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.

RENESAS

USER'S MANUAL

78K/II SERIES

8-BIT SINGLE-CHIP MICROCOMPUTER

INSTRUCTIONS

μPD78214 SUB-SERIES
μPD78218A SUB-SERIES
μPD78224 SUB-SERIES
μPD78234 SUB-SERIES
μPD78244 SUB-SERIES

Document No. U10228EJ6V0UM00 (6th edition)
(Previous No. IEU-1311)
Date Published December 1995 P
Printed in Japan

Cautions on CMOS Devices

① **Countermeasures against static electricity for all MOSs**

Caution When handling MOS devices, take care so that they are not electrostatically charged.

Strong static electricity may cause dielectric breakdown in gates. When transporting or storing MOS devices, use conductive trays, magazine cases, shock absorbers, or metal cases that NEC uses for packaging and shipping. Be sure to ground MOS devices during assembling. Do not allow MOS devices to stand on plastic plates or do not touch pins.

Also handle boards on which MOS devices are mounted in the same way.

② **CMOS-specific handling of unused input pins**

Caution Hold CMOS devices at a fixed input level.

Unlike bipolar or NMOS devices, if a CMOS device is operated with no input, an intermediate-level input may be caused by noise. This allows current to flow in the CMOS device, resulting in a malfunction. Use a pull-up or pull-down resistor to hold a fixed input level. Since unused pins may function as output pins at unexpected times, each unused pin should be separately connected to the V_{DD} or GND pin through a resistor.

If handling of unused pins is documented, follow the instructions in the document.

③ **Statuses of all MOS devices at initialization**

Caution The initial status of a MOS device is unpredictable when power is turned on.

Since characteristics of a MOS device are determined by the amount of ions implanted in molecules, the initial status cannot be determined in the manufacture process. NEC has no responsibility for the output statuses of pins, input and output settings, and the contents of registers at power on. However, NEC assures operation after reset and items for mode setting if they are defined.

When you turn on a device having a reset function, be sure to reset the device first.

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 Changes

Page	Description
p.3, 9, 18	A disaster/crime prevention unit has been added as a special product in applications of the μPD78214 sub-series, μPD78218A sub-series, and μPD78234 sub-series.
p.170 to p.177	(Z, AC, and CY flags also do not change) has been added to the sentence, "If the second operand (cnt) is 0, no processing is performed." in [Description] of ROR, ROL, RORC, ROLC, SHR, SHL, SHRW, and SHLW.
p.223	**Chapter 9** • Description related to the 3.5-inch 2HC has been added to IBM PC/AT. • The HP9000 series 300 has been changed to the HP9000 series 700.
p.237	• The screen debugger of IBM PC/AT and the 5.25-inch 2HC of the device file have been already developed.
p.239	• **(4) OS for the IBM PC** has been added.
p.248	• The Fuzzy inference debugger (FD78K/II) has been already developed.
p.253	**Appendix C** has been added.

The mark ⋆ shows major revised points.

PREFACE

Intended readership

This manual is intended for engineers who wish to gain an understanding of the functions of the 78K/II series[Note] and design application systems using a device in this series.

Note 78K/II series products:

μPD78214 sub-series	: μPD78212, 78213, 78214, 78P214, 78212(A), 78213(A), 78214(A), 78P214(A)
μPD78218A sub-series	: μPD78217A, 78218A, 78P218A, 78218A(A)
μPD78224 sub-series	: μPD78220, 78224, 78P224
μPD78234 sub-series	: μPD78233, 78234, 78237, 78238, 78P238 78234(A), 78238(A)
μPD78244 sub-series	: μPD78243, 78244

Purpose

The purpose of this manual is to give users an understanding of the various instruction functions of the 78K/II series products.

Organization

This manual is broadly organized as follows:

- 78K/II series features
- 78K/II series products
- CPU functions
- Instruction set
- Description of instructions
- Development tools
- Built-in Software

Using the manual

When reading this manual, a general knowledge of electrical and logic circuits and microcomputers is necessary.

- To check details of the function of an instruction when the mnemonic is known:
  -> Use the instruction indexes in **Appendix A** and **B**.
- To check an instruction when the function is generally known but the mnemonic is not known:
  -> Find the mnemonic from **Chapter 7**, then check the function of the instruction in **Chapter 8**.
- To get an overview of the function's of the 78K/II series instructions:
  -> Read the manual in accordance with the table of contents.
- To learn about the hardware functions of the 78K/II series:
  -> Refer to the relevant separate User's Manual.
  - μPD78214 Sub-Series User's Manual (IEM-1236)
  - μPD78218A Sub-Series User's Manual (IEU-1313)
  - μPD78224 Sub-Series User's Manual (IEM-1215)
  - μPD78234 Sub-Series Use's Manual (IEU-1290)
  - μPD78244 Sub-Series User's Manual (IEU-1316)

- To learn about the electrical specifications of the 78K/II series:
  -> Refer to the relevant separate Data Sheet.
- To learn about application examples of the various functions of the 78K/II series products:
  -> Refer to the relevant separate Application Note.

Legend

Weighting in data notation	:	High-order digit on the left, low-order digit on the right
Active-low notation	:	$\overline{xxx}$ (line over pin/signal name)
Note	:	Explanation of text marked **"Note"**
Caution	:	Information to be noted carefully
Remarks	:	Supplementary information
Numeric notations	:	Binary : xxxxB or xxxx
		Decimal : xxxx
		Hexadecimal : xxxxH

Related documentation

- **Documents for entire 78K/II series**

Document		Document number
User's Manual, Instruction		This manual
SBI User's Manual		EEU-1303
Application Note	Basic	IEA-1220
	Application	IEA-1282
	Floating-Point Arthmetic Operation Program	IEA-1273
Selection Guide		IF-1160
Instruction Application Table		—
Instruction Set		—
Development Tools Selection Guide		EF-1114

- **Individual documents**

  - **μPD78214 sub-series**

Document \ Product name	μPD78212	μPD78213	μPD78214	μPD78P214
Brochure	—			
Data Sheet	IC-2526			IC-2481
User's Manual, Hardware	IEM-1236			
Mode Register Application Table	—			

Document \ Product name	μPD78212(A)	μPD78213(A)	μPD78214(A)	μPD78P214(A)
Brochure	—			
Data Sheet	IC-2831			IC-3095
User's Manual, Hardware	IEU-1236			
Mode Register Application Table	—			

- **μPD78218A sub-series**

Document \ Product name	μPD78217A	μPD78218A	μPD78P218A	μPD78P218A(A)
Brochure	—			
Data Sheet	IC-2748		IC-2722	IC-3188
User's Manual, Hardware	IEU-1313			
Special Function Register Application Table	—			

- **μPD78224 sub-series**

Document \ Product name	μPD78220	μPD78224	μPD78P224
Brochure	—		
Data Sheet	IC-2374		IC-2475
User's Manual, Hardware	IEU-1215		
Special Function Register Application Table	—		

- **μPD78234A sub-series**

Document \ Product name	μPD78233	μPD78234	μPD78237	μPD78238	μPD78P238	μPD78234(A)	μPD78238(A)
Brochure	—						
Data Sheet	IC-2476				IC-2607	IC-2984	
User's Manual, Hardware	IEU-1290						
Special Function Register Application Table	—						

- **μPD78244 sub-series**

Document \ Product name	μPD78243	μPD78244
Brochure	—	
Data Sheet	IC-2774	
User's Manual, Hardware	IEU-1316	
Special Function Register Application Table	—	

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 78K/II SERIES FEATURES

The 78K series consists of the 5 series shown in Figure 1-1.

