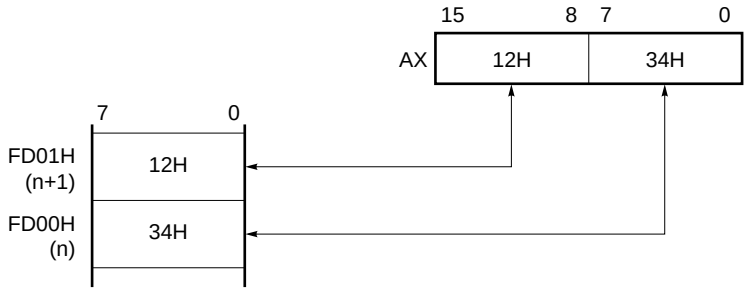


4: To write/read word data into/from even address (FD00H) in peripheral RAM area

When word data is written into an even address (n) in the peripheral RAM area, the low-order eight bits of the word data are written into the even address n and the high-order eight bits are written into the odd address n+1.

When word data is read from an even address (n) in the peripheral RAM area, word data is read from addresses n and n+1.

```
MOVW AX, #1234H
MOVW DE, #0FD00H
MOVW [DE], AX      ; Write word data into FD00H
MOVW AX, [DE]      ; Read word data from FD00H
```



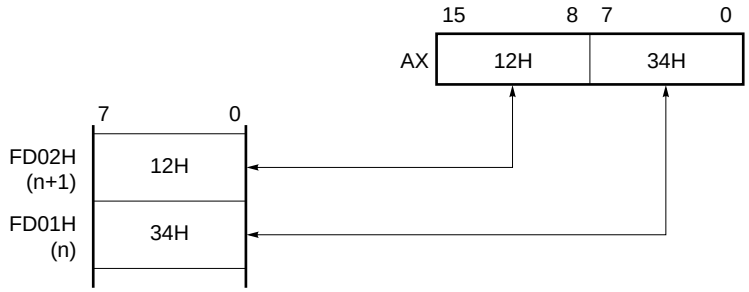
n: Reference address

5: To write/read word data into/from odd address (FD01H) in peripheral RAM area

When word data is written into an odd address (n) in the peripheral RAM area, the low-order eight bits of the word data are written into the odd address n and the high-order eight bits are written into the even address n+1.

When word data is read from an odd address (n) in the peripheral RAM area, word data is read from addresses n and n+1.

```
MOVW AX, #1234H
MOVW DE, #0FD01H
MOVW [DE], AX      ; Write word data into FD01H
MOVW AX, [DE]      ; Read word data from FD01H
```



n: Reference address

3.1.5 Special function register area

A group of registers to which special functions are assigned, such as peripheral hardware mode registers and control registers, are mapped in the FF00H-FFFFH area.

Cautions 1. Do not access addresses in which the special function registers are not mapped (except for external SFR area).

★ **2. The μPD78361A, 78362A, and 78P364A do not contain external SFR area.**

3.1.6 External memory area

The external memory area is a memory area that can be accessed by setting the memory expansion mode register (MM).

External devices (data memory, program memory, peripheral devices) can be connected to the external memory area of the μPD78356.

★ **Caution The μPD78361A, 78362A, and 78P364A do not contain the external memory area.**

★ **Table 3-4. External Memory Area List**

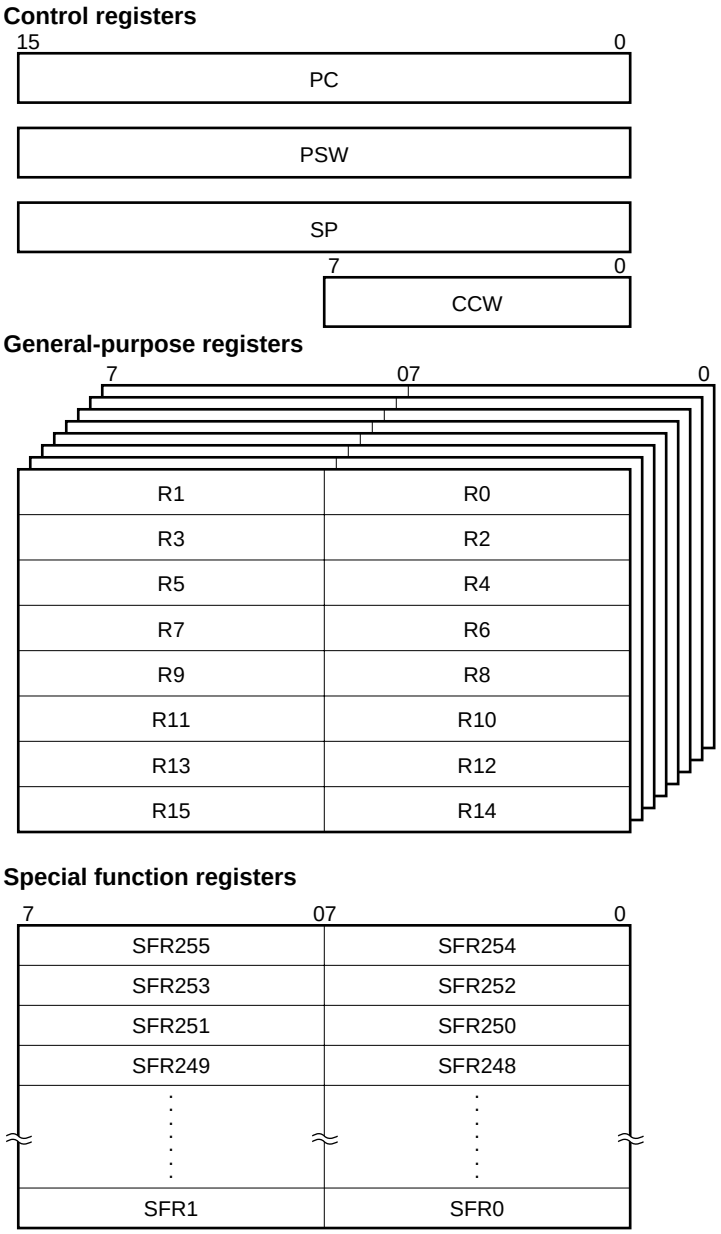
Part number	External memory area
μPD78350[Note] μPD78350A[Note]	64640 bytes (0000H-FC7FH)
μPD78352A μPD78P352	31872 bytes (8000H-FC7FH)
μPD78355[Note]	63232 bytes (0000H-F6FFH)
μPD78356 μPD78P356 μPD78356(A) μPD78P356(A)	14080 bytes (C000H-F6FFH)
μPD78363A	39936 bytes (6000H-FBFFH)
μPD78365A[Note]	63232 bytes (0000H-F6FFH)
μPD78366A	30464 bytes (8000H-F6FFH)
μPD78368A μPD78P368A	14080 bytes (C000H-F6FFH)
μPD78372(A), (A1), (A2) μPD78P372(A), (A1), (A2)	39936 bytes (6000H-FBFFH)

Note ROMless product

3.2 Processor Registers

The processor registers comprise three groups of registers: Control registers consisting of an 8-bit register and three 16-bit registers, general purpose registers consisting of eight banks each consisting of sixteen 8-bit registers, and special function registers to which special functions are assigned, such as I/O mode register of peripheral hardware.

Figure 3-2. Register Configuration



Remark The control register CCW is mapped in a special function register (SFR) area.

3.2.1 Control registers

The control registers have dedicated functions such as control of a program sequence, the status, and the stack memory and operand addressing modification.

The control registers consist of three 16-bit registers and one 8-bit register.

(1) Program counter (PC)

The program counter (PC) is a 16-bit register which holds address information of the next program to be executed. The PC operation is as follows:

- at normal operation
  PC is incremented automatically according to the number of bytes of fetched instruction;
- when a branch instruction is executed immediate data or register contents are set in the PC.

When $\overline{\text{RESET}}$ is input, the reset vector data at addresses 0000H and 0001H is set in the PC for branching.

(2) Program status word (PSW)

The program status word (PSW) is a 16-bit register which consists of flags set or reset according to the instruction execution result.

The PSW can be read/written in units of the high-order eight bits (PSWH) and the low-order eight bits (PSWL). Each of the flags can be handled by executing a bit manipulation instruction.

When an interrupt request occurs or a BRK instruction is executed, the PSW contents are automatically saved in a given stack and when a RETI or RETB instruction is executed, automatically restored.

When $\overline{\text{RESET}}$ is input, all bits of the PSW are reset to 0.

Figure 3-3. Format of Program Status Word (PSW)

Symbol	7	6	5	4	3	2	1	0
PSWH	UF	RBS2	RBS1	RBS0	0	0	0	0

	7	6	5	4	3	2	1	0
PSWL	S	Z	RSS	AC	IE	P/V	0	CY

UF : User flag
RBS0-RBS2 : Register bank selection flag
S : Sign flag (MSB of operation result)
Z : Zero flag
RSS : Register set selection flag
AC : Auxiliary carry flag
IE : Interrupt request enable flag
P/V : Parity/overflow flag
CY : Carry flag

The flags are described below:

(a) User flag (UF)

The user flag (UF) can be set or reset in a user program for program control.

(b) Register bank selection flag (RBS0-RBS2)

The register bank selection flag consists of three bits to select one of eight register banks (register banks 0-7).

(c) Sign flag (S)

The sign flag (S) is a flag for storing the MSB which is set to 1 as a result of operation.
When the MSB is 1 as a result of operation, the S flag is set to 1; when 0, it is reset to 0.
The S flag can be tested by executing a conditional branch instruction.

(d) Zero flag (Z)

The zero flag (Z) is a flag for storing the operation result which is 0.
When the operation result is 0, the Z flag is set to 1; otherwise, it is reset to 0.
The Z flag can be tested by executing a conditional branch instruction.

(e) Register set selection flag (RSS)

The register set selection flag (RSS) is a flag for specifying 8-bit general-purpose registers functioning as X, A, C, and B registers and 16-bit general-purpose register pairs functioning as AX and BC.
The function names and the absolute names (names enclosed in parentheses) are related to each other according to specification of the RSS flag as follows: (See **Table 3-5. Configuration of General-Purpose Registers**.)

- **When RSS=0**
  X (R0), A (R1), C (R2), B (R3), AX (RP0), BC (RP1)
- **When RSS=1**
  X (R4), A (R5), C (R6), B (R7), AX (RP2), BC (RP3)

To set or reset the RSS flag, be sure to describe an RSS pseudo instruction just before or after the set or reset instruction. (See below.)

<Program examples>

- To set RSS=0

```
RSS     0       ; RSS pseudo instruction
CLR1    PSWL.5
MOV     B, A    ; This description is equivalent to “MOV R3, R1”
```

- To set RSS=1

```
RSS     1       ; RSS pseudo instruction
SET1    PSWL.5
MOV     B, A    ; This description is equivalent to “MOV R7, R5”
```

A similar effect to having two register sets can be produced. Registers or register pairs not specified with the RSS flag can be accessed by describing their absolute names.

(f) Auxiliary carry flag (AC)

The auxiliary carry flag (AC) is a flag used for decimal adjustment for storing an underflow into bit 3 or an overflow out of bit 3.
When a carry is generated out of bit 3 as a result of operation (overflow) or a borrow is generated into bit 3 (underflow), the AC flag is set to 1; otherwise, it is reset to 0.
The AC flag can be tested by executing a conditional branch instruction.

(g) Interrupt request enable flag (IE)

The interrupt request enable flag (IE) is a flag indicating whether interrupt requests are enabled or disabled. When an EI instruction is executed, the IE flag is set to 1; when a DI instruction is executed or an interrupt is acknowledged, it is reset to 0.

(h) Parity/overflow flag (P/V)

The P/V flag operates as a parity flag or overflow flag according to execution of a logical or arithmetic operation instruction, as described below.
The P/V flag state can be tested by executing a conditional branch instruction.

- **Parity flag operation**
  When the number of bits set to 1 is even as a result of execution of a logical operation instruction, the P/V flag is set to 1; when odd, it is reset to 0.
  However, only the low-order eight bits of the operation result are effective for the parity flag regardless of 16-bit or 8-bit operations.

- **Overflow flag operation**
  Only when the execution result of an arithmetic operation instruction exceeds the numeric value range represented by two's complements, the P/V flag is set to 1; otherwise, it is reset to 0.
  For example, the two's complement range is 80H (–128) to 7FH (+127) in 8-bit arithmetic operations, and if the operation result becomes outside the range, the P/V flag is set to 1; otherwise, it is reset to 0.

Example: The overflow flag operation when an 8-bit addition instruction is executed is shown below:

When 78H (+120) and 69H (+105) are added, the operation result becomes E1H (+225), exceeding the upper limit of two's complements. Thus, the P/V flag is set to 1. The two's complement representation of E1H is –31.

```
    78H (+120)     =      0111 1000
+)  69H (+105)     =  +)  0110 1001
                       0   1110 0001  =  –31   P/V=1
                       ↑
                       C
```

When the following two negative numbers are added, the operation result is within the two's complement range. Thus, the P/V flag is reset to 0.

```
    FBH (–5)       =      1111 1011
+)  F0H (–16)      =  +)  1111 0000
                       1   1110 1011  =  –21   P/V=0
                       ↑
                       C
```

(i) Carry flag (CY)

The carry flag (CY) is a flag for storing an overflow or underflow of the operation instruction execution result. When an operation instruction generates a carry out of bit 7 (overflow) or a borrow into bit 7 (underflow), the CY flag is set to 1. In word operation, when a carry is generated out of bit 15 (overflow) or a borrow is generated into bit 15 (underflow), the CY flag is set to 1; otherwise, it is reset to 0.

The CY flag can be tested by executing a conditional branch instruction. When a bit manipulation instruction is executed, the CY flag serves as a 1-bit accumulator.

(3) Stack pointer (SP)

The stack pointer (SP) is a 16-bit register which holds the top address of a stack area (LIFO) of memory.

The SP is handled by executing dedicated instructions (stack handling instructions).

The SP is decremented before write (save) operation into the stack memory; it is incremented after read (restore) operation from the stack memory.

When $\overline{\text{RESET}}$ is input, the SP becomes undefined. Be sure to set the SP before executing a subroutine call, etc.

Caution To make a word access to the main RAM area (FE00H-FEFFH), only even addresses can be specified in operands.

(4) CPU control word (CCW)

The CPU control word (CCW) is an 8-bit register which consists of CPU control flags.

The CCW is mapped in the special function register area (FFC1H) and can be controlled by software.

When $\overline{\text{RESET}}$ is input, all bits of the CCW are reset to 0.

Figure 3-4. Format of CPU Control Word

Symbol	7	6	5	4	3	2	1	0	Address	When reset	R/W
CCW	0	0	0	0	0	0	TPF	0	FFC1H	00H	R/W

TPF: Table position flag

The table position flag (TPF) is a flag indicating the locations of a vector table referenced when a CALLT instruction or an interrupt request is executed. The vector table locations are changed according to specification of the TPF flag as follows:

- TPF=0 (reset)
  0000H-007FH
- TPF=1 (set)
  8000H-807FH

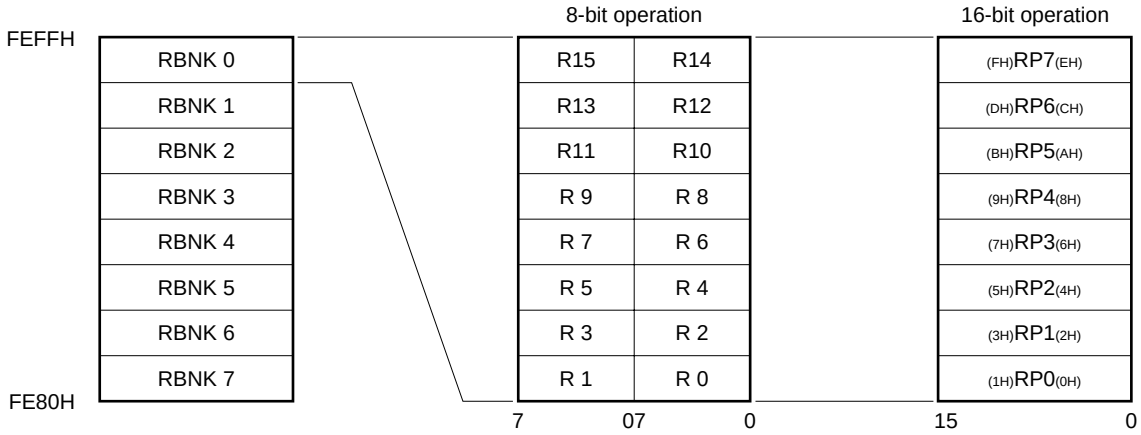
Cautions 1. Vector table entries for $\overline{\text{RESET}}$ input, BRK instruction, and OPE code trap interrupt are fixed to 0000H, 003EH, and 003CH respectively, and not affected by the TPF.

2. The μPD78361A, 78362A, and 78P364A do not contain the CPU control word.

3.2.2 General-purpose registers

The general-purpose registers are eight 128-byte register banks mapped in a specific area of internal RAM space (FE80H-FEFFH), each register bank consisting of sixteen 8-bit registers.

Figure 3-5. Process Bits of General-Purpose Registers



8-bit registers can also be paired as eight 16-bit register pairs (RP0-RP7).

In addition to the absolute names, the function names can be used to describe the sixteen 8-bit registers, as listed in Table 3-5. The X register serves as the low-order part of a 16-bit accumulator; the A register serves as an 8-bit accumulator or the high-order part of the 16-bit accumulator; the B and C registers serve as a counter; and DE, HL, VP, and UP register pairs serve as address registers. Particularly, the VP register pair has a function as a base register and the UP register pair has a function as a user stack pointer.

The registers having a unique function change according to the value of the register set selection flag (RSS) in the program status word (PSW), as listed in Table 3-5.

Therefore, when the function names are used to describe a program, a similar effect to having two register sets (X, A, B, C, AX, BC) can be produced by handling the RSS flag. Registers not specified with the RSS flag, for example, R4 register when RSS=0, can be accessed by describing the absolute names (in the example, R4).

The μPD78356 enables implied addressing using the function names attaching importance to the unique function of each register and register addressing using the absolute names for high-speed processing with less data transfer count and preparing highly descriptive programs as process data addressing.

Table 3-5. Configuration of General-Purpose Registers

(a) Correspondence between absolute and function names of 8-bit registers

Absolute name	Function name	
	RSS=0	RSS=1
R0	X	
R1	A	
R2	C	
R3	B	
R4		X
R5		A
R6		C
R7		B
R8	VP_L	VP_L
R9	VP_H	VP_H
R10	UP_L	UP_L
R11	UP_H	UP_H
R12	E	E
R13	D	D
R14	L	L
R15	H	H

(b) Correspondence between absolute and function names of 16-bit register pairs

Absolute name	Function name	
	RSS=0	RSS=1
RP0	AX	
RP1	BC	
RP2		AX
RP3		BC
RP4	VP	VP
RP5	UP	UP
RP6	DE	DE
RP7	HL	HL

3.2.3 Special function registers (SFRs)

Unlike the general-purpose registers, the special function registers (SFRs) have special functions. The SFRs are allocated to FF00H-FFFFH memory space (256-byte special function register area).

Short direct addressing can be applied to the 32 byte area FF00H-FF1FH. Therefore, the SFRs allocated to the area can be processed with the shorter word length and a fewer number of clocks than the SFRs allocated to another area. The frequently accessed SFRs, such as capture registers, compare registers, and ports, are allocated to the 32-byte area.

For the 16-byte area FFD0H-FFDFH, an access to the external is made by SFR addressing. Therefore, an access to the external memory and bit manipulation of external devices can be performed with instructions having short word length.

The SFRs can be handled as the general-purpose registers by executing instructions such as operation, transfer, and bit manipulation instructions. Bit units in which the SFRs can be handled (1, 8, 16 bits) vary depending on the SFR.

3.3 Data Memory Addressing

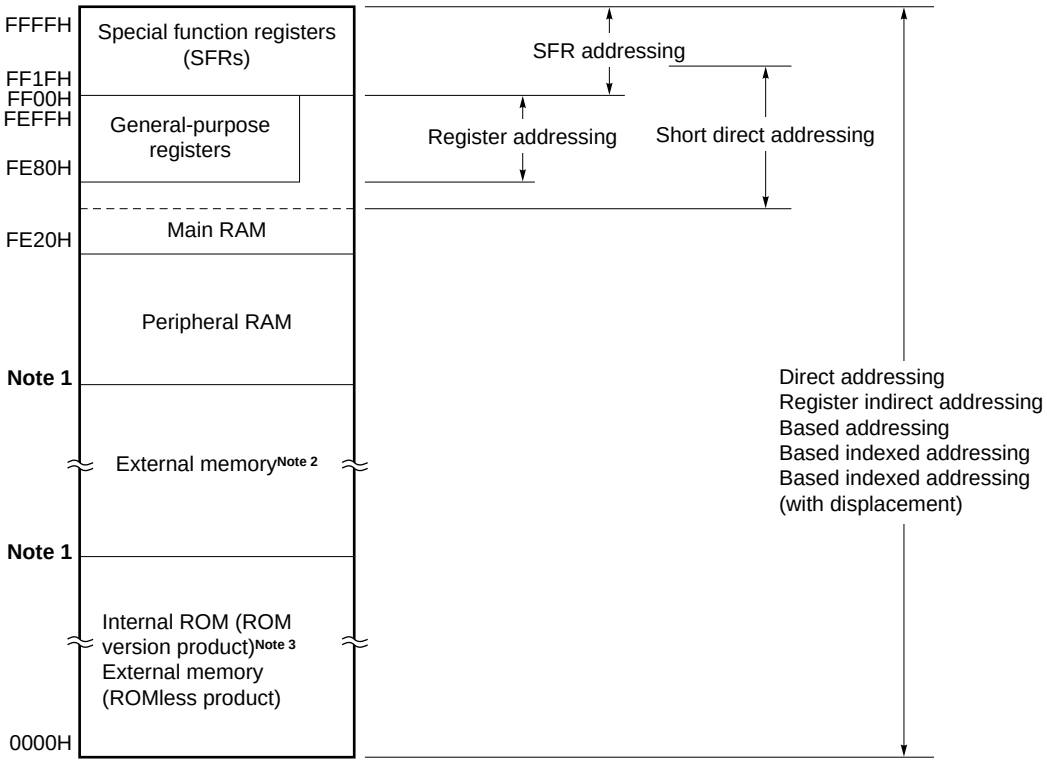
The μPD78356 provides various addressing modes considering operability of memory and high-level languages.

Particularly in the data memory area, proper addressing modes can be used conforming to the functions of the special function registers (SFRs), general-purpose registers, etc.

Figure 3-6 shows the data memory addressing modes.

For details of the addressing modes, see **4.2 Operand Addressing**.

Figure 3-6. Data Memory Addressing



Notes 1. The addresses vary depending on the product.

★ **2.** Since the μPD78361A, 78362A, and 78P364A do not contain the external memory expansion function, the area cannot be used.

3. External memory in the ROMless mode.

Caution To make a word access to the main RAM area (FE00H-FEFFH) (containing stack handling), only even addresses can be specified in operands.

3.3.1 General-purpose register addressing

(1) Implied addressing

In the implied addressing, the registers functioning as the accumulators (A, AX) and loop counters (B, C) in the general-purpose register area are addressed automatically by instructions.

Description example **MULU r**

Assuming that the multiplier is the value stored in the B register in an 8-bit × 8-bit multiplication instruction, describe the following.

When the instruction is executed, data in the accumulator (A register) is multiplied by data in the 16-bit accumulator (AX register) and the result is stored in the 16-bit accumulator (AX register).

MULU B ; AX ← A × B

(2) Register addressing

In the register addressing, the register to be used is addressed directly in an instruction.

Description example **ADD r, r**

To specify D and E registers as registers to store addition instruction operand values in an 8-bit addition instruction, describe the following:

ADD D, E ; D ← D + E

3.3.2 Short direct addressing

The short direct addressing is addressing to access addresses FE20H-FEFFH in the internal RAM space and addresses FF00H-FF1FH in the SFR space. The short direct addressing enables the areas to be accessed with short operation codes at high speed.

To handle 16-bit data, specify even addresses.

Description example **ADD A, saddr**

When one of addition instruction operand values is stored at address FE80H in the internal data memory space in an 8-bit addition instruction, describe the following:

ADD A, 0FE80H ; A ← A + (FE80H)

3.3.3 Special function register (SFR) addressing

The special function register (SFR) addressing is used to handle the special function registers (SFRs) mapped in the SFR area (FF00H-FFFFH).

Description example **MOV A, sfr**

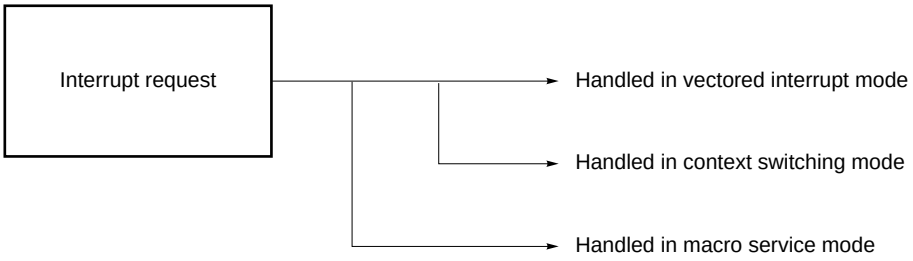
To specify port 0 in the SFR area for the special function register used as the source in an 8-bit transfer instruction, describe the following:

MOV A, P0 ; A ← P0

3.4 Interrupt Function

The μPD78356 handles interrupt requests occurring from on-chip peripheral hardware and the external in the three processing modes shown in Figure 3-7.

Figure 3-7. Handling Interrupt Requests



The interrupt requests are classified into the following four types. Table 3-6 lists the relationships among the interrupt requests and the processing modes.

- Nonmaskable interrupt requests
- Maskable interrupt requests
- Software interrupt requests
- OPE code trap interrupt requests

Table 3-6. Interrupt Requests and Processing Modes

Processing mode / Interrupt request	Vectored interrupt processing	Macro service	Context switching
Nonmaskable interrupt	○	–	–
Maskable interrupt	○	○	○
Software interrupt	○	–	○
OPE code trap interrupt	○	–	–

Caution The μPD78356 enters the complete interrupt disable state during execution of a write access instruction to an interrupt control register (see 3.4.6) or program status word (PSW). In the state, nonmaskable interrupt requests and macro service requests are not acknowledged and kept pending.

3.4.1 Interrupt request types

The interrupt requests are classified into the following four types:

- Nonmaskable interrupt
- Maskable interrupt
- Software interrupt
- OPE code trap interrupt

The interrupt requests are described below:

(1) Nonmaskable interrupts

The nonmaskable interrupts are nonmaskable interrupt requests whose acknowledge cannot be disabled. The interrupts can always be acknowledged. This means that they are unconditionally acknowledged even in the DI (disable interrupt) state. Vectored interrupt processing can be performed for the nonmaskable interrupts. The nonmaskable interrupt sources are the following two:

- NMI pin input (NMI)
- Watchdog timer output (WDT)

Unlike maskable interrupts, programmable priority level control is not applied to the nonmaskable interrupts. However, the priority levels between the interrupt requests caused by NMI pin input (NMI) and watchdog timer output (WDT) can be specified in a watchdog timer mode register (WDM).
To return from the NMI or WDT interrupt, use a RETI instruction.

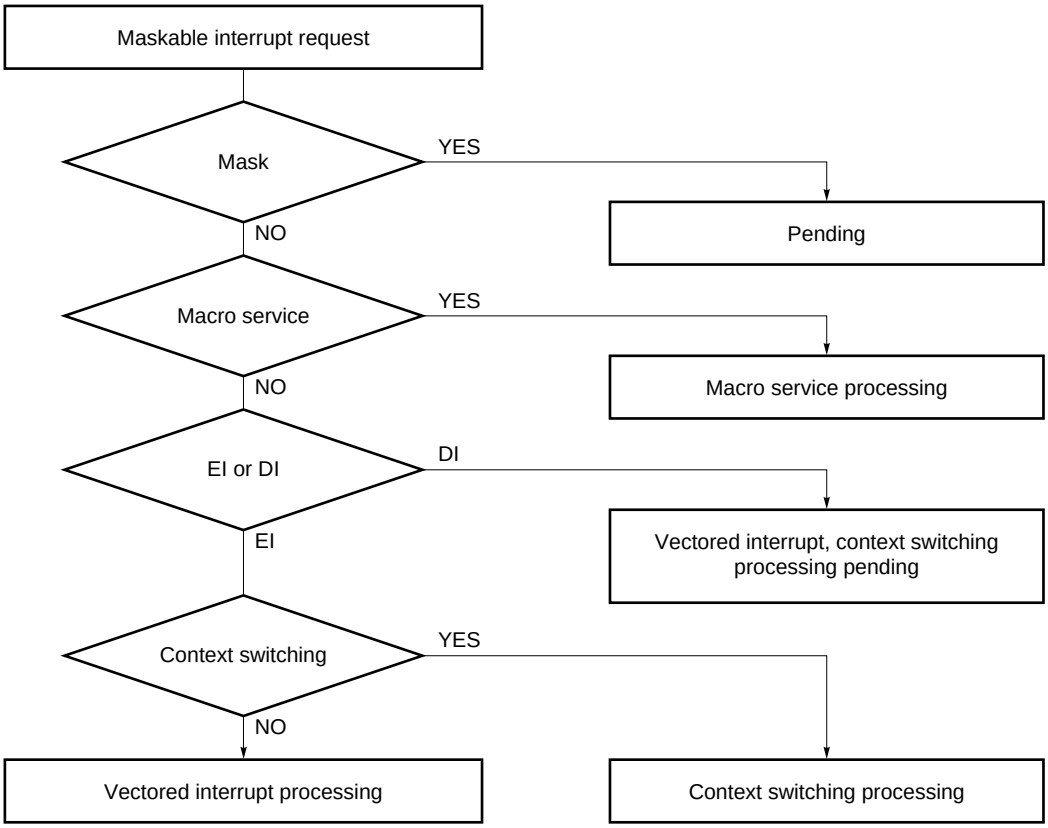
(2) Maskable interrupts

The maskable interrupts are interrupt requests whose acknowledge can be masked by setting the control register. Since they are interrupts occurring from peripheral hardware, their interrupt sources vary depending on the product. For the maskable interrupts, the processing mode can be selected among the following three:

- Vectored interrupt processing
- Macro service
- Context switching

When more than one maskable interrupt occurs at the same time, their priority levels are determined according to the default priority levels. Apart from the default priority levels, four interrupt priority levels can be set by setting the interrupt control register (programmable priority level control).
When an interrupt request is acknowledged, the DI state is entered, disabling the subsequent maskable interrupt requests.
When an EI instruction is executed in an interrupt service routine, the EI state is entered, enabling an interrupt request having the higher priority level than that of the current interrupt request being acknowledged (specified in the interrupt control register).
Interrupts having the same priority level cannot be nested. However, if the PRSL bit of an interrupt mode control register (IMC) is set to 0, an interrupt request having the same interrupt level is enabled only at the lowest level (level 3).
The macro service is acknowledged even in the DI state independently of the priority level.

Figure 3-8. Process Flow of Maskable Interrupt



(3) Software interrupts

The software interrupt is an interrupt request occurring when a CPU break instruction is executed, and can always be acknowledged. Vectored interrupt processing can be performed for the software interrupt.
The following two instructions cause a software interrupt to occur:

- BRK instruction : Branch to the address indicated by the contents of memory addresses 003EH, 003FH.
- BRKCS instruction : Branch by context switching processing. Switch to the register bank specified in the instruction.

If the instructions are executed in the DI state, the corresponding interrupts occur. Control of interrupt priority level specification is not applied to the interrupts.
When a BRK instruction is executed, unconditionally the vector table contents are set in the PC for causing the program flow to branch.
By executing a new BRK instruction in the interrupt service routine of a BRK instruction, its own routine can be nested.
If an EI instruction is executed in the interrupt service routine of a BRK instruction, the EI state is entered and a maskable interrupt request can be acknowledged.

(4) OPE code trap interrupts

The OPE code trap interrupt request occurs when a write into a watchdog timer mode register (WDM) or a standby control register (STBC) is not normally executed.

Write instructions into the WDM register and the STBC register consist of special operation codes. A write is enabled only when operation codes at the three and fourth bytes of the instruction are complementary to each other. If they are not complementary to each other, an OPE code trap interrupt is generated.

3.4.2 Interrupt processing modes

The following three interrupt processing modes can be used:

- Vectored interrupt processing
- Macro service
- Context switching

(1) Vectored interrupt processing

When an interrupt is acknowledged, automatically the program counter (PC) and the program status word (PSW) are saved in a stack memory and the program flow branches to the address indicated by the data stored in the vector table for executing the interrupt service routine.

To return from the interrupt service routine, use a RETI instruction.

(2) Macro service

When an interrupt is acknowledged, execution of the CPU is temporarily stopped and data is transferred by the hardware. Since the macro service is executed without intervention of the CPU, the CPU status of the PC PSW, etc., need not be saved or restored.

Therefore, the macro service has a large effect on improvement of the CPU service time.

(3) Context switching

When an interrupt is acknowledged, a predetermined register bank is selected by the hardware and the program flow branches to the vector address preset in the register bank. At the same time, the current PC and PSW are saved in the register bank.

Remark The context refers to the CPU registers that can be accessed from the current program being executed. The registers include the general purpose registers, PC, PSW, and SP (stack pointer).

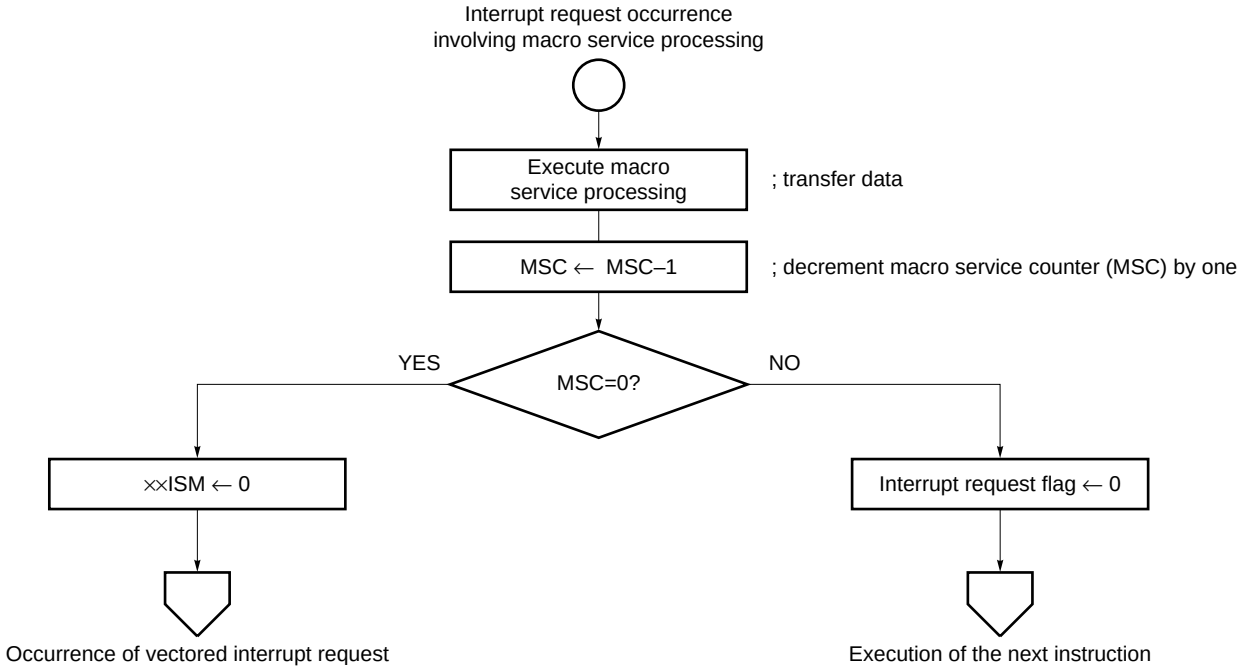
3.4.3 Macro service function

The macro service function is a function for transferring data between the special function register area and the memory space by the hardware when an interrupt request occurs.

When a macro service request occurs, the CPU temporarily stops program execution and 1/2-byte data transfer is automatically executed between the special function register (SFR) area and the memory. When the data transfer is completed, the interrupt request flag is reset to 0 and the CPU restarts program execution.

Further, data transfer is executed as many times as the count set in the macro service counter (MSC), then a vectored interrupt request is generated.

★ **Figure 3-9. Macro Service Process Sequence Example**



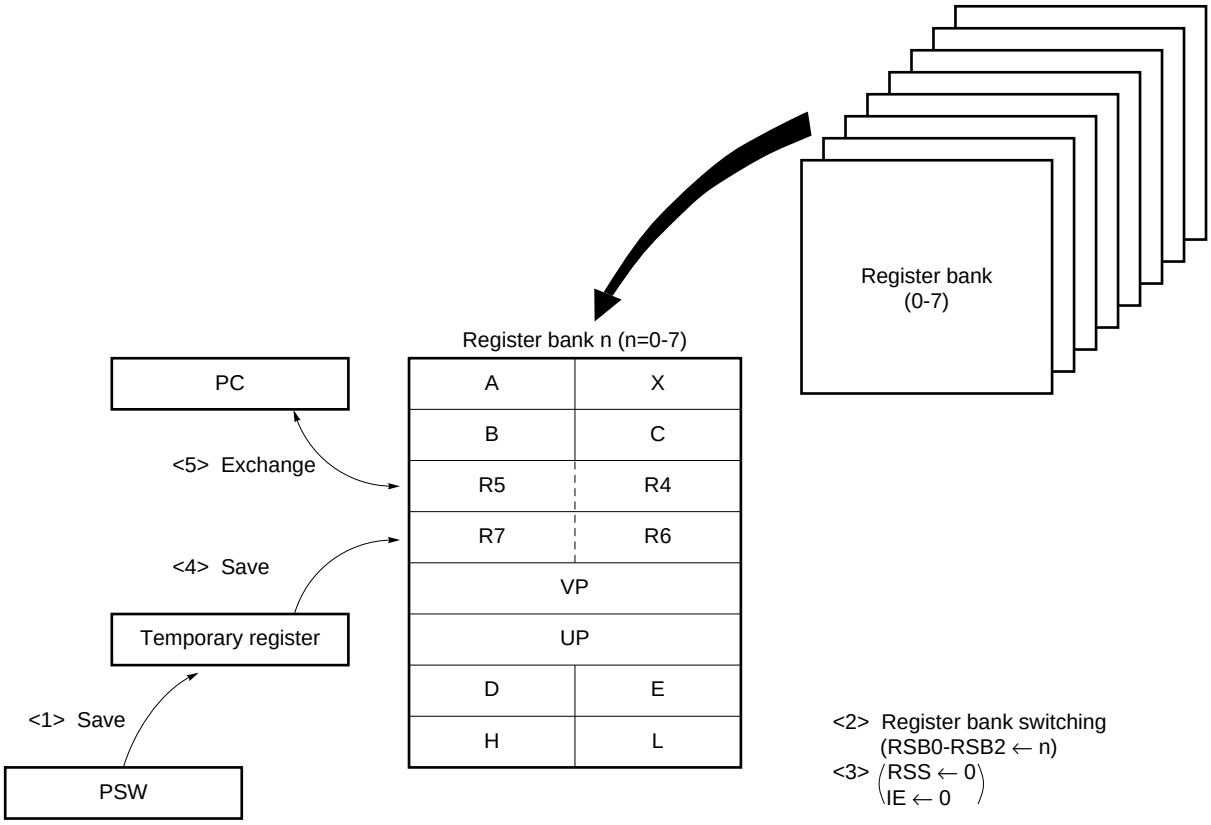
Unlike other interrupt processing, macro service function processing is automatically performed without starting any interrupt service program, thus does not involve a sequence of steps of branch to an interrupt service routine, saving/restoring register, and return from the interrupt service routine. Therefore, the CPU service time can be improved and the number of program steps can be reduced.

3.4.4 Context switching function

When an interrupt request occurs or a BRKCS instruction is executed, a predetermined register bank is selected by the hardware. At the same time as the program flow branches to the vector address prestored in the register bank, the current program counter (PC) and program status word (PSW) contents are stacked in the register bank.

To return from the service routine, use a RETCS instruction (if the function is started by the BRKCS instruction, RETCSB instruction).

Figure 3-10. Context Switching Operation when an Interrupt Request Occurs



3.4.5 Interrupt execution rates

The interrupt execution rates are listed below.

However, the priority level determination time is not contained. Since the priority level is determined every two clocks, the determination time changes in the range of 0 to two clocks depending on the interrupt occurrence timing.

n denotes the number of wait states specified in a programmable wait control register (PWC).

Caution In the three subseries of μPD78356, 78366A, and 78372, the number of clocks at the normal fetch is the same as that at the high-speed fetch. See the column "Normal fetch" in the interrupt execution rate lists given below.

(1) Vectored interrupt processing

Stack	No. of clocks	
	Normal fetch	High-speed fetch
Main RAM	21+2n	
Peripheral RAM	25+2n/33+2n[Note]	27+2n/37+2n[Note]
External memory	33+6n	37+2n

Note Even address/odd address

(2) Context switching processing

(a) Normal fetch: 17+2n

(b) High-speed fetch: 19

(3) Macro service processing

Macro service		No. of clocks			
		Normal fetch		High-speed fetch	
		Byte operation	Word operation	Byte operation	Word operation
EVTCNT		12		12	
BLKTRS mem → SFR		18	19	18	19
BLKTRS SFR → mem		17	18	17	18
BLKTRS-P mem → SFR	(IRAM)	20	21	20	21
	(PRAM)	22	23/27[Note]	23	24/29[Note]
	(EMEM)	22+n	27+2n	28	29
BLKTRS-P SFR → mem	(IRAM)	20	21	20	21
	(PRAM)	22	23/27[Note]	23	24/29[Note]
	(EMEM)	22+n	27+2n	28	29
DTADIF		–	22	–	22
DTADIF-P	(IRAM)	–	26	–	26
	(PRAM)		28/32[Note]		29/34[Note]
	(EMEM)		32+2n		34

Note Even address/odd address

3.4.6 Control registers

78K/III series interrupt processing is controlled for each interrupt request with control registers in which interrupt processing is specified. Tables 3-7 to 3-10 list the control registers for each subseries.

Table 3-7. Control Register List (μPD78352A Subseries)

Register name	Symbols
Interrupt control registers	OVIC, PIC0, PIC1, CMIC10, CMIC20, PIC2, PIC3
Interrupt mask flag register	MKL
In-service priority register	ISPR
Interrupt mode control register	IMC

Table 3-8. Control Register List (μPD78356 Subseries)

Register name	Symbols
Interrupt control registers	OIC0, OVIC3, PIC0, PIC1, PIC2, PIC3, PIC4, CMIC00, CMIC01, CMIC02, CMIC03, CMIC10, CMIC11, CMIC20, CMIC21, CMIC40, CMICUD0, CMICUD1, SERIC, SRIC, STIC, CSIIC0, CSIIC1, ADIC
Interrupt mask flag register	MK0, MK1, MK0H, MK0L, MK1L
In-service priority register	ISPR
Interrupt mode control register	IMC

Table 3-9. Control Register List (μPD78366A Subseries)

Register name	Symbols
Interrupt control registers	OVIC3, PIC0, PIC1, PIC2, PIC3, PIC4, TMIC0, CMIC03, CMIC10, CMIC40, CMIC41, SERIC, SRIC, STIC, CSIIC, ADIC
Interrupt mask flag register	MK0, MK0H, MK0L
In-service priority register	ISPR
Interrupt mode control register	IMC

Table 3-10. Control Register List (μPD78372 Subseries)

Register name	Symbols
Interrupt control registers	OVIC0, OVIC1, PLIC0, PHIC0, PLIC1, PHIC1, PLIC2, PHIC2, PLIC3, PHIC3, PIC4, PIC5, CMIC10, CMIC11, CMIC12, CMIC13, SERIC, SRIC, STIC, CSIIC, ADIC
Interrupt mask flag register	MK0, MK1, MK0H, MK0L, MK1L
In-service priority register	ISPR
Interrupt mode control register	IMC

[MEMO]

4.1 Instruction Addressing

The instruction address is automatically determined by the program counter (PC) contents and is automatically incremented according to the number of bytes of the fetched instruction (by one for one byte) each time one instruction is executed. However, when a branch instruction is executed, branch destination address information is set in the PC for branching by the addressing as listed below.

The flowing five instruction addressing modes can be used:

- Relative addressing
- Immediate addressing
- Table indirect addressing
- Register addressing
- Register indirect addressing

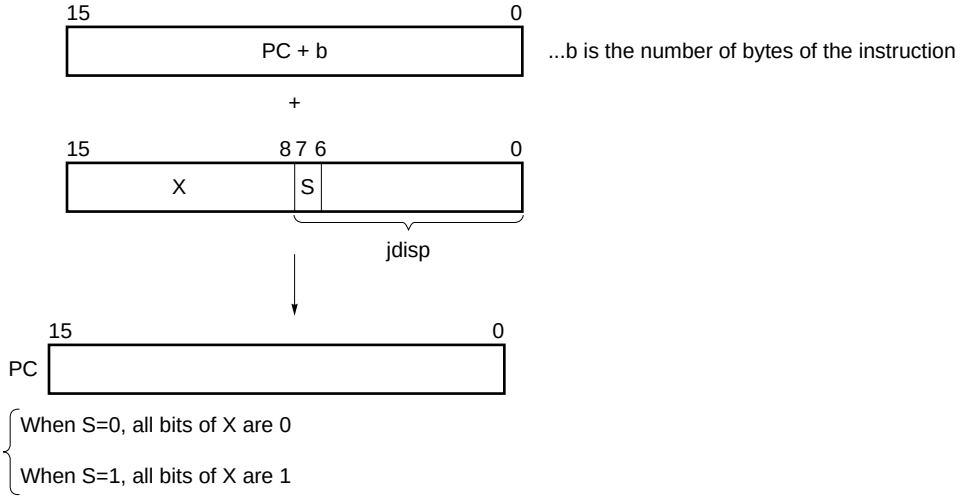
The addressing modes are described.

4.1.1 Relative addressing

The value resulting from adding 8-bit immediate data of operation code (displacement value: jdisp) to the top address of the following instruction is set in the program counter (PC) for branching. The displacement value is handled as signed two's complement data (from –128 to +127) and bit 7 is used as a sign bit.

The relative addressing is used to execute a BR $addr16 instruction or conditional branch instruction.

Figure 4-1. Relative Addressing



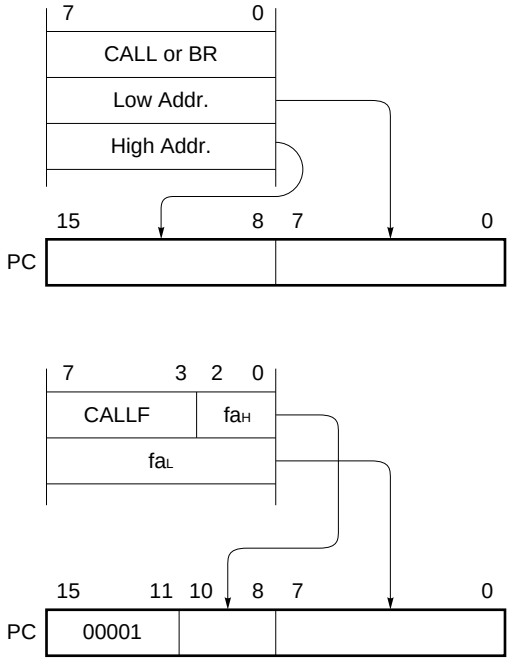
4.1.2 Immediate addressing

Immediate data in an instruction is transferred to the program counter (PC) for branching.

The immediate addressing is used to execute a CALL !addr16, BR !addr16, or CALLF !addr11 instruction.

When the CALLF !addr11 instruction is executed, a branch is taken to a fixed area whose high-order 5-bit address part is determined.

Figure 4-2. Immediate Addressing

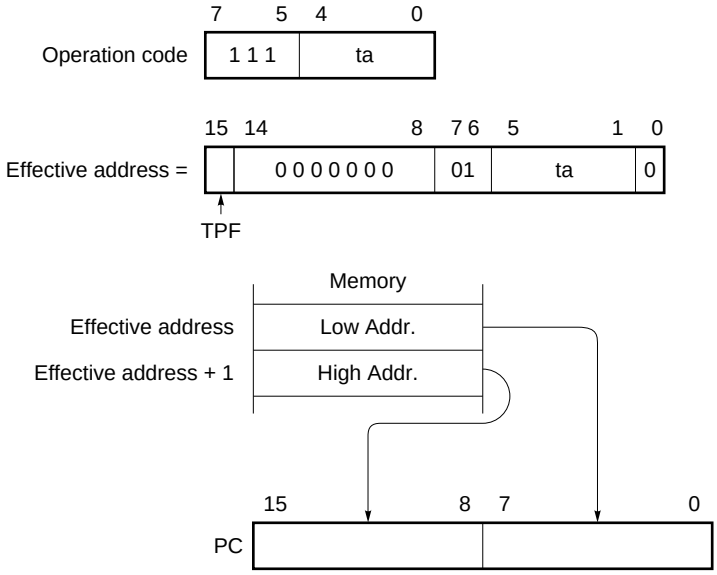


4.1.3 Table indirect addressing

The contents of the table in a specific location addressed by the immediate data of the low-order five bits of operation code are transferred to the program counter (PC) for branching.

The table indirect addressing is used to execute a CALLT [addr5] instruction.

Figure 4-3. Table Indirect Addressing

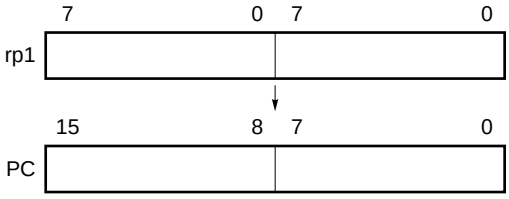


4.1.4 Register addressing

The contents of the register pair (RP0-RP7) specified by an instruction are transferred to the program counter (PC) for branching.

The register addressing is used to execute a BR rp1 or CALL rp1 instruction.

Figure 4-4. Register Addressing

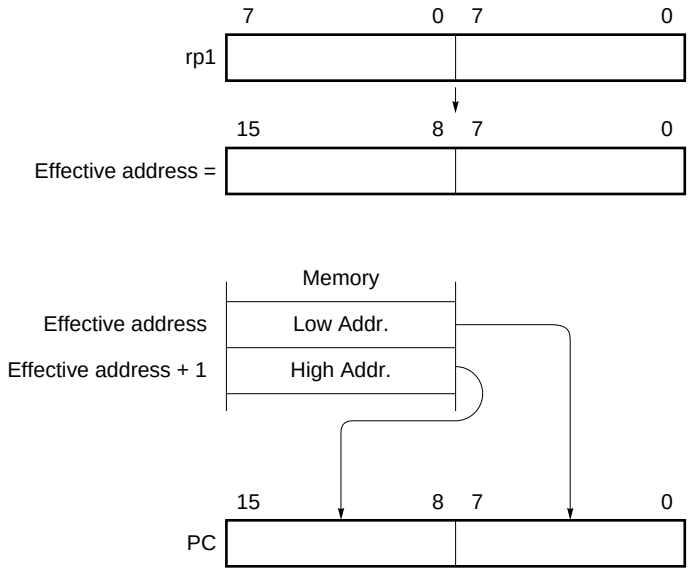


4.1.5 Register indirect addressing

The continuous 2-byte data in the memory addressed by the contents of the register pair (RP0-RP7) specified by an instruction are transferred to the program counter (PC) for branching.

The register indirect addressing is used to execute a BR [rp1] or CALL [rp1] instruction.

Figure 4-5. Register Indirect Addressing



4.2 Operand Addressing

The following ten modes can be used for addressing registers and memories on which operations are to be performed when instructions are executed:

- Register addressing
- Immediate addressing
- Direct addressing
- Short direct addressing
- Special function register (SFR) addressing
- Short direct memory indirect addressing
- Register indirect addressing
- Based addressing
- Indexed addressing
- Based indexed addressing

These addressing modes are described below.

4.2.1 Register addressing

In the register addressing, the general-purpose register specified in the register set selection flag (RSS) in the register bank specified in the register bank selection flag bits (RBS0-RBS2) and the register specification code (Rn, Pn, Qn) in an instruction is used as an operand for accessing.

The register addressing is used to execute instructions having the operand identifiers listed below. To specify an 8-bit register, three bits in an operation code are used to specify one of eight registers or four bits are used to specify one of 16 registers. To specify a 16-bit register pair, three bits in an operation code are used to specify one of eight register pairs.

Identifier	Description
r	R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15
r1	R0, R1, R2, R3, R4, R5, R6, R7
r2	C, B
rp	RP0, RP1, RP2, RP3, RP4, RP5, RP6, RP7
rp1	RP0, RP1, RP2, RP3, RP4, RP5, RP6, RP7
rp2	DE, HL, VP, UP

As r, r1, rp, and rp1, function names (X, A, C, B, E, D, L, H, AX, BC, DE, HL, VP, and UP) can be described in addition to the absolute names (R0-R15 and RP0-RP7).

The function names corresponding to the absolute names are as listed in Tables 5-2 and 5-3.

Example 1: MOV A, r1

Operation code | 1 1 0 1 0 R_2 R_1 R_0 |

To select the R2 register as r1, describe the following: (The R2 register becomes the C register when RSS=0.)

MOV A, R2

The following operation code is generated corresponding to the instruction:

Operation code | 1 1 0 1 0 0 1 0 |

2: INCW rp2

Operation code | 0 1 0 0 0 1 S_1 S_0 |

To select the DE register pair as rp2, describe the following:

INCW DE

The following operation code is generated corresponding to the instruction:

Operation code | 0 1 0 0 0 1 1 0 |

4.2.2 Immediate addressing

In the immediate addressing, 8-bit or 16-bit data is contained in an operation code.

The immediate addressing is used to execute instructions having the operand identifiers listed below.

Identifier	Description
byte	label, numeric value of up to eight bits word,
word	label, numeric value of up to 16 bits

Example: ADD A, #byte

Operation code | 1 0 1 0 1 0 0 0 |

| Data |

To take 77H as byte, describe the following:

ADD A, #77H

The following operation code is generated corresponding to the instruction:

Operation code | 1 0 1 0 1 0 0 0 |

| 0 1 1 1 0 1 1 1 |

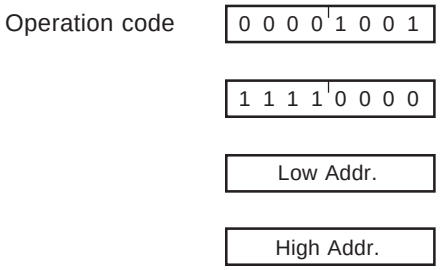
4.2.3 Direct addressing

In the direct addressing, immediate data in an instruction addresses the memory on which an operation is to be performed as operand address.

The direct addressing is used to execute instructions having the operand identifier listed below.

Identifier	Description
addr16	label, numeric value of up to 16 bits

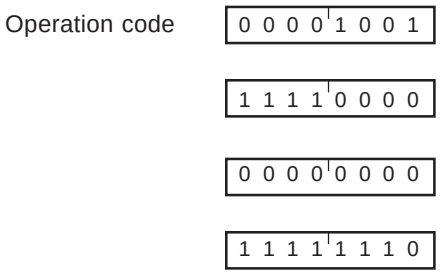
Example: MOV A, !addr16



To take FE00H as addr16, describe the following:

MOV A, !0FE00H

The following operation code is generated corresponding to the instruction:



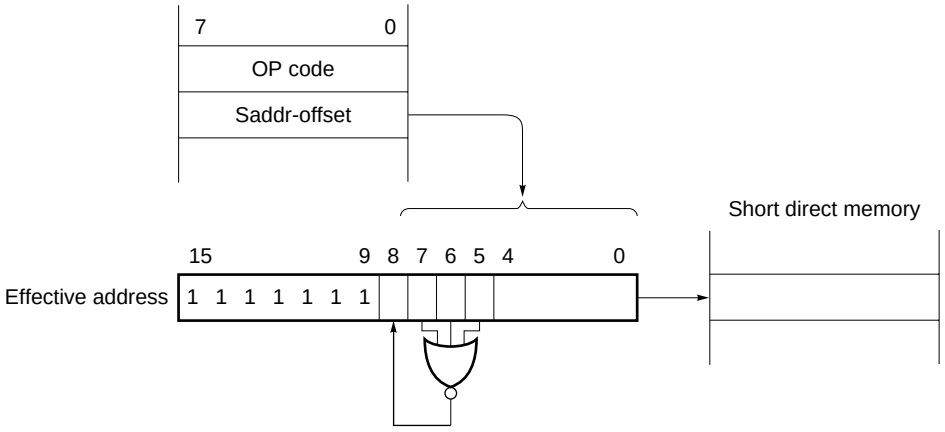
4.2.4 Short direct addressing

In the short direct addressing, 8-bit immediate data in an instruction directly addresses the fixed memory space on which an operation is to be performed.

The short direct addressing is applied to the 256-byte space of FE20H-FF1FH. The internal RAM (short direct memory) is mapped in FE20H-FEFFH and the special function registers (SFRs) are mapped in FF00H-FF1FH.

Bit 8 of an effective address is set to 0 when 8-bit immediate data is 20H-FFH or 1 when the data is 00H-1FH.

Figure 4-6. Short Direct Addressing



The short direct addressing is used to execute instructions containing saddr or saddrp as an operand.

When an instruction containing saddrp is executed, 2-byte data in the memory location addressed by the effective address and the following memory location (data at even-odd addresses where the least significant bit of the effective address is ignored) is accessed.

Identifier	Description
saddr	label, numeric value ranging from FE20H to FF1FH
saddrp	label, numeric value ranging from FE20H to FF1EH (even)

Example: MOV saddr, saddr

Operation code	0 0 1 1 1 0 0 0
	Saddr-offset
	Saddr-offset

To take FE30H as the first operand saddr and FE50H as the second operand saddr, describe the following:

MOV 0FE20H, 0FE50H

The following operation code is generated corresponding to the instruction:

Operation code	0 0 1 1 1 0 0 0
	0 0 1 0 0 0 0 0
	0 1 1 0 0 0 0 0

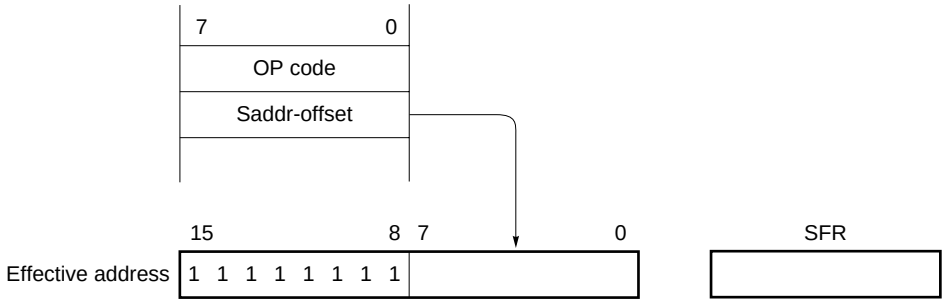
4.2.5 Special function register (SFR) addressing

In the special function register (SFR) addressing, 8-bit immediate data in an operation instruction addresses a memory-mapped special function register (SFR).

The space in which the SFRs are mapped to which the SFR addressing is applied is the 256-byte space of FF00H-FFFFH. However, the SFRs mapped in FF00H-FF1FH are accessed not only in the SFR addressing mode, but also in the short direct addressing mode.

Remark With the assembler package manufactured by NEC (RA78K3), the short direct addressing is used automatically (forcibly) for instructions for the SFRs mapped in FF00H-FF1FH.

Figure 4-7. Special Function Register Addressing



Identifier	Description
sfr	Symbol of special function register
sfrp	Symbol of special function register where 16-bit operation can be performed

Example: MOV sfr, A

Operation code | 0 0 0 1 0 0 1 0 |

| Sfr-offset |

To specify PM0 as sfr, describe the following:

MOV PM0, A

The following operation code is generated corresponding to the instruction:

Operation code | 0 0 0 1 0 0 1 0 |

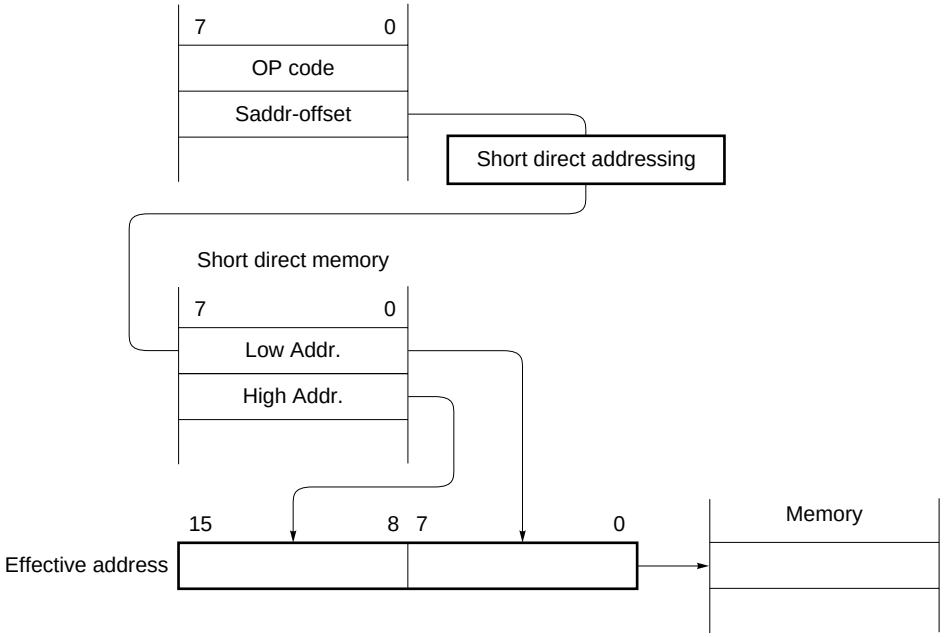
| 0 0 1 0 0 0 0 0 |

4.2.6 Short direct memory indirect addressing

In the short direct memory indirect addressing, the contents of the contiguous 2-byte short direct memory area addressed by 8-bit immediate data in an instruction address the memory on which an operation is to be performed as operand address.

The short direct memory indirect addressing is used to execute instructions having [saddrp] as operands.

Figure 4-8. Short Direct Memory Indirect Addressing



Identifier	Description
[saddrp]	[label, numeric value ranging from FE20H to FF1F (even)]

Example: XCH A, [saddrp]

Operation code | 0 0 1 0 0 0 1 1

Saddr-offset

To take FEA0H as saddrp, describe the following:

XCH A, [0FEA0H]

The following operation code is generated corresponding to the instruction:

Operation code | 0 0 1 0 0 0 1 1

1 0 1 0 0 0 0 0

4.2.7 Register indirect addressing

In the register indirect addressing, the contents of the register pair specified in the register set selection flag (RSS) in the register bank specified in the register bank selection flag bits (RBS1-RBS2) and the register pair specification code in an instruction address the memory on which an operation is to be performed as operand address.

The register indirect addressing is used to execute instructions having the following operand identifiers:

Identifier	Description
mem	[DE], [HL], [DE+], [HL+], [DE–], [HL–], [VP], [UP]
[rp1]	[RP0], [RP1], [RP2], [RP3], [RP4], [RP5], [RP6], [RP7]

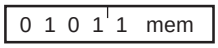
The register indirect addressing with the register pair DE or HL enables the register pair contents to be incremented or decremented by one for the next addressing after addressing the memory.

To use this function, describe [DE+], [HL+], [DE–], or [HL–] in the operand field mem.

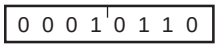
Example 1: MOV A, mem

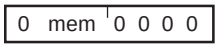
Operation code

- When [DE], [HL], [DE+], [HL+], [DE-], or [HL–] in register indirect mode is described as mem



- When the instruction is described in register indirect mode other than the above

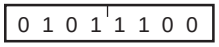




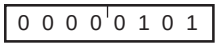
To specify [DE] as mem, describe the following:

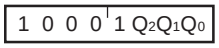
MOV A, [DE]

The following operation code is generated corresponding to the instruction:

Operation code 

2: ROR4 [rp1]

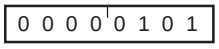
Operation code 

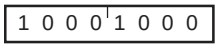


To select RP0 as rp1, describe the following:

ROR4 [RP0]

The following operation code is generated corresponding to the instruction:

Operation code 



3: ADD A, mem

Operation code (in register indirect mode)

0	0	0	1	0	1	1	0

0	mem	1	0	0	0

To specify [HL+] as mem, describe the following:

ADD A, [HL+]

The following operation code is generated corresponding to the instruction:

Operation code

0	0	0	1	0	1	1	0

0	0	0	1	1	0	0	0

4.2.8 Based addressing

In the based addressing, the sum of the contents of the 16-bit register or register pair (DE, SP, HL, UP, or VP) specified in the register set selection flag (RSS) in the register bank specified in the register bank selection flag bits (RBS0-RBS2) and the addressing code (mem) in an instruction and 8-bit immediate data specified as operand addresses the memory on which an operation is to be performed as operand address.

The based addressing is used to execute instructions having the following operand identifier:

Identifier	Description
mem	[DE+byte], [SP+byte], [HL+byte], [UP+byte], [VP+byte]

Example: AND A, mem

Operation code

0	0	0	0	0	1	1	0

0	mem	1	1	0	0

offset

To select based addressing of the sum of register pair VP and immediate data 10H as mem, describe the following:

AND A, [VP+10H]

The following operation code is generated corresponding to the instruction:

Operation code

0	0	0	0	0	1	1	0

0	1	0	0	1	1	0	0

0	0	0	1	0	0	0	0

4.2.9 Indexed addressing

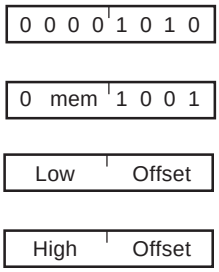
In the based addressing, the sum of the contents of the 8-bit register or 16-bit register pair (A, B, DE, or HL) specified in the register set selection flag (RSS) in the register bank specified in the register bank selection flag bits (RBS0-RBS2) and the addressing code (mem) in an instruction and 16-bit immediate data specified as operand addresses the memory on which an operation is to be performed as operand address.

The based addressing is used to execute instructions having the following operand identifier:

Identifier	Description
mem	word[DE], word[A], word[HL], word[B]

Example: ADDC A, mem

Operation code

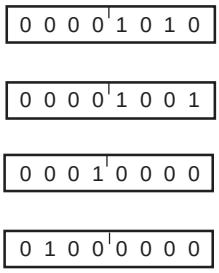


To select indexed addressing of the sum of register pair DE and immediate data 4010H as mem, describe the following:

ADDC A, 4010H[DE]

The following operation code is generated corresponding to the instruction:

Operation code



4.2.10 Based indexed addressing

In the based indexed addressing, the sum of the contents of the 16-bit register (DE, HL, or VP) specified in the register set selection flag (RSS) in the register bank specified in the register bank selection flag bits (RBS0-RBS2) and the addressing code (mem) in an instruction and the contents of 8-bit or 16-bit register (A, B, DE, or HL) addresses the memory on which an operation is to be performed as operand address.

The based addressing is used to execute instructions having the following operand identifier:

Identifier	Description
mem	[DE+A], [HL+A], [DE+B], [HL+B], [VP+DE], [VP+HL]

Example: OR A, mem

Operation code

0	0	0	1	0	1	1	1

0	mem	1	0	0	1

To select based indexed addressing of the sum of register pair HL and register B as mem, describe the following:

SUBC A, [HL+B]

The following operation code is generated corresponding to the instruction:

Operation code

0	0	0	1	0	1	1	1

0	0	1	1	1	0	1	1

CHAPTER 5 INSTRUCTION SET LIST

This chapter describes the μPD78356 instruction repertoire.

5.1 List of Operations

5.1.1 Operand identifier and description

Operands are coded in the operand field of each instruction as listed in the description column of Table 20-1. For details of the operand format, refer to the relevant assembler specifications. When several coding forms are presented, any one of them is selected. Uppercase letters and the symbols, +, –, #, $, !, and [ ], are keywords and must be written as they are.

For immediate data, an appropriate numeric or label must be written. The symbols #, $, !, and [ ] must not be omitted when describing labels.

Table 5-1. Operand Identifier and Description

Identifier	Description
r r1 r2	R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15 R0, R1, R2, R3, R4, R5, R6, R7 C, B
rp rp1 rp2	RP0, RP1, RP2, RP3, RP4, RP5, RP6, RP7 RP0, RP1, RP2, RP3, RP4, RP5, RP6, RP7 DE, HL, VP, UP
sfr sfrp	Special function register name Special function register name
post	RP0, RP1, RP2, RP3, RP4, RP5/PSW, RP6, RP7 (Can be coded more than once. However, RP5 can only be used in a PUSH or POP instruction and PSW can only be used in a PUSHU or POPU instruction.)
mem	[DE], [HL], [DE+], [HL+], [DE–], [HL–], [VP], [UP] : Register indirect mode [DE+A], [HL+A], [DE+B], [HL+B], [VP+DE], [VP+HL] : Based indexed mode [DE+byte], [HL+byte], [VP+byte], [UP+byte], [SP+byte] : Based mode word[A], word[B], word[DE], word[HL] : Indexed mode
saddr saddrp	FE20H-FF1FH Immediate data or label FE20H-FE1EH Immediate data (bit 0 = 0, however) or label (for 16-bit manipulation)
$addr16 !addr16 addr11 addr5	0000H-FDFFH Immediate data or label : Relative addressing 0000H-FDFFH Immediate data or label : Immediate addressing (Data at an address up to FFFFH can be coded in an MOV instruction. Data at an address from FE00H to FEFFH can be coded in an MOVTBLW instruction.) 800H-FFFH Immediate data or label 40H-7EH Immediate data (bit 0 = 0, however)[Note] or label
word byte bit n	16-bit immediate data or label 8-bit immediate data or label 3-bit immediate data or label 3-bit immediate data (0 to 7)

Note Do not attempt to access word data at an odd-numbered address (bit 0 = 1).

Remarks 1. The same register name can be specified in rp and rp1, but different codes are generated. (See **5.2 Instruction Codes**.)

2. Immediate addressing is effective for entire address spaces. Relative addressing is effective for the locations within a displacement range of –128 to +127 from the starting address of the next instruction.

The 8-bit registers (r, r1) and 16-bit register pairs (rp, rp1, post) can be represented by either absolute names (R0-R15, RP0-RP7) or function names. Table 5-2 lists the absolute names and corresponding function names of an 8-bit register. Table 5-3 lists those of a 16-bit register.

Table 5-2. Absolute Names and Their Corresponding Function Names of an 8-bit Register

Absolute name	Function name		Absolute name	Function name	
	RSS=0	RSS=1		RSS=0	RSS=1
R0	X		R8	VP_L	VP_L
R1	A		R9	VP_H	VP_H
R2	C		R10	UP_L	UP_L
R3	B		R11	UP_H	UP_H
R4		X	R12	E	E
R5		A	R13	D	D
R6		C	R14	L	L
R7		B	R15	H	H

Table 5-3. Absolute Names and Their Corresponding Function Names of a 16-bit Register

Absolute name	Function name	
	RSS=0	RSS=1
RP0	AX	
RP1	BC	
RP2		AX
RP3		BC
RP4	VP	VP
RP5	UP	UP
RP6	DE	DE
RP7	HL	HL

RSS stands for the register set selection flag (bit 5 of PSW). Setting or resetting RSS switches function names for correspondence with an absolute name.

5.1.2 Legend

A	: A register; 8-bit accumulator
X	: X register
B	: B register
C	: C register
D	: D register
E	: E register
H	: H register
L	: L register
R0-R15	: Register 0 to register 15 (absolute name)
AX	: Register pair (AX); 16-bit accumulator
BC	: Register pair (BC)
DE	: Register pair (DE)
HL	: Register pair (HL)
RP0-RP7	: Register pair 0 to register pair 7 (absolute name)
PC	: Program counter
SP	: Stack pointer
UP	: User stack pointer
PSW	: Program status word
CY	: Carry flag
AC	: Auxiliary carry flag
Z	: Zero flag
P/V	: Parity/overflow flag
S	: Sign flag
TPF	: Table position flag
RBS	: Register bank selecting flag
RSS	: Register set selecting flag
IE	: Interrupt request enable flag
STBC	: Standby control register
WDM	: Watchdog timer mode register
jdisp8	: Signed 8-bit data (displacement value: –128 to +127)
( )	: Contents at an address enclosed in parentheses or at an address indicated in a register indicated in parentheses. (+) and (–) indicate that an address or the contents of a register indicated in parentheses are incremented and decremented by one after execution of the instruction, respectively.
(( ))	: Contents at an address indicated by the contents at an address indicated in parentheses (( )).
xxH	: Hexadecimal number
x_H, x_L	: High-order 8 bits and low-order 8 bits of 16-bit register

5.1.3 Notational symbols in flag operation field

Table 5-4. Notational Symbols in Flag Operation Field

Symbol	Explanation
(Blank)	No change
0	Cleared to zero.
1	Set to 1.
x	Set or reset according to the result.
P	P/V flag operates as a parity flag.
V	P/V flag operates as an overflow flag.
R	Saved value are restored.

5.1.4 Instruction set differences among 78K/III Series products

Table 5-5. Instruction Set Differences among 78K/III Series Products

Parameter \ Subseries name / Typical product	μPD78356 μPD78366A μPD78372	μPD78352A	μPD78322 μPD78328 μPD78334	μPD78312A
	μPD78356	μPD78352A	μPD78322	μPD78312A
No. of basic instructions	115	113	111	104
Instruction set	Addition of the following instructions to μPD78322. • Sum-of-products instruction • Table shift instruction • Sum-of-products instruction with saturation function • Correlation instruction	Addition of the following instruction to μPD78322 • Sum-of-products instruction • Table shift instruction	Addition of a large number of instructions to μPD78312A	—

5.1.5 Operations of basic instructions

(1) 8-bit data transfer instructions: MOV, XCH

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
MOV	r1,#byte	2	r1←byte	
	saddr,#byte	3	(saddr)←byte	
	sfr[Note],#byte	3	sfr←byte	
	r,r1	2	r←r1	
	A,r1	1	A←r1	
	A,saddr	2	A←(saddr)	
	saddr,A	2	(saddr)←A	
	saddr,saddr	3	(saddr)←(saddr)	
	A,sfr	2	A←sfr	
	sfr,A	2	sfr←A	
	A,mem	1-4	A←(mem)	
	mem,A	1-4	(mem)←A	
	a,[saddrp]	2	A←((saddrp))	
	[saddrp],A	2	((saddrp))←A	
	A,!addr16	4	A←(addr16)	
	!addr16,A	4	(addr16)←A	
	PSWL,#byte	3	PSW_L←byte	× × × × ×
	PSWH,#byte	3	PSW_H←byte	
	PSWL,A	2	PSW_L←A	× × × × ×
	PSWH,A	2	PSW_H←A	
	A,PSWL	2	A←PSW_L	
	A,PSWH	2	A←PSW_H	
XCH	A,r1	1	A↔r1	
	r,r1	2	r↔r1	
	A,mem	2-4	A↔(mem)	
	A,saddr	2	A↔(saddr)	
	A,sfr	3	A↔sfr	
	A,[saddrp]	2	A↔((saddrp))	
	saddr,saddr	3	(saddr)↔(saddr)	

Note If STBC or WDM is coded in sfr, a different instruction having the different byte count is generated.

(2) 16-bit data transfer instructions: MOVW, XCHW

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
MOVW	rp1,#word	3	rp1←word	
	saddrp,#word	4	(saddrp)←word	
	sfrp,#word	4	sfrp←word	
	rp,rp1	2	rp←rp1	
	AX,saddrp	2	AX←(saddrp)	
	saddrp,AX	2	(saddrp)←AX	
	saddrp,saddrp	3	(saddrp)←(saddrp)	
	AX,sfrp	2	AX←sfrp	
	sfrp,AX	2	sfrp←AX	
	rp1,!addr16	4	rp1←(addr16)	
	!addr16,rp1	4	(addr16)←rp1	
	AX,mem	2-4	AX←(mem)	
	mem,AX	2-4	(mem)←AX	
XCHW	AX,saddrp	2	AX↔(saddrp)	
	AX,sfrp	3	AX↔sfrp	
	saddrp,saddrp	3	(saddrp)↔(saddrp)	
	rp,rp1	2	rp↔rp1	
	AX,mem	2-4	AX↔(mem)	

(3) 8-bit arithmetic/logical instructions: ADD, ADDC, SUB, SUBC, AND, OR, XOR, CMP

Mnemonic	Operand	Byte	Operation	Flag				
				S	Z	AC	P/V	CY
ADD	A,#byte	2	A,CY←A+byte	×	×	×	V	×
	saddr,#byte	3	(saddr),CY←(saddr)+byte	×	×	×	V	×
	sfr,#byte	4	sfr,CY←sfr+byte	×	×	×	V	×
	r,r1	2	r,CY←r+r1	×	×	×	V	×
	A,saddr	2	A,CY←A+(saddr)	×	×	×	V	×
	A,sfr	3	A,CY←A+sfr	×	×	×	V	×
	saddr,saddr	3	(saddr),CY←(saddr)+(saddr)	×	×	×	V	×
	A,mem	2-4	A,CY←A+(mem)	×	×	×	V	×
	mem,A	2-4	(mem),CY←(mem)+A	×	×	×	V	×
ADDC	A,#byte	2	A,CY←A+byte+CY	×	×	×	V	×
	saddr,#byte	3	(saddr),CY←(saddr)+byte+CY	×	×	×	V	×
	sfr,#byte	4	sfr,CY←sfr+byte+CY	×	×	×	V	×
	r,r1	2	r,CY←r+r1+CY	×	×	×	V	×
	A,saddr	2	A,CY←A+(saddr)+CY	×	×	×	V	×
	A,sfr	3	A,CY←A+sfr+CY	×	×	×	V	×
	saddr,saddr	3	(saddr),CY←(saddr)+(saddr)+CY	×	×	×	V	×
	A,mem	2-4	A,CY←A+(mem)+CY	×	×	×	V	×
	mem,A	2-4	(mem),CY←(mem)+A+CY	×	×	×	V	×
SUB	A,#byte	2	A,CY←A–byte	×	×	×	V	×
	saddr,#byte	3	(saddr),CY←(saddr)–byte	×	×	×	V	×
	sfr,#byte	4	sfr,CY←sfr–byte	×	×	×	V	×
	r,r1	2	r,CY←r–r1	×	×	×	V	×
	A,saddr	2	A,CY←A–(saddr)	×	×	×	V	×
	A,sfr	3	A,CY←A–sfr	×	×	×	V	×
	saddr,saddr	3	(saddr),CY←(saddr)–(saddr)	×	×	×	V	×
	A,mem	2-4	A,CY←A–(mem)	×	×	×	V	×
	mem,A	2-4	(mem),CY←(mem)–A	×	×	×	V	×
SUBC	A,#byte	2	A,CY←A–byte–CY	×	×	×	V	×
	saddr,#byte	3	(saddr),CY←(saddr)–byte–CY	×	×	×	V	×
	sfr,#byte	4	sfr,CY←sfr–byte–CY	×	×	×	V	×
	r,r1	2	r,CY←r–r1–CY	×	×	×	V	×
	A,saddr	2	A,CY←A–(saddr)–CY	×	×	×	V	×
	A,sfr	3	A,CY←A–sfr–CY	×	×	×	V	×
	saddr,saddr	3	(saddr),CY←(saddr)–(saddr)–CY	×	×	×	V	×

Mnemonic	Operand	Byte	Operation	Flag				
				S	Z	AC	P/V	CY
SUBC	A,mem	2-4	A,CY←A–(mem)–CY	×	×	×	V	×
	mem,A	2-4	(mem),CY←(mem)–A–CY	×	×	×	V	×
AND	A,#byte	2	A←A ∧ byte	×	×		P	
	saddr,#byte	3	(saddr)←(saddr) ∧ byte	×	×		P	
	sfr,#byte	4	sfr←sfr ∧ byte	×	×		P	
	r,r1	2	r←r ∧ r1	×	×		P	
	A,saddr	2	A←A ∧ (saddr)	×	×		P	
	A,sfr	3	A←A ∧ sfr	×	×		P	
	saddr,saddr	3	(saddr)←(saddr) ∧ (saddr)	×	×		P	
	A,mem	2-4	A←A ∧ (mem)	×	×		P	
	mem,A	2-4	(mem)←(mem) ∧ A	×	×		P	
OR	A,#byte	2	A←A ∨ byte	×	×		P	
	saddr,#byte	3	(saddr)←(saddr) ∨ byte	×	×		P	
	sfr,#byte	4	sfr←sfr ∨ byte	×	×		P	
	r,r1	2	r←r ∨ r1	×	×		P	
	A,saddr	2	A←A ∨ (saddr)	×	×		P	
	A,sfr	3	A←A ∨ sfr	×	×		P	
	saddr,saddr	3	(saddr)←(saddr) ∨ (saddr)	×	×		P	
	A,mem	2-4	A←A ∨ (mem)	×	×		P	
	mem,A	2-4	(mem)←(mem) ∨ A	×	×		P	
XOR	A,#byte	2	A←A ⊻ byte	×	×		P	
	saddr,#byte	3	(saddr)←(saddr) ⊻ byte	×	×		P	
	sfr,#byte	4	sfr←sfr ⊻ byte	×	×		P	
	r,r1	2	r←r ⊻ r1	×	×		P	
	A,saddr	2	A←A ⊻ (saddr)	×	×		P	
	A,sfr	3	A←A ⊻ sfr	×	×		P	
	saddr,saddr	3	(saddr)←(saddr) ⊻ (saddr)	×	×		P	
	A,mem	2-4	A←A ⊻ (mem)	×	×		P	
	mem,A	2-4	(mem)←(mem) ⊻ A	×	×		P	
CMP	A,#byte	2	A–byte	×	×	×	V	×
	saddr,#byte	3	(saddr)–byte	×	×	×	V	×
	sfr,#byte	4	sfr–byte	×	×	×	V	×
	r,r1	2	r–r1	×	×	×	V	×
	A,saddr	2	A–(saddr)	×	×	×	V	×
	A,sfr	3	A–sfr	×	×	×	V	×
	saddr,saddr	3	(saddr)–(saddr)	×	×	×	V	×
	A,mem	2-4	A–(mem)	×	×	×	V	×
	mem,A	2-4	(mem)–A	×	×	×	V	×

(4) 16-bit arithmetic/logical instructions: ADDW, SUBW, CMPW

Mnemonic	Operand	Byte	Operation	Flag				
				S	Z	AC	P/V	CY
ADDW	AX,#word	3	AX,CY←AX+word	×	×	×	V	×
	saddrp,#word	4	(saddrp),CY←(saddrp)+word	×	×	×	V	×
	sfrp,#word	5	sfrp,CY←sfrp+word	×	×	×	V	×
	rp,rp1	2	rp,CY←rp+rp1	×	×	×	V	×
	AX,saddrp	2	AX,CY←AX+(saddrp)	×	×	×	V	×
	AX,sfrp	3	AX,CY←AX+sfrp	×	×	×	V	×
	saddrp,saddrp	3	(saddrp),CY←(saddrp)+(saddrp)	×	×	×	V	×
SUBW	AX,#word	3	AX,CY←AX–word	×	×	×	V	×
	saddrp,#word	4	(saddrp),CY←(saddrp)–word	×	×	×	V	×
	sfrp,#word	5	sfrp,CY←sfrp–word	×	×	×	V	×
	rp,rp1	2	rp,CY←rp–rp1	×	×	×	V	×
	AX,saddrp	2	AX,CY←AX–(saddrp)	×	×	×	V	×
	AX,sfrp	3	AX,CY←AX–sfrp	×	×	×	V	×
	saddrp,saddrp	3	(saddrp),CY←(saddrp)–(saddrp)	×	×	×	V	×
CMPW	AX,#word	3	AX–word	×	×	×	V	×
	saddrp,#word	4	(saddrp)–word	×	×	×	V	×
	sfrp,#word	5	sfrp–word	×	×	×	V	×
	rp,rp1	2	rp–rp1	×	×	×	V	×
	AX,saddrp	2	AX–(saddrp)	×	×	×	V	×
	AX,sfrp	3	AX–sfrp	×	×	×	V	×
	saddrp,saddrp	3	(saddrp)–(saddrp)	×	×	×	V	×

(5) Multiply/divide instructions: MULU, DIVUW, MULUW, DIVUX

Mnemonic	Operand	Byte	Operation	Flag				
				S	Z	AC	P/V	CY
MULU	r1	2	AX←Axr1					
DIVUW	r1	2	AX(quotient),r1(remainder)←AX÷r1					
MULUW	rp1	2	AX(high-order 16 bits), rp1 (low-order 16 bits)←AXxrp1					
DIVUX	rp1	2	AXDE(quotient), rp1(remainder)←AXDE÷rp1					

(6) Signed multiply instruction: MULW

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
MULW	rp1	2	AX(high-order 16 bits), rp1(low-order 16 bits)←AXxrp1	

(7) Sum-of-products instruction: MACW

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
MACW	n	3	AXDE←(B)×(C)+AXDE B←B+2, C←C+2, n←n–1 End if n=0 or P/V=1	× × × V ×

(8) Sum-of-products instruction with saturation function: MACSW

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
MACSW	n	3	AXDE←(B)×(C)+AXDE B←B+2, C←C+2, n←n–1 if overflow (P/V=1) then AXDE←7FFFFFFFH if underflow (P/V=1) then AXDE←80000000H end if n=0 or P/V=1	× × × V ×

Remark The μPD78352A Subseries does not provide the sum-of-products instruction with saturation function.

(9) Correlation instruction: SACW

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
SACW	[DE+], [HL+]	4	AX←AX+I(DE)–(HL)I DE←DE+2, HL←HL+2, C←C–1 end if C=0 or CY=1	× × × V ×

Remark The μPD78352A Subseries does not provide the correlation instruction.

(10) Table shift instruction: MOVTBLW

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
MOVTBLW	!addr16,n	4	(addr16+2)←(addr16), n←n–1 addr16←addr16–2, End if n=0	

Remark The addressing range of the table shift instruction is FE00H to FEFFH.

(11) Increment/decrement instructions: INC, DEC, INCW, DECW

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
INC	r1	1	r1←r1+1	× × × V
	saddr	2	(saddr)←(saddr)+1	× × × V
DEC	r1	1	r1←r1–1	× × × V
	saddr	2	(saddr)←(saddr)–1	× × × V
INCW	rp2	1	rp2←rp2+1	
	saddrp	3	(saddrp)←(saddrp)+1	
DECW	rp2	1	rp2←rp2–1	
	saddrp	3	(saddrp)←(saddrp)–1	

(12) Shift/rotate instructions: ROR, ROL, RORC, ROLC, SHR, SHL, SHRW, SHLW, ROR4, ROL4

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
ROR	r1,n	2	(CY, $r1_7$←$r1_0$, $r1_{m-1}$←$r1_m$) × n times (n=0–7)	P ×
ROL	r1,n	2	(CY, $r1_0$←$r1_7$, $r1_{m+1}$←$r1_m$) × n times (n=0-7)	P ×
RORC	r1,n	2	(CY←$r1_0$, $r1_7$←CY, $r1_{m-1}$←$r1_m$) × n times (n=0–7)	P ×
ROLC	r1,n	2	(CY←$r1_7$, $r1_0$←CY, $r1_{m+1}$←$r1_m$) × n times (n=0–7)	P ×
SHR	r1,n	2	(CY←$r1_0$, $r1_7$←0, $r1_{m-1}$←$r1_m$) × n times (n=0–7)	× × 0 P ×
SHL	r1,n	2	(CY←$r1_7$, $r1_0$←0, $r1_{m+1}$←$r1_m$) × n times (n=0–7)	× × 0 P ×
SHRW	rp1,n	2	(CY←$rp1_0$, $rp1_{15}$←0, $rp1_{m-1}$←$rp1_m$) × n times (n=0–7)	× × 0 P ×
SHLW	rp1,n	2	(CY←$rp1_{15}$,$rp1_0$←0, $rp1_{m+1}$←$rp1_m$) × n times (n=0–7)	× × 0 P ×
ROR4	[rp1]	2	A_{3-0}←$(rp1)_{3-0}$, $(rp1)_{7-4}$←A_{3-0}, $(rp1)_{3-0}$←$(rp1)_{7-4}$	
ROL4	[rp1]	2	A_{3-0}←$(rp1)_{7-4}$, $(rp1)_{3-0}$←A_{3-0}, $(rp1)_{7-4}$←$(rp1)_{3-0}$	

Remark n indicates the number of shifts or rotations.

(13) BCD correction instructions: ADJBA, ADJBS

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
ADJBA		2	Decimal adjust accumulator	× × × P ×
ADJBS				

(14) Data conversion instruction: CVTBW

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
CVTBW		1	When A_7=0, X←A, A←00H When A_7=1, X←A, A←FFH	

(15) Bit manipulation instructions: MOV1, AND1, OR1, XOR1, SET1, CLR1, NOT1

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
MOV1	CY,saddr.bit	3	CY←(saddr.bit)	×
	CY,sfr.bit	3	CY←sfr.bit	×
	CY,A.bit	2	CY←A.bit	×
	CY,X.bit	2	CY←X.bit	×
	CY,PSWH.bit	2	CY←PSW_H.bit	×
	CY,PSWL.bit	2	CY←PSW_L.bit	×
	saddr.bit,CY	3	(saddr.bit)←CY	
	sfr.bit,CY	3	sfr.bit←CY	
	A.bit,CY	2	A.bit←CY	
	X.bit,CY	2	X.bit←CY	
	PSWH.bit,CY	2	PSW_H.bit←CY	
	PSWL.bit, CY	2	PSW_L.bit←CY	
AND1	CY,saddr.bit	3	CY←CY ∧ (saddr.bit)	×
	CY,/saddr.bit	3	CY←CY ∧ ($\overline{\text{saddr.bit}}$)	×
	CY,sfr.bit	3	CY←CY ∧ sfr.bit	×
	CY,/sfr.bit	3	CY←CY ∧ $\overline{\text{sfr.bit}}$	×
	CY,A.bit	2	CY←CY ∧ A.bit	×
	CY,/A.bit	2	CY←CY ∧ $\overline{\text{A.bit}}$	×
	CY,X.bit	2	CY←CY ∧ X.bit	×
	CY,/X.bit	2	CY←CY ∧ $\overline{\text{X.bit}}$	×
	CY,PSWH.bit	2	CY←CY ∧ PSW_H.bit	×
	CY,/PSWH.bit	2	CY←CY ∧ $\overline{PSW_H.bit}$	×
	CY,PSWL.bit	2	CY←CY ∧ PSW_L.bit	×
	CY,/PSWL.bit	2	CY←CY ∧ $\overline{PSW_L.bit}$	×
OR1	CY,saddr.bit	3	CY←CY ∨ (saddr.bit)	×
	CY,/saddr.bit	3	CY←CY ∨ ($\overline{\text{saddr.bit}}$)	×
	CY,sfr.bit	3	CY←CY ∨ sfr.bit	×

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
OR1	CY,/sfr.bit	3	CY←CY ∨ $\overline{\text{sfr.bit}}$	×
	CY,A.bit	2	CY←CY ∨ A.bit	×
	CY,/A.bit	2	CY←CY ∨ $\overline{\text{A.bit}}$	×
	CY,X.bit	2	CY←CY ∨ X.bit	×
	CY,/X.bit	2	CY←CY ∨ $\overline{\text{X.bit}}$	×
	CY,PSWH.bit	2	CY←CY ∨ PSW_H.bit	×
	CY,/PSWH.bit	2	CY←CY ∨ $\overline{PSW_H.bit}$	×
	CY,PSWL.bit	2	CY←CY ∨ PSW_L.bit	×
	CY,/PSWL.bit	2	CY←CY ∨ $\overline{PSW_L.bit}$	×
XOR1	CY,saddr.bit	3	CY←CY ∀ (saddr.bit)	×
	CY,sfr.bit	3	CY←CY ∀ sfr.bit	×
	CY,A.bit	2	CY←CY ∀ A.bit	×
	CY,X.bit	2	CY←CY ∀ X.bit	×
	CY,PSWH.bit	2	CY←CY ∀ PSW_H.bit	×
	CY,PSWL.bit	2	CY←CY ∀ PSW_L.bit	×
SET1	saddr.bit	2	(saddr.bit)←1	
	sfr.bit	3	sfr.bit←1	
	A.bit	2	A.bit←1	
	X.bit	2	X.bit←1	
	PSWH.bit	2	PSW_H.bit←1	
	PSWL.bit	2	PSW_L.bit←1	× × × × ×
	CY	1	CY←1	1
CLR1	saddr.bit	2	(saddr.bit)←0	
	sfr.bit	3	sfr.bit←0	
	A.bit	2	A.bit←0	
	X.bit	2	X.bit←0	
	PSWH.bit	2	PSW_H.bit←0	
	PSWL.bit	2	PSW_L.bit←0	× × × × ×
	CY	1	CY←0	0
NOT1	saddr.bit	3	(saddr.bit)←$\overline{\text{(saddr.bit)}}$	
	sfr.bit	3	sfr.bit←$\overline{\text{sfr.bit}}$	
	A.bit	2	A.bit←$\overline{\text{A.bit}}$	
	X.bit	2	X.bit←$\overline{\text{X.bit}}$	
	PSWH.bit	2	PSW_H.bit←$\overline{PSW_H.bit}$	
	PSWL.bit	2	PSW_L.bit←$\overline{PSW_L.bit}$	× × × × ×
	CY	1	CY←$\overline{\text{CY}}$	×

(16) Call/return instructions: CALL, CALLF, CALLT, BRK, RET, RETB, RETI

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
CALL	!addr16	3	$(SP-1)\leftarrow(PC+3)_H$, $(SP-2)\leftarrow(PC+3)_L$, PC←addr16, SP←SP–2	
	rp1	2	$(SP-1)\leftarrow(PC+2)_H$, $(SP-2)\leftarrow(PC+2)_L$, $PC_H\leftarrow rp1_H$, $PC_L\leftarrow rp1_L$, SP←SP–2	
	[rp1]	2	$(SP-1)\leftarrow(PC+2)_H$, $(SP-2)\leftarrow(PC+2)_L$, $PC_H\leftarrow(rp1+1)$, $PC_L\leftarrow(rp1)$, SP←SP–2	
CALLF	!addr11	2	$(SP-1)\leftarrow(PC+2)_H$, $(SP-2)\leftarrow(PC+2)_L$, $PC_{15-11}\leftarrow$00001, $PC_{10-0}\leftarrow$addr11, SP←SP–2	
CALLT	[addr5]	1	$(SP-1)\leftarrow(PC+1)_H$, $(SP-2)\leftarrow(PC+1)_L$, $PC_H\leftarrow$(TPF, 000000001, addr5+1), $PC_L\leftarrow$(TPF, 000000001, addr5), SP←SP–2	
BRK		1	$(SP-1)\leftarrow PSW_H$, $(SP-2)\leftarrow PSW_L$, $(SP-3)\leftarrow(PC+1)_H$, $(SP-4)\leftarrow(PC+1)_L$, $PC_L\leftarrow$(003EH), $PC_H\leftarrow$(003FH), SP←SP–4, IE←0	
RET		1	$PC_L\leftarrow$(SP), $PC_H\leftarrow$(SP+1), SP←SP+2	
RETB		1	$PC_L\leftarrow$(SP), $PC_H\leftarrow$(SP+1), $PSW_L\leftarrow$(SP+2), $PSW_H\leftarrow$(SP+3), SP←SP+4	R R R R R
RETI		1	$PC_L\leftarrow$(SP), $PC_H\leftarrow$(SP+1), $PSW_L\leftarrow$(SP+2), $PSW_H\leftarrow$(SP+3), SP←SP+4, ISPRn←0**Note**	R R R R R

★ **Note** A RETI instruction resets (0) the bit corresponding to the interrupt request with the highest priority among bits (n = 0 to 3) set (1) in an ISPR register.

(17) Stack manipulation instructions: PUSH, PUSHU, POP, POPU, MOVW, INCW, DECW

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
PUSH	sfrp	3	(SP–1)←sfr$_H$, (SP–2)←sfr$_L$, SP←SP–2	
	post	2	{(SP–1)←post$_H$, (SP–2)←post$_L$, SP←SP–2} × n times	
	PSW	1	(SP–1)←PSW$_H$, (SP–2)←PSW$_L$, SP←SP–2	
PUSHU	post	2	{(UP–1)←post$_H$, (UP–2)←post$_L$, UP←UP–2} × n times	
POP	sfrp	3	sfr$_L$←(SP), sfr$_H$←(SP+1), SP←SP+2	
	post	2	{post$_L$←(SP), post$_H$←(SP+1), SP←SP+2} × n times	
	PSW	1	PSW$_L$←(SP), PSW$_H$←(SP+1), SP←SP+2	R R R R R
POPU	post	2	{post$_L$←(UP), post$_H$←(UP+1), UP←UP+2} × n times	
MOVW	SP,#word	4	SP←word	
	SP,AX	2	SP←AX	
	AX,SP	2	AX←SP	
INCW	SP	2	SP←SP+1	
DECW	SP	2	SP←SP–1	

Remark n indicates the number of registers specified in post.

(18) Special instructions: CHKL, CHKLA

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
CHKL	sfr	3	(Pin level) ⊻ (Signal level before output buffer)	× × P
CHKLA	sfr	3	A←{(Pin level) ⊻ (Signal level before output buffer)}	× × P

(19) Unconditional branch instruction: BR

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
BR	!addr16	3	PC←addr16	
	rp1	2	PC$_H$←rp1$_H$, PC$_L$←rp1$_L$	
	[rp1]	2	PC$_H$←(rp1+1), PC$_L$←(rp1)	
	$addr16	2	PC←PC+2+jdisp8	

(20) Conditional branch instructions: BC, BL, BNC, BNL, BZ, BE, BNZ, BNE, BV, BPE, BNV, BPO, BN, BP, BGT, BGE, BLT, BLE, BH, BNH, BT, BF, BTCLR, BFSET, DBNZ

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
BC	$addr16	2	PC←PC+2+jdisp8 if CY=1	
BL				
BNC	$addr16	2	PC←PC+2+jdisp8 if CY=0	
BNL				
BZ	$addr16	2	PC←PC+2+jdisp8 if Z=1	
BE				
BNZ	$addr16	2	PC←PC+2+jdisp8 if Z=0	
BNE				
BV	$addr16	2	PC←PC+2+jdisp8 if P/V=1	
BPE				
BNV	$addr16	2	PC←PC+2+jdisp8 if P/V=0	
BPO				
BN	$addr16	2	PC←PC+2+jdisp8 if S=1	
BP	$addr16	2	PC←PC+2+jdisp8 if S=0	
BGT	$addr16	3	PC←PC+3+jdisp8 if (P/V ⊻ S) ∨ Z=0	
BGE	$addr16	3	PC←PC+3+jdisp8 if P/V ⊻ S=0	
BLT	$addr16	3	PC←PC+3+jdisp8 if P/V ⊻ S=1	
BLE	$addr16	3	PC←PC+3+jdisp8 if (P/V ⊻ S) ∨ Z=1	
BH	$addr16	3	PC←PC+3+jdisp8 if Z ∨ CY=0	
BNH	$addr16	3	PC←PC+3+jdisp8 if Z ∨ CY=1	
BT	saddr.bit,$addr16	3	PC←PC+3+jdisp8 if (saddr.bit)=1	
	sfr.bit,$addr16	4	PC←PC+4+jdisp8 if sfr.bit=1	
	A.bit,$addr16	3	PC←PC+3+jdisp8 if A.bit=1	
	X.bit,$addr16	3	PC←PC+3+jdisp8 if X.bit=1	
	PSWH.bit,$addr16	3	PC←PC+3+jdisp8 if PSW_H.bit=1	
	PSWL.bit,$addr16	3	PC←PC+3+jdisp8 if PSW_L.bit=1	
BF	saddr.bit,$addr16	4	PC←PC+4+jdisp8 if (saddr.bit)=0	
	sfr.bit,$addr16	4	PC←PC+4+jdisp8 if sfr.bit=0	
	A.bit,$addr16	3	PC←PC+3+jdisp8 if A.bit=0	
	X.bit,$addr16	3	PC←PC+3+jdisp8 if X.bit=0	
	PSWH.bit,$addr16	3	PC←PC+3+jdisp8 if PSW_H.bit=0	
	PSWL.bit,$addr16	3	PC←PC+3+jdisp8 if PSW_L.bit=0	

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
BTCLR	saddr.bit,$addr16	4	PC←PC+4+jdisp8 if (saddr.bit)=1 then reset (saddr.bit)	
	sfr.bit,$addr16	4	PC←PC+4+jdisp8 if sfr.bit=1 then reset sfr.bit	
	A.bit,$addr16	3	PC←PC+3+jdisp8 if A.bit=1 then reset A.bit	
	X.bit,$addr16	3	PC←PC+3+jdisp8 if X.bit=1 then reset X.bit	
	PSWH.bit,$addr16	3	PC←PC+3+jdisp8 if PSW_H.bit=1 then reset PSW_H.bit	
	PSWL.bit,$addr16	3	PC←PC+3+jdisp8 if PSW_L.bit=1 then reset PSW_L.bit	× × × × ×
BFSET	saddr.bit,$addr16	4	PC←PC+4+jdisp8 if (saddr.bit)=0 then set (saddr.bit)	
	sfr.bit,$addr16	4	PC←PC+4+jdisp8 if sfr.bit=0 then set sfr.bit	
	A.bit,$addr16	3	PC←PC+3+jdisp8 if A.bit=0 then set A.bit	
	X.bit,$addr16	3	PC←PC+3+jdisp8 if X.bit=0 then set X.bit	
	PSWH.bit,$addr16	3	PC←PC+3+jdisp8 if PSW_H.bit=0 then set PSW_H.bit	
	PSWL.bit,$addr16	3	PC←PC+3+jdisp8 if PSW_L.bit=0 then set PSW_L.bit	× × × × ×
DBNZ	r2,$addr16	2	r2←r2–1, then PC←PC+2+jdisp8 if r2≠0	
	saddr,$addr16	3	(saddr)←(saddr)–1, then PC←PC+3+jdisp8 if (saddr)≠0	

(21) Context switching instructions: BRKCS, RETCS, RETCSB

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
BRKCS	RBn	2	RBS2–0←n, PC_H↔R5, PC_L↔R4, R7←PSW_H, R6←PSW_L, RSS←0, IE←0	
RETCS	!addr16	3	PC_H←R5, PC_L←R4, R5←addr16$_H$, R4←addr16$_L$, PSW_H←R7, PSW_L←R6	R R R R R
RETCSB	!addr16	4	PC_H←R5, PC_L←R4, R5←addr16$_H$, R4←addr16$_L$, PSW_H←R7, PSW_L←R6	R R R R R

(22) String instructions: MOVM, MOVBK, XCHM, XCHBK, CMPME, CMPBKE, CMPMNE, CMPBKNE, CMPMC, CMPBKC, CMPMNC, CMPBKNC

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
MOVM	[DE+],A	2	(DE+)←A, C←C–1, End if C=0	
	[DE–],A	2	(DE–)←A, C←C–1, End if C=0	
MOVBK	[DE+],[HL+]	2	(DE+)←(HL+), C←C–1, End if C=0	
	[DE–],[HL–]	2	(DE–)←(HL–), C←C–1, End if C=0	
XCHM	[DE+],A	2	(DE+)↔A, C←C–1, End if C=0	
	[DE–],A	2	(DE–)↔A, C←C–1, End if C=0	
XCHBK	[DE+],[HL+]	2	(DE+)↔(HL+), C←C–1, End if C=0	
	[DE–],[HL–]	2	(DE–)↔(HL–), C←C–1, End if C=0	
CMPME	[DE+],A	2	(DE+)–A, C←C–1, End if C=0 or Z=0	× × × V ×
	[DE–],A	2	(DE–)–A, C←C–1, End if C=0 or Z=0	
CMPBKE	[DE+],[HL+]	2	(DE+)–(HL+), C←C–1, End if C=0 or Z=0	× × × V ×
	[DE–],[HL–]	2	(DE–)–(HL–), C←C–1, End if C=0 or Z=0	
CMPMNE	[DE+],A	2	(DE+)–A, C←C–1, End if C=0 or Z=1	× × × V ×
	[DE–],A	2	(DE–)–A, C←C–1, End if C=0 or Z=1	
CMPBKNE	[DE+],[HL+]	2	(DE+)–(HL+), C←C–1, End if C=0 or Z=1	× × × V ×
	[DE–],[HL–]	2	(DE–)–(HL–), C←C–1, End if C=0 or Z=1	
CMPMC	[DE+],A	2	(DE+)–A, C←C–1, End if C=0 or CY=0	× × × V ×
	[DE–],A	2	(DE–)–A, C←C–1, End if C=0 or CY=0	
CMPBKC	[DE+],[HL+]	2	(DE+)–(HL+), C←C–1, End if C=0 or CY=0	× × × V ×
	[DE–],[HL–]	2	(DE–)–(HL–), C←C–1, End if C=0 or CY=0	
CMPMNC	[DE+],A	2	(DE+)–A, C←C–1, End if C=0 or CY=1	× × × V ×
	[DE–],A	2	(DE–)–A, C←C–1, End if C=0 or CY=1	
CMPBKNC	[DE+],[HL+]	2	(DE+)–(HL+), C←C–1, End if C=0 or CY=1	× × × V ×
	[DE–],[HL–]	2	(DE–)–(HL–), C←C–1, End if C=0 or CY=1	

(23) CPU control instructions: MOV, SWRS, SEL, NOP, EI, DI

Mnemonic	Operand	Byte	Operation	Flag
				S Z AC P/V CY
MOV	STBC,#byte	4	STBC←byte**Note**	
	WDM,#byte	4	WDM←byte**Note**	
SWRS		1	RSS←$\overline{\text{RSS}}$	
SEL	RBn	2	RBS2–0←n, RSS←0	
	RBn,ALT	2	RBS2–0←n, RSS←1	
NOP		1	No Operation	
EI		1	IE←1 (Enable Interrupt)	
DI		1	IE←0 (Disable Interrupt)	

Note An op-code trap interrupt occurs if an invalid op-code is specified in an STBC or WDM register manipulation instruction.

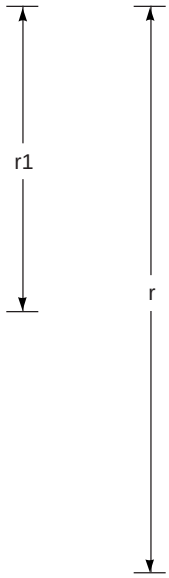
Trap operation: $(SP-1)\leftarrow PSW_H$, $(SP-2)\leftarrow PSW_L$, $(SP-3)\leftarrow (PC-4)_H$, $(SP-4)\leftarrow (PC-4)_L$, $PC_L\leftarrow$(003CH), $PC_H\leftarrow$(003DH), SP←SP–4, IE←0

5.2 Instruction Codes

5.2.1 Symbols of instruction codes

r,r1

R_3	R_2	R_1	R_0	reg
0	0	0	0	R0
0	0	0	1	R1
0	0	1	0	R2
0	0	1	1	R3
0	1	0	0	R4
0	1	0	1	R5
0	1	1	0	R6
0	1	1	1	R7
1	0	0	0	R8
1	0	0	1	R9
1	0	1	0	R10
1	0	1	1	R11
1	0	0	0	R12
1	1	0	1	R13
1	1	1	0	R14
1	1	1	1	R15



r2

C_0	reg
0	C
1	B

rp

P_2	P_1	P_0	reg-pair
0	0	0	RP0
0	0	1	RP1
0	1	0	RP2
0	1	1	RP3
1	0	0	RP4
1	0	1	RP5
1	1	0	RP6
1	1	1	RP7

rp1

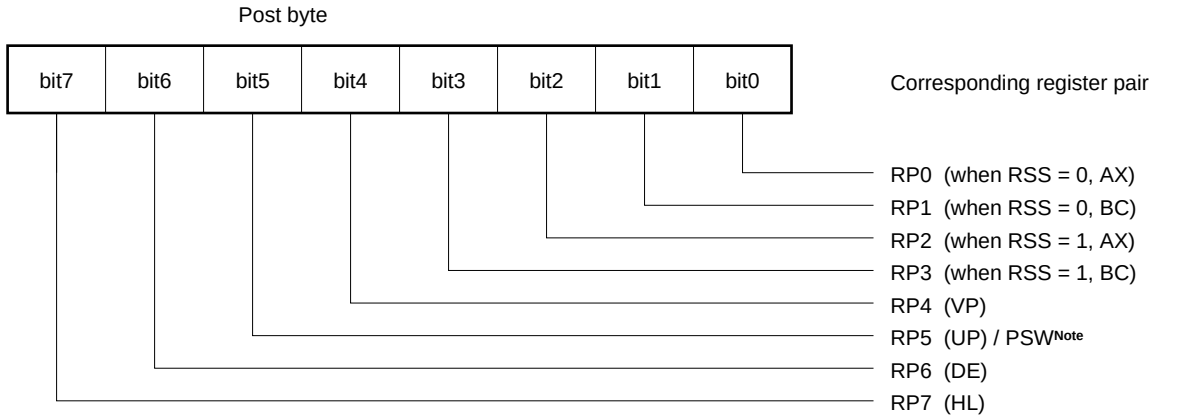
Q_2	Q_1	Q_0	reg-pair
0	0	0	RP0
0	0	1	RP4
0	1	0	RP1
0	1	1	RP5
1	0	0	RP2
1	0	1	RP6
1	1	0	RP3
1	1	1	RP7

rp2

S_1	S_0	reg-pair
0	0	VP
0	1	UP
1	0	DE
1	1	HL

B_n : Immediate data for the bit operand
N_n : Immediate data for the n operand
Data : 8-bit immediate data for the byte operand
Low/high byte : 16-bit immediate data for the word operand
Saddr-offset : Offset data for eight low-order bits of 16-bit address for the saddr operand
Sfr-offset : Offset data for eight low-order bits of 16-bit address of special function register (sfr)
Low/high offset : 8-/16-bit offset data for memory addressing in based/indexed mode
Low/high Addr. : 16-bit immediate data for the addr16 operand
jdisp : Signed 2's complement data (8 bits) indicating the relative address displacement from the starting address of the next instruction to the branch address
fa : 11 low-order bits of immediate data for the addr11 operand
ta : Five low-order bits of immediate data for 'addr5 × 1/2'
Post byte : 8-bit data that specifies the register pair which performs stack operations
A register pair is assigned to each bit and is specified according to the contents (0/1). (See **Figure 5-1**.)

Figure 5-1. 8-bit Data that Specifies the Register Pair which Performs Stack Operations



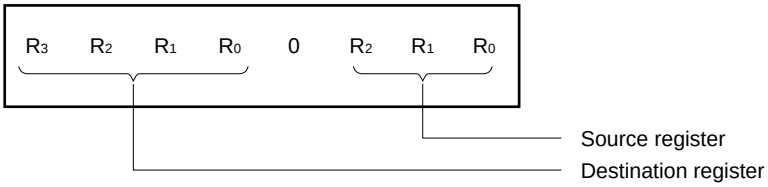
0	Save/restore operations are not performed with stack memory.
1	Save/restore operations are performed with stack memory.

Note RP5 (UP) is set for the PUSH and POP instructions and PSW is set for the PUSHU and POPU instructions.

Cautions 1. If registers are specified as both the source and destination in the operand field (such as 'r, r1' in the MOV instruction and 'saddr, saddr' in the ADD instruction), or saddr or saddrp is specified as both the source and destination in the operand field, the instruction code is as follows:

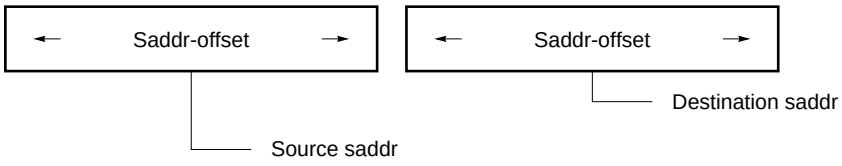
- **If registers are specified as both the source and destination, the destination specification code comes first and the source specification comes next. (This is the same as for register pairs.)**

  Example:



- **For saddr and saddrp, the first 1-byte data is the offset data that specifies a source and the next 1-byte data is the offset data that specifies a destination.**

  Example:



Cautions 2. When the special function register (SFR) mapped at addresses FF00H to FF1FH is specified for operand sfr or sfrp, short direct addressing is applied instead of SFR addressing, and the instruction code whose operand is saddr or saddrp is issued.

Example:

AND A, P5

Instruction code	10011100	00000101

AND A, PM5

Instruction code	00000001	10011100	00100101

In this example, the instruction code of instruction AND A, P5 for short direct addressing is shorter than that for SFR addressing.

5.2.2 Instruction codes in various memory addressing modes

mod and mem codes in the instruction code field are determined according to the contents of mem in the operand field as shown in Table 5-6.

Table 5-6. mod and mem Codes in the Instruction Code Field

mem		mod	1 0110	1 0111	0 0110	0 1010
			Register indirect mode	Based indexed mode	Based mode	Indexed mode
0	0	0	[DE+]**Note**	[DE+A]	[DE+byte]	word [DE]
0	0	1	[HL+]**Note**	[HL+A]	[SP+byte]	word [A]
0	1	0	[DE–]**Note**	[DE+B]	[HL+byte]	word [HL]
0	1	1	[HL–]**Note**	[HL+B]	[UP+byte]	word [B]
1	0	0	[DE]**Note**	[VP+DE]	[VP+byte]	–
1	0	1	[HL]**Note**	[VP+HL]	–	–
1	1	0	[VP]	–	–	–
1	1	1	[UP]	–	–	–

Note If these codes are specified in mem for an MOV instruction, a dedicated 1-byte instruction is generated.

Remark If the based mode or indexed mode is specified in mem, 8- or 16-bit offset data for the byte or word operand is added to the third or the subsequent bytes.

5.2.3 List of instruction codes

(1) 8-bit data transfer instructions: MOV, XCH

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
MOV	r1,#byte	1 0 1 1 1 R_2 R_1 R_0	← Data →	
	saddr,#byte	0 0 1 1 1 0 1 0	← Saddr-offset →	← Data →
	sfr,#byte	0 0 1 0 1 0 1 1	← Sfr-offset →	← Data →
	r,r1	0 0 1 0 0 1 0 0	R_3 R_2 R_1 R_0 0 R_2 R_1 R_0	
	A,r1	1 1 0 1 0 R_2 R_1 R_0		
	A,saddr	0 0 1 0 0 0 0 0	← Saddr-offset →	
	saddr,A	0 0 1 0 0 0 1 0	← Saddr-offset →	
	saddr,saddr	0 0 1 1 1 0 0 0	← Saddr-offset →	← Saddr-offset →
	A,sfr	0 0 0 1 0 0 0 0	← Sfr-offset →	
	sfr,A	0 0 0 1 0 0 1 0	← Sfr-offset →	
	A,mem **Note**	0 1 0 1 1 mem		
	A,mem	0 0 0 mod	0 mem 0 0 0 0	← Low offset →
	A,mem	← High offset →		
	mem,A **Note**	0 1 0 1 0 mem		
	mem,A	0 0 0 mod	1 mem 0 0 0 0	← Low offset →
	mem,A	← High offset →		
	A,[saddrp]	0 0 0 1 1 0 0 0	← Saddr-offset →	
	[saddrp],A	0 0 0 1 1 0 0 1	← Saddr-offset →	
	A,!addr16	0 0 0 0 1 0 0 1	1 1 1 1 0 0 0 0	← Low Addr. →
	A,!addr16	← High Addr. →		
	!addr16,A	0 0 0 0 1 0 0 1	1 1 1 1 0 0 0 1	← Low Addr. →
	!addr16,A	← High Addr. →		
	PSWL,#byte	0 0 1 0 1 0 1 1	1 1 1 1 1 1 1 0	← Data →
	PSWH,#byte	0 0 1 0 1 0 1 1	1 1 1 1 1 1 1 1	← Data →
	PSWL,A	0 0 0 1 0 0 1 0	1 1 1 1 1 1 1 0	
	PSWH,A	0 0 0 1 0 0 1 0	1 1 1 1 1 1 1 1	
	A,PSWL	0 0 0 1 0 0 0 0	1 1 1 1 1 1 1 0	
	A,PSWH	0 0 0 1 0 0 0 0	1 1 1 1 1 1 1 1	

Note This one-bite code is generated by coding [DE], [HL], [DE+], [DE–], [HL+], or [HL–] in mem.

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
XCH	A,r1	1 1 0 1 1 R_2 R_1 R_0		
	r,r1	0 0 1 0 0 1 0 1	R_3 R_2 R_1 R_0 0 R_2 R_1 R_0	
	A,mem	0 0 0 mod	0 mem 0 1 0 0	← Low offset →
		← High offset →		
	A,saddr	0 0 1 0 0 0 0 1	← Saddr-offset →	
	A,sfr	0 0 0 0 0 0 0 1	0 0 1 0 0 0 0 1	← Sfr-offset →
	A,[saddrp]	0 0 1 0 0 0 1 1	← Saddr-offset →	
	saddr,saddr	0 0 1 1 1 0 0 1	← Saddr-offset →	← Saddr-offset →

(2) 16-bit data transfer instructions: MOVW, XCHW

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
MOVW	rp1,#word	0 1 1 0 0 Q_2 Q_1 Q_0	← Low byte →	← High byte →
	saddrp,#word	0 0 0 0 1 1 0 0	← Saddr-offset →	← Low byte →
		← High byte →		
	sfrp,#word	0 0 0 0 1 0 1 1	← Sfr-offset →	← Low byte →
		← High byte →		
	rp,rp1	0 0 1 0 0 1 0 0	P_2 P_1 P_0 0 1 Q_2 Q_1 Q_0	
	AX,saddrp	0 0 0 1 1 1 0 0	← Saddr-offset →	
	saddrp,AX	0 0 0 1 1 0 1 0	← Saddr-offset →	
	saddrp,saddrp	0 0 1 1 1 1 0 0	← Saddr-offset →	← Saddr-offset →
	AX,sfrp	0 0 0 1 0 0 0 1	← Sfr-offset →	
	sfrp,AX	0 0 0 1 0 0 1 1	← Sfr-offset →	
	rp1,!addr16	0 0 0 0 1 0 0 1	1 0 0 0 0 Q_2 Q_1 Q_0	← Low Addr. →
		← High-Addr. →		
	!addr16,rp1	0 0 0 0 1 0 0 1	1 0 0 1 0 Q_2 Q_1 Q_0	← Low Addr. →
		← High-Addr. →		
	AX,mem	0 0 0 mod	0 mem 0 0 0 1	← Low offset →
		← High offset →		
	mem,AX	0 0 0 mod	1 mem 0 0 0 1	← Low offset →
		← High offset →		

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
XCHW	AX,saddrp	0 0 0 1 1 0 1 1	← Saddr-offset →	
	AX,sfrp	0 0 0 0 0 0 0 1	0 0 0 1 1 0 1 1	← Sfr-offset →
	saddrp,saddrp	0 0 1 0 1 0 1 0	← Saddr-offset →	← Saddr-offset →
	rp,rp1	0 0 1 0 0 1 0 1	P_2 P_1 P_0 0 1 Q_2 Q_1 Q_0	
	AX,mem	0 0 0 mod	0 mem 0 1 0 1	← Low offset →
		← High offset →		

(3) 8-bit arithmetic/logical instructions: ADD, ADDC, SUB, SUBC, AND, OR, XOR, CMP

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
ADD	A,#byte	1 0 1 0 1 0 0 0	← Data →	
	saddr,#byte	0 1 1 0 1 0 0 0	← Saddr-offset →	← Data →
	sfr,#byte	0 0 0 0 0 0 0 1	0 1 1 0 1 0 0 0	← Sfr-offset →
		← Data →		
	r,r1	1 0 0 0 1 0 0 0	R_3 R_2 R_1 R_0 0 R_2 R_1 R_0	
	A,saddr	1 0 0 1 1 0 0 0	← Saddr-offset →	
	A,sfr	0 0 0 0 0 0 0 1	1 0 0 1 1 0 0 0	← Sfr-offset →
	saddr,saddr	0 1 1 1 1 0 0 0	← Saddr-offset →	← Saddr-offset →
	A,mem	0 0 0 mod	0 mem 1 0 0 0	← Low offset →
		← High offset →		
	mem,A	0 0 0 mod	1 mem 1 0 0 0	← Low offset →
		← High offset →		
ADDC	A,#byte	1 0 1 0 1 0 0 1	← Data →	
	saddr,#byte	0 1 1 0 1 0 0 1	← Saddr-offset →	← Data →
	sfr,#byte	0 0 0 0 0 0 0 1	0 1 1 0 1 0 0 1	← Sfr-offset →
		← Data →		
	r,r1	1 0 0 0 1 0 0 1	R_3 R_2 R_1 R_0 0 R_2 R_1 R_0	
	A,saddr	1 0 0 1 1 0 0 1	← Saddr-offset →	
	A,sfr	0 0 0 0 0 0 0 1	1 0 0 1 1 0 0 1	← Sfr-offset →
	saddr, saddr	0 1 1 1 1 0 0 1	← Saddr-offset →	← Saddr-offset →
	A,mem	0 0 0 mod	0 mem 1 0 0 1	← Low offset →
		← High offset →		
	mem,A	0 0 0 mod	1 mem 1 0 0 1	← Low offset →
		← High offset →		

Mnemonic	Operand	Instruction code			
		B1	B2	B3	
		B4	B5		
SUB	A,#byte	1 0 1 0 1 0 1 0	← Data →		
	saddr,#byte	0 1 1 0 1 0 1 0	← Saddr-offset →	← Data →	
	sfr,#byte	0 0 0 0 0 0 0 1	0 1 1 0 1 0 1 0	← Sfr-offset →	
		← Data →			
	r,r1	1 0 0 0 1 0 1 0	R_3 R_2 R_1 R_0 0 R_2 R_1 R_0		
	A,saddr	1 0 0 1 1 0 1 0	← Saddr-offset →		
	A,sfr	0 0 0 0 0 0 0 1	1 0 0 1 1 0 1 0	← Sfr-offset →	
	saddr,saddr	0 1 1 1 1 0 1 0	← Saddr-offset →	← Saddr-offset →	
	A,mem	0 0 0 mod	0 mem 1 0 1 0	← Low offset →	
		← High offset →			
	mem,A	0 0 0 mod	1 mem 1 0 1 0	← Low offset →	
		← High offset →			
SUBC	A,#byte	1 0 1 0 1 0 1 1	← Data →		
	saddr,#byte	0 1 1 0 1 0 1 1	← Saddr-offset →	← Data →	
	sfr,#byte	0 0 0 0 0 0 0 1	0 1 1 0 1 0 1 1	← Sfr-offset →	
		← Data →			
	r,r1	1 0 0 0 1 0 1 1	R_3 R_2 R_1 R_0 0 R_2 R_1 R_0		
	A,saddr	1 0 0 1 1 0 1 1	← Saddr-offset →		
	A,sfr	0 0 0 0 0 0 0 1	1 0 0 1 1 0 1 1	← Sfr-offset →	
	saddr,saddr	0 1 1 1 1 0 1 1	← Saddr-offset →	← Saddr-offset →	
	A,mem	0 0 0 mod	0 mem 1 0 1 1	← Low offset →	
		← High offset →			
	mem,A	0 0 0 mod	1 mem 1 0 1 1	← Low offset →	
		← High offset →			
AND	A,#byte	1 0 1 0 1 1 0 0	← Data →		
	saddr,#byte	0 1 1 0 1 1 0 0	← Saddr-offset →	← Data →	
	sfr,#byte	0 0 0 0 0 0 0 1	0 1 1 0 1 1 0 0	← Sfr-offset →	
		← Data →			
	r,r1	1 0 0 0 1 1 0 0	R_3 R_2 R_1 R_0 0 R_2 R_1 R_0		
	A,saddr	1 0 0 1 1 1 0 0	← Saddr-offset →		
	A,sfr	0 0 0 0 0 0 0 1	1 0 0 1 1 1 0 0	← Sfr-offset →	
	saddr,saddr	0 1 1 1 1 1 0 0	← Saddr-offset →	← Saddr-offset →	

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
AND	A,mem	0 0 0 mod	0 mem 1 1 0 0	← Low offset →
		← High offset →		
	mem,A	0 0 0 mod	1 mem 1 1 0 0	← Low offset →
		← High offset →		
OR	A,#byte	1 0 1 0 1 1 1 0	← Data →	
	saddr,#byte	0 1 1 0 1 1 1 0	← Saddr-offset →	← Data →
	sfr,#byte	0 0 0 0 0 0 0 1	0 1 1 0 1 1 1 0	← Sfr-offset →
		← Data →		
	r,r1	1 0 0 0 1 1 1 0	R_3 R_2 R_1 R_0 0 R_2 R_1 R_0	
	A,saddr	1 0 0 1 1 1 1 0	← saddr-offset →	
	A,sfr	0 0 0 0 0 0 0 1	1 0 0 1 1 1 1 0	← Sfr-offset →
	saddr,saddr	0 1 1 1 1 1 1 0	← Saddr-offset →	← Saddr-offset →
	A,mem	0 0 0 mod	0 mem 1 1 1 0	← Low offset →
		← High offset →		
	mem,A	0 0 0 mod	1 mem 1 1 1 0	← Low offset →
		← High offset →		
XOR	A,#byte	1 0 1 0 1 1 0 1	← Data →	
	saddr,#byte	0 1 1 0 1 1 0 1	← Saddr-offset →	← Data →
	sfr,#byte	0 0 0 0 0 0 0 1	0 1 1 0 1 1 0 1	← Sfr-offset →
		← Data →		
	r,r1	1 0 0 0 1 1 0 1	R_3 R_2 R_1 R_0 0 R_2 R_1 R_0	
	A,saddr	1 0 0 1 1 1 0 1	← Saddr-offset →	
	A,sfr	0 0 0 0 0 0 0 1	1 0 0 1 1 1 0 1	← Sfr-offset →
	saddr,saddr	0 1 1 1 1 1 0 1	← Saddr-offset →	← Saddr-offset →
	A,mem	0 0 0 mod	0 mem 1 1 0 1	← Low offset →
		← High offset →		
	mem,A	0 0 0 mod	1 mem 1 1 0 1	← Low offset →
		← High offset →		
CMP	A,#byte	1 0 1 0 1 1 1 1	← Data →	
	saddr,#byte	0 1 1 0 1 1 1 1	← Saddr-offset →	← Data →
	sfr,#byte	0 0 0 0 0 0 0 1	0 1 1 0 1 1 1 1	← Sfr-offset →
		← Data →		
	r,r1	1 0 0 0 1 1 1 1	R_3 R_2 R_1 R_0 0 R_2 R_1 R_0	

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
CMP	A,saddr	1 0 0 1 1 1 1 1	← Saddr-offset →	
	A,sfr	0 0 0 0 0 0 0 1	1 0 0 1 1 1 1 1	← Sfr-offset →
	saddr,saddr	0 1 1 1 1 1 1 1	← Saddr-offset →	← Saddr-offset →
	A,mem	0 0 0 mod	0 mem 1 1 1 1	← Low offset →
		← High offset →		
	mem,A	0 0 0 mod	1 mem 1 1 1 1	← Low offset →
		← High offset →		

(4) 16-bit arithmetic/logical instructions: ADDW, SUBW, CMPW

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
ADDW	AX,#word	0 0 1 0 1 1 0 1	← Low byte →	← High byte →
	saddrp,#word	0 0 0 0 1 1 0 1	← Saddr-offset →	← Low byte →
		← High byte →		
	sfrp,#word	0 0 0 0 0 0 0 1	0 0 0 0 1 1 0 1	← Sfr-offset →
		← Low byte →	← High byte →	
	rp,rp1	1 0 0 0 1 0 0 0	P_2 P_1 P_0 0 1 Q_2 Q_1 Q_0	
	AX,saddrp	0 0 0 1 1 1 0 1	← Saddr-offset →	
	AX,sfrp	0 0 0 0 0 0 0 1	0 0 0 1 1 1 0 1	← Sfr-offset →
	saddrp,saddrp	0 0 1 1 1 1 0 1	← Saddr-offset →	← Saddr-offset →
SUBW	AX,#word	0 0 1 0 1 1 1 0	← Low byte →	← High byte →
	saddrp,#word	0 0 0 0 1 1 1 0	← Saddr-offset →	← Low byte →
		← High byte →		
	sfrp,#word	0 0 0 0 0 0 0 1	0 0 0 0 1 1 1 0	← Sfr-offset →
		← Low byte →	← High byte →	
	rp,rp1	1 0 0 0 1 0 1 0	P_2 P_1 P_0 0 1 Q_2 Q_1 Q_0	
	AX,saddrp	0 0 0 1 1 1 1 0	← Saddr-offset →	
	AX,sfrp	0 0 0 0 0 0 0 1	0 0 0 1 1 1 1 0	← Sfr-offset →
	saddrp,saddrp	0 0 1 1 1 1 1 0	← Saddr-offset →	← Saddr-offset →
CMPW	AX,#word	0 0 1 0 1 1 1 1	← Low byte →	← High byte →
	saddrp,#word	0 0 0 0 1 1 1 1	← Saddr-offset →	← Low byte →
		← High byte →		

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
CMPW	sfrp,#word	0 0 0 0 0 0 0 1	0 0 0 0 1 1 1 1	← Sfr-offset →
		← Low byte →	← High byte →	
	rp,rp1	1 0 0 0 1 1 1 1	P_2 P_1 P_0 0 1 Q_2 Q_1 Q_0	
	AX,saddrp	0 0 0 1 1 1 1 1	← Saddr-offset →	
	AX,sfrp	0 0 0 0 0 0 0 1	0 0 0 1 1 1 1 1	← Sfr-offset →
	saddrp,saddrp	0 0 1 1 1 1 1 1	← Saddr-offset →	← Saddr-offset →

(5) Multiply/divide instructions: MULU, DIVUW, MULUW, DIVUX

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
MULU	r1	0 0 0 0 0 1 0 1	0 0 0 0 1 R_2 R_1 R_0	
DIVUW	r1	0 0 0 0 0 1 0 1	0 0 0 1 1 R_2 R_1 R_0	
MULUW	rp1	0 0 0 0 0 1 0 1	0 0 1 0 1 Q_2 Q_1 Q_0	
DIVUX	rp1	0 0 0 0 0 1 0 1	1 1 1 0 1 Q_2 Q_1 Q_0	

(6) Signed multiply instruction: MULW

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
MULW	rp1	0 0 0 0 0 1 0 1	0 0 1 1 1 Q_2 Q_1 Q_0	

(7) Sum-of-products instruction: MACW

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
MACW	n	0 0 0 0 0 1 1 1	1 0 0 0 0 1 0 1	$N_7N_6N_5N_4N_3N_2N_1N_0$

(8) Sum-of-products instruction with saturation function: MACSW

Mnemonic	Operand	Instruction code		
		B1	B2	B3
MACSW	n	0 0 0 0 0 1 1 1	0 0 0 0 0 1 0 1	**Note**

Note Number of products to be added (n times)

Remark The μPD78352A Subseries does not provide this sum-of-products instruction with saturation function.

(9) Correlation instruction: SACW

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	B6
SACW	[DE+],[HL+]	0 0 0 0 1 0 0 1	1 0 1 1 0 0 0 0	0 1 0 0 0 0 0 1
		0 1 0 0 0 1 1 0		

Remark The μPD78352A subseries does not provide the correlation instruction.

(10) Table shift instruction: MOVTBLW

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
MOVTBLW	!addr16,n	0 0 0 0 1 0 0 1	1 0 1 0 0 0 0 0	← Low byte →
		N_7 N_6 N_5 N_4 N_3 N_2 N_1 N_0		

(11) Increment/decrement instructions: INC, DEC, INCW, DECW

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
INC	r1	1 1 0 0 0 R_2 R_1 R_0		
	saddr	0 0 1 0 0 1 1 0	← Saddr-offset →	
DEC	r1	1 1 0 0 1 R_2 R_1 R_0		
	saddr	0 0 1 0 0 1 1 1	← Saddr-offset →	
INCW	rp2	0 1 0 0 0 1 S_1 S_0		
	saddrp	0 0 0 0 0 1 1 1	1 1 1 0 1 0 0 0	← Saddr-offset →
DECW	rp2	0 1 0 0 1 1 S_1 S_0		
	saddrp	0 0 0 0 0 1 1 1	1 1 1 0 1 0 0 1	← Saddr-offset →

(12) Shift/rotate instructions: ROR, ROL, RORC, ROLC, SHR, SHL, SHRW, SHLW, ROR4, ROL4

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
ROR	r1,n	0 0 1 1 0 0 0 0	0 1 N_2 N_1 N_0 R_2 R_1 R_0	
ROL	r1,n	0 0 1 1 0 0 0 1	0 1 N_2 N_1 N_0 R_2 R_1 R_0	
RORC	r1,n	0 0 1 1 0 0 0 0	0 0 N_2 N_1 N_0 R_2 R_1 R_0	
ROLC	r1,n	0 0 1 1 0 0 0 1	0 0 N_2 N_1 N_0 R_2 R_1 R_0	
SHR	r1,n	0 0 1 1 0 0 0 0	1 0 N_2 N_1 N_0 R_2 R_1 R_0	
SHL	r1,n	0 0 1 1 0 0 0 1	1 0 N_2 N_1 N_0 R_2 R_1 R_0	
SHRW	rp1,n	0 0 1 1 0 0 0 0	1 1 N_2 N_1 N_0 Q_2 Q_1 Q_0	
SHLW	rp1,n	0 0 1 1 0 0 0 1	1 1 N_2 N_1 N_0 Q_2 Q_1 Q_0	
ROR4	[rp1]	0 0 0 0 0 1 0 1	1 0 0 0 1 Q_2 Q_1 Q_0	
ROL4	[rp1]	0 0 0 0 0 1 0 1	1 0 0 1 1 Q_2 Q_1 Q_0	

(13) BCD correction instructions: ADJBA, ADJBS

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
ADJBA		0 0 0 0 0 1 0 1	1 1 1 1 1 1 1 0	
ADJBS		0 0 0 0 0 1 0 1	1 1 1 1 1 1 1 1	

(14) Data conversion instruction: CVTBW

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
CVTBW		0 0 0 0 0 1 0 0		

(15) Bit manipulation instructions: MOV1, AND1, OR1, XOR1, SET1, CLR1, NOT1

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
MOV1	CY,saddr.bit	0 0 0 0 1 0 0 0	0 0 0 0 0 B_2 B_1 B_0	← Saddr-offset →
	CY,sfr.bit	0 0 0 0 1 0 0 0	0 0 0 0 1 B_2 B_1 B_0	← Sfr-offset →
	CY,A.bit	0 0 0 0 0 0 1 1	0 0 0 0 1 B_2 B_1 B_0	

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
MOV1	CY,X.bit	0 0 0 0 0 0 1 1	0 0 0 0 0 B_2 B_1 B_0	
	CY,PSWH.bit	0 0 0 0 0 0 1 0	0 0 0 0 1 B_2 B_1 B_0	
	CY,PSWL.bit	0 0 0 0 0 0 1 0	0 0 0 0 0 B_2 B_1 B_0	
	saddr.bit,CY	0 0 0 0 1 0 0 0	0 0 0 1 0 B_2 B_1 B_0	← Saddr-offset →
	sfr.bit,CY	0 0 0 0 1 0 0 0	0 0 0 1 1 B_2 B_1 B_0	← Sfr-offset →
	A.bit,CY	0 0 0 0 0 0 1 1	0 0 0 1 1 B_2 B_1 B_0	
	X.bit,CY	0 0 0 0 0 0 1 1	0 0 0 1 0 B_2 B_1 B_0	
	PSWH.bit,CY	0 0 0 0 0 0 1 0	0 0 0 1 1 B_2 B_1 B_0	
	PSWL.bit,CY	0 0 0 0 0 0 1 0	0 0 0 1 0 B_2 B_1 B_0	
AND1	CY,saddr.bit	0 0 0 0 1 0 0 0	0 0 1 0 0 B_2 B_1 B_0	← Saddr-offset →
	CY,/saddr.bit	0 0 0 0 1 0 0 0	0 0 1 1 0 B_2 B_1 B_0	← Saddr-offset →
	CY,sfr.bit	0 0 0 0 1 0 0 0	0 0 1 0 1 B_2 B_1 B_0	← Sfr-offset →
	CY,/sfr.bit	0 0 0 0 1 0 0 0	0 0 1 1 1 B_2 B_1 B_0	← Sfr-offset →
	CY,A.bit	0 0 0 0 0 0 1 1	0 0 1 0 1 B_2 B_1 B_0	
	CY,/A.bit	0 0 0 0 0 0 1 1	0 0 1 1 1 B_2 B_1 B_0	
	CY,X.bit	0 0 0 0 0 0 1 1	0 0 1 0 0 B_2 B_1 B_0	
	CY,/X.bit	0 0 0 0 0 0 1 1	0 0 1 1 0 B_2 B_1 B_0	
	CY,PSWH.bit	0 0 0 0 0 0 1 0	0 0 1 0 1 B_2 B_1 B_0	
	CY,/PSWH.bit	0 0 0 0 0 0 1 0	0 0 1 1 1 B_2 B_1 B_0	
	CY,PSWL.bit	0 0 0 0 0 0 1 0	0 0 1 0 0 B_2 B_1 B_0	
	CY,/PSWL.bit	0 0 0 0 0 0 1 0	0 0 1 1 0 B_2 B_1 B_0	
OR1	CY,saddr.bit	0 0 0 0 1 0 0 0	0 1 0 0 0 B_2 B_1 B_0	← Saddr-offset →
	CY,/saddr.bit	0 0 0 0 1 0 0 0	0 1 0 1 0 B_2 B_1 B_0	← Saddr-offset →
	CY,sfr.bit	0 0 0 0 1 0 0 0	0 1 0 0 1 B_2 B_1 B_0	← Sfr-offset →
	CY,/sfr.bit	0 0 0 0 1 0 0 0	0 1 0 1 1 B_2 B_1 B_0	← Sfr-offset →
	CY,A.bit	0 0 0 0 0 0 1 1	0 1 0 0 1 B_2 B_1 B_0	
	CY,/A.bit	0 0 0 0 0 0 1 1	0 1 0 1 1 B_2 B_1 B_0	
	CY,X.bit	0 0 0 0 0 0 1 1	0 1 0 0 0 B_2 B_1 B_0	
	CY,/X.bit	0 0 0 0 0 0 1 1	0 1 0 1 0 B_2 B_1 B_0	
	CY,PSWH.bit	0 0 0 0 0 0 1 0	0 1 0 0 1 B_2 B_1 B_0	
	CY,/PSWH.bit	0 0 0 0 0 0 1 0	0 1 0 1 1 B_2 B_1 B_0	
	CY,PSWL.bit	0 0 0 0 0 0 1 0	0 1 0 0 0 B_2 B_1 B_0	
	CY,/PSWL.bit	0 0 0 0 0 0 1 0	0 1 0 1 0 B_2 B_1 B_0	

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
XOR1	CY,saddr.bit	0 0 0 0 1 0 0 0	0 1 1 0 0 B_2 B_1 B_0	← Saddr-offset →
	CY,sfr.bit	0 0 0 0 1 0 0 0	0 1 1 0 1 B_2 B_1 B_0	← Sfr-offset →
	CY,A.bit	0 0 0 0 0 0 1 1	0 1 1 0 1 B_2 B_1 B_0	
	CY,X.bit	0 0 0 0 0 0 1 1	0 1 1 0 0 B_2 B_1 B_0	
	CY,PSWH.bit	0 0 0 0 0 0 1 0	0 1 1 0 1 B_2 B_1 B_0	
	CY,PSWL.bit	0 0 0 0 0 0 1 0	0 1 1 0 0 B_2 B_1 B_0	
SET1	saddr.bit	1 0 1 1 0 B_2 B_1 B_0	← Saddr-offset →	
	sfr.bit	0 0 0 0 1 0 0 0	1 0 0 0 1 B_2 B_1 B_0	← Sfr-offset →
	A.bit	0 0 0 0 0 0 1 1	1 0 0 0 1 B_2 B_1 B_0	
	X.bit	0 0 0 0 0 0 1 1	1 0 0 0 0 B_2 B_1 B_0	
	PSWH.bit	0 0 0 0 0 0 1 0	1 0 0 0 1 B_2 B_1 B_0	
	PSWL.bit	0 0 0 0 0 0 1 0	1 0 0 0 0 B_2 B_1 B_0	
	CY	0 1 0 0 0 0 0 1		
CLR1	saddr.bit	1 0 1 0 0 B_2 B_1 B_0	← Saddr-offset →	
	sfr.bit	0 0 0 0 1 0 0 0	1 0 0 1 1 B_2 B_1 B_0	← Sfr-offset →
	A.bit	0 0 0 0 0 0 1 1	1 0 0 1 1 B_2 B_1 B_0	
	X.bit	0 0 0 0 0 0 1 1	1 0 0 1 0 B_2 B_1 B_0	
	PSWH.bit	0 0 0 0 0 0 1 0	1 0 0 1 1 B_2 B_1 B_0	
	PSWL.bit	0 0 0 0 0 0 1 0	1 0 0 1 0 B_2 B_1 B_0	
	CY	0 1 0 0 0 0 0 0		
NOT1	saddr.bit	0 0 0 0 1 0 0 0	0 1 1 1 0 B_2 B_1 B_0	← Saddr-offset →
	sfr.bit	0 0 0 0 1 0 0 0	0 1 1 1 1 B_2 B_1 B_0	← Sfr-offset →
	A.bit	0 0 0 0 0 0 1 1	0 1 1 1 1 B_2 B_1 B_0	
	X.bit	0 0 0 0 0 0 1 1	0 1 1 1 0 B_2 B_1 B_0	
	PSWH.bit	0 0 0 0 0 0 1 0	0 1 1 1 1 B_2 B_1 B_0	
	PSWL.bit	0 0 0 0 0 0 1 0	0 1 1 1 0 B_2 B_1 B_0	
	CY	0 1 0 0 0 0 1 0		

(16) Call/return instructions: CALL, CALLF, CALLT, BRK, RET, RETB, RETI

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
CALL	!addr16	0 0 1 0 1 0 0 0	← Low Addr. →	← High-Addr. →
	rp1	0 0 0 0 0 1 0 1	0 1 0 1 1 Q_2 Q_1 Q_0	
	[rp1]	0 0 0 0 0 1 0 1	0 1 1 1 1 Q_2 Q_1 Q_0	
CALLF	!addr11	1 0 0 1 0 ←	fa →	
CALLT	[addr5]	1 1 1 ← ta →		
BRK		0 1 0 1 1 1 1 0		
RET		0 1 0 1 0 1 1 0		
RETB		0 1 0 1 1 1 1 1		
RETI		0 1 0 1 0 1 1 1		

(17) Stack manipulation instructions: PUSH, PUSHU, POP, POPU, MOVW, INCW, DECW

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
PUSH	sfrp	0 0 0 0 0 1 1 1	1 1 0 1 1 0 0 1	← Sfr-offset →
	post	0 0 1 1 0 1 0 1	← Post byte →	
	PSW	0 1 0 0 1 0 0 1		
PUSHU	post	0 0 1 1 0 1 1 1	← Post byte →	
POP	sfrp	0 0 0 0 0 1 1 1	1 1 0 1 1 0 0 0	← Sfr-offset →
	post	0 0 1 1 0 1 0 0	← Post byte →	
	PSW	0 1 0 0 1 0 0 0		
POPU	post	0 0 1 1 0 1 1 0	← Post byte →	
MOVW	SP,#word	0 0 0 0 1 0 1 1	1 1 1 1 1 1 0 0	← Low-byte →
		← High byte →		
	SP,AX	0 0 0 1 0 0 1 1	1 1 1 1 1 1 0 0	
	AX,SP	0 0 0 1 0 0 0 1	1 1 1 1 1 1 0 0	
INCW	SP	0 0 0 0 0 1 0 1	1 1 0 0 1 0 0 0	
DECW	SP	0 0 0 0 0 1 0 1	1 1 0 0 1 0 0 1	

(18) Special instructions: CHKL, CHKLA

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
CHKL	sfr	0 0 0 0 0 1 1 1	1 1 0 0 1 0 0 0	← Sfr-offset →
CHKLA	sfr	0 0 0 0 0 1 1 1	1 1 0 0 1 0 0 1	← Sfr-offset →

(19) Unconditional branch instruction: BR

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
BR	!addr16	0 0 1 0 1 1 0 0	← Low Addr. →	← High Addr. →
	rp1	0 0 0 0 0 1 0 1	0 1 0 0 1 Q_2 Q_1 Q_0	
	[rp1]	0 0 0 0 0 1 0 1	0 1 1 0 1 Q_2 Q_1 Q_0	
	$addr16	0 0 0 1 0 1 0 0	← jdisp →	

(20) Conditional branch instructions: BC, BL, BNC, BNL, BZ, BE, BNZ, BNE, BV, BPE, BNV, BPO, BN, BP, BGT, BGE, BLT, BLE, BH, BNH, BT, BF, BTCLR, BFSET, DBNZ

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
BR	$addr16	1 0 0 0 0 0 1 1	← jdisp →	
BL				
BNC	$addr16	1 0 0 0 0 0 1 0	← jdisp →	
BNL				
BZ	$addr16	1 0 0 0 0 0 0 1	← jdisp →	
BE				
BNZ	$addr16	1 0 0 0 0 0 0 0	← jdisp →	
BNE				
BV	$addr16	1 0 0 0 0 1 0 1	← jdisp →	
BPE				
BNV	$addr16	1 0 0 0 0 1 0 0	← jdisp →	
BPO				
BN	$addr16	1 0 0 0 0 1 1 1	← jdisp →	
BP	$addr16	1 0 0 0 0 1 1 0	← jdisp →	

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
BGT	$addr16	0 0 0 0 0 1 1 1	1 1 1 1 1 0 1 1	← jdisp →
BGE	$addr16	0 0 0 0 0 1 1 1	1 1 1 1 1 0 0 1	← jdisp →
BLT	$addr16	0 0 0 0 0 1 1 1	1 1 1 1 1 0 0 0	← jdisp →
BLE	$addr16	0 0 0 0 0 1 1 1	1 1 1 1 1 0 1 0	← jdisp →
BH	$addr16	0 0 0 0 0 1 1 1	1 1 1 1 1 1 0 1	← jdisp →
BNH	$addr16	0 0 0 0 0 1 1 1	1 1 1 1 1 1 0 0	← jdisp →
BT	saddr.bit,$addr16	0 1 1 1 0 B_2 B_1 B_0	← Saddr-offset →	← jdisp →
	sfr.bit,$addr16	0 0 0 0 1 0 0 0	1 0 1 1 1 B_2 B_1 B_0	← Sfr-offset →
		← jdisp →		
	A.bit,$addr16	0 0 0 0 0 0 1 1	1 0 1 1 1 B_2 B_1 B_0	← jdisp →
	X.bit,$addr16	0 0 0 0 0 0 1 1	1 0 1 1 0 B_2 B_1 B_0	← jdisp →
	PSWH.bit,$addr16	0 0 0 0 0 0 1 0	1 0 1 1 1 B_2 B_1 B_0	← jdisp →
	PSWL.bit,$addr16	0 0 0 0 0 0 1 0	1 0 1 1 0 B_2 B_1 B_0	← jdisp →
BF	saddr.bit,$addr16	0 0 0 0 1 0 0 0	1 0 1 0 0 B_2 B_1 B_0	← Saddr-offset →
		← jdisp →		
	sfr.bit,$addr16	0 0 0 0 1 0 0 0	1 0 1 0 1 B_2 B_1 B_0	← Sfr-offset →
		← jdisp →		
	A.bit,$addr16	0 0 0 0 0 0 1 1	1 0 1 0 1 B_2 B_1 B_0	← jdisp →
	x.bit,$addr16	0 0 0 0 0 0 1 1	1 0 1 0 0 B_2 B_1 B_0	← jdisp →
	PSWH.bit,$addr16	0 0 0 0 0 0 1 0	1 0 1 0 1 B_2 B_1 B_0	← jdisp →
	PSWL.bit,$addr16	0 0 0 0 0 0 1 0	1 0 1 0 0 B_2 B_1 B_0	← jdisp →
BTCLR	saddr.bit,$addr16	0 0 0 0 1 0 0 0	1 1 0 1 0 B_2 B_1 B_0	← Saddr-offset →
		← jdisp →		
	sfr.bit,$addr16	0 0 0 0 1 0 0 0	1 1 0 1 1 B_2 B_1 B_0	← Sfr-offset →
		← jdisp →		
	A.bit,$addr16	0 0 0 0 0 0 1 1	1 1 0 1 1 B_2 B_1 B_0	← jdisp →
	X.bit,$addr16	0 0 0 0 0 0 1 1	1 1 0 1 0 B_2 B_1 B_0	← jdisp →
	PSWH.bit,$addr16	0 0 0 0 0 0 1 0	1 1 0 1 1 B_2 B_1 B_0	← jdisp →
	PSWL.bit,$addr16	0 0 0 0 0 0 1 0	1 1 0 1 0 B_2 B_1 B_0	← jdisp →
BFSET	saddr.bit,$addr16	0 0 0 0 1 0 0 0	1 1 0 0 0 B_2 B_1 B_0	← Saddr-offset →
		← jdisp →		
	sfr.bit,$addr16	0 0 0 0 1 0 0 0	1 1 0 0 1 B_2 B_1 B_0	← Sfr-offset →
		← jdisp →		

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
BFSET	A.bit,$addr16	0 0 0 0 0 0 1 1	1 1 0 0 1 B_2 B_1 B_0	← jdisp →
	X.bit,$addr16	0 0 0 0 0 0 1 1	1 1 0 0 0 B_2 B_1 B_0	← jdisp →
	PSWH.bit,$addr16	0 0 0 0 0 0 1 0	1 1 0 0 1 B_2 B_1 B_0	← jdisp →
	PSWL.bit,$addr16	0 0 0 0 0 0 1 0	1 1 0 0 0 B_2 B_1 B_0	← jdisp →
DBNZ	r2,$addr16	0 0 1 1 0 0 1 C_0	← jdisp →	
	saddr,$addr16	0 0 1 1 1 0 1 1	← Saddr-offset →	← jdisp →

(21) Context switching instructions: BRKCS, RETCS, RETCSB

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
BRKCS	RBn	0 0 0 0 0 1 0 1	1 1 0 1 1 N_2 N_1 N_0	
RETCS	!addr16	0 0 1 0 1 0 0 1	← Low Addr. →	← High Addr. →
RETCSB	!addr16	0 0 0 0 1 0 0 1	1 1 1 0 0 0 0 0	← Low Addr. →
		← High Addr. →		

(22) String instructions: MOVM, MOVBK, XCHM, XCHBK, CMPME, CMPBKE, CMPMNE, CMPBKNE, CMPMC, CMPBKC, CMPMNC, CMPBKNC

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
MOVM	[DE+],A	0 0 0 1 0 1 0 1	0 0 0 0 0 0 0 0	
	[DE–],A	0 0 0 1 0 1 0 1	0 0 0 1 0 0 0 0	
MOVBK	[DE+],[HL+]	0 0 0 1 0 1 0 1	0 0 1 0 0 0 0 0	
	[DE–],[HL–]	0 0 0 1 0 1 0 1	0 0 1 1 0 0 0 0	
XCHM	[DE+],A	0 0 0 1 0 1 0 1	0 0 0 0 0 0 0 1	
	[DE–],A	0 0 0 1 0 1 0 1	0 0 0 1 0 0 0 1	
XCHBK	[DE+],[HL+]	0 0 0 1 0 1 0 1	0 0 1 0 0 0 0 1	
	[DE–],[HL–]	0 0 0 1 0 1 0 1	0 0 1 1 0 0 0 1	
CMPME	[DE+],A	0 0 0 1 0 1 0 1	0 0 0 0 0 1 0 0	
	[DE–],A	0 0 0 1 0 1 0 1	0 0 0 1 0 1 0 0	
CMPBKE	[DE+],[HL+]	0 0 0 1 0 1 0 1	0 0 1 0 0 1 0 0	
	[DE–],[HL–]	0 0 0 1 0 1 0 1	0 0 1 1 0 1 0 0	

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
CMPMNE	[DE+],A	0 0 0 1 0 1 0 1	0 0 0 0 0 1 0 1	
	[DE–],A	0 0 0 1 0 1 0 1	0 0 0 1 0 1 0 1	
CMPBKNE	[DE+],[HL+]	0 0 0 1 0 1 0 1	0 0 1 0 0 1 0 1	
	[DE–],[HL–]	0 0 0 1 0 1 0 1	0 0 1 1 0 1 0 1	
CMPMC	[DE+],A	0 0 0 1 0 1 0 1	0 0 0 0 0 1 1 1	
	[DE–],A	0 0 0 1 0 1 0 1	0 0 0 1 0 1 1 1	
CMPBKC	[DE+],[HL+]	0 0 0 1 0 1 0 1	0 0 1 0 0 1 1 1	
	[DE–],[HL–]	0 0 0 1 0 1 0 1	0 0 1 1 0 1 1 1	
CMPMNC	[DE+],A	0 0 0 1 0 1 0 1	0 0 0 0 0 1 1 0	
	[DE–],A	0 0 0 1 0 1 0 1	0 0 0 1 0 1 1 0	
CMPBKNC	[DE+],[HL+]	0 0 0 1 0 1 0 1	0 0 1 0 0 1 1 0	
	[DE–],[HL–]	0 0 0 1 0 1 0 1	0 0 1 1 0 1 1 0	

(23) CPU control instructions: MOV, SWRS, SEL, NOP, EI, DI

Mnemonic	Operand	Instruction code		
		B1	B2	B3
		B4	B5	
MOV	STBC,#byte	0 0 0 0 1 0 0 1	1 1 0 0 0 0 0 0	← $\overline{\text{Data}}$ →
		← Data →		
	WDM,#byte	0 0 0 0 1 0 0 1	1 1 0 0 0 0 1 0	← $\overline{\text{Data}}$ →
		← Data →		
SWRS		0 1 0 0 0 0 1 1		
SEL	RBn	0 0 0 0 0 1 0 1	1 0 1 0 1 N_2 N_1 N_0	
	RBn,ALT	0 0 0 0 0 1 0 1	1 0 1 1 1 N_2 N_1 N_0	
NOP		0 0 0 0 0 0 0 0		
EI		0 1 0 0 1 0 1 1		
DI		0 1 0 0 1 0 1 0		

5.3 Number of Clocks of the Instructions

5.3.1 Description of clock columns

Caution The following series products use the same number of clock cycles for normal fetch and high-speed fetch. See the "Normal fetch" column in the clock cycle list.
μPD78356 Subseries, 78366 Subseries, 78372 Subseries

(1) Conditions of the number of clocks for instruction execution

The following conditions are required for calculating the number of clocks for instruction execution in Section 5.3.2.

(a) Sufficient instruction codes are always stored in an instruction queue. When the EXU requires instruction codes, it can immediately read the instruction codes from the queue.

(b) The stack pointer indicates an address from FE00H to FEFFH on main RAM.

(c) The addresses represented by mem, !addr16, [saddrp], [DE+], [DE–], [HL+], [HL–], and [rp1] indicate FE00H to FEFFH on main RAM.

(d) Only the number of clocks for microprogram execution in the EXU is counted (excluding the time required for the procedure from clearing the instruction queue to reading the instruction codes of the branch destination if a branch is taken while an instruction such as BR, CALL, RET, BRK, or RETI is issued or during interrupt handling).

The number of clocks for instruction execution is calculated under the above conditions. When an actual program is executed, the number of clocks may therefore become greater than that shown in Section 5.3.2. The following items describe the reason.

(a) When instruction codes are read from the instruction queue
If no instruction codes are stored in the instruction queue when the EXU tries to read them, the EXU waits till they are stored in the instruction queue. In particular, when processing branches, the EXU always enters the wait state because the instruction queue becomes empty till the instruction codes of the branch destination are read after the instruction queue is cleared.

(b) When data is referenced from memory other than main RAM

<1> When data is read
The EXU waits till the BCU starts the bus cycle and stops reading data.

<2> When data is written
If the EXU issues a request to the BCU for writing data, it can immediately execute the next instruction. Because the BCU cannot accept any other request from the EXU for processing while it writes data, however, the EXU waits. In this state, it cannot reference data in memory other than main RAM or in SFR, or perform branch processing till the BCU finishes writing data.
In particular, if the instruction queue is empty, it cannot be determined when the bus cycle for writing is started because fetching instruction codes has priority over writing on the external memory or the peripheral RAM. Therefore, it cannot be determined when the EXU waits because the timing corresponds with write processing.

<3> Concurrence of referencing data in memory other than main RAM and branch processing while fetching instruction codes.
If the BCU executes the bus cycle of fetching instruction codes while the EXU issues a request to the BCU for referencing data in memory other than main RAM and performing branch processing, the BCU does not accept the request till it ends the bus cycle of fetching the instruction codes. In this case, the EXU enters the wait state.

(2) Dividing the clock column into the following columns

The instructions differ in the number of clocks according to the memory area to be accessed or to which a branch is taken by the individual instructions.

- Internal ROM:
  When the instruction is fetched from internal ROM
- IRAM:
  When the internal dual-port RAM (0FE00H to 0FEFFH) is accessed
- PRAM:
  When the area of internal RAM other than IRAM is accessed
- SFR:
  When the special function register is accessed
- EMEM: When the external memory is accessed

(3) n in a clock column indicates the following:

- Shift/rotate instructions:
  Number of bits the data is shifted
- Stack manipulation instructions:
  Number of registers whose contents are saved or returned
- String instructions:
  Number of times the instruction is executed till it exits from the loop after the condition is satisfied
- Sum-of-products instructions, sum-of-products instructions with saturation function, correlation instructions, table shift instructions:
  Number of multiplication and accumulation terms or number of shift terms.

(4) w in a clock column indicates the following:

- Wait count specified in the PWC register

(5) / in a clock column indicates the following:

- Or. For example, a/b indicates a or b.

5.3.2 Numbers of clocks

(1) 8-bit data transfer instructions: MOV, XCH

Mnemonic	Operand	Byte	Clocks									
			High-speed fetch					Normal fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
MOV	r1, #byte	2	-	2	-	-	-	-	2	-	-	-
	saddr, #byte	3		3		7			3		7	
	sfr[Note], #byte	3		-			7		-			7
	r, r1	2		2		-	-		2		-	-
	A, r1	1										
	A, saddr	2		3		7			3		7	
	saddr, A	2		2		6			2		6	
	saddr, saddr	3		4		8			4		8	
	A, sfr	2		-		7	7		-		7	7
	sfr, A	2				6	6				6	6
	A, mem	1-4	See **Table 5-7 (1/5)** for details.									
	mem, A	1-4										
	A, [saddrp]	2	-	7	10	13	9	-	7	9	13	9+w
	[saddrp], A	2		6	9	12			6	8	12	8+w
	A, !addr16	4	11	8	11	10	11	10+w	8	10	10	10+w
	!addr16, A	4	-	7	10	9	10	-	7	9	9	9+w
	PSWL, #byte	3		-	-	8	-		-	-	7	-
	PSWH, #byte	3										
	PSWL, A	2				7					6	
	PSWH, A	2										
	A, PSWL	2				8					7	
	A, PSWH	2										
XCH	A, r1	1	-	4	-	-	-	-	4	-	-	-
	r, r1	2										
	A, mem	2-4	See **Table 5-7 (2/5)** for details.									
	A, saddr	2	-	5	-	13	-	-	5	-	13	-
	A, sfr	3		-		14	14		-		14	14
	A, [saddrp]	2		10	16	18	16		10	14	18	14+2w
	saddr, saddr	3		7	-	15	-		7	-	15	-

Note If STBC or WDM is coded in sfr, a different instruction having the different byte and clock counts is generated.

(2) 16-bit data transfer instructions: MOVW, XCHW

Mnemonic	Operand	Byte	Clocks									
			High-speed fetch					Normal fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
MOVW	rp1, #word	3	-	3	-	-	-	-	3	-	-	-
	saddrp, #word	4		4		8			4		8	
	sfrp, #word	4		-					-			
	rp, rp1	2		2		-			2		-	
	AX, saddrp	2		3		7			3		7	
	saddrp, AX	2		2		6			2		6	
	saddrp, saddrp	3		4		8			4		8	
	AX, sfrp	2		-		7			-		7	
	sfrp, AX	2				6					6	
	rp1, !addr16	4	11/16	8	11/16	10	11/16	10/14	8	10/14	10	10+w /14+2w
	!addr16, rp1	4	-	7	10/15	9	10/15	-	7	9/13	9	9+w /13+2w
	AX, mem	2-4	See **Tables 5-7 (2/5)** and **5-7 (3/5)** for details.									
	mem, AX	2-4										
XCHW	AX, saddrp	2	-	5	-	13	-	-	5	-	13	-
	AX, sfrp	3		-		14			-		14	
	saddrp, saddrp	3		7		15			7		15	
	rp, rp1	2		4		-			4		-	
	AX, mem	2-4	See **Table 5-7 (3/5)** for details.									

(3) 8-bit arithmetic/logical instructions: ADD, ADDC, SUB, SUBC, AND, OR, XOR, CMP

Mnemonic	Operand	Byte	Clocks									
			High-speed fetch					Normal fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
ADD	A, #byte	2	-	2	-	-	-	-	2	-	-	-
	saddr, #byte	3		4		12			4		12	
	sfr, #byte	4		-		13	13		-		13	13
	r, r1	2		3		-	-		3		-	-
	A, saddr	2				7					7	
	A, sfr	3		-		8	8		-		8	8
	saddr, saddr	3		5		9	-		5		9	-
	A, mem	2-4	See **Table 5-7 (4/5)** for details.									
	mem, A	2-4										
ADDC	A, #byte	2	-	2	-	-	-	-	2	-	-	-
	saddr, #byte	3		4		12			4		12	
	sfr, #byte	4		-		13	13		-		13	13
	r, r1	2		3		-	-		3		-	-
	A, saddr	2				7					7	
	A, sfr	3		-		8	8		-		8	8
	saddr, saddr	3		5		9	-		5		9	-
	A, mem	2-4	See **Table 5-7 (4/5)** for details.									
	mem, A	2-4										
SUB	A, #byte	2	-	2	-	-	-	-	2	-	-	-
	saddr, #byte	3		4		12			4		12	
	sfr, #byte	4		-		13	13		-		13	13
	r, r1	2		3		-	-		3		-	-
	A, saddr	2				7					7	
	A, sfr	3		-		8	8		-		8	8
	saddr, saddr	3		5		9	-		5		9	-
	A, mem	2-4	See **Table 5-7 (4/5)** for details.									
	mem, A	2-4										
SUBC	A, #byte	2	-	2	-	-	-	-	2	-	-	-
	saddr, #byte	3		4		12			4		12	
	sfr, #byte	4		-		13	13		-		13	13
	r, r1	2		3		-	-		3		-	-
	A, saddr	2				7					7	
	A, sfr	3		-		8	8		-		8	8
	saddr, saddr	3		5		9	-		5		9	-
	A, mem	2-4	See **Table 5-7 (4/5)** for details.									
	mem, A	2-4										

Mnemonic	Operand	Byte	Clocks									
			High-speed fetch					Normal fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
AND	A, #byte	2	-	2	-	-	-	-	2	-	-	-
	saddr, #byte	3		4		12			4		12	
	sfr, #byte	4		-		13	13		-		13	13
	r, r1	2		3		-	-		3		-	-
	A, saddr	2				7					7	
	A, sfr	3		-		8	8		-		8	8
	saddr, saddr	3		5		9	-		5		9	-
	A, mem	2-4	See **Table 5-7 (4/5)** for details.									
	mem, A	2-4										
OR	A, #byte	2	-	2	-	-	-	-	2	-	-	-
	saddr, #byte	3		4		12			4		12	
	sfr, #byte	4		-		13	13		-		13	13
	r, r1	2		3		-	-		3		-	-
	A, saddr	2				7					7	
	A, sfr	3		-		8	8		-		8	8
	saddr, saddr	3		5		9	-		5		9	-
	A, mem	2-4	See **Table 5-7 (4/5)** for details.									
	mem, A	2-4										
XOR	A, #byte	2	-	2	-	-	-	-	2	-	-	-
	saddr, #byte	3		4		12			4		12	
	sfr, #byte	4		-		13	13		-		13	13
	r, r1	2		3		-	-		3		-	-
	A, saddr	2				7					7	
	A, sfr	3		-		8	8		-		8	8
	saddr, saddr	3		5		9	-		5		9	-
	A, mem	2-4	See **Table 5-7 (4/5)** for details.									
	mem, A	2-4										
CMP	A, #byte	2	-	2	-	-	-	-	2	-	-	-
	saddr, #byte	3		4		8			4		8	
	sfr, #byte	4		-		9	9		-		9	9
	r, r1	2		3		-	-		3		-	-
	A, saddr	2				7					7	
	A, sfr	3		-		8	8		-		8	8
	saddr, saddr	3		5		9	-		5		9	-
	A, mem	2-4	See **Tables 5-7 (4/5)** and **5-7 (5/5)** for details.									
	mem, A	2-4										

(4) 16-bit arithmetic/logical instructions: ADDW, SUBW, CMPW

Mnemonic	Operand	Byte	Clocks									
			High-speed fetch					Normal fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
ADDW	AX, #word	3	-	3	-	-	-	-	3	-	-	-
	saddrp, #word	4		5		13			5		13	
	sfrp, #word	5		-		14			-		14	
	rp, rp1	2		3		-			3		-	
	AX, saddrp	2				7					7	
	AX, sfrp	3		-		8			-		8	
	saddrp, saddrp	3		5		9			5		9	
SUBW	AX, #word	3	-	3	-	-	-	-	3	-	-	-
	saddrp, #word	4		5		13			5		13	
	sfrp, #word	5		-		14			-		14	
	rp, rp1	2		3		-			3		-	
	AX, saddrp	2				7					7	
	AX, sfrp	3		-		8			-		8	
	saddrp, saddrp	3		5		9			5		9	
CMPW	AX, #word	3	-	3	-	-	-	-	3	-	-	-
	saddrp, #word	4		5		9			5		9	
	sfrp, #word	5		-		10			-		10	
	rp, rp1	2		3		-			3		-	
	AX, saddrp	2				7					7	
	AX, sfrp	3		-		8			-		8	
	saddrp, saddrp	3		5		9			5		9	

(5) Multiply/divide instructions: MULU, DIVUW, MULUW, DIVUX

Mnemonic	Operand	Byte	Clocks									
			High-speed fetch					Normal fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
MULU	r1	2	-	11	-	-	-	-	11	-	-	-
DIVUW	r1	2	-	23	-	-	-	-	23	-	-	-
MULUW	rp1	2	-	15	-	-	-	-	15	-	-	-
DIVUX	rp1	2	-	43	-	-	-	-	43	-	-	-

(6) Signed multiply instruction: MULW

Mnemonic	Operand	Byte	Clocks									
			High-speed fetch					Normal fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
MULW	rp1	2	-	14	-	-	-	-	14	-	-	-

(7) Sum-of-products instruction: MACW

Mnemonic	Operand	Byte	Clocks									
			High-speed fetch					Normal fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
MACW	n	3	-	5+21n (2+21n)	-	-	-	-	5+21n (2+21n)	-	-	-

Remark The clock count enclosed in parentheses applies when execution of the instruction is forcibly stopped due to an overflow.

(8) Sum-of-products instruction with saturation function: MACSW

Mnemonic	Operand	Byte	Clocks				
			Internal ROM	IRAM	PRAM	SFR	EMEM
MACSW	n	3	-	5+21n (7+21n)	-	-	-

Remarks 1. If instruction execution is forced to be stopped when an overflow or underflow occurs, the number of clock cycles enclosed in parentheses is applicable.

2. The μPD78352A Subseries does not provide the sum-of-products instruction with saturation function.

(9) Correlation instruction: SACW

Mnemonic	Operand	Byte	Clocks				
			Internal ROM	IRAM	PRAM	SFR	EMEM
SACW	[DE+], [HL+]	4	4+(27+4w)n	4+(27+4w)n	4+(27+4w)n	4+(27+4w)n	4+(27+4w)n

Remark The μPD78352A Subseries does not provide the correlation instruction.

(10) Table shift instruction: MOVTBLW

Mnemonic	Operand	Byte	Clocks									
			High-speed fetch					Normal fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
MOVTBLW	!addr16, n	4	-	2+5n	-	-	-	-	2+5n	-	-	-

(11) Increment/decrement instructions: INC, DEC, INCW, DECW

Mnemonic	Operand	Byte	Clocks									
			High-speed fetch					Normal fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
INC	r1	1	-	2	-	-	-	-	2	-	-	-
	saddr	2		3		11			3		11	
DEC	r1	1	-	2	-	-	-	-	2	-	-	-
	saddr	2		3		11			3		11	
INCW	rp2	1	-	2	-	-	-	-	2	-	-	-
	saddrp	3		4		12			4		12	
DECW	rp2	1	-	2	-	-	-	-	2	-	-	-
	saddrp	3		4		12			4		12	

(12) Shift/rotate instructions: ROR, ROL, RORC, ROLC, SHR, SHL, SHRW, SHLW, ROR4, ROL4

Mnemonic	Operand	Byte	Clocks									
			High-speed fetch					Normal fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
ROR	r1, n	2	-	5+n	-	-	-	-	5+n	-	-	-
ROL	r1, n	2	-	5+n	-	-	-	-	5+n	-	-	-
RORC	r1, n	2	-	5+n	-	-	-	-	5+n	-	-	-
ROLC	r1, n	2	-	5+n	-	-	-	-	5+n	-	-	-
SHR	r1, n	2	-	5+n	-	-	-	-	5+n	-	-	-
SHL	r1, n	2	-	5+n	-	-	-	-	5+n	-	-	-
SHRW	rp1, n	2	-	5+n	-	-	-	-	5+n	-	-	-
SHLW	rp1, n	2	-	5+n	-	-	-	-	5+n	-	-	-
ROR4	[rp1]	2	-	11	17	15	17	-	11	15	15	15+2w
ROL4	[rp1]	2	-	11	17	15	17	-	11	15	15	15+2w

(13) BCD correction instructions: ADJBA, ADJBS

Mnemonic	Operand	Byte	Clocks									
			High-speed fetch					Normal fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
ADJBA		2	-	4	-	-	-	-	4	-	-	-
ADJBS		2	-	4	-	-	-	-	4	-	-	-

(14) Data conversion instruction: CVTBW

Mnemonic	Operand	Byte	Clocks									
			High-speed fetch					Normal fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
CVTBW		1	-	3	-	-	-	-	3	-	-	-

(15) Bit manipulation instructions: MOV1, AND1, OR1, XOR1, SET1, CLR1, NOT1

Mnemonic	Operand	Byte	Clocks									
			High-speed fetch					Normal fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
MOV1	CY, saddr.bit	3	-	6	-	10	-	-	6	-	10	-
	CY, sfr.bit	3		-			10		-			10
	CY, A.bit	2		5		-	-		5		-	-
	CY, X.bit	2										
	CY, PSWH.bit	2		-		5			-		5	
	CY, PSWL.bit	2										
	saddr.bit, CY	3		5		13			5		13	
	sfr.bit, CY	3		-			13		-			13
	A.bit, CY	2		6		-	-		6		-	-
	X.bit, CY	2										
	PSWH.bit, CY	2		-		7			-		7	
	PSWL.bit, CY	2				8					8	
AND1	CY, saddr.bit	3	-	6	-	10	-	-	6	-	10	-
	CY, /saddr.bit	3										
	CY, sfr.bit	3		-			10		-			10
	CY, /sfr.bit	3										
	CY, A.bit	2		5		-	-		5		-	-
	CY, /A.bit	2										
	CY, X.bit	2										
	CY, /X.bit	2										
	CY, PSWH.bit	2		-		5			-		5	
	CY, /PSWH.bit	2										
	CY, PSWL.bit	2										
	CY, /PSWL.bit	2										
OR1	CY, saddr.bit	3	-	6	-	10	-	-	6	-	10	-
	CY, /saddr.bit	3										
	CY, sfr.bit	3		-			10		-			10
	CY, /sfr.bit	3										
	CY, A.bit	2		5		-	-		5		-	-
	CY, /A.bit	2										
	CY, X.bit	2										
	CY, /X.bit	2										
	CY, PSWH.bit	2		-		5			-		5	
	CY, /PSWH.bit	2										
	CY, PSWL.bit	2										
	CY, /PSWL.bit	2										

Mnemonic	Operand	Byte	Clocks									
			High-speed fetch					Normal fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
XOR1	CY, saddr.bit	3	-	6	-	10	-	-	6	-	10	-
	CY, sfr.bit	3		-			10		-			10
	CY, A.bit	2		5		-	-		5		-	-
	CY, X.bit	2										
	CY, PSWH.bit	2		-		5			-		5	
	CY, PSWL.bit	2										
SET1	saddr.bit	2	-	4	-	12	-	-	4	-	12	-
	sfr.bit	3		-		13	13		-		13	13
	A.bit	2		5		-	-		5		-	-
	X.bit	2										
	PSWH.bit	2		-		6			-		6	
	PSWL.bit	2				7					7	
	CY	1				2					2	
CLR1	saddr.bit	2	-	4	-	12	-	-	4	-	12	-
	sfr.bit	3		-		13	13		-		13	13
	A.bit	2		5		-	-		5		-	-
	X.bit	2										
	PSWH.bit	2		-		6			-		6	
	PSWL.bit	2				7					7	
	CY	1				2					2	
NOT1	saddr.bit	3	-	5	-	13	-	-	5	-	13	-
	sfr.bit	3		-			13		-			13
	A.bit	2		5		-	-		5		-	-
	X.bit	2										
	PSWH.bit	2		-		6			-		6	
	PSWL.bit	2				7					7	
	CY	1				2					2	

(16) Call/return instructions: CALL, CALLF, CALLT, BRK, RET, RETB, RETI

Mnemonic	Operand	Byte	Clocks					
			High-speed fetch			Normal fetch		
			Internal ROM	PRAM	EMEM	Internal ROM	PRAM	EMEM
CALL	!addr16	3	17/22	17/22	17/22	15+2w/19+3w	15/19	15+2w/19+3w
	rp1	2	18/23	18/23	18/23	16+2w/20+3w	16/20	16+2w/20+3w
	[rp1]	2	24/34	24/34	24/34	21+3w/29+5w	21/29	21+3w/29+5w
CALLF	!addr11	2	17/22	17/22	17/22	15+2w/19+3w	15/19	15+2w/19+3w
CALLT	[addr5]	1	23/33	-	23/33	20+3w/28+5w	-	20+3w/28+5w
BRK		1	32/47	-	32/47	28+4w/40+7w	-	28+4w/40+7w
RET		1	15/20	15/20	15/20	13+2w/17+3w	13/17	13+2w/17+3w
RETB		1	22/32	22/32	22/32	19+2w/27+5w	19/27	19+2w/27+5w
RETI		1						

(17) Stack manipulation instructions: PUSH, PUSHU, POP, POPU, MOVW, INCW, DECW

Mnemonic	Operand	Byte	Clocks					
			High-speed fetch			Normal fetch		
			Internal ROM	PRAM	EMEM	Internal ROM	PRAM	EMEM
PUSH	sfrp	3	11	11	11	11	11	11
	post	2	3+5n	3+5n	3+5n	3+5n	3+5n	3+5n
	PSW	1	5	5	5	5	5	5
PUSHU	post	2	5+5n	5+5n	5+5n	5+5n	5+5n	5+5n
POP	sfrp	3	13	13	13	13	13	13
	post	2	3+6n	3+6n	3+6n	3+6n	3+6n	3+6n
	PSW	1	6	6	6	6	6	6
POPU	post	2	7+6n	7+6n	7+6n	7+6n	7+6n	7+6n
MOVW	SP, #word	4	9	9	9	8	8	8
	SP, AX	2	7	7	7	6	6	6
	AX, SP	2	8	8	8	7	7	7
INCW	SP	2	4	4	4	4	4	4
DECW	SP	2	4	4	4	4	4	4

(18) Special instructions: CHKL, CHKLA

Mnemonic	Operand	Byte	Clocks	
			High-speed fetch	Normal fetch
CHKL	sfr	3	14	14
CHKLA	sfr	3	14	14

(19) Unconditional branch instruction: BR

Mnemonic	Operand	Byte	Clocks					
			High-speed fetch			Normal fetch		
			Internal ROM	PRAM	EMEM	Internal ROM	PRAM	EMEM
BR	!addr16	3	10	10	10	9+w	9	9+w
	rp1	2	10	10	10	9+w	9	9+w
	[rp1]	2	14	14	14	13+w	13	13+w
	$addr16	2	10	10	10	9+w	9	9+w

(20) Conditional branch instructions: BC, BL, BNC, BNL, BZ, BE, BNZ, BNE, BV, BPE, BNV, BPO, BN, BP, BGT, BGE, BLT, BLE, BH, BNH, BT, BF, BTCLR, BFSET, DBNZ

Mnemonic	Operand	Byte	Clocks						
			No branch	Branch					
				High-speed fetch			Normal fetch		
				Internal ROM	PRAM	EMEM	Internal ROM	PRAM	EMEM
BC	$addr16	2	3	10	10	10	9+w	9	9+w
BL									
BNC	$addr16	2	3	10	10	10	9+w	9	9+w
BNL									
BZ	$addr16	2	3	10	10	10	9+w	9	9+w
BE									
BNZ	$addr16	2	3	10	10	10	9+w	9	9+w
BNE									
BV	$addr16	2	3	10	10	10	9+w	9	9+w
BPE									
BNV	$addr16	2	3	10	10	10	9+w	9	9+w
BPO									
BN	$addr16	2	3	10	10	10	9+w	9	9+w
BP	$addr16	2	3	10	10	10	9+w	9	9+w
BGT	$addr16	3	4	11	11	11	10+w	10	10+w
BGE	$addr16	3	4	11	11	11	10+w	10	10+w
BLT	$addr16	3	4	11	11	11	10+w	10	10+w
BLE	$addr16	3	4	11	11	11	10+w	10	10+w
BH	$addr16	3	4	11	11	11	10+w	10	10+w
BNH	$addr16	3	4	11	11	11	10+w	10	10+w
BT	saddr.bit, $addr16	3	6	13	13	13	12+w	12	12+w
	sfr.bit, $addr16	4	11	18	18	18	17+w	17	17+w
	A.bit, $addr16	3	6	13	13	13	12+w	12	12+w
	X.bit, $addr16	3							
	PSWH.bit, $addr16	3							
	PSWL.bit, $addr16	3							
BF	saddr.bit, $addr16	4	7	14	14	14	13+w	13	13+w
	sfr.bit, $addr16	4	11	18	18	18	17+w	17	17+w
	A.bit, $addr16	3	6	13	13	13	12+w	12	12+w
	X.bit, $addr16	3							
	PSWH.bit, $addr16	3							
	PSWL.bit, $addr16	3							

Mnemonic	Operand	Byte	Clocks						
			No branch	Branch					
				High-speed fetch			Normal fetch		
				Internal ROM	PRAM	EMEM	Internal ROM	PRAM	EMEM
BTCLR	saddr.bit, $addr16	4	7	16	16	16	15+w	15	15+w
	sfr.bit, $addr16	4	15	24	24	24	23+w	23	23+w
	A.bit, $addr16	3	6	15	15	15	14+w	14	14+w
	X.bit, $addr16	3							
	PSWH.bit, $addr16	3		16	16	16	15+w	15	15+w
	PSWL.bit, $addr16	3		15	15	15	14+w	14	14+w
BFSET	saddr.bit, $addr16	4	7	16	16	16	15+w	15	15+w
	sfr.bit, $addr16	4	15	24	24	24	23+w	23	23+w
	A.bit, $addr16	3	6	15	15	15	14+w	14	14+w
	X.bit, $addr16	3							
	PSWH.bit, $addr16	3		16	16	16	15+w	15	15+w
	PSWL.bit, $addr16	3		15	15	15	14+w	14	14+w
DBNZ	r2, $addr16	2	4	12	12	12	11+w	11	11+w
	saddr, $addr16	3	5	13	13	13	12+w	12	12+w

(21) Context switching instructions: BRKCS, RETCS, RETCSB

Mnemonic	Operand	Byte	Clocks					
			High-speed fetch			Normal fetch		
			Internal ROM	PRAM	EMEM	Internal ROM	PRAM	EMEM
BRKCS	RBn	2	13	13	13	12+w	12	12+w
RETCS	!addr16	3	11	11	11	10+w	10	10+w
RETCSB	!addr16	4	12	12	12	11+w	11	11+w

(22) String instructions: MOVM, MOVBK, XCHM, XCHBK, CMPME, CMPBKE, CMPMNE, CMPBKNE, CMPMC, CMPBKC, CMPMNC, CMPBKNC

Mnemonic	Operand	Byte	Clocks									
			High-speed fetch					Normal fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
MOVM	[DE+], A	2	2+11n	2+8n	2+11n	2+10n	2+11n	2+(10+w)n	2+8n	2+10n	2+10n	2+(10+w)n
	[DE–], A	2										
MOVBK	[DE+], [HL+]	2	2+19n	2+13n	2+19n	2+17n	2+19n	2+(17+2w)n	2+13n	2+17n	2+17n	2+(17+2w)n
	[DE–], [HL–]	2										
XCHM	[DE+], A	2	2+18n	2+12n	2+18n	2+16n	2+18n	2+(16+2w)n	2+12n	2+16n	2+16n	2+(16+2w)n
	[DE–], A	2										
XCHBK	[DE+], [HL+]	2	2+33n	2+21n	2+33n	2+29n	2+33n	2+(29+4w)n	2+21n	2+29n	2+29n	2+(29+4w)n
	[DE–], [HL–]	2										
CMPME	[DE+], A	2	2+13n	2+10n	2+13n	2+12n	2+13n	2+(12+w)n	2+10n	2+12n	2+12n	2+(12+w)n
	[DE–], A	2										
CMPBKE	[DE+], [HL+]	2	2+21n	2+15n	2+21n	2+19n	2+21n	2+(19+2w)n	2+15n	2+19n	2+19n	2+(19+2w)n
	[DE–], [HL–]	2										
CMPMNE	[DE+], A	2	2+13n	2+10n	2+13n	2+12n	2+13n	2+(12+w)n	2+10n	2+12n	2+12n	2+(12+w)n
	[DE–], A	2										
CMPBKNE	[DE+], [HL+]	2	2+21n	2+15n	2+21n	2+19n	2+21n	2+(19+2w)n	2+15n	2+19n	2+19n	2+(19+2w)n
	[DE–], [HL–]	2										
CMPMC	[DE+], A	2	2+13n	2+10n	2+13n	2+12n	2+13n	2+(12+w)n	2+10n	2+12n	2+12n	2+(12+w)n
	[DE–], A	2										
CMPBKC	[DE+], [HL+]	2	2+21n	2+15n	2+21n	2+19n	2+21n	2+(19+2w)n	2+15n	2+19n	2+19n	2+(19+2w)n
	[DE–], [HL–]	2										
CMPMNC	[DE+], A	2	2+13n	2+10n	2+13n	2+12n	2+13n	2+(12+w)n	2+10n	2+12n	2+12n	2+(12+w)n
	[DE–], A	2										
CMPBKNC	[DE+], [HL+]	2	2+21n	2+15n	2+21n	2+19n	2+21n	2+(19+2w)n	2+15n	2+19n	2+19n	2+(19+2w)n
	[DE–], [HL–]	2										

(23) CPU control instructions: MOV, SWRS, SEL, NOP, EI, DI

Mnemonic	Operand	Byte	Clocks
MOV	STBC, #byte	4	13
	WDM, #byte	4	13
SWRS		1	2
SEL	RBn	2	3
	RBn, ALT	2	3
NOP		1	2
EI		1	2
DI		1	2

Table 5-7. Instruction Execution Cycles (1/5)

Instruction set	Mnemonic	Operand	Byte	Clocks									
				High-speed fetch					Normal fetch				
				Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
8-bit data transfer	**MOV**	A, [DE] A, [HL]	1	9	6	9	8	9	8+W	6	8	8	8+W
		A, [DE+] A, [HL+] A, [DE–] A, [HL–]	1	10	7	10	9	10	9+W	7	9	9	9+W
		A, [VP] A, [UP]	2	10	7	10	9	10	9+W	7	9	9	9+W
		A, [DE+A] A, [HL+A] A, [DE+B] A, [HL+B] A, [VP+DE] A, [VP+HL]	2	11	8	11	10	11	10+W	8	10	10	10+W
		A, [DE+byte] A, [HL+byte] A, [VP+byte] A, [UP+byte]	3	11	8	11	10	11	10+W	8	10	10	10+W
		A, [SP+byte]	3	12	9	12	11	12	11+W	9	11	11	11+W
		A, word [A] A, word [B] A, word [DE] A, word [HL]	4	12	9	12	11	12	11+W	9	11	11	11+W
		[DE], A [HL], A	1	-	5	8	7	8	-	5	7	7	7+W
		[DE+], A [HL+], A [DE–], A [HL–], A	1	-	6	9	8	9	-	6	8	8	8+W
		[VP], A [UP], A	2	-	7	10	9	10	-	7	9	9	9+W
		[DE+A], A [HL+A], A [DE+B], A [HL+B], A [VP+DE], A [VP+HL], A	2	-	8	11	10	11	-	8	10	10	10+W
		[DE+byte], A [HL+byte], A [VP+byte], A [UP+byte], A	3	-	8	11	10	11	-	8	10	10	10+W
		[SP+byte], A	3	-	9	12	11	12	-	9	11	11	11+W
		word [A], A word [B], A	4	-	9	12	11	12	-	9	11	11	11+W
		word [DE], A word [HL], A	4	-	8	11	10	11	-	8	10	10	10+W

Remark w is the wait count specified in the PWC register.

Table 5-7. Instruction Execution Cycles (2/5)

Instruction set	Mnemonic	Operand	Byte	Clocks									
				High-speed fetch					Normal fetch				
				Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
16-bit data transfer	**XCH**	A, [DE] A, [HL]	2	-	12	18	16	18	-	12	16	16	16+2W
		A, [DE+] A, [HL+] A, [DE–] A, [HL–]	2	-	14	20	18	20	-	14	18	18	18+2W
		A, [VP] A, [UP]	2	-	12	18	16	18	-	12	16	16	16+2W
		A, [DE+A] A, [HL+A] A, [DE+B] A, [HL+B] A, [VP+DE] A, [VP+HL]	2	-	13	19	17	19	-	13	17	17	17+2W
		A, [DE+byte] A, [HL+byte] A, [VP+byte] A, [UP+byte]	3	-	13	19	17	19	-	13	17	17	17+2W
		A, [SP+byte]	3	-	14	20	18	20	-	14	18	18	18+2W
		A, word [A] A, word [B] A, word [DE] A, word [HL]	4	-	13	19	17	19	-	13	17	17	17+2W
	MOVW	AX, [DE] AX, [HL]	2	15	7	15	9	15	13+2W	7	13	9	13+2W
		AX, [DE+] AX, [HL+] AX, [DE–] AX, [HL–]	2	17	9	17	11	17	15+2W	9	15	11	15+2W
		AX, [VP] AX, [UP]	2	15	7	15	9	15	13+2W	7	13	9	13+2W
		AX, [DE+A] AX, [HL+A] AX, [DE+B] AX, [HL+B] AX, [VP+DE] AX, [VP+HL]	2	16	8	16	10	16	14+2W	8	14	10	14+2W
		AX, [DE+byte] AX, [HL+byte] AX, [VP+byte] AX, [UP+byte]	3	16	8	16	10	16	14+2W	8	14	10	14+2W
		AX, [SP+byte]	3	17	9	17	11	17	15+2W	9	15	11	15+2W
		AX, word [A] AX, word [B] AX, word [DE] AX, word [HL]	4	17	9	17	11	17	15+2W	9	15	11	15+2W

Remark w is the wait count specified in the PWC register.

Table 5-7. Instruction Execution Cycles (3/5)

Instruction set	Mnemonic	Operand	Byte	Clocks									
				High-speed fetch					Normal fetch				
				Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
16-bit data transfer	**MOVW**	[DE], AX [HL], AX	2	-	7	15	9	15	-	7	13	9	13+2W
		[DE+], AX [HL+], AX [DE–], AX [HL–], AX	2	-	9	17	11	17	-	9	15	11	15+2W
		[VP], AX [UP], AX	2	-	7	15	9	15	-	7	13	9	13+2W
		[DE+A], AX [HL+A], AX [DE+B], AX [HL+B], AX [VP+DE], AX [VP+HL], AX	2	-	8	16	10	16	-	8	14	10	14+2W
		[DE+byte], AX [HL+byte], AX [VP+byte], AX [UP+byte], AX	3	-	8	16	10	16	-	8	14	10	14+2W
		[SP+byte], AX	3	-	9	17	11	17	-	9	15	11	15+2W
		word [A], AX word [B], AX	4	-	9	17	11	17	-	9	15	11	15+2W
		word [DE], AX word [HL], AX	4	-	8	16	10	16	-	8	4	10	14+2W
	XCHW	AX, [DE] AX, [HL]	2	-	12	28	16	28	-	12	24	16	24+4W
		AX, [DE+] AX, [HL+] AX, [DE–] AX, [HL–]	2	-	14	30	18	30	-	14	26	18	26+4W
		AX, [VP] AX, [UP]	2	-	12	28	16	28	-	12	24	16	24+4W
		AX, [DE+A] AX, [HL+A] AX, [DE+B] AX, [HL+B] AX, [VP+DE] AX, [VP+HL]	2	-	13	29	17	29	-	13	25	17	25+4W
		AX, [DE+byte] AX, [HL+byte] AX, [VP+byte] AX, [UP+byte]	3	-	13	29	17	29	-	13	25	17	25+4W
		AX, [SP+byte]	3	-	14	30	18	30	-	14	26	18	26+4W
		AX, word [A] AX, word [B] AX, word [DE] AX, word [HL]	4	-	13	29	17	29	-	13	25	17	25+4W

Remark w is the wait count specified in the PWC register.

Table 5-7. Instruction Execution Cycles (4/5)

Instruction set	Mnemonic	Operand	Byte	Clocks									
				High-speed fetch					Normal fetch				
				Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
8-bit arithmetic/ logical	**ADD ADDC SUB SUBC AND OR XOR CMP**	A, [DE] A, [HL]	2	11	8	11	10	11	10+w	8	10	10	10+W
		A, [DE+] A, [HL+] A, [DE–] A, [HL–]	2	12	9	12	11	12	11+w	9	11	11	11+W
		A, [VP] A, [UP]	2	11	8	11	10	11	10+w	8	10	10	10+W
		A, [DE+A] A, [HL+A] A, [DE+B] A, [HL+B] A, [VP+DE] A, [VP+HL]	2	12	9	12	11	12	11+w	9	11	11	11+W
		A, [DE+byte] A, [HL+byte] A, [VP+byte] A, [UP+byte]	3	12	9	12	11	12	11+w	9	11	11	11+W
		A, [SP+byte]	3	13	10	13	12	13	12+w	10	12	12	12+2W
		A, word [A] A, word [B] A, word [DE] A, word [HL]	4	13	10	13	12	13	12+w	10	12	12	12+W
	ADD ADDC SUB SUBC AND OR XOR	[DE], A [HL], A [DE+], A [HL+], A [DE–], A [HL–], A [VP], A [UP], A	2	-	10	16	14	16	-	10	14	14	14+2W
		[DE+A], A [HL+A], A [DE+B], A [HL+B], A [VP+DE], A [VP+HL], A	2	-	11	17	15	17	-	11	15	15	15+2W
		[DE+byte], A [HL+byte], A [VP+byte], A [UP+byte], A	3	-	11	17	15	17	-	11	15	15	15+2W
		[SP+byte], A	3	-	12	18	16	18	-	12	16	16	16+2W
		word [A], A word [B], A	4	-	12	18	16	18	-	12	16	16	16+2W
		word [DE], AX word [HL], AX	4	-	11	17	15	17	-	11	15	15	15+2W

Remark w is the wait count specified in the PWC register.

Table 5-7. Instruction Execution Cycles (5/5)

Instruction set	Mnemonic	Operand	Byte	Clocks									
				High-speed fetch					Normal fetch				
				Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
8-bit arithmetic/ logical	**CMP**	[DE], A [HL], A [DE+], A [HL+], A [DE–], A [HL–], A [VP], A [UP], A	2	11	8	11	10	11	10+w	8	10	10	10+w
		[DE+A], A [HL+A], A [DE+B], A [HL+B], A [VP+DE], A [VP+HL], A	2	12	9	12	11	12	11+w	9	11	11	11+w
		[DE+byte], A [HL+byte], A [VP+byte], A [UP+byte], A	3	12	9	12	11	12	11+w	9	11	11	11+w
		[SP+byte], A	3	13	10	13	12	13	12+w	10	12	12	12+w
		word [A], A word [B], A	4	13	10	13	12	13	12+w	10	12	12	12+w
		word [DE], AX word [HL], AX	4	12	9	12	11	12	11+w	9	11	11	11+W

Remark w is the wait count specified in the PWC register.

CHAPTER 6 EXPLANATION OF INSTRUCTIONS

This chapter explains the μPD78356 instructions.

Operands are collected for each instruction mnemonic for explanation.

See **CHAPTER 5 INSTRUCTION SET LIST** for the number of bytes, the operation code, and the number of clocks of each instruction.

Description example

Mnemonic

MOV

Full name

Move

Transfer byte data

Meaning of instruction

[Instruction format] **MOV dst, src** : Shows the basic description format of the instruction.

[Operation] **dst ← src** : Shows the operation of the instruction with symbols.

[Operands] : Shows the operands that can be used with the instruction. See **CHAPTER 5 INSTRUCTION SET LIST** for the description of the symbols of the operands.

Mnemonic	Operands
MOV	r1, #byte
	saddr, #byte
	A, saddr
	sfr, A
	A, mem

Mnemonic	Operands
MOV	mem, A
	A, [saddrp]
	PSWL, #byte
	A, PSWL
	A, PSWH

[Flags] : Shows the operation of flags changed as the instruction is executed. Operation symbols of the flags are listed in legend.

S	Z	AC	P/V	CY

Legend

Symbol	Description
(blank)	No change
0	Clear to 0
1	Set to 1
×	Set or clear according to the result
P	P/V flag operates as parity flag
V	P/V flag operates as overflow flag
R	Previously saved value is restored

[Explanation] : Explains the operation of the instruction in detail.

- MOV moves the contents of the source operand (src) specified in the second operand to the destination operand (dst) specified in the first operand.

[Description example]

MOV A, #4DH ; Transfer 4DH to A register

6.1 8-Bit Data Transfer Instructions

The following 8-bit data transfer instructions can be used:

MOV ... 140
XCH ... 141

MOV

Exchange

Exchange byte data

[Instruction format] **MOV dst, src**

[Operation] **dst ← src**

[Operands]

Mnemonic	Operands (dst, src)
XCH	r1, #byte
	saddr, #byte
	sfr, #byte
	r, r1
	A, r1
	A, saddr
	saddr, A
	saddr, saddr
	A, sfr
	sfr, A
	A, mem

Mnemonic	Operands (dst, src)
MOV	mem, A
	A, [saddrp]
	[saddrp], A
	A, !addr16
	!addr16, A
	PSWL, #byte
	PSWH, #byte
	PSWL, A
	PSWH, A
	A, PSWL
	A, PSWH

byte=00H-FFH
addr16=0000H-FFFFH
saddr=FE20H-FF1FH
saddrp=FE20H-FF1EH (limited to even addresses)

[Flags]

For PSWL, #byte and PSWL, A operands

S	Z	AC	P/V	CY
×	×	×	×	×

Other than left

S	Z	AC	P/V	CY

[Explanation]

- MOV moves the contents of the source operand (src) specified in the second operand to the destination operand (dst) specified in the first operand.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) or PSW and its following instruction.

Caution If STBC or WDM is described as sfr, a dedicated operation code different from the instruction is generated. (See 6.23 CPU Control Instructions.)

[Description example]

MOV A, #4DH ; Transfer 4DH to A register

XCH

Exchange

Exchange byte data

[Instruction format] **XCH dst, src**

[Operation] **dst ↔ src**

[Operands]

Mnemonic	Operands (dst, src)
XCH	A, r1
	r, r1
	A, mem
	A, saddr
	A, sfr
	A, [saddrp]
	saddr, saddr

saddr=FE20H-FF1FH
saddrp=FE20H-FF1EH (limited to even addresses)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- XCH exchanges the contents of the first and second operands.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) and its following instruction.

[Description example]

XCH D, B ; Exchange the contents of B and D registers.

6.2 16-Bit Data Transfer Instructions

The following 16-bit data transfer instructions can be used:

MOVW ... 143
XCHW ... 144

MOVW

Move Word

Transfer word data

[Instruction format] **MOVW dst, src**

[Operation] **dst ← src**

[Operands]

Mnemonic	Operands (dst, src)
MOVW	rp1, #word
	saddrp, #word
	sfrp, #word
	rp, rp1
	AX, saddrp
	saddrp, AX
	saddrp, saddrp

Mnemonic	Operands (dst, src)
MOVW	AX, sfrp
	sfrp, AX
	rp1, !addr16
	!addr16, rp1
	AX, mem
	mem, AX

word = 0000H-FFFFH
saddrp = FE20H-FF1EH (limited to even addresses)
mem = 0000H-FDFFH (any address in the range can be specified)
mem = FE00H-FFFFH (limited to even addresses)
addr16 = 0000H-FDFFH (any address in the range can be specified)
addr16 = FE00H-FFFFH (limited to even addresses)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- MOVW moves the contents of the source operand (src) specified in the second operand to the destination operand (dst) specified in the first operand.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) and its following instruction.

[Description example]

MOVW AX, [HL] ; Transfer contents of memory addressed by HL register contents to AX register

XCHW

Exchange Word

Exchange word data

[Instruction format] **XCHW dst, src**

[Operation] **dst ↔ src**

[Operands]

Mnemonic	Operands (dst, src)
XCHW	AX, saddrp
	AX, sfrp
	saddrp, saddrp
	rp, rp1
	AX, mem

saddrp = FE20H-FF1EH (limited to even addresses)
mem = 0000H-FDFFH (any address in the range can be specified)
mem = FE00H-FFFFH (limited to even addresses)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- XCHW exchanges the contents of the first and second operands.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) and its following instruction.

[Description example]

XCHW AX, mem ; Exchange the contents of AX register and the contents of memory addressed by memory addressing.

6.3 8-Bit Operation Instructions

The following 8-bit operation instructions can be used:

ADD ... 146
ADDC ... 147
SUB ... 149
SUBC ... 150
AND ... 152
OR ... 153
XOR ... 154
CMP ... 155

ADD

Add

Add byte data

[Instruction format] **ADD dst, src**

[Operation] **dst, CY ← dst+src**

[Operands]

Mnemonic	Operands (dst, src)
ADD	A, #byte
	saddr, #byte
	sfr, #byte
	r, r1
	A, saddr
	A, sfr
	saddr, saddr
	A, mem
	mem, A

byte = 00H-FFH
saddr = FE20H-FF1FH

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- ADD adds the destination operand (dst) specified in the first operand and the source operand (src) specified in the second operand and stores the result in the CY flag and the destination operand (dst).
- When bit 7 of dst is set to 1 as a result of the addition, the S flag is set to 1; otherwise, cleared to 0.
- When dst becomes 0 as a result of the addition, the Z flag is set to 1; otherwise, cleared to 0.
- If the addition instruction generates a carry into bit 4 out of bit 3, the AC flag is set to 1; otherwise, cleared to 0.
- If the addition instruction generates a carry into bit 7 out of bit 6 and does not generate a carry out of bit 7 (when an overflow occurs by operation in the two's complement format) or if the addition instruction does not generate a carry into bit 7 out of bit 6 and generates a carry out of bit 7 (when an underflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.
- If the addition instruction generates a carry out of bit 7, the CY flag is set to 1; otherwise, cleared to 0.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) and its following instruction.

[Description example]

ADD A, #56H ; Add A register contents and 56H and store the result in A register.

ADDC

Add with Carry

Add byte data with carry

[Instruction format] **ADDC dst, src**

[Operation] **dst, CY ← dst+src+CY**

[Operands]

Mnemonic	Operands (dst, src)
ADDC	A, #byte
	saddr, #byte
	sfr, #byte
	r, r1
	A, saddr
	A, sfr
	saddr, saddr
	A, mem
	mem, A

byte = 00H-FFH
saddr = FE20H-FF1FH

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- ADDC adds the destination operand (dst) specified in the first operand, the source operand (src) specified in the second operand, and the CY flag together and stores the result in and the destination operand (dst) and the CY flag. The CY flag is added to the least significant bit. The instruction is used mainly to add a number of bytes.
- When bit 7 of dst is set to 1 as a result of the addition, the S flag is set to 1; otherwise, cleared to 0.
- When dst becomes 0 as a result of the addition, the Z flag is set to 1; otherwise, cleared to 0.
- If the addition instruction generates a carry into bit 4 out of bit 3, the AC flag is set to 1; otherwise, cleared to 0.
- If the addition instruction generates a carry into bit 7 out of bit 6 and does not generate a carry out of bit 7 (when an overflow occurs by operation in the two's complement format) or if the addition instruction does not generate a carry into bit 7 out of bit 6 and generates a carry out of bit 7 (when an underflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.
- If the addition instruction generates a carry out of bit 7, the CY flag is set to 1; otherwise, cleared to 0.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) and its following instruction.

[Description example]

ADDC A, 1234H[B] ; Add contents of A register and address (1234H + (B register)) and CY flag and store the result in A register.

SUB

Subtract

Subtract byte data

[Instruction format] **SUB dst, src**

[Operation] **dst, CY ← dst–src**

[Operands]

Mnemonic	Operands (dst, src)
SUB	A, #byte
	saddr, #byte
	sfr, #byte
	r, r1
	A, saddr
	A, sfr
	saddr, saddr
	A, mem
	mem, A

byte = 00H-FFH
saddr = FE20H-FF1FH

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- SUB subtracts the source operand (src) specified in the second operand from the destination operand (dst) specified in the first operand and stores the result in the destination operand (dst) and the CY flag.
- If the same contents are specified in the source operand (src) and the destination operand (dst), the destination operand (dst) can be cleared to 0.
- When bit 7 of dst is set to 1 as a result of the subtraction, the S flag is set to 1; otherwise, cleared to 0.
- When dst is 0 as a result of the subtraction, the Z flag is set to 1; otherwise, cleared to 0.
- If the subtraction instruction generates a borrow into bit 3 out of bit 4, the AC flag is set to 1; otherwise, cleared to 0.
- If the subtraction instruction generates a borrow into bit 7 out of bit 6 and does not generate a borrow in bit 7 (when an underflow occurs by operation in the two's complement format) or if the subtraction instruction does not generate a borrow into bit 7 out of bit 6 and generates a borrow in bit 7 (when an overflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.
- If the subtraction instruction generates a borrow in bit 7, the CY flag is set to 1; otherwise, cleared to 0.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) and its following instruction.

[Description example]

SUB D, C ; Subtract C register from D register and store the result in D register.

SUBC

Subtract with Carry

Subtract byte data with carry

[Instruction format] **SUBC dst, src**

[Operation] **dst, CY ← dst–src–CY**

[Operands]

Mnemonic	Operands (dst, src)
SUBC	A, #byte
	saddr, #byte
	sfr, #byte
	r, r1
	A, saddr
	A, sfr
	saddr, saddr
	A, mem
	mem, A

byte = 00H-FFH
saddr = FE20H-FF1FH

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- SUBC subtracts the source operand (src) specified in the second operand and the CY flag from the destination operand (dst) specified in the first operand and stores the result in the destination operand (dst) and the CY flag. The CY flag is subtracted from the least significant bit. The instruction is used mainly to subtract a number of bytes.
- When bit 7 of dst is set to 1 as a result of the subtraction, the S flag is set to 1; otherwise, cleared to 0.
- When dst is 0 as a result of the subtraction, the Z flag is set to 1; otherwise, cleared to 0.
- If the subtraction instruction generates a borrow into bit 3 out of bit 4, the AC flag is set to 1; otherwise, cleared to 0.
- If the subtraction instruction generates a borrow into bit 7 out of bit 6 and does not generate a borrow in bit 7 (when an underflow occurs by operation in the two's complement format) or if the subtraction instruction does not generate a borrow into bit 7 out of bit 6 and generates a borrow in bit 7 (when an overflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.
- If the subtraction instruction generates a borrow in bit 7, the CY flag is set to 1; otherwise, cleared to 0.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) and its following instruction.

[Description example]

SUBC A, [DE+] ; Subtract contents of address (DE register) and CY flag from A register and store the result in A register and after the subtraction, increment DE register.

AND

And

AND byte data

[Instruction format] **AND dst, src**

[Operation] **dst ← dst ∧ src**

[Operands]

Mnemonic	Operands (dst, src)
AND	A, #byte
	saddr, #byte
	sfr, #byte
	r, r1
	A, saddr
	A, sfr
	saddr, saddr
	A, mem
	mem, A

byte = 00H-FFH
saddr = FE20H-FF1FH

[Flags]

S	Z	AC	P/V	CY
×	×		P	

[Explanation]

- AND ANDs the destination operand (dst) specified in the first operand and the source operand (src) specified in the second operand for each bit and stores the result in the destination operand (dst).
- When bit 7 of dst is set to 1 as a result of the ANDing, the S flag is set to 1; otherwise, cleared to 0.
- When all bits are 0 as a result of the ANDing, the Z flag is set to 1; otherwise, cleared to 0.
- When the number of bits set to 1 in dst is even as a result of the ANDing, the P/V flag is set to 1; otherwise, cleared to 0.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) and its following instruction.

[Description example]

AND SADG, #11011100B ; AND contents of SADG address that can be accessed by short direct addressing and bit string 11011100B for each bit and store the result in SADG.

OR

Or

OR byte data

[Instruction format] **OR dst, src**

[Operation] **dst ← dst ∨ src**

[Operands]

Mnemonic	Operands (dst, src)
OR	A, #byte
	saddr, #byte
	sfr, #byte
	r, r1
	A, saddr
	A, sfr
	saddr, saddr
	A, mem
	mem, A

byte = 00H-FFH
saddr = FE20H-FF1FH

[Flags]

S	Z	AC	P/V	CY
×	×		P	

[Explanation]

- OR ORs the destination operand (dst) specified in the first operand and the source operand (src) specified in the second operand for each bit and stores the result in the destination operand (dst).
- When bit 7 of dst is set to 1 as a result of the ORing, the S flag is set to 1; otherwise, cleared to 0.
- When all bits are 0 as a result of the ORing, the Z flag is set to 1; otherwise, cleared to 0.
- When the number of bits set to 1 in dst is even as a result of the ORing, the P/V flag is set to 1; otherwise, cleared to 0.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) and its following instruction.

[Description example]

OR A, FE98H ; OR A register contents and FE98H for each bit and store the result in A register.

XOR

Exclusive Or

Exclusive-OR byte data

[Instruction format] **XOR dst, src**

[Operation] **dst ← dst ∀ src**

[Operands]

Mnemonic	Operands (dst, src)
XOR	A, #byte
	saddr, #byte
	sfr, #byte
	r, r1
	A, saddr
	A, sfr
	saddr, saddr
	A, mem
	mem, A

byte = 00H-FFH
saddr = FE20H-FF1FH

[Flags]

S	Z	AC	P/V	CY
×	×		P	

[Explanation]

- XOR exclusive-ORs the destination operand (dst) specified in the first operand and the source operand (src) specified in the second operand for each bit and stores the result in the destination operand (dst).
- If #0FFH is specified in the source operand (src) of the instruction, all bits of the destination operand (dst) can be negated.
- When bit 7 of dst is set to 1 as a result of the exclusive-ORing, the S flag is set to 1; otherwise, cleared to 0.
- When all bits are 0 as a result of the exclusive-ORing, the Z flag is set to 1; otherwise, cleared to 0.
- When the number of bits set to 1 in dst is even as a result of the exclusive-ORing, the P/V flag is set to 1; otherwise, cleared to 0.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) and its following instruction.

[Description example]

XOR A, #0FFH ; Exclusive-OR A register contents and 0FFH for each bit and store the result in A register.

CMP

Compare

Compare byte data

[Instruction format] **CMP dst, src**

[Operation] **dst–src**

[Operands]

Mnemonic	Operands (dst, src)
CMP	A, #byte
	saddr, #byte
	sfr, #byte
	r, r1
	A, saddr
	A, sfr
	saddr, saddr
	A, mem
	mem, A

byte = 00H-FFH
saddr = FE20H-FF1FH

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- CMP subtracts the source operand (src) specified in the second operand from the destination operand (dst) specified in the first operand and stores the result nowhere and changes the S, Z, AC, P/V, CY flags only.
- When bit 7 is set to 1 as a result of the subtraction, the S flag is set to 1; otherwise, cleared to 0.
- When the subtraction result is 0, the Z flag is set to 1; otherwise, cleared to 0.
- If the subtraction instruction generates a borrow into bit 3 out of bit 4, the AC flag is set to 1; otherwise, cleared to 0.
- If the instruction generates a borrow in bit 7 and does not generate a borrow in bit 6 (when an underflow occurs by operation in the two's complement format) or if the instruction does not generate a borrow in bit 7 and generates a borrow in bit 6 (when an overflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.
- If the subtraction instruction generates a borrow in bit 7, the CY flag is set to 1; otherwise, cleared to 0.

[Description example]

CMP 0FE38H, 0FED0H ; Subtract contents of address FE38H from contents of address FED0H and change flags only (compare contents of address FE38H with contents of address FED0H)

6.4 16-Bit Operation Instructions

The following 16-bit operation instructions can be used:

ADDW ... 157
SUBW ... 158
CMPW ... 159

ADDW

Add Word

Add word data

[Instruction format] **ADDW dst, src**

[Operation] **dst, CY → dst+src**

[Operands]

Mnemonic	Operands (dst, src)
ADDW	AX, #word
	saddrp, #word
	sfrp, #word
	rp, rp1
	AX, saddrp
	AX, sfrp
	saddrp, saddrp

word = 0000H-FFFFH
saddrp = FE20H-FF1EH (limited to even addresses)

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- ADDW adds the destination operand (dst) specified in the first operand and the source operand (src) specified in the second operand and stores the result in the CY flag and the destination operand (dst).
- When bit 15 of dst is set to 1 as a result of the addition, the S flag is set to 1; otherwise, cleared to 0.
- When dst becomes 0 as a result of the addition, the Z flag is set to 1; otherwise, cleared to 0.
- The AC flag becomes undefined as a result of the addition.
- If the addition instruction generates a carry into bit 15 out of bit 14 and does not generate a carry out of bit 15 (when an overflow occurs by operation in the two's complement format) or if the addition instruction does not generate a carry into bit 15 out of bit 14 and generates a carry out of bit 15 (when an underflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.
- If the addition instruction generates a carry out of bit 15, the CY flag is set to 1; otherwise, cleared to 0.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) and its following instruction.

[Description example]

ADDW AX, #0ABCDH ; Add AX register contents and ABCDH and store the result in AX register.

SUBW

Subtract Word

Subtract word data

[Instruction format] **SUBW dst, src**

[Operation] **dst, CY ← dst–src**

[Operands]

Mnemonic	Operands (dst, src)
SUBW	AX, #word
	saddrp, #word
	sfrp, #word
	rp, rp1
	AX, saddrp
	AX, sfrp
	saddrp, saddrp

word = 0000H-FFFFH
saddrp = FE20H-FF1EH (limited to even addresses)

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- SUBW subtracts the source operand (src) specified in the second operand from the destination operand (dst) specified in the first operand and stores the result in the destination operand (dst) and the CY flag.
- If the same contents are specified in the source operand (src) and the destination operand (dst), the destination operand (dst) can be cleared to 0.
- When bit 15 of dst is set to 1 as a result of the subtraction, the S flag is set to 1; otherwise, cleared to 0.
- When dst is 0 as a result of the subtraction, the Z flag is set to 1; otherwise, cleared to 0.
- The AC flag becomes undefined as a result of the subtraction.
- If the subtraction instruction generates a borrow into bit 15 out of bit 14 and does not generate a borrow in bit 15 (when an underflow occurs by operation in the two's complement format) or if the subtraction instruction does not generate a borrow into bit 15 out of bit 14 and generates a borrow in bit 15 (when an overflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.
- If the subtraction instruction generates a borrow in bit 15, the CY flag is set to 1; otherwise, cleared to 0.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) and its following instruction.

[Description example]

SUBW AX, BC ; Subtract BC register contents from AX register contents and store the result in AX register.

CMPW

Compare Word

Compare word data

[Instruction format] **CMPW dst, src**

[Operation] **dst–src**

[Operands]

Mnemonic	Operands (dst, src)
CMPW	AX, #word
	saddrp, #word
	sfrp, #word
	rp, rp1
	AX, saddrp
	AX, sfrp
	saddrp, saddrp

word = 0000H-FFFFH
saddrp = FE20H-FF1EH (limited to even addresses)

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- CMPW subtracts the source operand (src) specified in the second operand from the destination operand (dst) specified in the first operand and stores the result nowhere and changes the Z, AC, CY flags only.
- When bit 15 is set to 1 as a result of the subtraction, the S flag is set to 1; otherwise, cleared to 0.
- When the subtraction result is 0, the Z flag is set to 1; otherwise, cleared to 0.
- The AC flag becomes undefined as a result of the subtraction.
- If the subtraction instruction generates a borrow into bit 15 out of bit 14 and does not generate a borrow in bit 15 (when an underflow occurs by operation in the two's complement format) or if the subtraction instruction does not generate a borrow into bit 15 out of bit 14 and generates a borrow in bit 15 (when an overflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.
- If the subtraction instruction generates a borrow in bit 15, the CY flag is set to 1; otherwise, cleared to 0.

[Description example]

CMPW AX, SADG ; Subtract word data at SADG address that can be accessed by short direct addressing from AX register contents and change flags only (compare AX register contents with word data at SADG address)

6.5 Multiplication and Division Instructions

The following multiplication and division instructions can be used:

MULU ... 161
DIVUW ... 162
MULUW ... 163
DIVUX ... 164

MULU

Multiply Unsigned byte

Multiply unsigned byte data

[Instruction format] **MULU src**

[Operation] **AX ← A × src**

[Operands]

Mnemonic	Operands (src)
MULU	r1

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- MULU multiplies the A register contents by the data in the source operand (src) as unsigned data and stores the result in the AX register.

[Description example]

MULU B ; Multiply A register contents by B register contents and store the result in AX register.

DIVUW

Divide Unsigned Word

Divide unsigned word data

[Instruction format] **DIVUW dst**

[Operation] **AX (quotient), dst (remainder) ← AX ÷ dst**

[Operands]

Mnemonic	Operands (src)
DIVUW	r1

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- DIVUW divides the AX register contents by the destination operand (dst) contents and stores the quotient in the AX register and the remainder in the destination operand (dst).
  The division is executed by handling the AX register and destination operand (dst) contents as unsigned data.
- When the AX register contents are divided by 0 (dst=0),
  - AX (quotient) = FFFFH
  - dst (remainder) = original X register value

[Description example]

DIVUW C ; Divide AX register contents by C register contents and store the quotient in AX register and the remainder in C register.

MULUW

Multiply Unsigned Word

Multiply unsigned word data

[Instruction format] **MULUW src**

[Operation] **AX (high order), src (low order) ← AX × src**

[Operands]

Mnemonic	Operands (src)
MULUW	rp1

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- MULUW multiplies the AX register contents by the data in the source operand (src) as unsigned data and stores the high-order 16 bits of the result in the AX register and the low-order 16 bits in the source operand.

[Description example]

MULUW HL ; Multiply AX register contents by HL register contents and store the result in AX and HL registers.

DIVUX

Divide Unsigned Word Expansion Word

Divide unsigned double word data

[Instruction format] **DIVUX dst**

[Operation] **AXDE (quotient), dst (remainder) ← AXDE ÷ dst**

[Operands]

Mnemonic	Operands (dst)
DIVUX	rp1

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- DIVUX divides 32-bit data consisting of the AX register contents as the high-order 16 bits and the DE register contents as the low-order 16 bits by the destination operand (dst) contents and stores the high-order 16 bits of the quotient in the AX register and the low-order 16 bits in the DE register and the remainder in the destination operand (dst).
  The division is executed by handling the 32-bit data contained in the AX and DE registers and the destination operand (dst) contents as unsigned data.
- When the 32-bit data is divided by 0 (dst=0),
  - AXDE (quotient) = FFFFFFFFH
  - dst (remainder) = original DE register value

[Description example]

DIVUX BC ; Divide AXDE register contents by BC register contents and store the high-order 16 bits of the quotient in AX register and the low-order 16 bits in DE register and the remainder in BC register.

6.6 Signed Multiplication Instruction

The following signed multiplication instruction can be used:

MULW ... 166

MULW

Multiply Word

Multiply signed word data

[Instruction format] **MULW src**

[Operation] **AX (high order), src (low order) ← AX × src**

[Operands]

Mnemonic	Operands (src)
MULW	rp1

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- MULW multiplies the AX register contents by the data in the source operand (src) as signed data and stores the high-order 16 bits of the result in the AX register and the low-order 16 bits in the source operand.

[Description example]

MULW HL ; Multiply AX register contents by HL register contents and store the result in AX and HL registers.

6.7 Multiplication and Accumulation Instruction

The following multiplication and accumulation instruction can be used:

MACW ... 168

MACW

Multiply and Accumulate Word

Multiply and accumulate word data

[Instruction format] **MACW byte**

[Operation] **AXDE ← (B) × (C) + AXDE, B ← B+2, C ← C+2, byte ← byte–1**
End if (byte = 0 or P/V = 1)

[Operands]

Mnemonic	Operands
MACW	byte

[Flags]

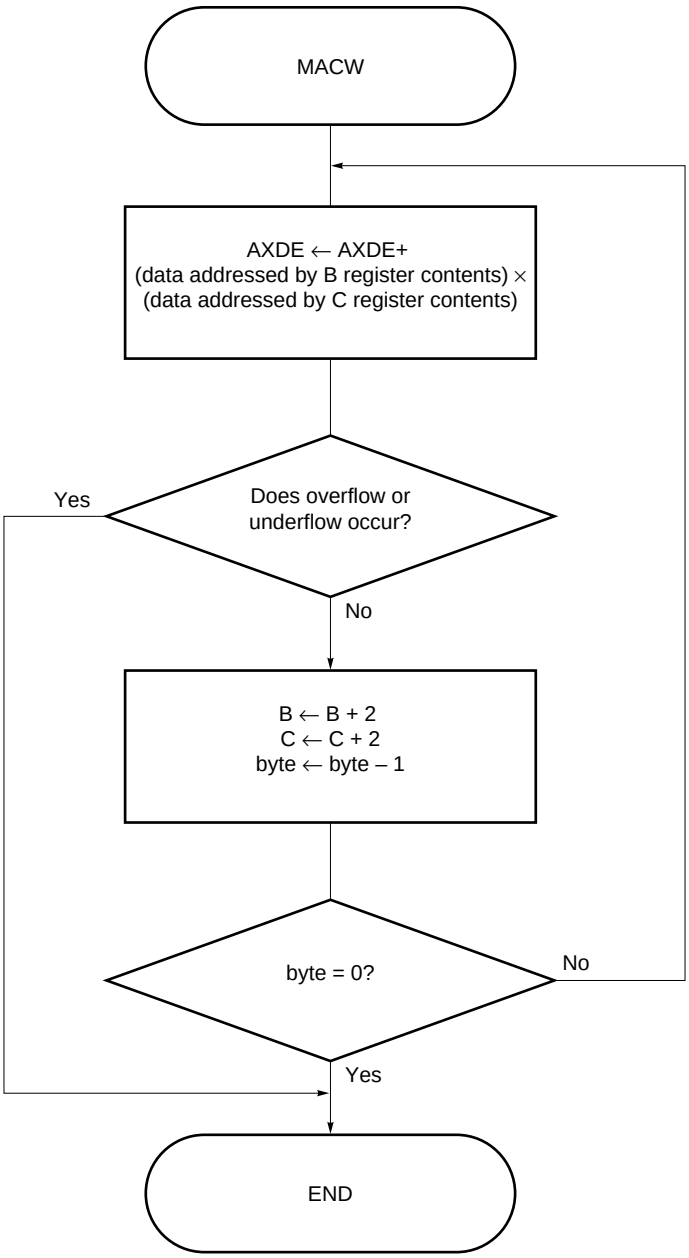
S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- MACW multiplies the contents of the 2-byte area addressed by the B register contents by the contents of the 2-byte area addressed by the C register contents as signed data and adds the result and the AXDE register contents in binary notation.
- After storing the addition result, MACW adds 2 to the B register and the C register.
- MACW repeats the operations as many times as the 8-bit immediate data described in the operand.
- If an overflow or underflow occurs as a result of the binary addition, the AXDE register value becomes undefined. The B and C registers hold the values just before the overflow occurs.
- The area addressed in the MACW instruction is limited to addresses FE00H-FEFFH. The low-order one byte of the address is specified in the B register and C register.
  Addresses FE80H-FEFFH are also used as general-purpose register area.
- Interrupts or macro service is not accepted during execution of the MACW instruction.
- The MACW instruction does not automatically clear the AXDE register pair value. Clear the value by a program if necessary.
- As a result of the operations, the S, Z, AC, and CY flags become undefined.
- When an overflow or underflow occurs, the P/V flag is set to 1; otherwise, cleared to 0.

[Description example]

MACW 5 ; Executes multiplication and accumulation operation five times.



6.8 Multiplication and Accumulation Instruction With Saturation Function

The following multiplication and accumulation instruction with saturation function can be used:

MACSW ... 171

Caution The μPD78352A Subseries does not contain the multiplication and accumulation instruction with saturation function.

MACSW

Multiply and Accumulate with Saturation Word

Multiply and accumulate with saturation function

[Instruction format] **MACSW byte**

[Operation] **AXDE ← (B) × (C) + AXDE, B ← B+2, C ← C+2, byte ← byte–1**
if byte=0 then End, if P/V=1, then if overflow AXDE ← 7FFFFFFFH,
end, if underflow AXDE ← 80000000H, end

[Operands]

Mnemonic	Operands ($addr16)
MACSW	byte

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

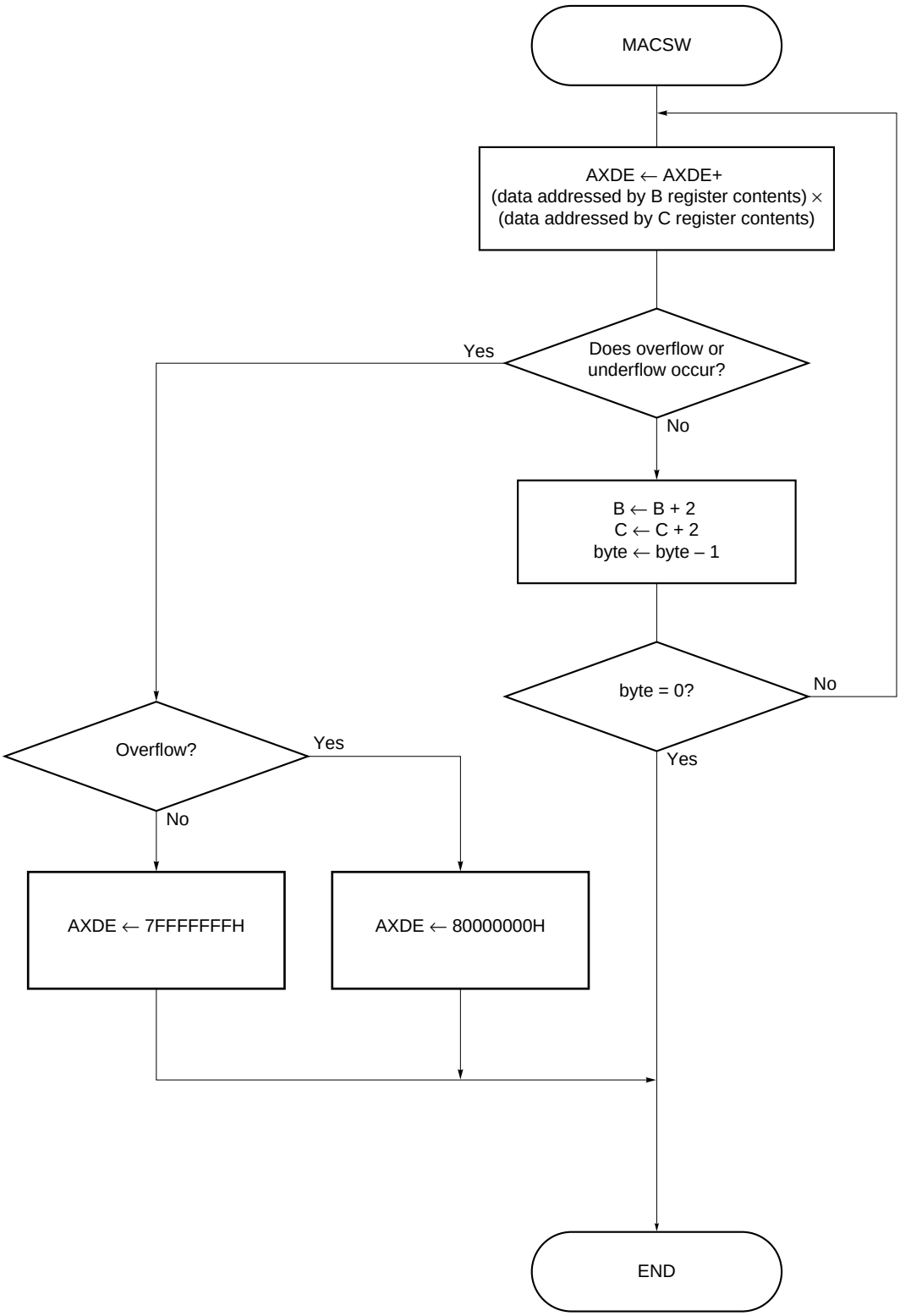
[Explanation]

- MACSW multiplies the contents of the 2-byte area addressed by the B register contents by the contents of the 2-byte area addressed by the C register contents as signed data and adds the result and the AXDE register contents in binary notation.
- After storing the addition result, MACSW adds 2 to the B register and the C register.
- MACSW repeats the operations as many times as the 8-bit immediate data described in the operand.
- If an overflow occurs as a result of the binary addition, the AXDE register is set to 7FFFFFFFH. If an underflow occurs, the P/V flag is set to 1 and the AXDE register contains 80000000H. The B and C registers hold the values just before the overflow or underflow occurs.
- The area addressed in the MACSW instruction is limited to addresses FE00H-FEFFH. The low-order one byte of the address is specified in the B register and C register.
  Addresses FE80H-FEFFH are also used as general-purpose register area.
- Interrupts or macro service is not accepted during execution of the MACSW instruction.
- The MACSW instruction does not automatically clear the AXDE register pair value. Clear the value by a program if necessary.
- As a result of the operations, the S, Z, AC, and CY flags become undefined.
- When an overflow or underflow occurs, the P/V flag is set to 1; otherwise, cleared to 0.

[Description example]

MACSW 6 ; Executes multiplication and accumulation operation with saturation function six times.

Phase-out/Discontinued



6.9 Correlation Operation Instruction

The following correlation operation instruction can be used:

SACW ... 174

Caution The μPD78352A Subseries does not contain the correlation operation instruction.

SACW

Subtract, Absolute and Accumulate Word

Correlation operation

[Instruction format] **SACW [DE+], [HL+]**

[Operation] **AX ← | (DE)–(HL) | +AX, DE ← DE+2, HL ← HL+2, C ← C–1,**
End if (C=0 or CY=1)

[Operands]

Mnemonic	Operands ($addr16)
SACW	[DE+], [HL+]

[Flags]

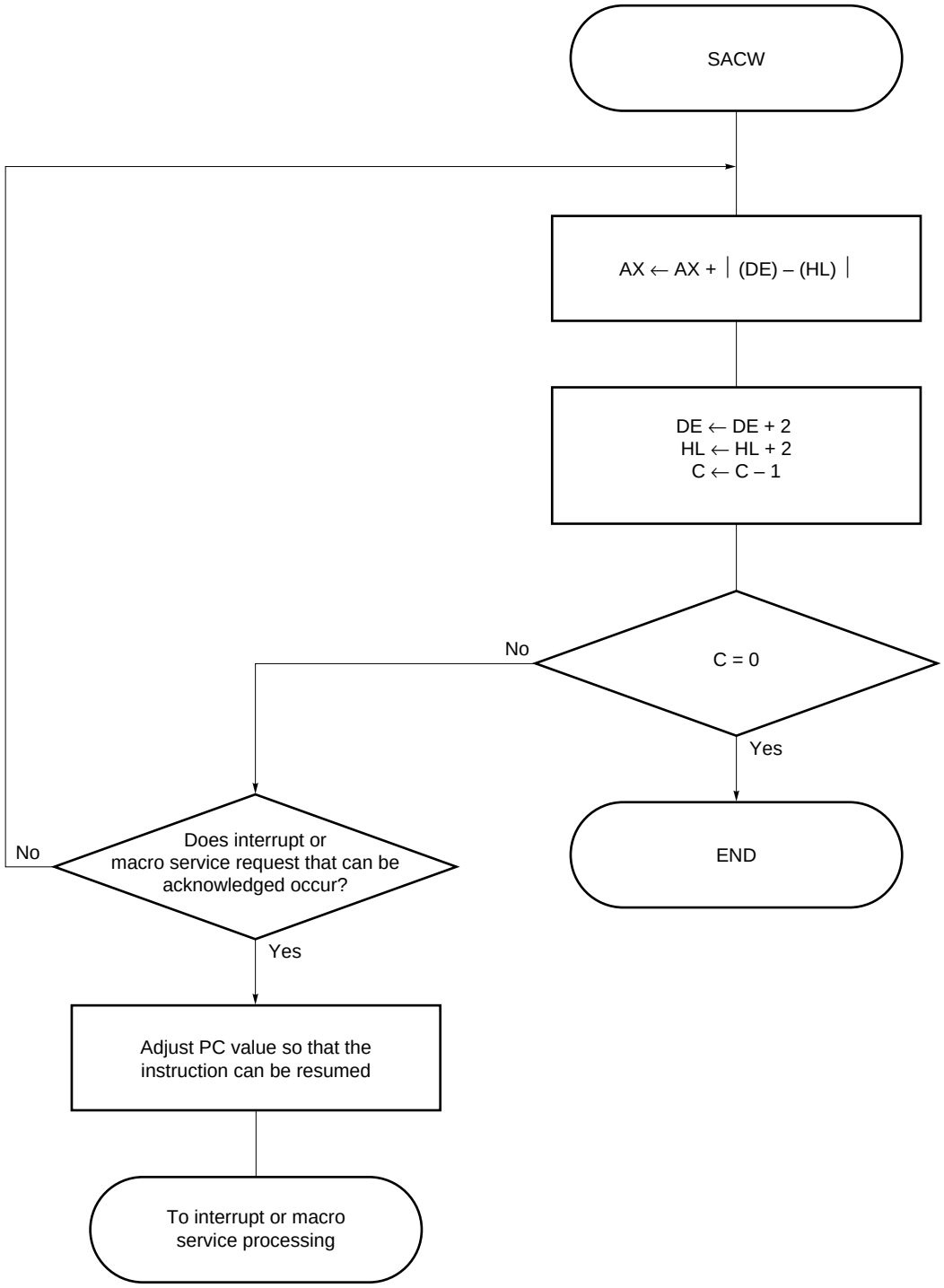
S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- SACW subtracts 16-bit data addressed by the HL register contents from 16-bit data addressed by the DE register contents and adds the absolute value of the result and the AX register contents and stores the result in the AX register.
- Whenever the operation is performed, two is added to the DE register and the HL register and one is subtracted from the C register.
- The steps are repeated until the C register is set to 0 or until a carry is generated out of bit 16 as a result of the addition.
- If a carry is generated out of bit 16 as a result of the addition and the repetitive operation is stopped, the DE register and HL register hold the values just before the carry is generated.
- If an interrupt or macro service request that can be acknowledged occurs during execution of the SACW instruction, processing of the interrupt or macro service is performed before a sequence of the operations. When the interrupt is acknowledged, the value of the program counter (PC) saved in a given stack is the top address of the SACW instruction. Therefore, when control is returned from the interrupt, execution of the interrupted SACW instruction can be continued.
- If a carry is generated out of bit 16 as a result of the last addition, the CY flag is set to 1; otherwise, cleared to 0.
- The S, Z, AC, and P/V flag contents become undefined.
- The SACW instruction does not automatically clear the AX register contents. Clear the contents by a program if necessary.

[Description example]

SACW [DE+], [HL+] ; Execute SACW instruction.



6.10 Table Shift Instruction

The following table shift instruction can be used:

MOVTBLW ... 177

MOVTBLW

Move Table Word

Transfer table word

[Instruction format] **MOVTBLW !addr16, byte**

[Operation] **(addr16+2) ← (addr16), addr16 ← addr16–2, byte ← byte–1, End if byte=0**

[Operands]

Mnemonic	Operands
MOVTBLW	!addr16, byte

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- MOVTBLW transfers the contents of the memory addressed by the 16-bit immediate data specified in the first operand to the location indicated by adding 2 to the memory address, then subtracts 2 from addr16. The operation is repeated as many times as the 8-bit immediate data described in the second operand.
- The MOVTBLW instruction is used to shift a data table used with the MACW or MACSW instruction.
- Describe the address of the least significant data of the data to be transferred in the first operand !addr16 with a label or numeric value directly.
- The area addressed in the MOVTBLW instruction is limited to addresses FE00H-FEFFH. The low-order one byte of the address is specified in the first operand !addr16. Addresses FE80H-FEFFH are also used as general-purpose register area.
- Interrupts or macro service is not accepted during execution of the MOVTBLW instruction.

[Description example]

MOVTBLW !0FE60H, 5 ; Transfer data at FE54H-FE60H to FE56H-FE62H

6.11 Increment and Decrement Instructions

The following increment and decrement instructions can be used:

INC ... 179
DEC ... 180
INCW ... 181
DECW ... 182

INC

Increment

Increment byte data

[Instruction format] **INC dst**

[Operation] **dst ← dst+1**

[Operands]

Mnemonic	Operands (dst)
INC	r1
	saddr

saddr=FE20H-FF1FH

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	

[Explanation]

- INC increments the destination operand (dst) contents by one.
- If the increment result is 0, the Z flag is set to 1;otherwise, cleared to 0.
- If the INC instruction generates a carry into bit 4 out of bit 3, the AC flag is set to 1; otherwise, cleared to 0.
- Since the INC instruction is frequently used to increment the counter of repetitive processing and the indexed addressing offset register, the CY flag contents are not changed (because the CY flag contents are held at operation on a number of bytes).
- When bit 7 of dst is set to 1 as a result of the increment, the S flag is set to 1; otherwise, cleared to 0.
- If the INC instruction generates a carry into bit 7 out of bit 6 and does not generate a carry out of bit 7 (when an overflow occurs by operation in the two's complement format) or if the INC instruction does not generate a carry into bit 7 out of bit 6 and generates a carry out of bit 7 (when an underflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.

[Description example]

INC B ; Increment B register

DEC

Decrement

Decrement byte data

[Instruction format] **DEC dst**

[Operation] **dst ← dst–1**

[Operands]

Mnemonic	Operands (dst)
DEC	r1
	saddr

saddr=FE20H-FF1FH

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	

[Explanation]

- DEC decrements the destination operand (dst) contents by one.
- If the decrement result is 0, the Z flag is set to 1; otherwise, cleared to 0.
- If the DEC instruction generates a carry into bit 3 out of bit 4, the AC flag is set to 1; otherwise, cleared to 0.
- Since the DEC instruction is frequently used to decrement the counter of repetitive processing and the indexed addressing offset register, the CY flag contents are not changed (because the CY flag contents are held at operation on a number of bytes).
- When bit 7 of dst is set to 1 as a result of the decrement, the S flag is set to 1; otherwise, cleared to 0.
- If the DEC instruction generates a borrow into bit 7 out of bit 6 and does not generate a borrow in bit 7 (when an underflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.
- To hold the S, Z, AC, and P/V flags unchanged when dst is the B register, C register, or saddr, a DBNZ instruction can be used.

[Description example]

DEC SAD1 ; Decrement the contents of SAD1 address that can be accessed by short direct addressing

INCW

Increment Word

Increment word data

[Instruction format] **INCW dst**

[Operation] **dst ← dst+1**

[Operands]

Mnemonic	Operands (dst)
INCW	rp2
	saddrp

saddrp=FE20H-FF1EH (limited to even addresses)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- INCW increments the destination operand (dst) contents by one.
- Since the INCW instruction is frequently used to increment the registers used in addressing using registers, the S, Z, AC, P/V, and CY flags are not changed.

[Description example]

INCW HL ; Increment HL register

DECW

Decrement Word

Decrement word data

[Instruction format] **DECW dst**

[Operation] **dst ← dst–1**

[Operands]

Mnemonic	Operands (dst)
DECW	rp2
	saddrp

saddrp=FE20H-FF1EH (limited to even addresses)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- DECW decrements the destination operand (dst) contents by one.
- Since the DECW instruction is frequently used to decrement the registers used in addressing using registers, the S, Z, AC, P/V, and CY flags are not changed.

[Description example]

DECW DE ; Decrement DE register

6.12 Shift and Rotate Instructions

The following shift and rotate instructions can be used:

ROR ... 184
ROL ... 185
RORC ... 186
ROLC ... 187
SHR ... 188
SHL ... 189
SHRW ... 190
SHLW ... 191
ROR4 ... 192
ROL4 ... 193

ROR

Rotate Right

Rotate byte data right

[Instruction format] **ROR dst, cnt**

[Operation] **(CY, $dst_7 \leftarrow dst_0$, $dst_{m-1} \leftarrow dst_m$) × cnt cnt=0–7**

[Operands]

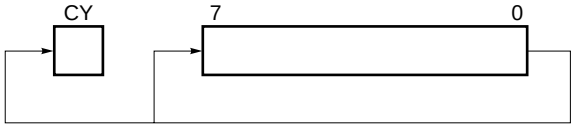
Mnemonic	Operands (dst, cnt)
ROR	r1, n

[Flags]

S	Z	AC	P/V	CY
			P	×

[Explanation]

- ROR rotates the contents of the destination operand (dst) specified in the first operand right as many times as cnt specified in the second operand.
- The LSB (bit 0) contents are rotated to the MSB (bit 7) and also transferred to the CY flag at the same time.
- If 0 is specified in the second operand (cnt), no operation is performed. (The S, Z, AC, P/V, and CY flags remain unchanged.)
- If the number of bits set to 1 in dst is even as a result of the rotating right, the P/V flag is set to 1; otherwise, cleared to 0.
- The S, Z, and AC flags remain unchanged regardless of the rotating result.



[Description example]

ROR R5, 4 ; Rotate R5 register contents right four bits

ROL

Rotate Left

Rotate byte data left

[Instruction format] **ROL dst, cnt**

[Operation] **(CY, $dst_0 \leftarrow dst_7$, $dst_{m+1} \leftarrow dst_m$) × cnt cnt=0–7**

[Operands]

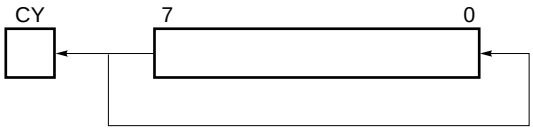
Mnemonic	Operands (dst, cnt)
ROL	r1, n

[Flags]

S	Z	AC	P/V	CY
			P	×

[Explanation]

- ROL rotates the contents of the destination operand (dst) specified in the first operand left as many times as cnt specified in the second operand.
- The MSB (bit 7) contents are rotated to the LSB (bit 0) and also transferred to the CY flag at the same time.
- If 0 is specified in the second operand (cnt), no operation is performed. (The S, Z, AC, P/V, and CY flags remain unchanged.)
- If the number of bits set to 1 in dst is even as a result of the rotating left, the P/V flag is set to 1; otherwise, cleared to 0.
- The S, Z, and AC flags remain unchanged regardless of the rotating result.



[Description example]

ROL R7, 2 ; Rotate R7 register contents left two bits

RORC

Rotate Right with Carry

Rotate byte data right with carry

[Instruction format] **RORC dst, cnt**

[Operation] **(CY ← dst_0, dst_7 ← CY, dst_{m-1} ← dst_m) × cnt cnt=0–7**

[Operands]

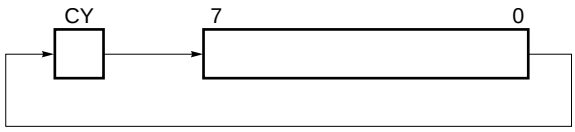
Mnemonic	Operands (dst, cnt)
RORC	r1, n

[Flags]

S	Z	AC	P/V	CY
			P	×

[Explanation]

- RORC rotates the contents of the destination operand (dst) specified in the first operand right with the CY flag as many times as cnt specified in the second operand.
- If 0 is specified in the second operand (cnt), no operation is performed. (The S, Z, AC, P/V, and CY flags remain unchanged.)
- If the number of bits set to 1 in dst is even as a result of the rotating right, the P/V flag is set to 1; otherwise, cleared to 0.
- The S, Z, and AC flags remain unchanged regardless of the rotating result.



[Description example]

RORC B, 1 ; Rotate B register contents with CY flag right one bit

ROLC

Rotate Left with Carry

Rotate byte data left with carry

[Instruction format] **ROLC dst, cnt**

[Operation] **(CY ← dst_7, dst_0 ← CY, dst_{m+1} ← dst_m) × cnt cnt=0–7**

[Operands]

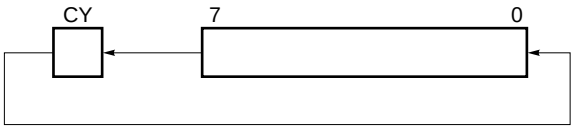
Mnemonic	Operands (dst, cnt)
ROLC	r1, n

[Flags]

S	Z	AC	P/V	CY
			P	×

[Explanation]

- ROLC rotates the contents of the destination operand (dst) specified in the first operand left with the CY flag as many times as cnt specified in the second operand.
- If 0 is specified in the second operand (cnt), no operation is performed. (The S, Z, AC, P/V, and CY flags remain unchanged.)
- To rotate left one bit only, if ADDC r, r1 is used, the execution time can be shortened.
- If the number of bits set to 1 in dst is even as a result of the rotating left, the P/V flag is set to 1; otherwise, cleared to 0.
- The S, Z, and AC flags remain unchanged regardless of the rotating result.



[Description example]

ROLC R7, 3 ; Rotate R7 register contents with CY flag left three bits

SHR

Shift Right (Logical)

Shift byte data right logically

[Instruction format] **SHR dst, cnt**

[Operation] **(CY ← dst_0, dst_7 ← 0, dst_{m-1} ← dst_m) × cnt cnt=0–7**

[Operands]

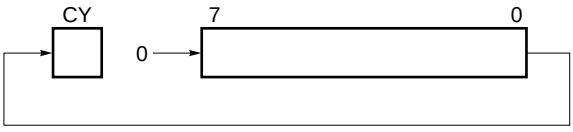
Mnemonic	Operands (dst, cnt)
SHR	r1, n

[Flags]

S	Z	AC	P/V	CY
×	×	0	P	×

[Explanation]

- SHR shifts the contents of the destination operand (dst) specified in the first operand right as many times as cnt specified in the second operand.
- Whenever the contents are shifted one bit, 0 is shifted in the MSB (bit 7).
- When 1 or more is specified as cnt, the S flag is cleared to 0.
- If the shifting result is 0, the Z flag is set to 1; otherwise, cleared to 0.
- The AC flag is always set to 0 regardless of the shifting result.
- If the number of bits set to 1 in dst is even as a result of the shifting, the P/V flag is set to 1; otherwise, cleared to 0.
- The last data shifted out from the LSB (bit 0) as a result of the shifting is set in the CY flag.
- If 0 is specified as cnt, no operation is performed. (The S, Z, AC, P/V, and CY flags remain unchanged.)
- Execution of this instruction generates the same result as dividing the destination operand (dst) as unsigned data by 2^{cnt}.



[Description example]

SHR X, 2 ; Shift X register contents right two bits

SHL

Shift Left (Logical)

Shift byte data left logically

[Instruction format] **SHL dst, cnt**

[Operation] **(CY ← dst_7, dst_0 ← 0, dst_{m+1} ← dst_m) × cnt cnt=0–7**

[Operands]

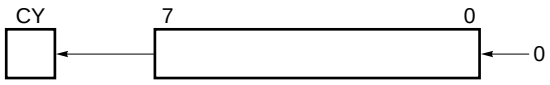
Mnemonic	Operands (dst, cnt)
SHL	r1, n

[Flags]

S	Z	AC	P/V	CY
×	×	0	P	×

[Explanation]

- SHL shifts the contents of the destination operand (dst) specified in the first operand left as many times as cnt specified in the second operand.
- Whenever the contents are shifted one bit, 0 is shifted in the LSB (bit 0).
- When bit 7 of dst is 1 as a result of the shifting, the S flag is set to 1; otherwise, cleared to 0.
- If the shifting result is 0, the Z flag is set to 1; otherwise, cleared to 0.
- The AC flag is always set to 0 regardless of the shifting result.
- If the number of bits set to 1 in dst is even as a result of the shifting, the P/V flag is set to 1; otherwise, cleared to 0.
- The last data shifted out from the LSB (bit 0) as a result of the shifting is set in the CY flag.
- If 0 is specified as cnt, no operation is performed. (The S, Z, AC, P/V, and CY flags remain unchanged.)
- To shift left one bit only, if the ADD r, r1 instruction is used, the execution time can be shortened.
- Execution of the instruction generates the same result as multiplying the destination operand (dst) by 2^{cnt} (when the multiplication result is eight bits or less).



[Description example]

SHL B, 1 ; Shift B register contents left one bit

SHRW

Shift Right (Logical) Word

Shift word data right logically

[Instruction format] **SHRW dst, cnt**

[Operation] **(CY ← dst_0, dst_{15} ← 0, dst_{m-1} ← dst_m) × cnt cnt=0–7**

[Operands]

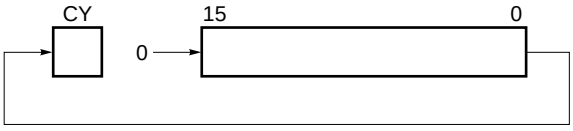
Mnemonic	Operands (dst, cnt)
SHRW	rp1, n

[Flags]

S	Z	AC	P/V	CY
×	×	0	P	×

[Explanation]

- SHRW shifts the contents of the destination operand (dst) specified in the first operand right as many times as cnt specified in the second operand.
- Whenever the contents are shifted one bit, 0 is shifted in the MSB (bit 7).
- When 1 or more is specified as cnt, the S flag is cleared to 0.
- If the shifting result is 0, the Z flag is set to 1; otherwise, cleared to 0.
- The AC flag is always set to 0 regardless of the shifting result.
- If the number of bits set to 1 of the low-order eight bits in dst is even as a result of the shifting, the P/V flag is set to 1; otherwise, cleared to 0.
- The last data shifted out from the LSB (bit 0) as a result of the shifting is set in the CY flag.
- If 0 is specified as cnt, no operation is performed. (The S, Z, AC, P/V, and CY flags remain unchanged.)
- Execution of this instruction generates the same result as dividing the destination operand (dst) as unsigned data by 2^{cnt}.



[Description example]

SHRW AX, 3 ; Shift AX register contents right three bits (divide AX register contents by 8)

SHLW

Shift Left (Logical) Word

Shift word data left logically

[Instruction format] **SHLW dst, cnt**

[Operation] **(CY ← dst_{15}, dst_0 ← 0, dst_{m+1} ← dst_m) × cnt cnt=0–7**

[Operands]

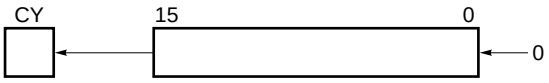
Mnemonic	Operands (dst, cnt)
SHLW	rp1, n

[Flags]

S	Z	AC	P/V	CY
×	×	0	P	×

[Explanation]

- SHLW shifts the contents of the destination operand (dst) specified in the first operand left as many times as cnt specified in the second operand.
- Whenever the contents are shifted one bit, 0 is shifted in the LSB (bit 0).
- When bit 15 of dst is 1 as a result of the shifting, the S flag is set to 1; otherwise, cleared to 0.
- If the shifting result is 0, the Z flag is set to 1; otherwise, cleared to 0.
- The AC flag is always set to 0 regardless of the shifting result.
- If the number of bits set to 1 of the low-order eight bits in dst is even as a result of the shifting, the P/V flag is set to 1; otherwise, cleared to 0.
- The last data shifted out from the LSB (bit 0) as a result of the shifting is set in the CY flag.
- If 0 is specified as cnt, no operation is performed. (The S, Z, AC, P/V, and CY flags remain unchanged.)



[Description example]

SHLW DE, 1 ; Shift DE register contents left one bit

ROR4

Rotate Right Digit

Rotate right digits

[Instruction format] **ROR4 dst**

[Operation] $A_{3-0} \leftarrow (dst)_{3-0}$, $(dst)_{7-4} \leftarrow A_{3-0}$, $(dst)_{3-0} \leftarrow (dst)_{7-4}$

[Operands]

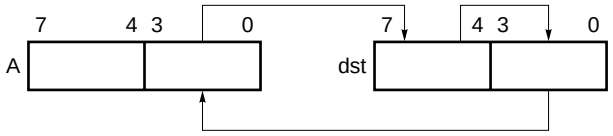
Mnemonic	Operands (dst)
ROR4	[rp1]

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- ROR4 rotates the low-order four bits of the A register and the 2-digit data (4-bit data) in the destination operand (dst) right.
  The high-order four bits of the A register do not change.



[Description example]

ROR4 [HL] ; Rotate the low-order four bits of A register and the contents of memory addressed by HL register contents right.

	A		(HL)	
	7 4 3 0		7 4 3 0	
Before execution	1010	0011	1100	0101
After execution	1010	0101	0011	1100

ROL4

Rotate Left Digit

Rotate left digits

[Instruction format] **ROL4 dst**

[Operation] $A_{3-0} \leftarrow (dst)_{7-4}$, $(dst)_{3-0} \leftarrow A_{3-0}$, $(dst)_{7-4} \leftarrow (dst)_{3-0}$

[Operands]

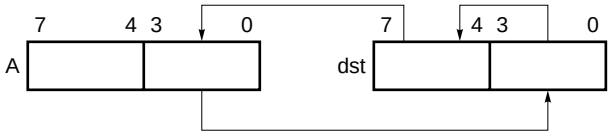
Mnemonic	Operands (dst)
ROL4	[rp1]

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- ROL4 rotates the low-order four bits of the A register and the 2-digit data (4-bit data) in the destination operand (dst) left.
  The high-order four bits of the A register do not change.



[Description example]

ROL4 [DE] ; Rotate the low-order four bits of A register and the contents of memory addressed by DE register contents left.

	A		(DE)	
	7 4 3 0		7 4 3 0	
Before execution	0001	0010	0100	1000
After execution	0001	0100	1000	0010

6.13 BCD Adjustment Instruction

The following BCD adjustment instructions can be used:

ADJBA ... 195
ADJBS ... 196

ADJBA

Decimal Adjust Register for Addition

Make decimal adjustment of addition result

[Instruction format] **ADJBA**

[Operation] **Decimal Adjust Accumulator for Addition**

[Operands] None

[Flags]

S	Z	AC	P/V	CY
×	×	×	P	×

[Explanation]

- ADJBA makes decimal adjustment of the A register, CY flag, and AC flag from the contents of the A register, CY flag, and AC flag. The ADJBA instruction is effective only when the addition result is stored in the A register after data in the BCD (binary decimal coded) format is added. The adjustment method is as listed below:

Condition		Operation
$A_{3-0} \leq 9$	$A_{7-4} \leq 9$ and CY=0	A←A, CY←0, AC←0
AC=0	$A_{7-4} \geq 10$ or CY=1	A←A+01100000B, CY←1, AC←0
$A_{3-0} \geq 10$	$A_{7-4} < 9$ and CY=0	A←A+00000110B, CY←0, AC←1
AC=0	$A_{7-4} \geq 9$ or CY=1	A←A+01100110B, CY←1, AC←1
AC=1	$A_{7-4} \leq 9$ and CY=0	A←A+00000110B, CY←0, AC←1
	$A_{7-4} \geq 10$ or CY=1	A←A+01100110B, CY←1, AC←1

- If the A register contains 0 as a result of the adjustment, the Z flag is set to 1; otherwise, cleared to 0.
- When the number of bits set to 1 in the A register is even as a result of the adjustment, the P/V flag is set to 1; otherwise, cleared to 0.
- When bit 7 of the A register is 1 as a result of the adjustment, the S flag is set to 1; otherwise, cleared to 0.

[Description example]

ADJBA ; Make decimal adjustment of A register contents.

ADJBS

Decimal Adjust Register for Subtraction

Make decimal adjustment of subtraction result

[Instruction format] **ADJBS**

[Operation] **Decimal Adjust Accumulator for Subtraction**

[Operands] None

[Flags]

S	Z	AC	P/V	CY
×	×	×	P	×

[Explanation]

- ADJBS makes decimal adjustment of the A register, CY flag, and AC flag from the contents of the A register, CY flag, and AC flag. The ADJBS instruction is effective only when the subtraction result is stored in the A register after data in the BCD (binary decimal coded) format is subtracted. The adjustment method is as listed below:

Condition		Operation
AC=0	CY=0	A←A, CY←0, AC←0
	CY=1	A←A–01100000B, CY←1, AC←0
AC=1	CY=0	A←A–00000110B, CY←0, AC←1
	CY=1	A←A–01100110B, CY←1, AC←1

- If the A register contains 0 as a result of the adjustment, the Z flag is set to 1; otherwise, cleared to 0.
- When the number of bits set to 1 in the A register is even as a result of the adjustment, the P/V flag is set to 1; otherwise, cleared to 0.
- When bit 7 of the A register is 1 as a result of the adjustment, the S flag is set to 1; otherwise, cleared to 0.

[Description example]

ADJBS ; Make decimal adjustment of A register contents.

6.14 Data Conversion Instruction

The following data conversion instruction can be used:

CVTBW ... 198

CVTBW

Convert Byte to Word

Convert byte data into word data

[Instruction format] **CVTBW**

[Operation] **When A_7=0, X ← A, A ← 00H**
When A_7=1, X ← A, A ← FFH

[Operands] None

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- CVTBW expands signed 8-bit data in the A register to signed 16-bit data in the AX register.
- When this instruction is executed, the S, Z, AC, P/V, and CY flags remain unchanged.

[Description example]

CVTBW ; Expand signed 8-bit data in the A register to signed 16-bit data and store the result in the AX register

6.15 Bit Manipulation Instructions

The following bit manipulation instructions can be used:

MOV1 ... 200
AND1 ... 201
OR1 ... 202
XOR1 ... 203
SET1 ... 204
CLR1 ... 205
NOT1 ... 206

MOV1

Move Single Bit

Transfer 1-bit data

[Instruction format] **MOV1 dst, src**

[Operation] **dst ← src**

[Operands]

Mnemonic	Operands (dst, src)
MOV1	CY, saddr.bit
	CY, sfr.bit
	CY, A.bit
	CY, X.bit
	CY, PSWH.bit
	CY, PSWL.bit

Mnemonic	Operands (dst, src)
MOV1	saddr.bit, CY
	sfr.bit, CY
	A.bit, CY
	X.bit, CY
	PSWH.bit, CY
	PSWL.bit, CY

saddr=FE20H-FF1FH
bit=0-7

[Flags]

When dst is CY

S	Z	AC	P/V	CY
				×

Other than left

S	Z	AC	P/V	CY

[Explanation]

- MOV1 moves the bit data of the source operand (src) specified in the second operand to the destination operand (dst) specified in the first operand.
- If the destination operand (dst) is CY, only the corresponding flags change.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) or PSW and its following instruction.

[Description example]

MOV1 P3.4, CY ; Transfer CY flag contents to bit 4 of port 3

AND1

And Single Bit

AND 1-bit data

[Instruction format] **AND1 dst, src** **AND1 dst, /src**

[Operation] **dst ← dst ∧ src** **dst ← dst ∧ $\overline{\text{src}}$**

[Operands]

Mnemonic	Operands (dst, src)
AND1	CY, saddr.bit
	CY, /saddr.bit
	CY, sfr.bit
	CY, /sfr.bit
	CY, A.bit
	CY, /A.bit

Mnemonic	Operands (dst, src)
AND1	CY, X.bit
	CY, /X.bit
	CY, PSWH.bit
	CY, /PSWH.bit
	CY, PSWL.bit
	CY, /PSWL.bit

saddr=FE20H-FF1FH
bit=0-7

[Flags]

S	Z	AC	P/V	CY
				×

[Explanation]

- AND1 ANDs the destination operand (dst) specified in the first operand and the bit data of the source operand (src) specified in the second operand and stores the result in the destination operand (dst).
- If "/" is prefixed to the second operand, the source operand (src) is negated and the destination operand (dst) is ANDed with the result.
- The operation result is stored in the CY flag because it is the destination operand (dst).

[Description example]

AND1 CY, SADR.3 ; AND bit 3 of SADR address that can be accessed by short direct addressing and CY flag and store the result in CY flag.

AND1 CY, /PSW.6 ; Negate bit 6 of PSW (Z flag) and AND CY flag with the result and store the result in CY flag.

OR1

Or Single Bit

OR 1-bit data

[Instruction format] **OR1 dst, src** **OR1 dst, /src**

[Operation] **dst ← dst ∨ src** **dst ← dst ∨ $\overline{\text{src}}$**

[Operands]

Mnemonic	Operands (dst, src)
OR1	CY, saddr.bit
	CY, /saddr.bit
	CY, sfr.bit
	CY, /sfr.bit
	CY, A.bit
	CY, /A.bit

Mnemonic	Operands (dst, src)
OR1	CY, X.bit
	CY, /X.bit
	CY, PSWH.bit
	CY, /PSWH.bit
	CY, PSWL.bit
	CY, /PSWL.bit

saddr=FE20H-FF1FH
bit=0-7

[Flags]

S	Z	AC	P/V	CY
				×

[Explanation]

- OR1 ORs the destination operand (dst) specified in the first operand and the bit data of the source operand (src) specified in the second operand and stores the result in the destination operand (dst).
- If "/" is prefixed to the second operand, the source operand (src) is negated and the destination operand (dst) is ORed with the result.
- The operation result is stored in the CY flag because it is the destination operand (dst).

[Description example]

OR1 CY, A.5 ; OR bit 5 of A register and CY flag and store the result in CY flag.
OR1 CY, /X.0 ; Negate bit 0 of X register and OR CY flag with the result and store the result in CY flag.

XOR1

Exclusive Or Single Bit

Exclusive-OR 1-bit data

[Instruction format] **XOR1 dst, src**

[Operation] **dst ← dst ∀ src**

[Operands]

Mnemonic	Operands (dst, src)
XOR1	CY, saddr.bit
	CY, sfr.bit
	CY, A.bit
	CY, X.bit
	CY, PSWH.bit
	CY, PSWL.bit

saddr=FE20H-FF1FH
bit=0-7

[Flags]

S	Z	AC	P/V	CY
				×

[Explanation]

- XOR1 exclusive-ORs the destination operand (dst) specified in the first operand and the bit data of the source operand (src) specified in the second operand and stores the result in the destination operand (dst).
- The operation result is stored in the CY flag because it is the destination operand (dst).

[Description example]

XOR1 CY, A.7 ; Exclusive-OR bit 7 of A register and CY flag and store the result in CY flag.

SET1

Set Single Bit (Carry Flag)

Set 1-bit data

[Instruction format] **SET1 dst**

[Operation] **dst ← 1**

[Operands]

Mnemonic	Operands (dst, src)
SET1	saddr.bit
	sfr.bit
	A.bit
	X.bit
	PSWH.bit
	PSWL.bit
	CY

saddr=FE20H-FF1FH
bit=0-7

[Flags]

When dst is PSWL.bit

S	Z	AC	P/V	CY
×	×	×	×	×

When dst is CY

S	Z	AC	P/V	CY
				1

Other than above

S	Z	AC	P/V	CY

[Explanation]

- SET1 sets the destination operand (dst) to 1.
- If the destination operand (dst) is CY or PSWL.bit, only the corresponding flags are set to 1.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) or PSW and its following instruction.

[Description example]

SET1 BITSYM ; Set contents of bit allocated in area that can be accessed by short direct addressing to 1

CLR1

Clear Single Bit (Carry Flag)

Clear 1-bit data

[Instruction format] **CLR1 dst**

[Operation] **dst ← 0**

[Operands]

Mnemonic	Operands (dst, src)
CLR1	saddr.bit
	sfr.bit
	A.bit
	X.bit
	PSWH.bit
	PSWL.bit
	CY

saddr=FE20H-FF1FH
bit=0-7

[Flags]

When dst is PSWL.bit

S	Z	AC	P/V	CY
×	×	×	×	×

When dst is CY

S	Z	AC	P/V	CY
				0

Other than above

S	Z	AC	P/V	CY

[Explanation]

- CLR1 clears the destination operand (dst) to 0.
- If the destination operand (dst) is CY or PSWL.bit, only the corresponding flags are cleared to 0.

- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) or PSW and its following instruction.

[Description example]

CLR1 A.7 ; Clear bit 7 of A register to 0

NOT1

Not Single Bit (Carry Flag)

Negate 1-bit data

[Instruction format] **NOT1 dst**

[Operation] **dst ← $\overline{\text{dst}}$**

[Operands]

Mnemonic	Operands (dst, src)
NOT1	saddr.bit
	sfr.bit
	A.bit
	X.bit
	PSWH.bit
	PSWL.bit
	CY

saddr=FE20H-FF1FH
bit=0-7

[Flags]

When dst is PSWL.bit

S	Z	AC	P/V	CY
×	×	×	×	×

When dst is CY

S	Z	AC	P/V	CY
				×

Other than above

S	Z	AC	P/V	CY

[Explanation]

- NOT1 negates the bit specified in the destination operand (dst) and stores the result in the destination operand (dst).
- If the destination operand (dst) is CY or PSWL.bit, only the corresponding flags change.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) or PSW and its following instruction.

[Description example]

NOT1 A.2 ; Invert bit 2 of A register

6.16 Call and Return Instructions

The following call and return instructions can be used:

CALL ... 237
CALLF ... 238
CALLT ... 239
BRK ... 240
RET ... 241
RETB ... 242
RETI ... 243

CALL

Call

Call subroutine (16-bit direct or register indirect specification)

[Instruction format] **CALL target**

[Operation] **(SP–1) ← (PC+n)$_H$,**
(SP–2) ← (PC+n)$_L$,
SP ← SP–2,
PC ← target
n: Number of bytes of instruction

[Operands]

Mnemonic	Operand (target)
CALL	!addr16
	rp1
	[rp1]

addr16=0000H-FDFFH

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- CALL calls the subroutine at the location addressed by the specified 16-bit absolute address or register direct or indirect addressing.
- The top address of the next instruction (PC+n) is saved in a given stack and branches to the address specified in the target operand (target).

Caution Instructions cannot be fetched from addresses FE00H-FFFFH. Do not describe any of the addresses in addr16.

[Description example]

CALL !3059H ; Call subroutine at 3059H

CALLF

Call Flag

Call subroutine (11-bit direct specification)

[Instruction format] **CALLF target**

[Operation] **(SP–1) ← (PC+2)H,**
(SP–2) ← (PC+2)L,
SP ← SP–2,
PC ← target

[Operands]

Mnemonic	Operand (target)
CALLF	!addr11

addr11=0800H-0FFFH

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- CALLF can call subroutines only at addresses 0800H-0FFFH.
- The top address of the next instruction (PC+2) is saved in a given stack and branches to the address specified in the target operand (target) in the range of 0800H to 0FFFH.
- Specify only the low-order 11 bits of the address. (The high-order five bits are fixed to 00001B.)
- If subroutines are placed at addresses 0800H-0FFFH and the CALLF instruction is used, the program size can be compressed.

[Description example]

CALLF !0C2AH ; Call subroutine at 0C2AH

CALLT

Call Table

Call subroutine (reference call table)

[Instruction format] **CALLT [addr5]**

[Operation]

(SP–1) ← (PC+1)H,
(SP–2) ← (PC+1)L,
SP ← SP–2,
PCH ← (TPF, 000000001, addr5+1)
PCL ← (TPF, 000000001, addr5)

[Operands]

Mnemonic	Operand ([addr5])
CALLT	[addr5]

addr5=40H-7EH

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- CALLT calls a given subroutine by referencing the call table.
- The top address of the next instruction (PC+1) is saved in a given stack and branches to the address indicated by the word data in the specified call table entry. (Bits 14-6 of the call table address are fixed to 000000001B and the least significant bit is fixed to 0. Specify bits 5-1 in addr5.)
- The branch destination address table can be placed in the external memory area (8040H-807EH) by setting the TPF flag to 1.

[Description example]

CALLT [60H] ; Call subroutine at address indicated by word data at addresses 0060H and 0061H

BRK

Break

Software vectored interrupt

[Instruction format] **BRK**

[Operation]

(SP–1) ← PSW$_H$
(SP–2) ← PSW$_L$
(SP–3) ← (SP+1)$_H$
(SP–4) ← (SP+1)$_L$
IE ← 0,
SP ← SP–4,
PC$_H$ ← (003FH),
PC$_L$ ← (003EH)

[Operands]

None

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- BRK is a software interrupt instruction.
- PSW and the address of the next instruction (PC+1) are saved in a given stack, then the IE flag is cleared to 0 and branches to the address indicated by the word data at the vector address (003EH). As the IE flag is cleared to 0, maskable vector interrupts can not be made afterward.
- To return from the software vector interrupt caused by the BRK instruction, use a RETB instruction.

[Description example]

BRK

RET

Return

Return from subroutine

[Instruction format] **RET**

[Operation] **$PC_L \leftarrow (SP)$,**
$PC_H \leftarrow (SP+1)$,
$SP \leftarrow SP+2$

[Operands]

None

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- RET is a return instruction from the subroutine called by the CALL, CALLF, or CALLT instruction.
- The data saved in the stack is restored in the PC and control is returned from the subroutine.

RETB

Return from Break

Return from software vectored interrupt

[Instruction format] **RETB**

[Operation]

$PC_L \leftarrow (SP)$,
$PC_H \leftarrow (SP+1)$,
$PSW_L \leftarrow (SP+2)$,
$PSW_H \leftarrow (SP+3)$,
$SP \leftarrow SP+4$

[Operands]

None

[Flags]

S	Z	AC	P/V	CY
R	R	R	R	R

[Explanation]

- RETB is a return instruction from the software interrupt caused by a BRK instruction or an OPE code trap.
- The PC and PSW saved in the stack are restored and control is returned from the interrupt service routine.
- The RETB instruction cannot be used to return from a BRKCS instruction or a hardware interrupt.

Caution To return from the interrupt service routine accompanying the BRK instruction or OPE code trap, be sure to use the RETB instruction. If the RETI instruction is used, the interrupt control circuit does not operate normally.

RETI

Return from Interrupt

Return from hardware vectored interrupt

[Instruction format] **RETI**

[Operation]

$PC_L \leftarrow (SP)$,
$PC_H \leftarrow (SP+1)$,
$PSW_L \leftarrow (SP+2)$,
$PSW_H \leftarrow (SP+3)$,
$SP \leftarrow SP+4$
Clear the MSB of bits set to 1 in ISPR to 0

[Operands]

None

[Flags]

S	Z	AC	P/V	CY
R	R	R	R	R

[Explanation]

- RETI is a return instruction from a vectored interrupt.
- The data saved in the stack is restored in the PC and PSW and the most significant flag bit of the flags set to 1 in the ISPR register is cleared to 0, then control is returned from the interrupt service routine.
- The RETI instruction cannot be used to return from a software interrupt caused by a BRK or BRKCS instruction or an OPE code trap or an interrupt using context switching.

Caution To return from the interrupt service routine accompanying the BRK instruction or OPE code trap, do not use the RETI instruction.

6.17 Stack Handling Instructions

The following stack handling instructions can be used:

PUSH ... 216
PUSHU ... 217
POP ... 218
POPU ... 219
MOVW SP, src ... 220
MOVW AX, SP ... 220
INCW SP ... 221
DECW SP ... 222

PUSH

Push

Push

[Instruction format] **PUSH src**

[Operation]

When src is sfrp or PSW

(SP–1) ← src$_H$,

(SP–2) ← src$_L$,

SP ← SP–2

When src is post

{(SP–1) ← post$_H$, (SP–2) ← post$_L$, SP ← SP–2} × n

(n is the number of register pairs described as post)

[Operands]

Mnemonic	Operand (src)
PUSH	sfrp
	PSW
	post

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- PUSH saves the data in the register specified in the source operand (src) in a given stack.
- If post is specified as the source operand, any combination of the following registers can be saved in the stack by the instruction:

  RP0, RP1, RP2, RP3, RP4, RP5, RP6, RP7

  The saving is performed from the right to left registers in order.
- After the data in the register specified in the source operand (src), the stack pointer (SP) is decremented by the number of bytes of the saved data.

[Description example]

PUSH AX, BC, RP2, RP3 ; Save the contents of AX, BC, RP2, and RP3 registers in stack

PUSHU

Push to User Stack

Push register to user stack

[Instruction format] **PUSHU src**

[Operation] **{(UP–1) ← post$_H$, (UP–2) ← post$_L$, UP ← UP–2} × n**
(n is the number of register pairs described as post)

[Operands]

Mnemonic	Operand (src)
PUSHU	post

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- PUSHU saves the contents of the 16-bit register pair specified in the source operand (src) in the memory addressed by the user stack pointer (UP), then decrements the UP.
- Any desired combination of the following registers can be described in post as the source operand:
  RP0, RP1, RP2, RP3, RP4, RP5, RP6, RP7
  The saving is performed from the right to left registers in order.

[Description example]

PUSHU BC, PSW ; Save the contents of BC register and PSW in stack

POP

Pop

Pop

[Instruction format] **POP dst**

[Operation] When dst is sfrp or PSW

$dst_L \leftarrow (SP)$,
$dst_H \leftarrow (SP+1)$,
$SP \leftarrow SP+2$

When dst is post

$\{post_L \leftarrow (SP), post_H \leftarrow (SP+1), SP \leftarrow SP+2\} \times n$
(n is the number of register pairs described as post)

[Operands]

Mnemonic	Operand (dst)
POP	sfrp
	PSW
	post

[Flags]

When dst is PSW

S	Z	AC	P/V	CY
R	R	R	R	R

Others

S	Z	AC	P/V	CY

[Explanation]

- POP restores the data saved in the stack in the register specified in the destination operand (dst).
- If PSW is specified in the destination operand (dst), the flag contents are replaced with the data saved in the stack.
- If post is specified as the destination operand (dst), the data saved in the stack can be restored in any combination of the following registers by one POP instruction:
  RP0, RP1, RP2, RP3, RP4, RP5, RP6, RP7
  The restoring is performed from the left to right registers in order.
- After the data saved in the stack is restored in the register specified in the destination operand (dst), the stack pointer (SP) is incremented by the number of bytes of the restored data.
- No interrupts are acknowledged between a write access instruction to an interrupt function control register (see **3.4.6 Control registers**) or PSW and its following instruction.

[Description example]

POP AX ; Restore data saved in stack in AX register

POPU

Pop from User Stack

Pop register from user stack

[Instruction format] **POPU dst**

[Operation] **{post$_L$ ← (UP), post$_H$ ← (UP+1), UP ← UP+2} × n**
(n is the number of register pairs described as post)

[Operands]

Mnemonic	Operand (dst)
POPU	post

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- POPU restores the contents of the memory (stack) addressed by the user stack pointer (UP) in the register pair specified in the destination operand (dst), then increments the UP.
- Any desired combination of the following registers can be described in post as the destination operand (dst):
  RP0, RP1, RP2, RP3, RP4, RP5, RP6, RP7
  The restoring is performed from the left to right registers in order.

[Description example]

POPU AX, BC ; Restore data saved in stack in AX and BC registers

MOVW SP, src
MOVW AX, SP

Move Word

Transfer data to and from stack pointer

[Instruction format] **MOVW dst, src**

[Operation] **dst ← src**

[Operands]

Mnemonic	Operands (dst, src)
MOVW	SP, #word
	SP, AX
	AX, SP

word = 0000H-FDFFH (any addresses in the range can be specified)
word = FE00H-FEFFH (limited to even addresses)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- MOVW is an instruction for handling the contents of the stack pointer (SP).
- MOVW stores the source operand (src) specified in the second operand in the destination operand (dst) specified in the first operand.
- To store any of addresses FE00H-FEFFH in the SP, only even address in the range can be specified.

[Description example]

MOVW SP, #0FE1EH ; Store FE1EH in stack pointer

INCW SP

Increment Word

Increment stack pointer

[Instruction format] **INCW SP**

[Operation] **SP ← SP+1**

[Operands] None

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- INCW is an instruction for adding one to the SP (stack pointer) contents.
- If an interrupt is acknowledged when the address stored in the SP is an odd address in the range of FE00H-FEFFH, an error may be caused. Therefore, be sure to store an even address in the SP.

[Description example]

```
INCW SP
```

DECW SP

Decrement Word

Decrement stack pointer

[Instruction format] **DECW SP**

[Operation] **SP ← SP–1**

[Operands] None

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- DECW is an instruction for subtracting one from the SP (stack pointer) contents.
- If an interrupt is acknowledged when the address stored in the SP is an odd address in the range of FE00H-FEFFH, an error may be caused. Therefore, be sure to store an even address in the SP.

[Description example]

DECW SP

6.18 Special Instructions

The following special instructions can be used:

CHKL ... 223
CHKLA ... 225

CHKL

Check Level

Check pin for ouput level

[Instruction format] **CHKL sfr**

[Operation] **(pin level) ⩛ (output latch)**

[Operands]

Mnemonic	Operand
CHKL	sfr

[Flags]

S	Z	AC	P/V	CY
×	×		P	

[Explanation]

- CHKL exclusive-ORs the pin level of the specified output pin and the signal level at the preceding stage of the output buffer.
- If bit 7 is set to 1 as a result of the exclusive-ORing, the S flag is set to 1; if bit 7 is cleared to 0, the S flag is cleared to 0.
- If all bits are 0 as a result of the exclusive-ORing, the Z flag is set to 1; if not all bits are 0, the Z flag is cleared to 0.
- If the number of bits set to 1 in the data is even as a result of the exclusive-ORing, the P/V flag is set to 1; if odd, the P/V flag is cleared to 0.
- The CHKL instruction is an instruction for detecting an abnormal condition caused for some reason in which the pin level of the output pin differs from the signal level at the preceding stage of the output buffer. At the normal operation, the Z flag is always set to 1.
- To execute the CHKL instruction, the PRDC0 bit of the port read control register (PRDC) must be cleared to 0. If the PRDC0 bit is set to 1, no abnormal condition can be detected.
- To execute this instruction for a port containing a pin used as control output, be sure to specify the input mode as the input/output mode as a port for the pin used as control output. If the input/output mode as a port for the pin used as control output is set to the output mode, the normal operation may be erroneously judged abnormal condition.
- The operation of the pin for which the input/output mode as a port is specified as the input mode is always judged normal by this instruction.

[Description example]

```
CHKL, P0
BNZ $ERROR ; Check whether or not the pin level of port 0 matches the signal level at the preceding stage
             of the output buffer. If they do not match, branch to address ERROR.
```

CHKLA

Check Level and Transfer to Register

Check pin for ouput level and transfer result to register

[Instruction format] **CHKLA sfr**

[Operation] A ← (pin level)∀ (output latch)

[Operands]

Mnemonic	Operand
CHKLA	sfr

[Flags]

S	Z	AC	P/V	CY
×	×		P	

[Explanation]

- CHKLA exclusive-ORs the pin level of the specified output pin and the signal level at the preceding stage of the output buffer and stores the result in the A register.
- If bit 7 is set to 1 as a result of the exclusive-ORing, the S flag is set to 1; if bit 7 is cleared to 0, the S flag is cleared to 0.
- If all bits are 0 as a result of the exclusive-ORing, the Z flag is set to 1; if not all bits are 0, the Z flag is cleared to 0.
- If the number of bits set to 1 in the data is even as a result of the exclusive-ORing, the P/V flag is set to 1; if odd, the P/V flag is cleared to 0.
- The CHKLA instruction is an instruction for detecting an abnormal condition caused for some reason in which the pin level of the output pin differs from the signal level at the preceding stage of the output buffer. At the normal operation, the Z flag is always set to 1.
- To execute this CHKLA instruction, the PRDC0 bit of the port read control register (PRDC) must be cleared to 0. If the PRDC0 bit is set to 1, no abnormal condition can be detected.
- To execute this instruction for a port containing a pin used as control output, be sure to specify the input mode as the input/output mode as a port for the pin used as control output. If the input/output mode as a port for the pin used as control output is set to the output mode, the normal operation may be erroneously judged abnormal condition.
- The operation of the pin for which the input/output mode as a port is specified as the input mode is always judged normal by this instruction.

[Description example]

CHKLA, P0 ; Check whether or not the pin level of port 0 matches the signal level at the preceding stage of the output buffer and store the result in A register.

6.19 Unconditional Branch Instruction

The following unconditional branch instruction can be used:

BR ... 227

BR

Branch

Unconditional branch

[Instruction format] **BR target**

[Operation] **PC ← target**

[Operands]

Mnemonic	Operand (target)
BR	!addr16
	rp1
	[rp1]
	$addr16

When target is !addr16, addr16=0000H-FDFFH
When target is $addr16, addr16=(PC–126)-(PC+129)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- BR is an instruction for causing control or program flow to branch unconditionally.
- BR transfers the data in the target address operand (target) to the PC for branching.

Caution Instructions cannot be fetched from addresses FE00H-FFFFH. Do not describe any address in the range in addr16 or rp1 or the memory addressed by rp1.

[Description example]

BR DE ; Branch to the address indicated by the DE register contents.

6.20 Conditional Branch Instructions

The following conditional branch instructions can be used:

BC ... 229
BL ... 229
BNC ... 230
BNL ... 230
BZ ... 231
BE ... 231
BNZ ... 232
BNE ... 232
BV ... 233
BPE ... 233
BNV ... 234
BPO ... 234
BN ... 235
BP ... 236
BGT ... 237
BGE ... 238
BLT ... 239
BLE ... 240
BH ... 241
BNH ... 242
BT ... 243
BF ... 244
BTCLR ... 245
BFSET ... 246
DBNZ ... 247

BC
BL

Branch if Carry/Less than

Conditional branch according to carry flag (CY=1)

[Instruction format] **BC $addr16**
BL $addr16

[Operation] **PC ← PC+2+jdisp8 if CY=1**

[Operands]

Mnemonic	Operand ($addr16)
BC	$addr16
BL	

addr16=(PC–126)-(PC+129)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- When CY=1, a branch is taken to the address specified in the operand.
  When CY=0, no operation is performed and the next instruction is executed.
- The BC and BL instructions are the same in operation.
  The instructions are used as follows:
  - BC instruction: To check whether or not a carry occurs after execution of an addition instruction or a shift or rotate instruction.
    To determine the bit manipulation result.
  - BL instruction: To check whether or not a borrow occurs after execution of a subtraction instruction.
    To check to see if the first operand of comparison instruction is less than the second operand after execution of comparison instruction for unsigned data.

[Description example]

BC $300H ; Branch to 0300H if CY=1.
However, assume that the top address of the BC instruction is at addresses 027FH-037EH.

BNC
BNL

Branch if Not Carry/Less than

Conditional branch according to carry flag (CY=0)

[Instruction format] **BNC $addr16**
BNL $addr16

[Operation] **PC ← PC+2+jdisp8 if CY=0**

[Operands]

Mnemonic	Operand ($addr16)
BNC	$addr16
BNL	

addr16=(PC–126)-(PC+129)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- When CY=0, a branch is taken to the address specified in the operand.
  When CY=1, no operation is performed and the next instruction is executed.
- The BNC and BNL instructions are the same in operation. The instructions are used as follows:
  - BNC instruction: To check whether or not a carry occurs after execution of an addition instruction or a shift or rotate instruction.
    To determine the bit manipulation result.
  - BNL instruction: To check whether or not a borrow occurs after execution of a subtraction instruction.
    To check to see if the first operand of comparison instruction is not less than the second operand after execution of comparison instruction for unsigned data.

[Description example]

CMP A, B ; Compare A register contents with B register contents in size
BNL $1500H ; Branch to 1500H if the A register contents are not less than the B register contents.
However, assume that the top address of the BNL instruction is at addresses 147FH-157EH.

BZ
BE

Branch if Zero/Equal

Conditional branch according to zero flag (Z=1)

[Instruction format] **BZ $addr16**
BE $addr16

[Operation] **PC ← PC+2+jdisp8 if Z=1**

[Operands]

Mnemonic	Operand ($addr16)
BZ	$addr16
BE	

addr16=(PC–126)-(PC+129)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- When Z=1, a branch is taken to the address specified in the operand.
  When Z=0, no operation is performed and the next instruction is executed.
- The BZ and BE instructions are the same in operation.
  The instructions are used as follows:
  - BZ instruction : To check whether or not the execution result of an addition, subtraction, increment or decrement, 8-bit logical operation, or shift or rotate instruction is 0.
  - BE instruction : To check to see if a match is found after execution of a comparison instruction.
- When data in two's complement format is added, if –80H and –80H are added at 8-bit data addition or if –8000H and –8000H are added at 16-bit data addition, Z is set to 1. To determine whether or not it is 0 from the addition result of data in two's complement format, previously check the overflow flag for overflow.

[Description example]

```
DEC B
BZ $3C5H ; Branch to 03C5H if B register contains 0.
           However, assume that the top address of the BZ instruction is at addresses 0344H-0443H.
```

BNZ
BNE

Branch if Not Zero/Not Equal

Conditional branch according to zero flag (Z=0)

[Instruction format] **BNZ $addr16**
BNE $addr16

[Operation] **PC ← PC+2+jdisp8 if Z=0**

[Operands]

Mnemonic	Operand ($addr16)
BNZ	$addr16
BNE	

addr16=(PC–126)-(PC+129)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- When Z=0, a branch is taken to the address specified in the operand.
  When Z=1, no operation is performed and the next instruction is executed.
- The BNZ and BNE instructions are the same in operation. The instructions are used as follows:
  - BNZ instruction : To check whether or not the execution result of an addition, subtraction, increment or decrement, 8-bit logical operation, or shift or rotate instruction is 0.
  - BNE instruction : To check to see if a match is found after execution of a comparison instruction.
- When data in two's complement format is added, if –80H and –80H are added at 8-bit data addition or if –8000H and –8000H are added at 16-bit data addition, Z is set to 1. To determine whether or not it is 0 from the addition result of data in two's complement format, previously check the overflow flag for overflow.

[Description example]

```
CMP A, #55H
BNE $0A39H ; Branch to 0A39H if A register does not contain 055H.
             However, assume that the top address of the BNE instruction is at addresses 09B8H-0AB7H.
```

BV
BPE

Branch if Overflow/Branch if Parity Even

Conditional branch according to parity/overflow flag (P/V=1)

[Instruction format] **BV $addr16**
BPE $addr16

[Operation] **PC ← PC+2+jdisp8 if P/V=1**

[Operands]

Mnemonic	Operand ($addr16)
BV	$addr16
BPE	

addr16=(PC–126)-(PC+129)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- When P/V=1, a branch is taken to the address specified in the operand.
  When P/V=0, no operation is performed and the next instruction is executed.
- The BV and BPE instructions are the same in operation.
  The instructions are used as follows:
  - BV instruction : To check that the result overflows or underflows after operation on data in two's complement format.
  - BPE instruction : To check that parity of the execution result of an instruction such as a logical operation instruction or a shift or rotate instruction is even.

[Description example]

OR A, #055H ; OR A register contents and 055H for each bit
BPE $841EH ; Branch to 841EH if parity is even as a result of the ORing.
However, assume that the top address of the instruction is at addresses 839DH-849CH.

BNV
BPO

Branch if No Overflow/Branch if Parity Odd

Conditional branch according to parity/overflow flag (P/V=0)

[Instruction format] **BNV $addr16**
BPO $addr16

[Operation] **PC ← PC+2+jdisp8 if P/V=0**

[Operands]

Mnemonic	Operand ($addr16)
BNV	$addr16
BPO	

addr16=(PC–126)-(PC+129)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- When P/V=0, a branch is taken to the address specified in the operand.
  When P/V=1, no operation is performed and the next instruction is executed.
- The BNV and BPO instructions are the same in operation. The instructions are used as follows:
  - BNV instruction : To check that the result does not overflow or underflow after operation on data in two's complement format.
  - BPO instruction : To check that parity of the execution result of an instruction such as a logical operation instruction or a shift or rotate instruction is odd.

[Description example]

ADD B, C ; Add B and C register contents (data in two's complement format)
BNV $560H ; Branch to 560H if no overflow occurs in the addition result.
However, assume that the top address of the BNV instruction is at addresses 04DFH-05DEH.

BN

Branch if Negative

Conditional branch according to sign flag (S=1)

[Instruction format] **BN $addr16**

[Operation] **PC ← PC+2+jdisp8 if S=1**

[Operands]

Mnemonic	Operand ($addr16)
BN	$addr16

addr16=(PC–126)-(PC+129)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- When S=1, a branch is taken to the address specified in the operand.
  When S=0, no operation is performed and the next instruction is executed.
- The BN instruction is used to check that the result is negative after operation on data in two's complement format. However, if the operation result overflows or underflows, normal determination cannot be made. (Before using the BN instruction, execute a BV or BNV instruction to check that no overflow or underflow occurs, or use a BLT instruction.)

[Description example]

BN $TARGET ; Branch to address TARGET if the operation result is negative

BP

Branch if Positive

Conditional branch according to sign flag (S=0)

[Instruction format] **BP $addr16**

[Operation] **PC ← PC+2+jdisp8 if S=0**

[Operands]

Mnemonic	Operand ($addr16)
BP	$addr16

addr16=(PC–126)-(PC+129)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- When S=0, a branch is taken to the address specified in the operand.
  When S=1, no operation is performed and the next instruction is executed.
- The BP instruction is used to check that the result is positive (containing 0) after operation on data in two's complement format. However, if the operation result overflows or underflows, normal determination cannot be made. (Before using the BP instruction, execute a BV or BNV instruction to check that no overflow or underflow occurs, or use a BGE instruction.)

[Description example]

BV $OVER ; Branch to address OVER if the operation result overflows or underflows.
BP $TARGET ; Branch to address TARGET if the operation result is positive (containing 0).
However, assume that the TARGET address is in the range of –126 to +129 addresses of the top address of the BP instruction.

BGT

Branch if Greater than/Equal

Conditional branch according to numeric values (greater than)

[Instruction format] **BGT $addr16**

[Operation] **PC ← PC+3+jdisp8 if (P/V ⊻ S) ∨ Z=0**

[Operands]

Mnemonic	Operand ($addr16)
BGT	$addr16

addr16=(PC–125)-(PC+130)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- When (P/V ⊻ S) ∨ Z=0, a branch is taken to the address specified in the operand.
  When (P/V ⊻ S) ∨ Z=1, no operation is performed and the next instruction is executed.
- This instruction is used to determine the numeric value size relationship between data in two's complement format or check that the operation result is greater than 0. To determine the relationship, a check is made to see if the first operand of the CMP instruction executed immediately preceding the BGT instruction is greater than the second operand. The BGT instruction is also used to check that the operation result is greater than 0 containing an overflow.

[Description example]

```
CMP A, B
BGT $2FEDH ; Branch to address 2FEDH if A register contents are greater than B register contents.
             However, assume that the top address of the BGT instruction is at addresses 2F6BH-306DH.
```

BGE

Branch if Greater than/Equal

Conditional branch according to numeric values (greater than or equal to)

[Instruction format] **BGE $addr16**

[Operation] **PC ← PC+3+jdisp8 if P/V ⊻ S=0**

[Operands]

Mnemonic	Operand ($addr16)
BGE	$addr16

addr16=(PC–125)-(PC+130)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- When P/V ⊻ S=0, a branch is taken to the address specified in the operand.
  When P/V ⊻ S=1, no operation is performed and the next instruction is executed.
- This instruction is used to determine the numeric value size relationship between data in two's complement format or check that the operation result is 0 or positive. To determine the relationship, a check is made to see if the first operand of the CMP instruction executed immediately preceding the BGE instruction is equal to or greater than the second operand. The BGE instruction is also used to check that the operation result is equal to or greater than 0 containing an overflow.

[Description example]

```
ADDW AX, BC
BGE $3456H ; Branch to address 3456H if the execution result of the immediately preceding addition
             instruction is 0 or more.
             However, assume that the top address of the BGE instruction is at addresses 33D4H-34D3H.
```

BLT

Branch if Less than

Conditional branch according to numeric values (less than)

[Instruction format] **BLT $addr16**

[Operation] **PC ← PC+3+jdisp8 if P/V ∀ S=1**

[Operands]

Mnemonic	Operand ($addr16)
BLT	$addr16

addr16=(PC–125)-(PC+130)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- When P/V ∀ S=1, a branch is taken to the address specified in the operand.
  When P/V ∀ S=0, no operation is performed and the next instruction is executed.
- This instruction is used to determine the numeric value size relationship between data in two's complement format or check that the operation result is negative.
  To determine the relationship, a check is made to see if the first operand of the CMP instruction executed immediately preceding the BLT instruction is less than the second operand. The BLT instruction is also used to check that the operation result is negative containing an underflow.

[Description example]

CMPW AX, #3456H
BLT $8123H ; Branch to address 8123H if AX register contents are less than 3456H.
However, assume that the top address of the BLT instruction is at addresses 80A1H-81A0H.

BLE

Branch if Less than/Equal

Conditional branch according to numeric values (less than or equal to)

[Instruction format] **BLE $addr16**

[Operation] **PC ← PC+3+jdisp8 if (P/V ⊻ S) ∨ Z=1**

[Operands]

Mnemonic	Operand ($addr16)
BLE	$addr16

addr16=(PC–125)-(PC+130)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- When (P/V ⊻ S) ∨ Z=1, a branch is taken to the address specified in the operand.
  When (P/V ⊻ S) ∨ Z=0, no operation is performed and the next instruction is executed.
- This instruction is used to determine the numeric value size relationship between data in two's complement format or check that the operation result is negative containing 0. To determine the relationship, a check is made to see if the first operand of the CMP instruction executed immediately preceding the BLE instruction is equal to or less than the second operand. The BLE instruction is also used to check that the operation result is negative containing an underflow.

[Description example]

```
SUB H, A
BLE $89ABH ; Branch to address 89ABH if the execution result of the immediately preceding subtraction
             instruction is 0 or less containing an underflow.
             However, assume that the top address of the BLE instruction is at addresses 8929H-89ABH.
```

BH

Branch if Higher than

Conditional branch according to numeric values (greater than)

[Instruction format] **BH $addr16**

[Operation] **PC ← PC+3+jdisp8 if Z ∨ CY=0**

[Operands]

Mnemonic	Operand ($addr16)
BH	$addr16

addr16=(PC–125)-(PC+130)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- When Z ∨ CY=0, a branch is taken to the address specified in the operand.
  When Z ∨ CY=1, no operation is performed and the next instruction is executed.
- This instruction is used to determine the numeric value size relationship between unsigned data. A check is made to see if the first operand of the CMP instruction executed immediately preceding the BH instruction is greater than the second operand.

[Description example]

CMP B, C

BH $356H ; Branch to address 356H if B register contents are greater than C register contents.
However, assume that the top address of the BH instruction is at addresses 2D4H-3D3H.

BNH

Branch if Not Higher than

Conditional branch according to numeric values (not greater than)

[Instruction format] **BNH $addr16**

[Operation] **PC ← PC+3+jdisp8 if Z ∨ CY=1**

[Operands]

Mnemonic	Operand ($addr16)
BNH	$addr16

addr16=(PC–125)-(PC+130)

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- When Z∨CY=1, a branch is taken to the address specified in the operand.
  When Z∨CY=0, no operation is performed and the next instruction is executed.
- This instruction is used to determine the numeric value size relationship between unsigned data. A check is made to see if the first operand of the CMP instruction executed immediately preceding the BNH instruction is not greater than (equal to or less than) the second operand.

[Description example]

```
CMPW AX, #8921H
BNH $TARGET ; Branch to address TARGET if AX register contents are not greater than (equal to or less than)
              8921H.
              However, assume that the top address of the BNH instruction is at address that can branch
              to the TARGET address.
```

BT

Branch if True

Conditional branch according to bit test (byte data bit=1)

[Instruction format] **BT bit, $addr16**

[Operation] **PC ← PC+b+jdisp8 if bit=1**

[Operands]

Mnemonic	Operands (bit, $addr16)	b (No. of bytes)
BT	saddr. bit, $addr16	3
	sfr. bit, $addr16	4
	A. bit, $addr16	3
	X. bit, $addr16	3
	PSWH. bit, $addr16	3
	PSWL. bit, $addr16	3

When b=3, addr16=(PC–125)-(PC+130)
When b=4, addr16=(PC–124)-(PC+131)
saddr=FE20H-FF1FH
bit=0-7

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- When the contents of the first operand (bit) are set to 1, a branch is taken to the address specified in the second operand ($addr16).
  When the contents of the first operand (bit) are not set to 1, no operation is performed and the next instruction is executed.

[Description example]

BT 0FE47H.3, $55CH ; Branch to address 055CH if bit 3 of address FE47H is 1.
However, assume that the top address of the BT instruction is at addresses 04D9H-05D8H.

BF

Branch if False

Conditional branch according to bit test (byte data bit=0)

[Instruction format] **BF bit, $addr16**

[Operation] **PC ← PC+b+jdisp8 if bit=0**

[Operands]

Mnemonic	Operands (bit, $addr16)	b (No. of bytes)
BF	saddr. bit, $addr16	4
	sfr. bit, $addr16	4
	A. bit, $addr16	3
	X. bit, $addr16	3
	PSWH. bit, $addr16	3
	PSWL. bit, $addr16	3

When b=3, addr16=(PC–125)-(PC+130)
When b=4, addr16=(PC–124)-(PC+131)
saddr=FE20H-FF1FH
bit=0-7

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- When the contents of the first operand (bit) are cleared to 0, a branch is taken to the address specified in the second operand ($addr16).
  When the contents of the first operand (bit) are not cleared to 0, no operation is performed and the next instruction is executed.

[Description example]

BF A.2, $1549H ; Branch to address 1549H if bit 2 of A register is 0.
However, assume that the top address of the BF instruction is at addresses 14C6H-15C5H.

BTCLR

Branch if True and Clear

Conditional branch and clear according to bit test (byte data bit=1)

[Instruction format] **BTCLR bit, $addr16**

[Operation] **PC ← PC+b+jdisp8 if bit=1, then bit ← 0**

[Operands]

Mnemonic	Operands (bit, $addr16)	b (No. of bytes)
BTCLR	saddr. bit, $addr16	4
	sfr. bit, $addr16	4
	A. bit, $addr16	3
	X. bit, $addr16	3
	PSWH. bit, $addr16	3
	PSWL. bit, $addr16	3

When b=3, addr16=(PC–125)-(PC+130)
When b=4, addr16=(PC–124)-(PC+131)
saddr=FE20H-FF1FH bit=0-7

[Flags]

When bit is PSWL.bit

S	Z	AC	P/V	CY
×	×	×	×	×

Other than left

S	Z	AC	P/V	CY

[Explanation]

- When the contents of the first operand (bit) are set to 1, the contents of the first operand (bit) are cleared to 0 and a branch is taken to the address specified in the second operand.
  When the contents of the first operand (bit) are not set to 1, no operation is performed and the next instruction is executed.
- If the first operand (bit) is PSWL.bit, the corresponding flag contents are cleared to 0.

[Description example]

BTCLR PSWL.0, $356H ; Clear CY flag to 0 and branch to address 0356H if bit 0 of PSWL (CY flag) is 1. However, assume that the top address of the BTCLR instruction is at addresses 02D4H-03D3H.

BFSET

Branch if False and Set

Conditional branch and set according to bit test (byte data bit=0)

[Instruction format] **BFSET bit, $addr16**

[Operation] **PC ← PC+b+jdisp8 if bit=0, then bit ← 1**

[Operands]

Mnemonic	Operands (bit, $addr16)	b (No. of bytes)
BFSET	saddr. bit, $addr16	4
	sfr. bit, $addr16	4
	A. bit, $addr16	3
	X. bit, $addr16	3
	PSWH. bit, $addr16	3
	PSWL. bit, $addr16	3

When b=3, addr16=(PC–125)-(PC+130)
When b=4, addr16=(PC–124)-(PC+131)
saddr=FE20H-FF1FH bit=0-7

[Flags]

When bit is PSWL.bit

S	Z	AC	P/V	CY
×	×	×	×	×

Other than left

S	Z	AC	P/V	CY

[Explanation]

- When the contents of the first operand (bit) are cleared to 0, the contents of the first operand (bit) are set to 1 and a branch is taken to the address specified in the second operand.
  When the contents of the first operand (bit) are set to 1, no operation is performed and the next instruction is executed.
- If the first operand (bit) is PSWL.bit, the corresponding flag contents are set to 1.

[Description example]

BFSET A.6 $3FE1H ; Set bit 6 of A register to 1 and branch to address 3FE1H if bit 6 of A register is 0. However, assume that the top address of the BFSET instruction is at addresses 3F5FH-405EH.

DBNZ

Decrement and Branch if Not Zero

Conditional loop (dst ≠ 0)

[Instruction format] **DBNZ dst, $addr16**

[Operation] **dst ← dst–1,**
then PC ← PC+b+jdisp8 if dst≠0

[Operands]

Mnemonic	Operands (dst, $addr16)	b (No. of bytes)
DBNZ	r2, $addr16	2
	saddr, $addr16	3

When b=2, addr16=(PC–126)-(PC+129)
When b=3, addr16=(PC–126)-(PC+130)
saddr=FE20H-FF1FH

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- DBNZ subtracts 1 from the contents of the destination operand (dst) specified in the first operand and stores the result in the destination operand (dst).
- If the result of subtracting 1 from the destination operand (dst) is not 0, a branch is taken to the address specified in the second operand.
  If the result of subtracting 1 from the destination operand (dst) is 0, no operation is performed and the next instruction is executed.
- The flags do not change.

[Description example]

DBNZ B, $1215H ; Decrement B register contents and if the result is not 0, branch to address 1215H.
However, assume that the top address of the DBNZ instruction is at addresses 1194H-1293H.

6.21 Context Switching Instructions

The following context switching instructions can be used:

BRKCS ... 249
RETCS ... 250
RETCSB ... 251

BRKCS

Break Context Switch

Switch software context

[Instruction format] **BRKCS RBn**

[Operation]

RBS2-0 ← n,
PC_H ↔ R5, PC_L ↔ R4,
R7 ← PSW_H, R6 ← PSW_L,
RSS ← 0,
IE ← 0 **(n=0–7)**

[Operands]

Mnemonic	Operand
BRKCS	RBn

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- BRKCS is a software interrupt instruction.
- Register bank n described in the operand is selected and the contents of the 8-bit registers R5 and R4 in the register bank and the contents of the program counter (PC) are exchanged. The contents of the program status word (PSW) are saved in the 8-bit registers R7 and R6 and control branches to the address set in the R5 and R4 registers. Then, the RSS and IE flags are cleared to 0.
- To return from the software interrupt caused by the BRKCS instruction, use a RETCSB instruction.
- Do not change the contents of R7, R6, R5, R4 in the software interrupt program started by the BRKCS instruction. To use R7, R6, R5, R4, once save the register contents in a given stack, etc., and restore them to the original values before execution of the RETCSB instruction.

[Description example]

BRKCS RB3 ; Select register bank 3 and execute instruction at address indicated by R5 and R4 in the register bank.

RETCS

Return from Context Switch

Return from hardware context switching

[Instruction format] **RETCS**

[Operation] **$PC_H \leftarrow R5$, $PC_L \leftarrow R4$,**
$R5 \leftarrow addr16_H$, $R4 \leftarrow addr16_L$,
$PSW_H \leftarrow R7$, $PSW_L \leftarrow R6$
Clear the most significant one of bits set to 1 in ISPR to 0

[Operands]

Mnemonic	Operand
RETCS	!addr16

addr16=0000H-FDFFH

[Flags]

S	Z	AC	P/V	CY
R	R	R	R	R

[Explanation]

- The contents of 8-bit registers R7, R6, R5, and R4 in the register bank specified when the RETCS instruction is executed are transferred to the program status word (PSW) and the program counter (PC) and control is returned to the address set in R5 and R4. Then, the 16-bit immediate data specified in the operand is transferred to R5 and R4.
- The RETCS instruction is effective for context switching accompanying interrupt request occurrence and is used for returning from the context switching branch. addr16 described in the operand becomes the branch destination address when the same register bank is specified again by the context switching function.
- The RETCS instruction cannot be used for returning from a software interrupt caused by a BRK instruction, BRKCS instruction, or OPE code trap or returning from a vectored interrupt.
- Before execution of the RETCS instruction, the R7, R6, R5, and R4 registers must be set to the same values as just after the interrupt was acknowledged.

Cautions 1. Instructions cannot be fetched from addresses FE00H-FFFFH. Do not describe any of the addresses in addr16.

2. To return from the branch taken by the BRKCS instruction, do not use the RETCS instruction.

[Description example]

RETCS !3456H ; Return from interrupt caused by context switching and set the address when the next interrupt will be acknowledged to 3456H.

RETCSB

Return from Context Switch Break

Return from software context switching

[Instruction format] **RETCSB**

[Operation] **$PC_H \leftarrow R5$, $PC_L \leftarrow R4$,**
$R5 \leftarrow addr16_H$, $R4 \leftarrow addr16_L$,
$PSW_H \leftarrow R7$, $PSW_L \leftarrow R6$

[Operands]

Mnemonic	Operand
RETCSB	!addr16

addr16=0000H-FDFFH

[Flags]

S	Z	AC	P/V	CY
R	R	R	R	R

[Explanation]

- The contents of 8-bit registers R7, R6, R5, and R4 in the register bank specified when the RETCSB instruction is executed are transferred to the program status word (PSW) and the program counter (PC). Then, control is returned to the address set in R5 and R4.
- The RETCSB instruction is effective for context switching caused by a BRKCS instruction and is used for returning from the context switching branch. addr16 described in the operand becomes the branch destination address when the same register bank is specified again by the context switching function.
- The RETCSB instruction cannot be used for returning from a software interrupt caused by a BRK instruction or OPE code trap or a hardware interrupt.
- Before execution of the RETCSB instruction, the R7, R6, R5, and R4 registers must be set to the same values as just after the interrupt was acknowledged.

Caution To return from the interrupt service routine started by a BRKCS instruction, be sure to use the RETCSB instruction. If the RETCS instruction is used, the interrupt control circuit does not operate normally.

[Description example]

RETCSB !0ABCDH ; Return from interrupt caused by BRKCS instruction

6.22 String Instructions

The following string instructions can be used:

MOVM ... 253
MOVBK ... 254
XCHM ... 255
XCHBK ... 256
CMPME ... 257
CMPBKE ... 259
CMPMNE ... 261
CMPBKNE ... 263
CMPMC ... 265
CMPBKC ... 267
CMPMNC ... 269
CMPBKNC ... 271

MOVM

Move Multiple Byte

Block transfer of fixed byte data

[Instruction format] **MOVM [DE+], A**
MOVM [DE–], A

[Operation] **(DE) ← A, DE ← DE+1, C ← C–1 End if C=0**
(DE) ← A, DE ← DE–1, C ← C–1 End if C=0

[Operands]

Mnemonic	Operands
MOVM	[DE+], A
	[DE–], A

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- MOVM transfers the A register contents to the memory addressed by the DE register contents and increments or decrements the DE register contents, then decrements the C register contents and repeats these steps until the C register is set to 0.
- If an interrupt or macro service request that can be acknowledged occurs during execution of the MOVM instruction, the instruction execution is interrupted and the interrupt or macro service is acknowledged. When the interrupt is acknowledged, if the return address saved in the stack or R5 and R4 and the contents of the DE and C registers used in the MOVM instruction execution are not changed by the interrupt service program, the interrupted MOVM instruction execution is resumed when control is returned from the interrupt. When the macro service is acknowledged, the MOVM instruction execution is resumed after completion of the macro service.
- The MOVM instruction is mainly used for initializing a given area of memory to a specific value.

[Description example]

```
MOV C, #00H          ; C ← 00H
MOV A, #00H          ; A ← 00H
MOVW DE, #0FE00H     ; DE ← FE00H
MOVM [DE+], A        ; Clear RAM area FE00H-FEFFH to 0.
```

MOVBK

Move Block Byte

Block transfer of byte data

[Instruction format] **MOVBK [DE+], [HL+]**
MOVBK [DE–], [HL–]

[Operation] **(DE) ← (HL), DE ← DE+1, HL ← HL+1, C ← C–1 End if C=0**
(DE) ← (HL), DE ← DE–1, HL ← HL–1, C ← C–1 End if C=0

[Operands]

Mnemonic	Operands
MOVBK	[DE+], [HL+]
	[DE–], [HL–]

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- MOVBK transfers the contents of the memory addressed by the HL register contents to the memory addressed by the DE register contents and increments or decrements the DE and HL register contents, then decrements the C register contents and repeats these steps until the C register is set to 0.
- If an interrupt or macro service request that can be acknowledged occurs during execution of the MOVBK instruction, the instruction execution is interrupted and the interrupt or macro service is acknowledged. When the interrupt is acknowledged, if the return address saved in the stack or R5 and R4 and the contents of the DE, HL, and C registers used in the MOVBK instruction execution are not changed by the interrupt service program, the interrupted MOVBK instruction execution is resumed when control is returned from the interrupt. When the macro service is acknowledged, the MOVBK instruction execution is resumed after completion of the macro service.
- If the source data area overlaps the destination data area,

  - when the lowest address of the source area is smaller than the highest address of the destination area, the initial values of the DE and HL registers are set to their respective lowest addresses and MOVBK [DE+], [HL+] is used.
  - when the highest address of the source area is greater than the lowest address of the destination area, the initial values of the DE and HL registers are set to their respective highest addresses and MOVBK [DE–], [HL–] is used.

[Description example]

```
MOV C, #10H          ; C ← 10H
MOVW DE, #3000H      ; DE ← 3000H
MOVW HL, #5000H      ; HL ← 5000H
MOVBK [DE+], [HL+]   ; Transfer contents of memory area 5000H-500FH to memory area 3000H-300FH
```

XCHM

Exchange Multiple Byte

Block exchange of fixed byte data

[Instruction format] **XCHM [DE+], A**
XCHM [DE–], A

[Operation] **(DE) ↔ A, DE ← DE+1, C ← C–1 End if C=0**
(DE) ↔ A, DE ← DE–1, C ← C–1 End if C=0

[Operands]

Mnemonic	Operands
XCHM	[DE+], A
	[DE–], A

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- XCHM exchanges the A register contents and the memory addressed by the DE register contents and increments or decrements the DE register contents, then decrements the C register contents and repeats these steps until the C register is set to 0.
- If an interrupt or macro service request that can be acknowledged occurs during execution of the XCHM instruction, the instruction execution is interrupted and the interrupt or macro service is acknowledged. When the interrupt is acknowledged, if the return address saved in the stack or R5 and R4 and the contents of the DE and C registers used in the XCHM instruction execution are not changed by the interrupt service program, the interrupted XCHM instruction execution is resumed when control is returned from the interrupt. When the macro service is acknowledged, the XCHM instruction execution is resumed after completion of the macro service.
- The XCHM instruction is mainly used for moving data in memory one byte. To move data to the higher address side, use XCHM [DE+], A or to move data to the lower address side, use XCHM [DE–], A. To move data two bytes or more, use a MOVBK instruction.

[Description example]

```
MOV C, #10H       ; C ← 10H
MOV A, #00H       ; A ← 00H
MOVW DE, #3050H   ; DE ← 3050H
XCHM [DE+], A     ; Shift contents of memory area 3050H-305FH each byte to one later address (address
                    3050H contents are set to 0).
```

XCHBK

Exchange Block Byte

Block exchange of byte data

[Instruction format] **XCHBK [DE+], [HL+]**
XCHBK [DE–], [HL–]

[Operation] **(DE) ↔ (HL), DE ← DE+1, HL ← HL+1, C ← C–1 End if C=0**
(DE) ↔ (HL), DE ← DE–1, HL ← HL–1, C ← C–1 End if C=0

[Operands]

Mnemonic	Operands
XCHBK	[DE+], [HL+]
	[DE–], [HL–]

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- XCHBK exchanges the contents of the memory addressed by the HL register contents and the memory addressed by the DE register contents and increments or decrements the DE and HL register contents, then decrements the C register contents and repeats these steps until the C register is set to 0.
- If an interrupt or macro service request that can be acknowledged occurs during execution of the XCHBK instruction, the instruction execution is interrupted and the interrupt or macro service is acknowledged. When the interrupt is acknowledged, if the return address saved in the stack or R5 and R4 and the contents of the DE, HL, and C registers used in the XCHBK instruction execution are not changed by the interrupt service program, the interrupted XCHBK instruction execution is resumed when control is returned from the interrupt. When the macro service is acknowledged, the XCHBK instruction execution is resumed after completion of the macro service.
- If the source data area overlaps the destination data area,

  - when the lowest address of the source area is smaller than the highest address of the destination area, the initial values of the DE and HL registers are set to their respective lowest addresses and XCHBK [DE+], [HL+] is used.
  - when the highest address of the source area is greater than the lowest address of the destination area, the initial values of the DE and HL registers are set to their respective highest addresses and XCHBK [DE–], [HL–] is used.

[Description example]

```
MOV C, #20H
MOVW DE, #0FE00H
MOVW HL, #0FE70H
XCHBK [DE+], [HL+] ; Exchange 20H-byte data starting at address FE00H and data starting at address FE70H
```

CMPME

Compare Multiple Equal Byte

Block comparison with fixed byte data (match detection)

[Instruction format] **CMPME [DE+], A**
CMPME [DE–], A

[Operation] **(DE)–A, DE ← DE+1, C ← C–1 End if C=0 or Z=0**
(DE)–A, DE ← DE–1, C ← C–1 End if C=0 or Z=0

[Operands]

Mnemonic	Operands
CMPME	[DE+], A
	[DE–], A

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- CMPME compares the A register contents with the memory addressed by the DE register contents, increments or decrements the DE register contents, and decrements the C register contents. CMPME repeats these steps until a mismatch is found between them as a result of the comparison or the C register is set to 0.
- When the instruction is executed, the A register contents and the contents of the memory addressed by the DE register contents do not change.
- If an interrupt or macro service request that can be acknowledged occurs during execution of the CMPME instruction, the instruction execution is interrupted and the interrupt or macro service is acknowledged. When the interrupt is acknowledged, if the return address saved in the stack or R5 and R4 and the contents of the DE and C registers used in the CMPME instruction execution are not changed by the interrupt service program, the interrupted CMPME instruction execution is resumed when control is returned from the interrupt. When the macro service is acknowledged, the CMPME instruction execution is resumed after completion of the macro service.
- The S, Z, AC, P/V, and CY flags change according to the last comparison (subtraction) executed by the CMPME instruction.
- When bit 7 is set to 1 as a result of the subtraction, the S flag is set to 1; otherwise, cleared to 0.
- When the subtraction result is 0, the Z flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow into bit 3 out of bit 4, the AC flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow in bit 6 and does not generate a borrow in bit 7 (when an underflow occurs by operation in the two's complement format) or if the subtraction instruction does not generate a borrow in bit 6 and generates a borrow in bit 7 (when an overflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow in bit 7, the CY flag is set to 1; otherwise, cleared to 0.

[Description example]

```
MOV C, #20H
MOVW DE, #0FE00H
MOV A, #00H
CMPME [DE+], A ; Indicate whether or not 20H-byte data starting at address FE00H is all 00H
BNZ $JMP       ; Branch to address JMP if data that is not 00H is found
```

CMPBKE

Compare Block Equal Byte

Block comparison with byte data (match detection)

[Instruction format] **CMPBKE [DE+], [HL+]**
CMPBKE [DE–], [HL–]

[Operation] **(DE)–(HL), DE ← DE+1, HL ← HL+1, C ← C–1**
End if C=0 or Z=0
(DE)–(HL), DE ← DE–1, HL ← HL–1, C ← C–1
End if C=0 or Z=0

[Operands]

Mnemonic	Operands
CMPBKE	[DE+], [HL+]
	[DE–], [HL–]

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- CMPBKE compares the contents of the memory addressed by the HL register contents with the contents of the memory addressed by the DE register contents, increments or decrements the DE and HL register contents, and decrements the C register contents. CMPBKE repeats these steps until a mismatch is found between them as a result of the comparison or the C register is set to 0.
- When the instruction is executed, the contents of the memory areas addressed by the DE and HL register contents do not change.
- If an interrupt or macro service request that can be acknowledged occurs during execution of the CMPBKE instruction, the instruction execution is interrupted and the interrupt or macro service is acknowledged. When the interrupt is acknowledged, if the return address saved in the stack or R5 and R4 and the contents of the DE, HL, and C registers used in the CMPBKE instruction execution are not changed by the interrupt service program, the interrupted CMPBKE instruction execution is resumed when control is returned from the interrupt. When the macro service is acknowledged, the CMPBKE instruction execution is resumed after completion of the macro service.
- The S, Z, AC, P/V, and CY flags change according to the last comparison (subtraction) executed by the CMPBKE instruction.
- When bit 7 is set to 1 as a result of the subtraction, the S flag is set to 1; otherwise, cleared to 0.
- When the subtraction result is 0, the Z flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow into bit 3 out of bit 4, the AC flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow in bit 6 and does not generate a borrow in bit 7 (when an underflow occurs by operation in the two's complement format) or if the subtraction instruction does not generate a borrow in bit 6 and generates a borrow in bit 7 (when an overflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow in bit 7, the CY flag is set to 1; otherwise, cleared to 0.

[Description example]

```
MOV C, #20H
MOVW DE, #0FE00H
MOVW HL, #0FE70H
CMPBKE [DE+], [HL+]
BNE $DIFF          ; Compare 20H-byte data starting at address FE00H with data starting at address
                     FE70H and branch to address DIFF if different data is found.
```

CMPMNE

Compare Multiple Not Equal Byte

Block comparison with fixed byte data (mismatch detection)

[Instruction format] **CMPMNE [DE+], A**
CMPMNE [DE–], A

[Operation] **(DE)–A, DE ← DE+1, C ← C–1 End if C=0 or Z=1**
(DE)–A, DE ← DE–1, C ← C–1 End if C=0 or Z=1

[Operands]

Mnemonic	Operands
CMPMNE	[DE+], A
	[DE–], A

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- CMPMNE compares the A register contents with the memory addressed by the DE register contents, increments or decrements the DE register contents, and decrements the C register contents. CMPMNE repeats these steps until a match is found between them as a result of the comparison or the C register is set to 0.
- When the instruction is executed, the A register contents and the contents of the memory addressed by the DE register contents do not change.
- If an interrupt or macro service request that can be acknowledged occurs during execution of the CMPMNE instruction, the instruction execution is interrupted and the interrupt or macro service is acknowledged.
  When the interrupt is acknowledged, if the return address saved in the stack or R5 and R4 and the contents of the DE and C registers used in the CMPMNE instruction execution are not changed by the interrupt service program, the interrupted CMPMNE instruction execution is resumed when control is returned from the interrupt.
  When the macro service is acknowledged, the CMPMNE instruction execution is resumed after completion of the macro service.
- The S, Z, AC, P/V, and CY flags change according to the last comparison (subtraction) executed by the CMPMNE instruction.
- When bit 7 is set to 1 as a result of the subtraction, the S flag is set to 1; otherwise, cleared to 0.
- When the subtraction result is 0, the Z flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow into bit 3 out of bit 4, the AC flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow in bit 6 and does not generate a borrow in bit 7 (when an underflow occurs by operation in the two's complement format) or if the subtraction instruction does not generate a borrow in bit 6 and generates a borrow in bit 7 (when an overflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow in bit 7, the CY flag is set to 1; otherwise, cleared to 0.

[Description example]

```
MOV C, #00H       ; C ← 00H
MOVW DE, #3000H   ; DE ← 3000H
CMPMNE [DE+], A
BZ $IMP           ; Branch to address indicated by label IMP if the same value as the A register contents
                    exists at 3000H-30FFH
```

CMPBKNE

Compare Block Not Equal Byte

Block comparison with byte data (mismatch detection)

[Instruction format] **CMPBKNE [DE+], [HL+]**
CMPBKNE [DE–], [HL–]

[Operation] **(DE)–(HL), DE ← DE+1, HL ← HL+1, C ← C–1**
End if C=0 or Z=1
(DE)–(HL), DE ← DE–1, HL ← HL–1, C ← C–1
End if C=0 or Z=1

[Operands]

Mnemonic	Operands
CMPBKNE	[DE+], [HL+]
	[DE–], [HL–]

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- CMPBKNE compares the contents of the memory addressed by the HL register contents with the contents of the memory addressed by the DE register contents, increments or decrements the DE and HL register contents, and decrements the C register contents. CMPBKNE repeats these steps until a match is found between them as a result of the comparison or the C register is set to 0.
- When the instruction is executed, the contents of the memory areas addressed by the DE and HL register contents do not change.
- If an interrupt or macro service request that can be acknowledged occurs during execution of the CMPBKNE instruction, the instruction execution is interrupted and the interrupt or macro service is acknowledged. When the interrupt is acknowledged, if the return address saved in the stack or R5 and R4 and the contents of the DE, HL, and C registers used in the CMPBKNE instruction execution are not changed by the interrupt service program, the interrupted CMPBKNE instruction execution is resumed when control is returned from the interrupt. When the macro service is acknowledged, the CMPBKNE instruction execution is resumed after completion of the macro service.
- The S, Z, AC, P/V, and CY flags change according to the last comparison (subtraction) executed by the CMPBKNE instruction.
- When bit 7 is set to 1 as a result of the subtraction, the S flag is set to 1; otherwise, cleared to 0.
- When the subtraction result is 0, the Z flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow into bit 3 out of bit 4, the AC flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow in bit 6 and does not generate a borrow in bit 7 (when an underflow occurs by operation in the two's complement format) or if the subtraction instruction does not generate a borrow in bit 6 and generates a borrow in bit 7 (when an overflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow in bit 7, the CY flag is set to 1; otherwise, cleared to 0.

[Description example]

```
MOV C, #5H
MOVW DE, #0FE00H
MOVW HL, #0FE70H
CMPBKNE [DE+], [HL+]
BE $FIND          ; Compare 5-byte data starting at address FE00H with data starting at address FE70H
                    and branch to address FIND if match data is found.
```

CMPMC

Compare Multiple Carry Byte

Block comparison with fixed byte data (value size comparison)

[Instruction format] **CMPMC [DE+], A**
CMPMC [DE–], A

[Operation] **(DE)–A, DE ← DE+1, C ← C–1 End if C=0 or CY=0**
(DE)–A, DE ← DE–1, C ← C–1 End if C=0 or CY=0

[Operands]

Mnemonic	Operands
CMPMC	[DE+], A
	[DE–], A

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- CMPMC compares the A register contents with the memory addressed by the DE register contents, increments or decrements the DE register contents, and decrements the C register contents. CMPMC repeats these steps until the contents of the memory addressed by the DE register contents become equal to or greater than the A register contents as a result of the comparison or the C register is set to 0.
- When the instruction is executed, the A register contents and the contents of the memory addressed by the DE register contents do not change.
- If an interrupt or macro service request that can be acknowledged occurs during execution of the CMPMC instruction, the instruction execution is interrupted and the interrupt or macro service is acknowledged.
  When the interrupt is acknowledged, if the return address saved in the stack or R5 and R4 and the contents of the DE and C registers used in the CMPMC instruction execution are not changed by the interrupt service program, the interrupted CMPMC instruction execution is resumed when control is returned from the interrupt.
  When the macro service is acknowledged, the CMPMC instruction execution is resumed after completion of the macro service.
- The S, Z, AC, P/V, and CY flags change according to the last comparison (subtraction) executed by the CMPMC instruction.
- When bit 7 is set to 1 as a result of the subtraction, the S flag is set to 1; otherwise, cleared to 0.
- When the subtraction result is 0, the Z flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow into bit 3 out of bit 4, the AC flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow in bit 6 and does not generate a borrow in bit 7 (when an underflow occurs by operation in the two's complement format) or if the subtraction instruction does not generate a borrow in bit 6 and generates a borrow in bit 7 (when an overflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow in bit 7, the CY flag is set to 1; otherwise, cleared to 0.

[Description example]

```
MOV C, #10H
MOV A, #80H
MOVW DE, #0FE00H
CMPMC [DE+], A
BNC $BIG          ; Branch to address BIG if data of 80H or greater is contained in 10H-byte data starting
                    at address FE00H
```

CMPBKC

Compare Block Carry Byte

Block comparison with byte data (value size comparison)

[Instruction format] **CMPBKC [DE+], [HL+]**
CMPBKC [DE–], [HL–]

[Operation] **(DE)–(HL), DE ← DE+1, HL ← HL+1, C ← C–1**
End if C=0 or CY=0
(DE)–(HL), DE ← DE–1, HL ← HL–1, C ← C–1
End if C=0 or CY=0

[Operands]

Mnemonic	Operands
CMPBKC	[DE+], [HL+]
	[DE–], [HL–]

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- CMPBKC compares the contents of the memory addressed by the HL register contents with the contents of the memory addressed by the DE register contents, increments or decrements the DE and HL register contents, and decrements the C register contents. CMPBKC repeats these steps until the contents of the memory addressed by the DE register become equal to or greater than the contents of the memory addressed by the HL register as a result of the comparison or the C register is set to 0.
- When the instruction is executed, the contents of the memory areas addressed by the DE and HL register contents do not change.
- If an interrupt or macro service request that can be acknowledged occurs during execution of the CMPBKC instruction, the instruction execution is interrupted and the interrupt or macro service is acknowledged. When the interrupt is acknowledged, if the return address saved in the stack or R5 and R4 and the contents of the DE, HL, and C registers used in the CMPBKC instruction execution are not changed by the interrupt service program, the interrupted CMPBKC instruction execution is resumed when control is returned from the interrupt. When the macro service is acknowledged, the CMPBKC instruction execution is resumed after completion of the macro service.
- The S, Z, AC, P/V, and CY flags change according to the last comparison (subtraction) executed by the CMPBKC instruction.
- When bit 7 is set to 1 as a result of the subtraction, the S flag is set to 1; otherwise, cleared to 0.
- When the subtraction result is 0, the Z flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow into bit 3 out of bit 4, the AC flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow in bit 6 and does not generate a borrow in bit 7 (when an underflow occurs by operation in the two's complement format) or if the subtraction instruction does not generate a borrow in bit 6 and generates a borrow in bit 7 (when an overflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.

- If the subtraction operation generates a borrow in bit 7, the CY flag is set to 1; otherwise, cleared to 0.

[Description example]

```
MOV C, #3H
MOVW DE, #0FE00H
MOVW HL, #0FE70H
CMPBKC [DE+], [HL+]
BNC $BIG          ; Compare 3-byte data starting at address FE00H with 3-byte data starting at address
                    FE70H and branch to address BIG if the former 3-byte data is equal to or greater than
                    the latter 3-byte data
```

CMPMNC

Compare Multiple Not Carry Byte

Block comparison with fixed byte data (value size comparison)

[Instruction format] **CMPMNC [DE+], A**
CMPMNC [DE–], A

[Operation] **(DE)–A, DE ← DE+1, C ← C–1 End if C=0 or CY=1**
(DE)–A, DE ← DE–1, C ← C–1 End if C=0 or CY=1

[Operands]

Mnemonic	Operands
CMPMNC	[DE+], A
	[DE–], A

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- CMPMNC compares the A register contents with the memory addressed by the DE register contents, increments or decrements the DE register contents, and decrements the C register contents. CMPMNC repeats these steps until the A register contents become greater than the contents of the memory addressed by the DE register contents as a result of the comparison or the C register is set to 0.
- When the instruction is executed, the A register contents and the contents of the memory addressed by the DE register contents do not change.
- If an interrupt or macro service request that can be acknowledged occurs during execution of the CMPMNC instruction, the instruction execution is interrupted and the interrupt or macro service is acknowledged.
  When the interrupt is acknowledged, if the return address saved in the stack or R5 and R4 and the contents of the DE and C registers used in the CMPMNC instruction execution are not changed by the interrupt service program, the interrupted CMPMNC instruction execution is resumed when control is returned from the interrupt.
  When the macro service is acknowledged, the CMPMNC instruction execution is resumed after completion of the macro service.
- The S, Z, AC, P/V, and CY flags change according to the last comparison (subtraction) executed by the CMPMNC instruction.
- When bit 7 is set to 1 as a result of the subtraction, the S flag is set to 1; otherwise, cleared to 0.
- When the subtraction result is 0, the Z flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow into bit 3 out of bit 4, the AC flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow in bit 6 and does not generate a borrow in bit 7 (when an underflow occurs by operation in the two's complement format) or if the subtraction instruction does not generate a borrow in bit 6 and generates a borrow in bit 7 (when an overflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow in bit 7, the CY flag is set to 1; otherwise, cleared to 0.

[Description example]

```
MOV C, #00H          ; C ← 00H
MOVW DE, #8000H      ; DE ← 8000H
CMPMNC [DE+], A
BC $JMP              ; Branch to address indicated by label JMP if a value greater than the A register contents
                       exists at 8000H-80FFH
```

CMPBKNC

Compare Block Not Carry Byte

Block comparison with byte data (value size comparison)

[Instruction format] **CMPBKNC [DE+], [HL+]**
CMPBKNC [DE–], [HL–]

[Operation] **(DE)–(HL), DE ← DE+1, HL ← HL+1, C ← C–1**
End if C=0 or CY=1
(DE)–(HL), DE ← DE–1, HL ← HL–1, C ← C–1
End if C=0 or CY=1

[Operands]

Mnemonic	Operands
CMPBKNC	[DE+], [HL+]
	[DE–], [HL–]

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Explanation]

- CMPBKNC compares the contents of the memory addressed by the HL register contents with the contents of the memory addressed by the DE register contents, increments or decrements the DE and HL register contents, and decrements the C register contents. CMPBKNC repeats these steps until the contents of the memory addressed by the HL register become greater than the contents of the memory addressed by the DE register as a result of the comparison or the C register is set to 0.
- When the instruction is executed, the contents of the memory areas addressed by the DE and HL register contents do not change.
- If an interrupt or macro service request that can be acknowledged occurs during execution of the CMPBKNC instruction, the instruction execution is interrupted and the interrupt or macro service is acknowledged. When the interrupt is acknowledged, if the return address saved in the stack or R5 and R4 and the contents of the DE, HL, and C registers used in the CMPBKNC instruction execution are not changed by the interrupt service program, the interrupted CMPBKNC instruction execution is resumed when control is returned from the interrupt. When the macro service is acknowledged, the CMPBKNC instruction execution is resumed after completion of the macro service.
- The S, Z, AC, P/V, and CY flags change according to the last comparison (subtraction) executed by the CMPBKNC instruction.
- When bit 7 is set to 1 as a result of the subtraction, the S flag is set to 1; otherwise, cleared to 0.
- When the subtraction result is 0, the Z flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow into bit 3 out of bit 4, the AC flag is set to 1; otherwise, cleared to 0.
- If the subtraction operation generates a borrow in bit 6 and does not generate a borrow in bit 7 (when an underflow occurs by operation in the two's complement format) or if the subtraction instruction does not generate a borrow in bit 6 and generates a borrow in bit 7 (when an overflow occurs by operation in the two's complement format), the P/V flag is set to 1; otherwise, cleared to 0.

- If the subtraction operation generates a borrow in bit 7, the CY flag is set to 1; otherwise, cleared to 0.

[Description example]

```
MOV C, #4H
MOVW DE, #0FE00H
MOVW HL, #0FE70H
CMPBKNC [DE–], [HL–]
BC $LITTLE          ; Compare 4-byte data starting at address FE00H with data starting at address FE70H
                      and branch to address LITTLE if the former 4-byte data is less than the latter data
```

6.23 CPU Control Instructions

The following CPU control instructions can be used:

MOV STBC, #byte ... 274
MOV WDM, #byte ... 275
SWRS ... 276
SEL RBn ... 277
SEL RBn, ALT ... 278
NOP ... 279
EI ... 280
DI ... 281

MOV STBC, #byte

Move

Set standby mode

[Instruction format] **MOV STBC, #byte**

[Operation] **STBC ← byte**

[Operands]

Mnemonic	Operands
MOV	STBC, #byte

byte = 00H-FFH

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- MOV STBC, #byte, which is a dedicated write instruction into the standby control register (STBC), writes the immediate data specified in the second operand into the STBC register. Data can be written into the STBC register only by executing the instruction.
- The instruction takes a special format. In the operation code of the instruction, in addition to immediate data to be written, data resulting from negating the value of the immediate data must be provided. (See below.) (It is automatically generated by the NEC assembler (RA78K3).)

  - Format of operation code

0	0	0	0	1	0	0	1
1	1	0	0	0	0	0	0
←			$\overline{\text{byte}}$				→
←			byte				→

- The CPU checks the immediate data to be written and the data provided by negating the immediate data, and only when they are correct, writes the data; if they are not correct, the CPU does not write the data and generates an OPE code trap interrupt.

[Description example]

MOV STBC, #2 ; Write 2 into STBC register (set STOP mode)

MOV WDM, #byte

Move

Set watchdog timer

[Instruction format] **MOV WDM, #byte**

[Operation] **WDM ← byte**

[Operands]

Mnemonic	Operand
MOV	WDM, #byte

byte = 00H-FFH

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- MOV WDM, #byte, which is a dedicated write instruction into the watchdog timer mode register (WDM), writes the immediate data specified in the second operand into the STBC register. Data can be written into the WDM register only by executing the instruction.
- The instruction takes a special format. In the operation code of the instruction, in addition to immediate data to be written, data resulting from negating the value of the immediate data must be provided. (See below.) (It is automatically generated by the NEC assembler (RA78K3).)

  • Format of operation code

0	0	0	0	1	0	0	1
1	1	0	0	0	0	1	0
←			$\overline{\text{byte}}$				→
←			byte				→

- The CPU checks the immediate data to be written and the data provided by negating the immediate data, and only when they are correct, writes the data; if they are not correct, the CPU does not write the data and generates an OPE code trap interrupt.

[Description example]

MOV WDM, #0C0H ; Write 0C0H into WDM register

Phase-out/Discontinued

SWRS

Switch Register Set

Switch register bit

[Instruction format] **SWRS**

[Operation] **RSS** ← $\overline{\textbf{RSS}}$

[Operands] None

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- SWRS inverts the register set selection flag (RSS) contents.

SEL RBn

Select Register Bank

Select register bank

[Instruction format] **SEL RBn**

[Operation] **RBS2-0 ← n, RSS ← 0 (n=0-7)**

[Operands]

Mnemonic	Operand (RBn)
SEL	RBn

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- SEL RBn selects the register bank specified in the operand (RBn) as a new register bank used with the instructions following the SEL RBn instruction.
- RBn ranges from RB0 to RB7.

[Description example]

SEL RB2 ; Select register bank 2 as a new register bank used with the instructions following SEL RBn

Phase-out/Discontinued

SEL RBn, ALT

Select Register Bank

Select register bank

[Instruction format] **SEL RBn, ALT**

[Operation] **RBS2-0 ← n, RSS ← 1 (n=0-7)**

[Operands]

Mnemonic	Operand
SEL	RBn, ALT

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- SEL RBn, ALT selects the register bank specified in the operand (RBn) as a new register bank used with the instructions following the SEL RBn instruction and further sets the register set selection flag (RSS) to 1.
- RBn ranges from RB0 to RB7.

NOP

No Operation

No Operation

[Instruction format] **NOP**

[Operation] **No Operation**

[Operands] None

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- NOP performs no operation and consumes two clocks.

EI

Enable interrupt

Enable interrupt

[Instruction format] **EI**

[Operation] **IE ← 1 (Enable interrupt)**

[Operands] None

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- EI enables a maskable interrupt to be acknowledged (sets the interrupt request enable flag (IE) to 1).
- No interrupts or macro services are acknowledged between the EI instruction and its following one instruction.
- It is possible not to acknowledge a vectored interrupt by another source even if the instruction is executed. For details, refer to **"User's Manual: Hardware"** of each product.

DI

Disable interrupt

Disable interrupt

[Instruction format] **DI**

[Operation] **IE ← 0 (Disable interrupt)**

[Operands] None

[Flags]

S	Z	AC	P/V	CY

[Explanation]

- DI disables acknowledgement of a maskable interrupt by a vectored interrupt (clears the interrupt request enable flag (IE) to 0).
- No interrupts or macro services are acknowledged between the DI instruction and its following one instruction.
- For details of interrupt processing, refer to **"User's Manual: Hardware"** of each product.

[MEMO]

CHAPTER 7 CAUTIONS ON USE

This chapter collects cautions given in the preceding chapters. Use the chapter when designing application products. The pages enclosed in parentheses indicate the pages on which the caution is given.

7.1 Cautions on CHAPTER 3 CPU ARCHITECTURE

(1) To make a word access to the main RAM area (FE00H-FEFFH) (containing stack handling), only even addresses can be specified in operands. (p.30, 44, 48)

★ (2) The μPD78361A, 78362A, and 78P364A do not contain the CPU control word. (p.33, 34, 44)

(3) To make a word access to the main RAM area (FE00H-FEFFH) (containing stack handling), the access operation varies depending on whether the reference address is even or odd. Therefore, if an access to an even address and an access to an odd address are mixed, an error is caused to occur. Set only even reference addresses. To execute a 16-bit data transfer instruction, specify even addresses in the operands. If an odd address is specified, an error occurs in the assembler package (RA78K3). (p.35)

(4) Do not make a word access across the peripheral RAM area and the main RAM area. (p.35)

(5) Do not access addresses in which the special function registers are not mapped (except for external SFR area). (p.39)

★ (6) The μPD78361A, 78362A, and 78P364A do not contain external SFR area. (p.39)

★ (7) The μPD78361A, 78362A, and 78P364A do not contain external memory area. (p.39)

(8) Vector table entries for $\overline{\text{RESET}}$ input, BRK instruction, and OPE code trap interrupt are fixed to 0000H, 003EH, and 003CH respectively, and not affected by the TPF. (p.44)

(9) The μPD78356 enters the complete interrupt disable state during execution of a write access instruction to an interrupt control register or program status word (PSW). In this state, nonmaskable interrupt requests and macro service requests are not acknowledged and pending. (p.50)

7.2 Cautions on CHAPTER 5 INSTRUCTION SET LIST

(1) When both the source and destination in the operand field are registers or saddr, saddrp as in MOV r, r1 or ADD saddr, saddr, the codes are as follows: (p.94)

- when both are registers (or register pairs), the destination specification code precedes the source specification code;
- when both are saddr, saddrp, offset data with the preceding 1-byte data specifying the source and the following 1-byte data specifying the destination.

(2) If a special function register (SFR) mapped in FF00H-FF1FH is described as operand sfr, sfrp, short direct addressing rather than SFR addressing is applied and operation code of an instruction having an operand saddr, saddrp is generated. (p.95)

(3) The following series products are the same in the number of clocks at normal fetch and at high-speed fetch. See the column “Normal fetch” in the clock list. (p.113)

μPD78356 Subseries, 78366 Subseries, 78372 Subseries

7.3 Cautions on CHAPTER 6 EXPLANATION OF INSTRUCTIONS

(1) If STBC or WDM is described as sfr to execute the MOV sfr, #byte instruction, a dedicated operation code different from the instruction is generated. (p.140)

(2) The μPD78352A Subseries does not contain the multiplication and accumulation instruction with saturation function. (p.170)

(3) The μPD78352A Subseries does not contain the correlation operation instruction. (p.173)

(4) To return from the interrupt service routine accompanying the BRK instruction or OPE code trap, be sure to use the RETB instruction. If the RETI instruction is used, the interrupt control circuit does not operate normally. (p.213)

(5) To return from the interrupt service routine started by a BRKCS instruction, be sure to use the RETCSB instruction. If the RETCS instruction is used, the interrupt control circuit does not operate normally. (p.251)

APPENDIX A TOOLS

A.1 78K/III Series Common Tools

Language processors

<table>
<tr><td rowspan="10">78K/III Series
relocatable assembler
(RA78K3)</td><td colspan="4">RA78K/III is a relocatable assembler common to the 78K/III Series. Since it contains the macro function, the development efficiency can be improved. A structured assembler, which enables you to explicitly describe a program control structure, is also attached for improving program productivity and maintenance.</td></tr>
<tr><td rowspan="2">Host machine</td><td colspan="2" rowspan="2"></td><td rowspan="2">Ordering code
(product name)</td></tr>
<tr><td>OS</td><td>Distributed media</td></tr>
<tr><td>PC-9800 Series</td><td>MS-DOS™</td><td>3.5-inch 2HD</td><td>μS5A13RA78K3</td></tr>
<tr><td></td><td></td><td>5-inch 2HD</td><td>μS5A10RA78K3</td></tr>
<tr><td>IBM PC/AT™
and it's compatibles</td><td>PC DOS™</td><td>3.5-inch 2HC</td><td>μS7B13RA78K3</td></tr>
<tr><td></td><td></td><td>5-inch 2HD</td><td>μS7B10RA78K3</td></tr>
<tr><td>HP9000 Series 700™</td><td>HP-UX™</td><td>DAT</td><td>μS3P16RA78K3</td></tr>
<tr><td>SPARCstation™</td><td>SunOS™</td><td rowspan="2">CGMT (QIC-24)</td><td>μS3K15RA78K3</td></tr>
<tr><td>NEWS™</td><td>NEWS-OS™</td><td>μS3R15RA78K3</td></tr>
<tr><td rowspan="10">78K/III Series
C compiler
(CC78K3)</td><td colspan="4">CC78K/III is a C compiler common to the 78K/III Series. It is a program which converts programs written in C language into object codes that can be executed by microcontrollers. To use the compiler, the 78K/III Series relocatable assembler (RA78K3) is required.</td></tr>
<tr><td rowspan="2">Host machine</td><td colspan="2" rowspan="2"></td><td rowspan="2">Ordering code
(product name)</td></tr>
<tr><td>OS</td><td>Distributed media</td></tr>
<tr><td>PC-9800 Series</td><td>MS-DOS™</td><td>3.5-inch 2HD</td><td>μS5A13CC78K3</td></tr>
<tr><td></td><td></td><td>5-inch 2HD</td><td>μS5A10CC78K3</td></tr>
<tr><td>IBM PC/AT
and it's compatibles</td><td>PC DOS</td><td>3.5-inch 2HC</td><td>μS7B13CC78K3</td></tr>
<tr><td></td><td></td><td>5-inch 2HC</td><td>μS7B10CC78K3</td></tr>
<tr><td>HP9000 Series 700</td><td>HP-UX</td><td>DAT</td><td>μS3P16CC78K3</td></tr>
<tr><td>SPARCstation</td><td>SunOS</td><td rowspan="2">CGMT (QIC-24)</td><td>μS3K15CC78K3</td></tr>
<tr><td>NEWS</td><td>NEWS-OS</td><td>μS3R15CC78K3</td></tr>
</table>

Remark The operation of the relocatable assembler and C compiler is guaranteed only on the host machines under the operating systems listed above.

A.2 μPD78352A Subseries Tools

PROM write tools

Hardware	PG-1500	PG-1500 is a PROM programmer which enables you to program single chip microcomputers containing PROM by stand-alone or host machine operation by connecting an attached board and optional programmer adapter to PG-1500. It also enables you to program typical PROM devices of 256K bits to 1M bits.			
	PA-78P352G PA-78P352KK	PROM program adapters required to program the μPD78P352 on a general-purpose PROM programmer such as the PG-1500. PA-78P352G for μPD78P352G PA-78P352KK for μPD78P352KK			
Software	PG-1500 controller	PG-1500 and a host machine are connected by a serial or parallel interface for controlling the PG-1500 on the host machine.			
		Host machine	OS	Distributed media	Ordering code (product name)
		PC-9800 Series	MS-DOS	3.5-inch 2HD	μS5A13PG1500
				5-inch 2HD	μS5A10PG1500
★		IBM PC/AT and it's compatibles	PC DOS	3.5-inch 2HD	μS7B13PG1500
				5-inch 2HC	μS7B10PG1500

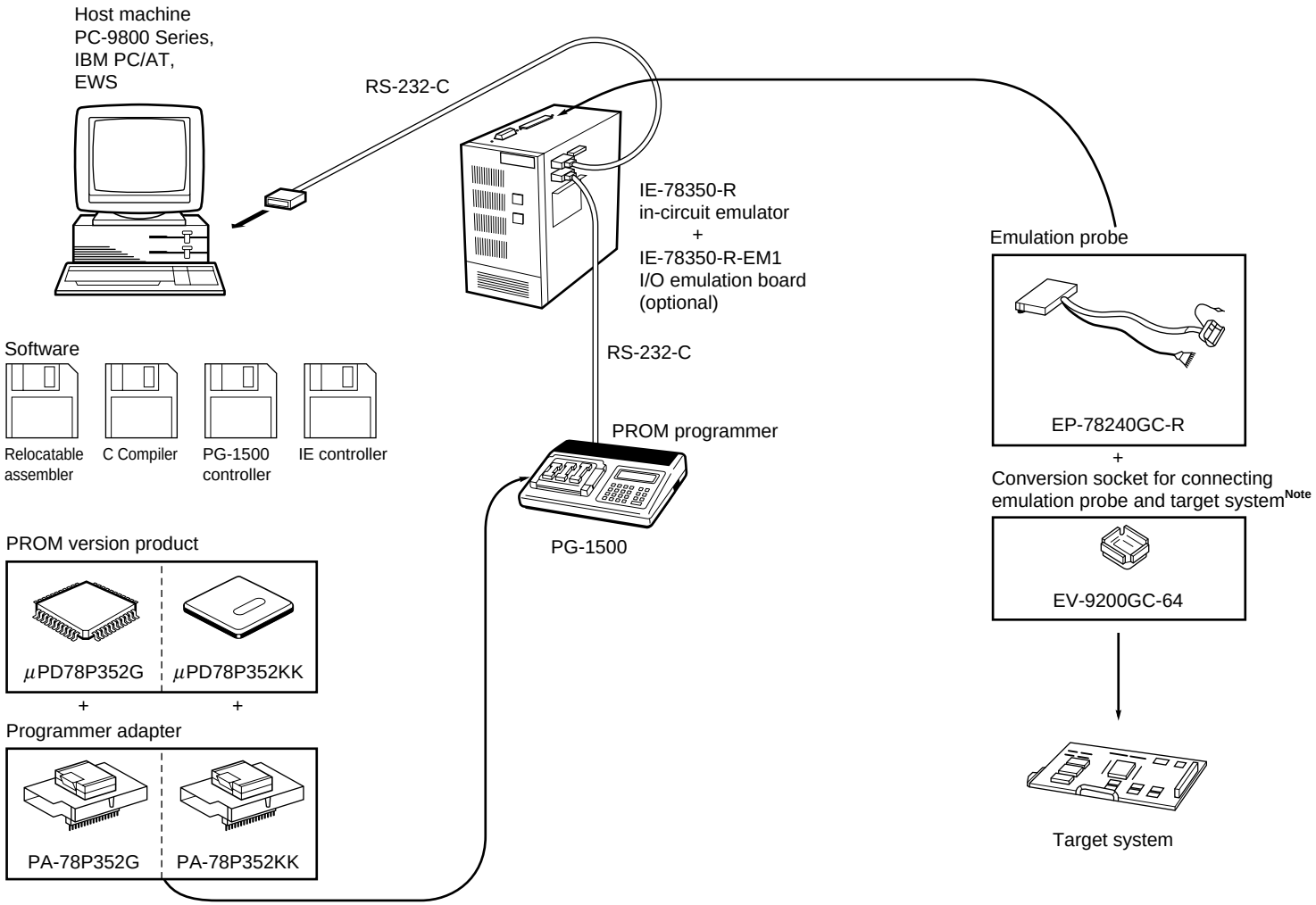
Remark The operation of the PG-1500 controller is guaranteed only on the host machines under the operating systems listed above.

Phase-out/Discontinued

Debugging tools (When using the IE controller)

Hardware	IE-78350-R	IE-78350-R is an in-circuit emulator that can be used to develop and debug application systems. For debugging, connect a host machine to the in-circuit emulator.			
	IE-78350-R-EM1	IE-78350-R-EM1 is an I/O emulation board to emulate peripheral functions of input/output ports, etc., of the μPD78352A Subseries.			
	EP-78240GC-R EV-9200GC-64	Emulation probe for 64-pin QFP of the μPD78352A Subseries. Use the emulation probe to connect IE-78350-R and target system. One piece of conversion socket EV-9200GC-64 used for connection to the target system is attached.			
Software	IE-78350-R control program (IE controller)	Program to control IE-78350-R on a host machine. Automatic execution of commands, etc., is enabled for more efficient debugging.			
		Host machine	OS	Distributed media	Ordering code (product name)
		PC-9800 Series	MS-DOS	3.5-inch 2HD	μS5A13IE8350
				5-inch 2HD	μS5A10IE78350
★		IBM PC/AT and it's compatibles	PC DOS	3.5-inch 2HC	μS7B13IE78350
				5-inch 2HC	μS7B10IE78350

Remark The operation of the IE controller is guaranteed only on the host machines under the operating systems listed above.



Note The conversion socket is attached to the emulation probe.

Remarks 1. The host machine and PG-1500 can also be connected directly by RS-232-C for use.

2. Medium supplying software is represented by 3.5-inch floppy diskette in this figure.

Phase-out/Discontinued

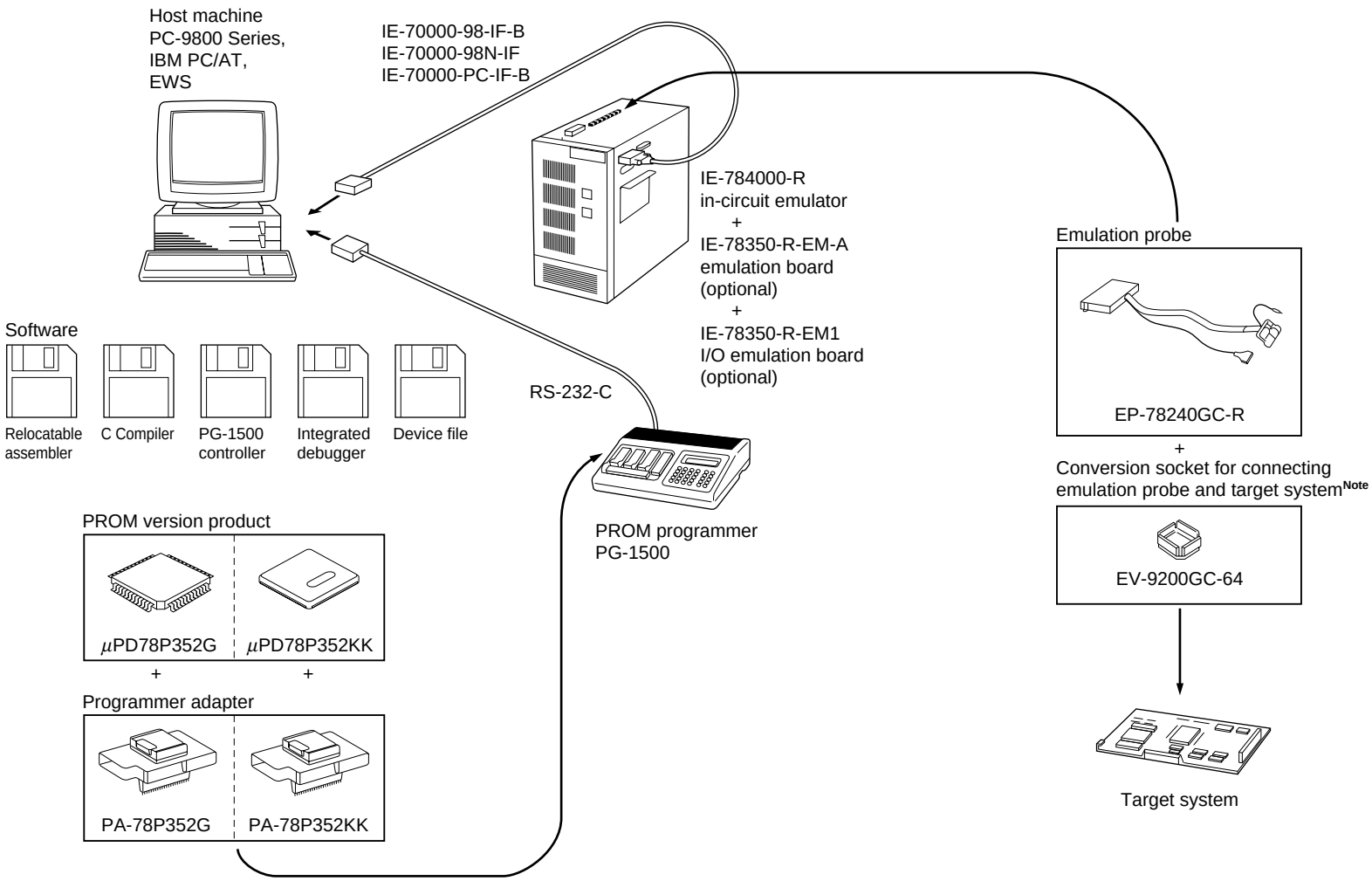
★ **Debugging Tools (When using integrated debuggers)**

Hardware	IE-784000-R	IE-78400-R is an in-circuit emulator that can be used to develop and debug application systems. For debugging, connect a host machine to the in-circuit emulator.			
	IE-78350-R-EM-A	IE-78355-R-EM-A is an emulation board to emulate peripheral functions such as input/output ports on the target device.			
	IE-78350-R-EM1	IE-78350-R-EM1 is an I/O emulation board to emulate peripheral functions such as input/output ports on the target device.			
	EP-78240GC-R EV-9200GC-64	Emulation probe for connecting IE-784000-R to the target system. One conversion socket EV-9200GC-64 is attached to connect the EP-78350-R-EM1 to the target system.			
	IE-70000-98-IF-B	Interface adapter and cable to use the PC-9800 series (except for a notebook computer) as a host machine.			
	IE-70000-98N-IF	Interface adapter and cable to use a notebook of the PC-9800 series as a host machine.			
	IE-70000-PC-IF-B	Interface adapter and cable to use the IBM PC/AT computer as a host machine.			
	IE-78000-R-SV3	Interface board to use the EWS machine as a host machine.			
Software	Integrated debugger (ID78K3)	Program to control the in-circuit emulator for the 78K/III Series. This debugger is used combined with the device file (DF78350). ID78K3 can debug the program at the source program level in the C language, structured assembly language, or assembly language. In addition, ID78K3 can divide the screen of the host machine and display a plenty of information at one time. This enables an efficient debugging.			
		Host machine	OS	Distributed media	Ordering code (Product name)
		PC-9800 Series	MS-DOS + Windows™	3.5-inch 2HD	μSAA13ID78K3
		IBM PC/AT and it's compatibles (Windows Japanese version)	PC DOS + Windows	3.5-inch 2HC	μSAB13ID78K3
		IBM PC/AT and it's compatibles (Windows English version)			μSBB13ID78K3
	Device file (DF78350)	This file contains the device-specific information. Use this file with the combination of Assembler (RA78K3) and C Compiler (CC78K3).			
		Host machine	OS	Distributed media	Ordering code (Product name)
		PC-9800 Series	MS-DOS	3.5-inch 2HD	μS5A13DF78350
				5-inch 2HD	μS5A10DF78350
		IBM PC/AT and it compatibles	PC DOS	3.5-inch 2HC	μS7B13DF78350
				5-inch 2HC	μS7B10DF78350

Remark The operation of the integrated debugger and device file is guaranteed only on the host machines under the operating systems listed above.

Phase-out/Discontinued

★ **Development tool configuration (When using the integrated debugger)**



Note The conversion socket is attached to the emulation probe

Remarks 1. Host machine is represented by desktop personal computer.

2. Medium supplying software is represented by 3.5-inch floppy diskette in this figure.

Evaluation Tools

The following evaluation tools (evaluation boards) are provided to evaluate the function of the μPD78352A Subseries. The evaluation board enables you to easily evaluate the μPD78352A Subseries function. However, application systems (application programs) cannot be developed by using the evaluation board. For this purpose, development tools are required.

Ordering code (product name)	Host machine	Function
EB-78350-98	PC-9800 Series	The function of the μPD78352A Subseries can be easily evaluated by connecting the evaluation tool to a host machine. Since the command system of the products is compliant with the IE-78350-R command system, easy movement to application system development process by IE-78350-R can be made.
EB-78350-PC	IBM PC/AT	

Caution EB-78350-98/PC is not a μPD78352A Subseries application system development tool.

A.3 μPD78356 Subseries Tools

PROM write tools

Hardware	PG-1500	PG-1500 is a PROM programmer which enables you to program single chip microcomputers containing PROM by stand-alone or host machine operation by connecting an attached board and optional programmer adapter to PG-1500. It also enables you to program typical PROM devices of 256K bits to 4M bits.			
	PA-78P356GC PA-78P356GD PA-78P356KP	PROM program adapters required to program the μPD78P356 on a general-purpose PROM programmer such as the PG-1500. PA-78P356GC for μPD78P356GC PA-78P356GD for μPD78P356GD PA-78P356KP for μPD78P356KP			
Software	PG-1500 controller	PG-1500 and a host machine are connected by a serial or parallel interface for controlling the PG-1500 on the host machine.			
		Host machine	OS	Distributed media	Ordering code (product name)
		PC-9800 Series	MS-DOS	3.5-inch 2HD	μS5A13PG1500
				5-inch 2HD	μS5A10PG1500
★		IBM PC/AT and it's compatibles	PC DOS	3.5-inch 2HD	μS7B13PG1500
				5-inch 2HC	μS7B10PG1500

Remark The operation of the PG-1500 controller is guaranteed only on the host machines under the operating systems listed above.

Phase-out/Discontinued

Debugging tools (When using the IE controller)

Hardware	IE-78350-R		IE-78350-R is an in-circuit emulator that can be used to develop and debug application systems. For debugging, connect a host machine to the in-circuit emulator.		
	IE-78355-R-EM1		IE-78355-R-EM1 is an I/O emulation board to emulate peripheral functions of input/output ports, etc., of the μPD78356 Subseries.		
	EP-78355GC-R		Emulation probe for 100-pin QFP of the μPD78356 Subseries. Use the emulation probe to connect IE-78350-R and target system.		
		TGC-100SDW	One piece of conversion socket TGC-100SDW used for connection to the target system is attached.		
	EP-78355GD-R		Emulation probe for 120-pin QFP of the μPD78356 Subseries. Use the emulation probe to connect IE-78350-R and target system.		
		EV-9200GD-120	One piece of conversion socket EV-9200GD-120 used for connection to the target system is attached.		
		EV-9501GC-100 + TGC-100SDW	By connecting to a 100-pin QFP conversion adapter EV-9501GC-100 (optional), 100-pin QFP of the μPD78356 Subseries can also be developed. However, to connect to the target system, use a conversion adapter TGC-100SDW (optional).		
Software	IE-78350-R control program (IE controller)		Program to control IE-78350-R on a host machine. Automatic execution of commands, etc., is enabled for more efficient debugging.		
			Host machine	OS	Distributed media	Ordering code (product name)
			PC-9800 Series	MS-DOS	3.5-inch 2HD	μS5A13IE78355
					5-inch 2HD	μS5A10IE78355
★			IBM PC/AT and it's compatibles	PC DOS	3.5-inch 2HC	μS7B13IE78355
					5-inch 2HC	μS7B10IE78355

Remarks 1. A conversion adapter TGC-100SDW is manufactured by TOKYO ELETECH Corporation. Consult your local NEC representative when you order.

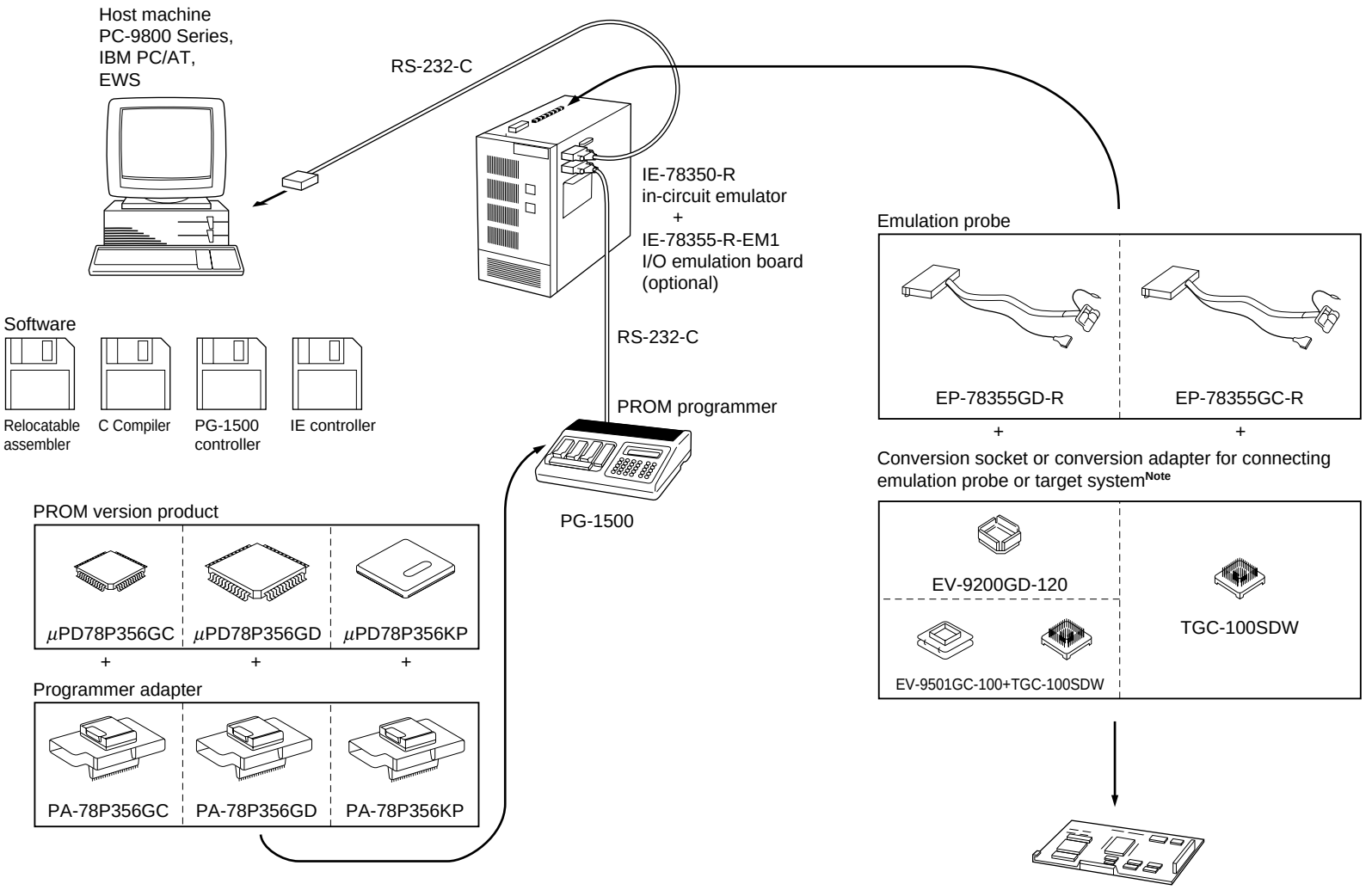
2. The operation of the IE controller is guaranteed only on the host machines under the operating systems listed above.

Phase-out/Discontinued

Development tool connection list

Development tool / Target device	In-circuit emulator	Emulation probe/ EPROM product	Conversion adapter	Conversion socket/ conversion adapter
GC package (100-pin QFP)	IE-78350-R + IE-78355-R-EM1	EP-78355GC-R	–	TGC-100SDW
		EP-78355GD-R	EV-9501GC-100	
	–	μPD78P356KP (120-pin WQFN)		
GD package (120-pin QFP)	IE-78350-R + IE-78355-R-EM1	EP-78355GD-R	–	EV-9200GD-120
	–	μPD78P356KP (120-pin WQFN)		

Development tool configuration (When using the integrated debugger)



Remarks 1. Host machine and PG-1500 can be directly connected by RS-232-C

2. Medium supplying software is represented by 3.5-inch floppy diskette in this figure.

Phase-out/Discontinued

★ **Debugging Tools (When using integrated debuggers)**

Hardware	IE-784000-R	IE-784000-R is an in-circuit emulator that can be used to develop and debug application systems. For debugging, connect a host machine to the in-circuit emulator.
	IE-78350-R-EM-A	IE-78350-R-EM-A is an I/O emulation board to emulate peripheral functions such as input/output ports on the target device.
	IE-78355-R-EM1	IE-78355R-EM1 is an I/O emulation board to emulate peripheral functions such as input/output ports on the target device.
	EP-78355GC-R TGC-100SDW	Emulation probe for 100-pin QFP to connect IE-784000-R to the target system. One conversion socket TGC-100SDW is attached to connect the EP-78355-GC-R to the target system.
	EP-78355GD-R EV-9200GD-120 EV-9501GC-100 + TGC-100SDW	Emulation probe for 120-pin QFP to connect the μPD78356 Subseries to the target system. One conversion socket EV-9200GD-120 is attached to connect the EP-78355-GD-R to the target system. In addition, connecting to a 100-pin QFP conversion adapter EV-9501GC-100 (optional) enables the development of 100-pin QFP of the μPD78356. Use the conversion adapter TGC-100SDW (optional) to connect the target system.
	IE-70000-98-IF-B	Interface adapter to use the PC-9800 series (except for a notebook computer) as a host machine.
	IE-70000-98N-IF	Interface adapter and cable to use a notebook of the PC-9800 series as a host machine.
	IE-70000-PC-IF-B	Interface adapter to use the IBM PC/AT computer as a host machine.
	IE-78000-R-SV3	Interface adapter and cable to use the EWS machine as a host machine.
Software	Integrated debugger (ID78K3)	Program to control the in-circuit emulator for the 78K/III Series. This debugger is used combined with the device file (DF78355). ID78K3 can debug the program at the source program level in the C language, structured assembly language, or assembly language. In addition, ID78K3 can divide the screen of the host machine and display a plenty of information at one time. This enables an efficient debugging.

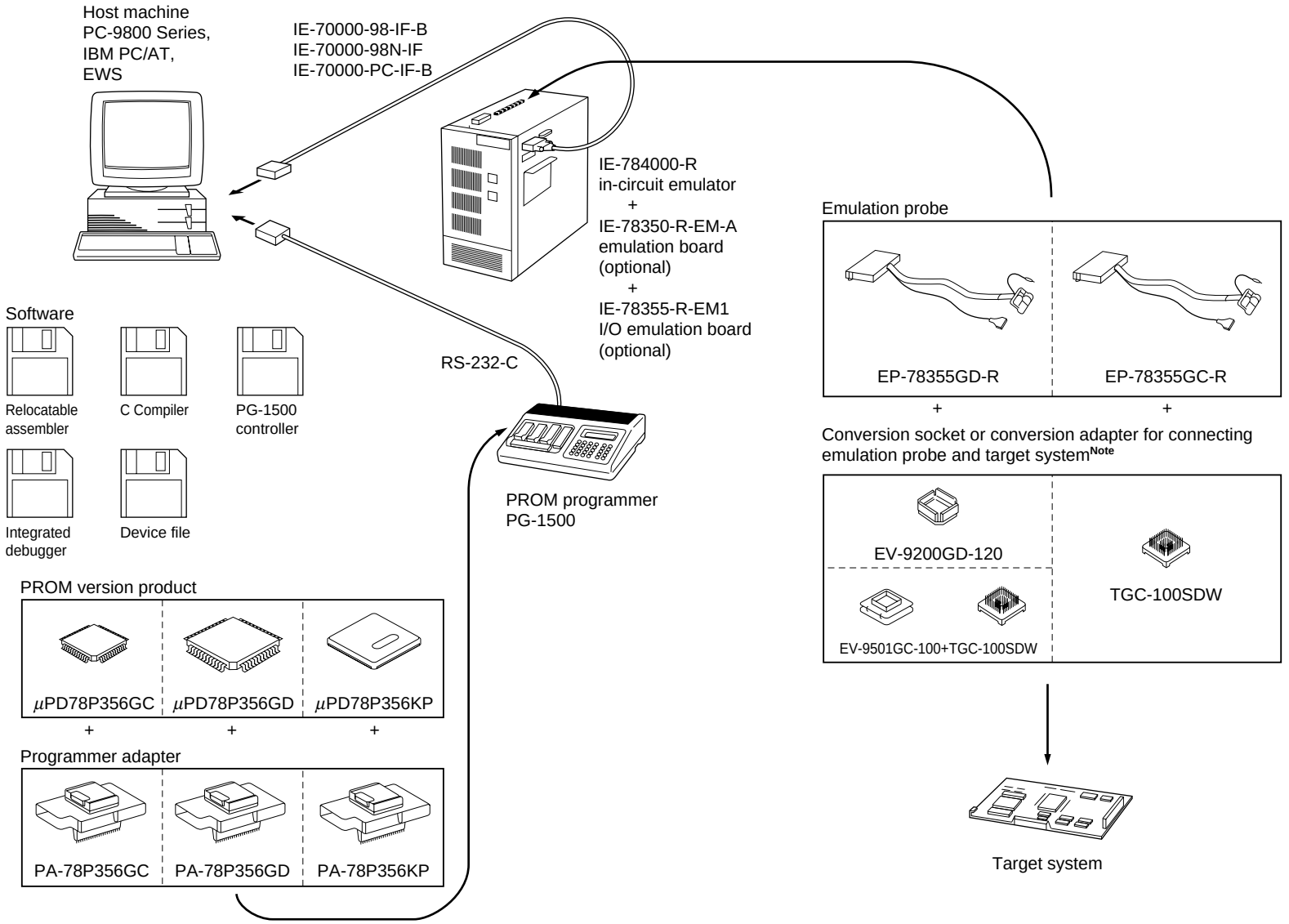
Host machine	OS	Distributed media	Ordering code (Product name)
PC-9800 Series	MS-DOS + Windows	3.5-inch 2HD	μSAA13ID78K3
IBM PC/AT and its compatibles (Windows Japanese version)	PC DOS + Windows	3.5-inch 2HC	μSAB13ID78K3
IBM PC/AT and its compatibles (Windows English version)			μSBB13ID78K3

	Devices file (DF78355)	This file contains the device-specific information. Use this file with the combination of Assembler (RA78K3) and C Compiler (CC78K3).

Host machine	OS	Distributed media	Ordering code (Product name)
PC-9800 Series	MS-DOS	3.5-inch 2HD	μS5A13DF78355
		5-inch 2HD	μS5A10DF78355
IBM PC/AT and it compatibles	PC DOS	3.5-inch 2HC	μS7B13DF78355
		5-inch 2HC	μS7B10DF78355

Remarks 1. A conversion adapter TGC-100SDW is manufactured by TOKYO ELETECH Corporation.

2. The operation of the integrated debugger and device file is guaranteed only on the host machines under the operating systems listed above.



Remarks 1. Host machine is represented by desktop personal computer.

2. Medium supplying software is represented by 3.5-inch floppy diskette in this figure.

Phase-out/Discontinued

A.4 μPD78366A Subseries Tools

PROM write tools

Hardware	PG-1500	PG-1500 is a PROM programmer which enables you to program single chip microcomputers containing PROM by stand-alone or host machine operation by connecting an attached board and optional programmer adapter to PG-1500. It also enables you to program typical PROM devices of 256K bits to 4M bits.			
	PA-78P364CW PA-78P368GF PA-78P368KL	PROM program adapters required to program the μPD78P364A, 78P368A on a general-purpose PROM programmer such as the PG-1500. PA-78P364CW for μPD78P364ACW PA-78P368GF for μPD78P368AGF PA-78P368KL for μPD78P368AKL			
Software	PG-1500 controller	PG-1500 and a host machine are connected by a serial or parallel interface for controlling the PG-1500 on the host machine.			
		Host machine	OS	Distributed media	Ordering code (product name)
		PC-9800 Series	MS-DOS	3.5-inch 2HD	μS5A13PG1500
				5-inch 2HD	μS5A10PG1500
★		IBM PC/AT and it's compatibles	PC DOS	3.5-inch 2HD	μS7B13PG1500
				5-inch 2HC	μS7B10PG1500

Remark The operation of the PG-1500 controller is guaranteed only on the host machines under the operating systems listed above.

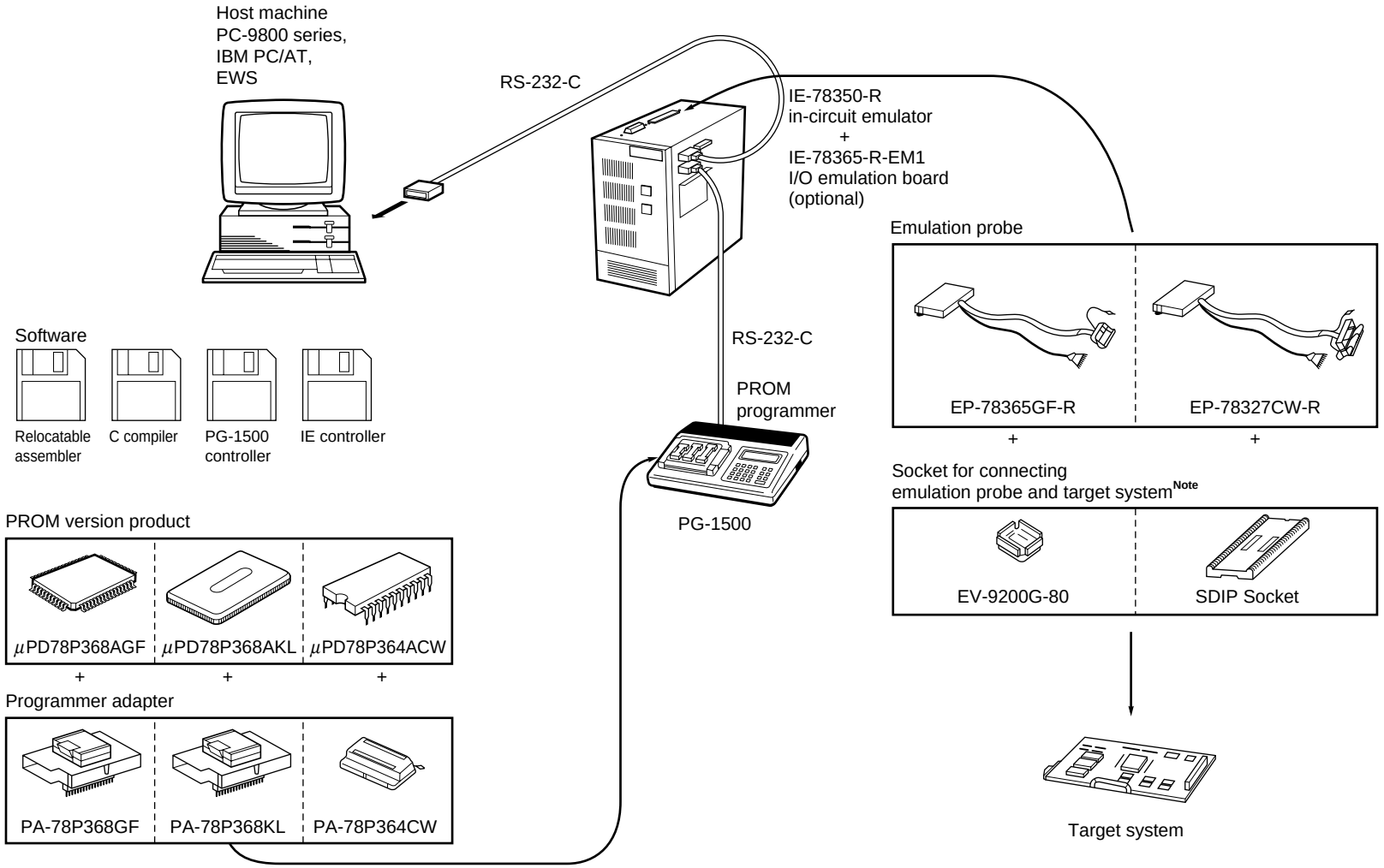
Phase-out/Discontinued

Debugging tools (When using the IE controller)

Hardware	IE-78350-R	IE-78350-R is an in-circuit emulator that can be used to develop and debug application systems. For debugging, connect a host machine to the in-circuit emulator.			
	IE-78365-R-EM1	IE-78365-R-EM1 is an I/O emulation board to emulate peripheral functions of input/output ports, etc., of the μPD78366A Subseries.			
	EP-78327CW-R	Emulation probe for 64-pin shrink DIP of the μPD78366A Subseries. Use the emulation probe to connect IE-78350-R and target system.			
	EP-78365GF-R EV-9200G-80	Emulation probe for 80-pin QFP of the μPD78366A Subseries. Use the emulation probe to connect IE-78350-R and target system. One piece of conversion socket EV-9200G-80 used for connection to the target system is attached.			
Software	IE-78350-R control program (IE controller)	Program to control IE-78350-R on a host machine. Automatic execution of commands, etc., is enabled for more efficient debugging.			
		Host machine	OS	Distributed media	Ordering code (product name)
		PC-9800 Series	MS-DOS	3.5-inch 2HD	μS5A13IE78365A
				5-inch 2HD	μS5A10IE78365A
★		IBM PC/AT and it's compatibles	PC DOS	3.5-inch 2HC	μS7B13IE78365A
				5-inch 2HC	μS7B10IE78365A

Remark The operation of the IE controller is guaranteed only on the host machines under the operating systems listed above.

Development tool configuration (When using the IE controller)



Note The EV-9200G-80 is attached to the emulation probe. Use the socket for SDIP which is sold on the market.

Remarks 1. The host machine and PG-1500 can also be connected directly by RS-232-C for use.

2. Medium supplying software is represented by 3.5-inch floppy diskette in this figure.

Phase-out/Discontinued

★ Debugging Tools (When using integrated debuggers)

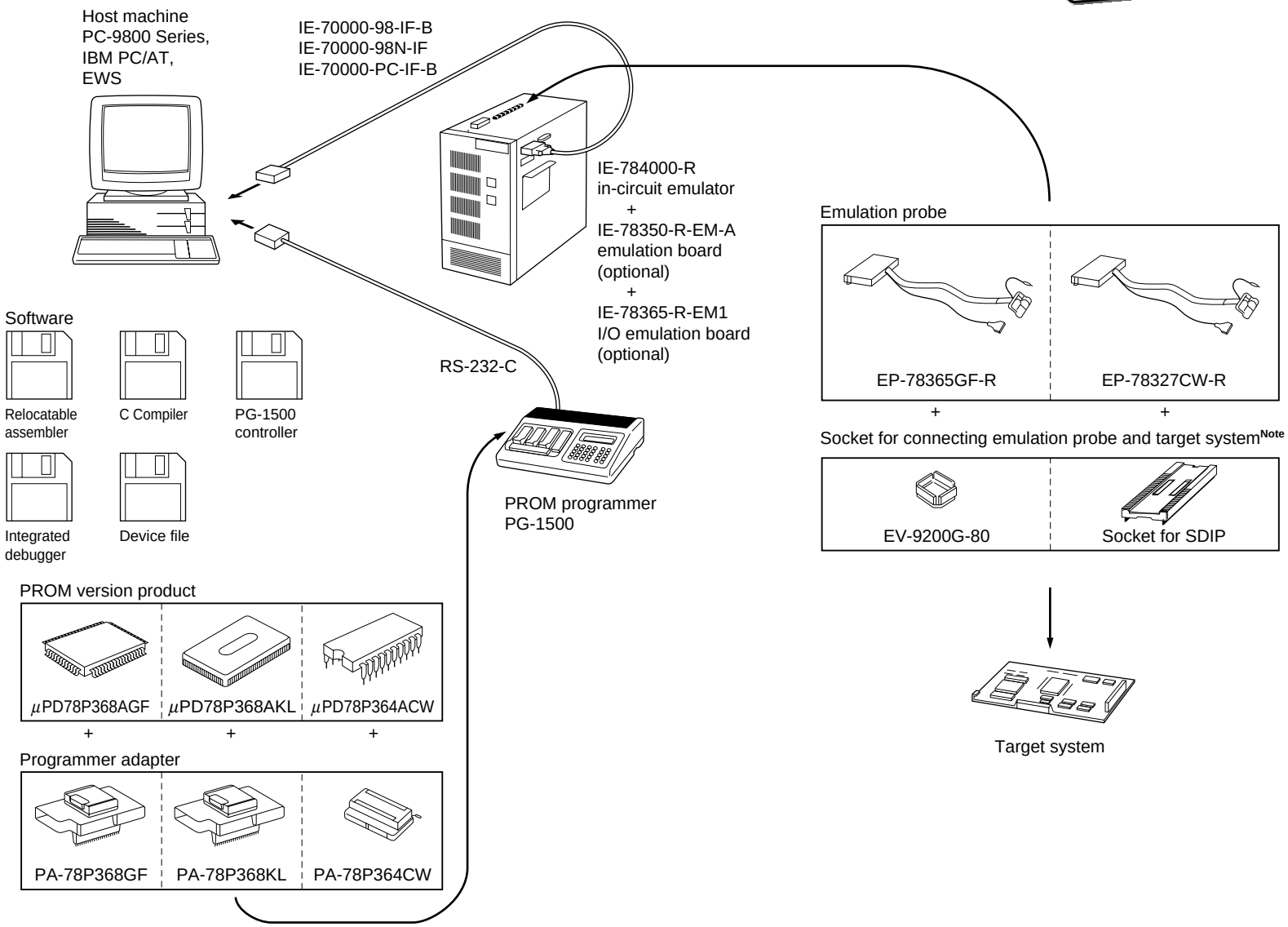
Hardware	IE-784000-R	IE-784000-R is an in-circuit emulator that can be used to develop and debug application systems. For debugging, connect a host machine to the in-circuit emulator.
	IE-78350-R-EM-A	IE-78350-R-EM-A is an I/O emulation board to emulate peripheral functions such as input/output ports on the target device.
	IE-78365-R-EM1	IE-78365R-EM1 is an I/O emulation board to emulate peripheral functions such as input/output ports on the target device.
	EP-78327CW-R	Emulation probe for 64-pin shrink DIP to connect IE-78400-R to the target system.
	EP-78365GF-R EV-9200G-80	Emulation probe for 80-pin QFP to connect IE-784000-R to the target system. One conversion socket EV-9200G-80 is attached to connect the EP-78365GF-R to the target system.
	IE-70000-98-IF-B	Interface adapter to use the PC-9800 series (except for a notebook computer) as a host machine.
	IE-70000-98N-IF	Interface adapter and cable to use a notebook of the PC-9800 series as a host machine.
	IE-70000-PC-IF-B	Interface adapter to use the IBM PC/AT computer as a host machine.
	IE-78000-R-SV3	Interface adapter and cable to use the EWS machine as a host machine.
Software	Integrated debugger (ID78K3)	Program to control the in-circuit emulator for the 78K/III Series. This debugger is used combined with the device file (DF78365). ID78K3 can debug the program at the source program level in the C language, structured assembly language, or assembly language. In addition, ID78K3 can divide the screen of the host machine and display a plenty of information at one time. This enables an efficient debugging.

Host machine	OS	Distributed media	Ordering code (Product name)
PC-9800 Series	MS-DOS + Windows	3.5-inch 2HD	μSAA13ID78K3
IBM PC/AT and its compatibles (Windows Japanese version)	PC DOS + Windows	3.5-inch 2HC	μSAB13ID78K3
IBM PC/AT and its compatibles (Windows English version)			μSBB13ID78K3

Software	Devices file (DF78365)	This file contains the device-specific information. Use this file with the combination of Assembler (RA78K3) and C Compiler (CC78K3).

Host machine	OS	Distributed media	Ordering code (Product name)
PC-9800 Series	MS-DOS	3.5-inch 2HD	μS5A13DF78365
		5-inch 2HD	μS5A10DF78365
IBM PC/AT and it compatibles	PC DOS	3.5-inch 2HC	μS7B13DF78365
		5-inch 2HC	μS7B10DF78365

Remark The operation of the integrated debugger and device file is guaranteed only on the host machines under the operating systems listed above.



Note EV-9200G-80 is attached to the emulation probe. Use the socket for SDIP which is sold on the market.

Remarks 1. Host machine is represented by desktop personal computer.

2. Medium supplying software is represented by 3.5-inch floppy diskette in this figure.

Phase-out/Discontinued

A.5 μPD78372 Subseries Tools

PROM write tools

Hardware	PG-1500	PG-1500 is a PROM programmer which enables you to program single chip microcomputers containing PROM by stand-alone or host machine operation by connecting an attached board and optional programmer adapter to PG-1500. It also enables you to program typical PROM devices of 256K bits to 4M bits.			
★ Hardware	PA-78P372GC PA-78P372GF PA-78P372KL	PROM program adapters required to program the μPD78P372(A) on a general-purpose PROM programmer such as the PG-1500. PA-78P372GC for μPD78P372GC(A) PA-78P372GF for μPD78P372GF(A) PA-78P372KL for μPD78P372KL			
Software	PG-1500 controller	PG-1500 and a host machine are connected by a serial or parallel interface for controlling the PG-1500 on the host machine.			
		Host machine	OS	Distributed media	Ordering code (product name)
		PC-9800 Series	MS-DOS	3.5-inch 2HD	μS5A13PG1500
				5-inch 2HD	μS5A10PG1500
★		IBM PC/AT and it's compatibles	PC DOS	3.5-inch 2HD	μS7B13PG1500
				5-inch 2HC	μS7B10PG1500

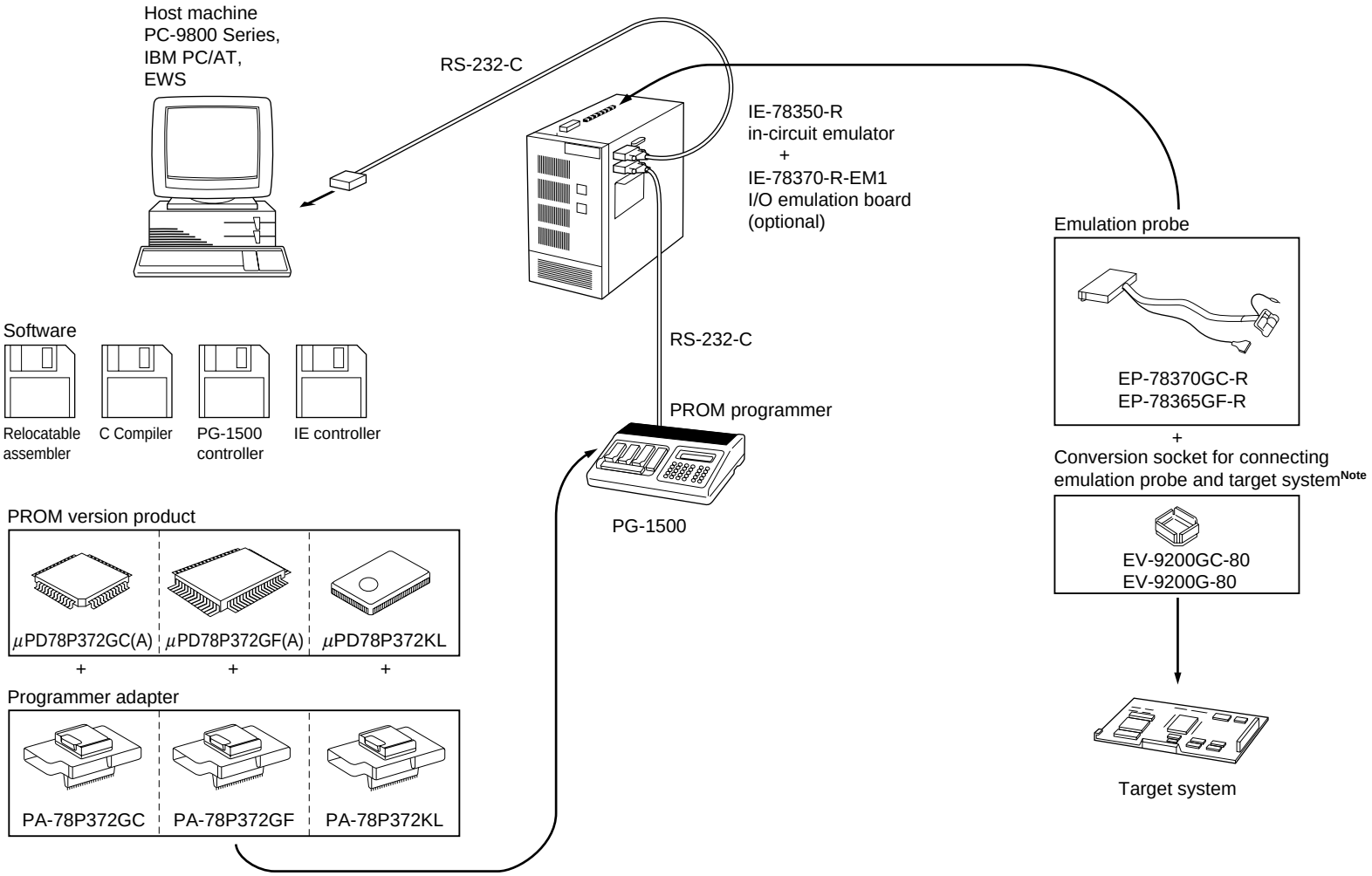
Remark The operation of the PG-1500 controller is guaranteed only on the host machines under the operating systems listed above.

Phase-out/Discontinued

Debugging tools (When using the IE controller)

Hardware	IE-78350-R	IE-78350-R is an in-circuit emulator that can be used to develop and debug application systems. For debugging, connect a host machine to the in-circuit emulator.			
	IIE-78370-R-EM1	IE-78370-R-EM1 is an I/O emulation board to emulate peripheral functions of input/output ports, etc., of the μPD78372 Subseries.			
★	EP-78370GC-R EV-9200GC-80	Emulation probe for 80-pin QFP (14 × 14 mm) of the μPD78372 Subseries. Use the emulation probe to connect IE-78350-R to the target system. One piece of conversion socket EV-9200GC-80 used for connection to the target system is attached.			
	EP-78365GF-R EV-9200G-80	Emulation probe for 80-pin QFP (14 × 20 mm) of the μPD78372 Subseries. Use the emulation probe to connect IE-78350-R to the target system. One piece of conversion socket EV-9200G-80 used for connection to the target system is attached.			
Software	IE-78350-R control program (IE controller)	Program to control IE-78350-R on a host machine. Automatic execution of commands, etc., is enabled for more efficient debugging.			
		Host machine	OS	Distributed media	Ordering code (product name)
		PC-9800 Series	MS-DOS	3.5-inch 2HD	μS5A13IE78370
				5-inch 2HD	μS5A10IE78370
★		IBM PC/AT and it's compatibles	PC DOS	3.5-inch 2HC	μS7B13IE78370
				5-inch 2HC	μS7B10IE78370

Remark The operation of the IE controller is guaranteed only on the host machines under the operating systems listed above.



Note The conversion socket is attached to the emulation probe.

Remarks 1. Host machine and PG-1500 can also be directly connected by RS-232-C.

2. Medium supplying software is represented by 3.5-inch floppy diskette in this figure.

Phase-out/Discontinued

★ **Debugging Tools (When using integrated debuggers)**

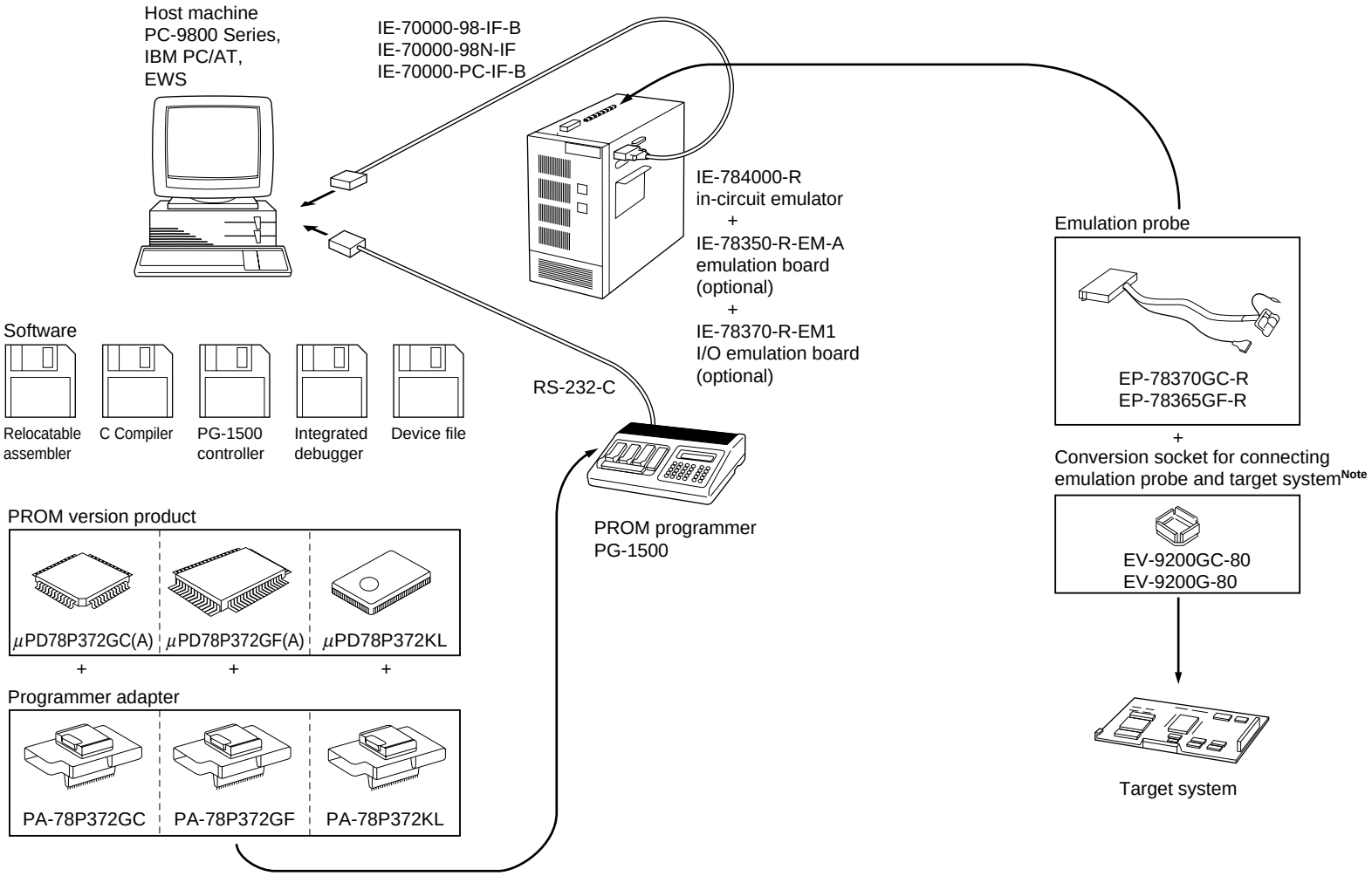
Hardware	IE-784000-R	IE-784000-R is an in-circuit emulator that can be used to develop and debug application systems. For debugging, connect a host machine to the in-circuit emulator.
	IE-78350-R-EM-A	IE-78350-R-EM-A is an I/O emulation board to emulate peripheral functions such as input/output ports on the target device.
	IE-78370-R-EM1	IE-78370R-EM1 is an I/O emulation board to emulate peripheral functions such as input/output ports on the target device.
	EP-78370GC-R EV-9200GC-80	Emulation probe for connecting IE-784000-R to the target system. One conversion socket EV-9200GC-80 is attached to connect the EP-78370GC-R to the target system.
	EP-78365GF-R EV-9200G-80	Emulation probe for connecting IE-784000-R to the target system. One conversion socket EV-9200G-80 is attached to connect the EP-78365GF-R to the target system.
	IE-70000-98-IF-B	Interface adapter to use the PC-9800 series (except for a notebook computer) as a host machine.
	IE-70000-98N-IF	Interface adapter and cable to use a notebook of the PC-9800 series as a host machine.
	IE-70000-PC-IF-B	Interface adapter to use the IBM PC/AT computer as a host machine.
	IE-78000-R-SV3	Interface adapter and cable to use the EWS machine as a host machine.
Software	Integrated debugger (ID78K3)	Program to control the in-circuit emulator for the 78K/III Series. This debugger is used combined with the device file (DF78370). ID78K3 can debug the program at the source program level in the C language, structured assembly language, or assembly language. In addition, ID78K3 can divide the screen of the host machine and display a plenty of information at one time. This enables an efficient debugging.

Host machine	OS	Distributed media	Ordering code (Product name)
PC-9800 Series	MS-DOS + Windows	3.5-inch 2HD	μSAA13ID78K3
IBM PC/AT and its compatibles (Windows Japanese version)	PC DOS + Windows	3.5-inch 2HC	μSAB13ID78K3
IBM PC/AT and its compatibles (Windows English version)			μSBB13ID78K3

Software	Devices file (DF78370)	This file contains the device-specific information. Use this file with the combination of Assembler (RA78K3) and C Compiler (CC78K3).

Host machine	OS	Distributed media	Ordering code (Product name)
PC-9800 Series	MS-DOS	3.5-inch 2HD	μS5A13DF78370
		5-inch 2HD	μS5A10DF78370
IBM PC/AT and it compatibles	PC DOS	3.5-inch 2HC	μS7B13DF78370
		5-inch 2HC	μS7B10DF78370

Remark The operation of the integrated debugger and device file is guaranteed only on the host machines under the operating systems listed above.



Note EV-9200G-80 is attached to the emulation probe.

Remarks 1. Host machine is represented by desktop personal computer.

2. Medium supplying software is represented by 3.5-inch floppy diskette in this figure.

★ A.6 Embedded Software

The following embedded software is available to perform the more efficient development or maintenance of the application program.

Real-time OS

<table>
<tr><td rowspan="6">Real-time OS
(RX78K/III)Note</td><td colspan="3">The RX78K/III is intended to realize multi-task environments in areas where real-time features are required. This OS can improve the performance of the entire system by allocating the idle time of the CPU to the other processing.
The RX78K/III supplies the system call conforming to the μITRON specifications.
RX78K/III package provides a tool (configurator) that is used to create the nucleus of the RX78K/III and several information tables.</td></tr>
<tr><td rowspan="2">Host machine</td><td colspan="2" rowspan="2">Ordering code
(Product name)</td></tr>
<tr><td>OS</td><td>Distributed media</td></tr>
<tr><td>PC-9800 Series</td><td>MS-DOS</td><td>3.5-inch 2HD</td><td>Undefined</td></tr>
<tr><td></td><td></td><td>5-inch 2HD</td><td>Undefined</td></tr>
<tr><td>IBM PC/AT
and its compatibles</td><td>PC DOS</td><td>3.5-inch 2HC</td><td>Undefined</td></tr>
<tr><td></td><td></td><td>5-inch 2HC</td><td>Undefined</td></tr>
</table>

Note Under development

Caution When purchasing the RX78K/III, fill out the necessary forms and conclude a contract with NEC.

Remark RX78K/III Real-time OS requires the RA78K3 assembler package (sold separately).

Phase-out/Discontinued

Fuzzy Inference Development Support System

Fuzzy knowledge data creation tool (FE9000, FE9200)	This program supports input of fuzzy knowledge data (fuzzy rule and membership function), editing (edit), and evaluation (simulation).			
	Host machine	OS	Distributed media	Ordering code (Product name)
	PC-9800 Series	MS-DOS	3.5-inch 2HD	μS5A13FE9000
			5-inch 2HD	μS5A10FE9000
	IBM PC/AT and its compatibles	PC DOS + Windows	3.5-inch 2HC	μS7B13FE9200
			5-inch 2HC	μS7B10FE9200
Translator (FT78K3)Note	This program converts fuzzy knowledge data obtained by using fuzzy knowledge data preparation tool to RA78K3 assembler source program.			
	Host machine	OS	Distributed media	Ordering code (Product name)
	PC-9800 Series	MS-DOS	3.5-inch 2HD	μS5A13FT78K3
			5-inch 2HD	μS5A10FT78K3
	IBM PC/AT and its compatibles	PC DOS	3.5-inch 2HC	μS7B13FT78K3
			5-inch 2HC	μS7B10FT78K3
Fuzzy inference module (FI78K/III)Note	This program executes fuzzy inference by linking fuzzy knowledge data converted by translator.			
	Host machine	OS	Distributed media	Ordering code (Product name)
	PC-9800 Series	MS-DOS	3.5-inch 2HD	μS5A13FI78K3
			5-inch 2HD	μS5A10FI78K3
	IBM PC/AT and its compatibles	PC DOS	3.5-inch 2HC	μS7B13FI78K3
			5-inch 2HC	μS7B10FI78K3
Fuzzy inference debugger (FD78K/III)	This software supports evaluating and adjusting fuzzy knowledge data at hardware level by using in-circuit emulator.			
	Host machine	OS	Distributed media	Ordering code (Product name)
	PC-9800 Series	MS-DOS	3.5-inch 2HD	μS5A13FD78K3
			5-inch 2HD	μS5A10FD78K3
	IBM PC/AT and its compatibles	PC DOS	3.5-inch 2HC	μS7B13FD78K3
			5-inch 2HC	μS7B10FD78K3

Note Under development

[MEMO]

APPENDIX B INSTRUCTION INDEX (MNEMONICS BY FUNCTION)

- **8-bit data transfer instructions**
  MOV ... 140
  XCH ... 141

- **16-bit data transfer instructions**
  MOVW ... 143
  XCHW ... 144

- **8-bit operation instructions**
  ADD ... 146
  ADDC ... 147
  SUB ... 149
  SUBC ... 150
  AND ... 152
  OR ... 153
  XOR ... 154
  CMP ... 155

- **16-bit operation instructions**
  ADDW ... 157
  SUBW ... 158
  CMPW ... 159

- **Multiplication and division instructions**
  MULU ... 161
  DIVUW ... 162
  MULUW ... 163
  DIVUX ... 164

- **Signed multiplication instruction**
  MULW ... 166

- **Multiplication and accumulation instruction**
  MACW ... 168

- **Multiplication and accumulation instruction with saturation function**
  MACSW ... 171

- **Correlation operation instruction**
  SACW ... 174

- **Table shift instruction**
  MOVTBLW ... 177

- **Increment and decrement instructions**
  INC ... 179
  DEC ... 180
  INCW ... 181
  DECW ... 182

- **Shift and rotate instructions**
  ROR ... 184
  ROL ... 185
  RORC ... 186
  ROLC ... 187
  SHR ... 188
  SHL ... 189
  SHRW ... 190
  SHLW ... 191
  ROR4 ... 192
  ROL4 ... 193

- **BCD adjustment instructions**
  ADJBA ... 195
  ADJBS ... 196

- **Data conversion instruction**
  CVTBW ... 198

- **Bit manipulation instructions**
  MOV1 ... 200
  AND1 ... 201
  OR1 ... 202
  XOR1 ... 203
  SET1 ... 204
  CLR1 ... 205
  NOT1 ... 206

- **Call and return instructions**
  CALL ... 237
  CALLF ... 238
  CALLT ... 239
  BRK ... 240
  RET ... 241
  RETB ... 242
  RETI ... 243

- **Stack handling instructions**
  - PUSH ... 216
  - PUSHU ... 217
  - POP ... 218
  - POPU ... 219
  - MOVW SP, src ... 220
  - MOVW AX, SP ... 220
  - INCW SP ... 221
  - DECW SP ... 222
- **Special functions**
  - CHKL ... 223
  - CHKLA ... 225
- **Unconditional branch instructions**
  - BR ... 227
- **Conditional branch instructions**
  - BC ... 229
  - BL ... 229
  - BNC ... 230
  - BNL ... 230
  - BZ ... 231
  - BE ... 231
  - BNZ ... 232
  - BNE ... 232
  - BV ... 233
  - BPE ... 233
  - BNV ... 234
  - BPO ... 234
  - BN ... 235
  - BP ... 236
  - BGT ... 237
  - BGE ... 238
  - BLT ... 239
  - BLE ... 240
  - BH ... 241
  - BNH ... 242
  - BT ... 243
  - BF ... 244
  - BTCLR ... 245
  - BFSET ... 246
  - DBNZ ... 247
- **Context switching instructions**
  - BRKCS ... 249
  - RETCS ... 250
  - RETCSB ... 251
- **String instructions**
  - MOVM ... 253
  - MOVBK ... 254
  - XCHM ... 255
  - XCHBK ... 256
  - CMPME ... 257
  - CMPBKE ... 259
  - CMPMNE ... 261
  - CMPBKNE ... 263
  - CMPMC ... 265
  - CMPBKC ... 267
  - CMPMNC ... 269
  - CMPBKNC ... 271
- **CPU control instructions**
  - MOV STBC, #byte ... 274
  - MOV WDM, #byte ... 275
  - SWRS ... 276
  - SEL RBn ... 277
  - SEL RBn, ALT ... 278
  - NOP ... 279
  - EI ... 280
  - DI ... 281

APPENDIX C INSTRUCTION INDEX (MNEMONICS BY ALPHABETICAL ORDER)

[A]

ADD ... 146
ADDC ... 147
ADDW ... 157
ADJBA ... 195
ADJBS ... 196
AND ... 152
AND1 ... 201

[B]

BC ... 229
BE ... 231
BF ... 244
BFSET ... 246
BGE ... 238
BGT ... 237
BH ... 241
BL ... 229
BLE ... 240
BLT ... 239
BN ... 235
BNC ... 230
BNE ... 232
BNH ... 242
BNL ... 230
BNV ... 234
BNZ ... 232
BP ... 236
BPE ... 233
BPO ... 234
BR ... 227
BRK ... 240
BRKCS ... 249
BT ... 243
BTCLR ... 245
BV ... 233
BZ ... 231

[C]

CALL ... 237
CALLF ... 238
CALLT ... 239
CHKL ... 223
CHKLA ... 225
CLR1 ... 205
CMP ... 155
CMPBKC ... 267
CMPBKE ... 259
CMPBKNC ... 271
CMPBKNE ... 263
CMPMC ... 265
CMPME ... 257
CMPMNC ... 269
CMPMNE ... 261
CMPW ... 159
CVTBW ... 198

[D]

DBNZ ... 247
DEC ... 180
DECW ... 182
DECW SP ... 222
DI ... 281
DIVUW ... 162
DIVUX ... 164

[E]

EI ... 280

[I]

INC ... 179
INCW ... 181
INCW SP ... 221

[M]

MACSW ... 171
MACW ... 168
MOV ... 140
MOV STBC, #byte ... 274
MOV WDM, #byte ... 275
MOV1 ... 200
MOVBK ... 254
MOVM ... 253
MOVTBLW ... 177
MOVW ... 143
MOVW AX, SP ... 220
MOVW SP, src ... 220

MULU ... 161
MULUW ... 163
MULW ... 166

[N]
NOP ... 279
NOT1 ... 206

[O]
OR ... 153
OR1 ... 202

[P]
POP ... 218
POPU ... 219
PUSH ... 216
PUSHU ... 217

[R]
RET ... 241
RETB ... 242
RETCS ... 250
RETCSB ... 251
RETI ... 243
ROL ... 185
ROL4 ... 193
ROLC ... 187
ROR ... 184
ROR4 ... 192
RORC ... 186

[S]
SACW ... 174
SEL RBn ... 277
SEL RBn, ALT ... 278
SET1 ... 204
SHL ... 189
SHLW ... 191
SHR ... 188
SHRW ... 190
SUB ... 149
SUBC ... 150
SUBW ... 158
SWRS ... 276

[X]
XCH ... 141
XCHBK ... 256
XCHM ... 255
XCHW ... 144
XOR ... 154
XOR1 ... 203

APPENDIX D REVISION HISTORY

The following table shows the revision history. The chapters appearing in the revised-chapter column indicate those of the corresponding edition.

Version	Major revisions from previous version	Range
Second	Adding the following products μPD78356(A), 78P356(A), 78361A, 78362A, 78P364A, 78363A, 78365A, 78366A, 78368A, 78P368A, 78372(A), 78372(A1), 78372(A2), 78P372(A), 78P372(A1), 78P372(A2)	Throughout
	Deleting the following products μPD78355A, 78356A, 78P356A, 78362, 78P364, 78365, 78366, 78P368, 78370, 78372, 78P372	
	Changing the status of the following products from developing to completed μPD78355, 78356, 78P356	
	Adding the description about the debugging tools when using the integrated debugger	APPENDIX A TOOLS
	Adding Section **A.6 Embedded Software**	

[MEMO]

Phase-out/Discontinued

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: +55-11-889-1689

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 Corporation
Semiconductor Solution Engineering Division
Technical Information Support Dept.
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	❑	❑	❑	❑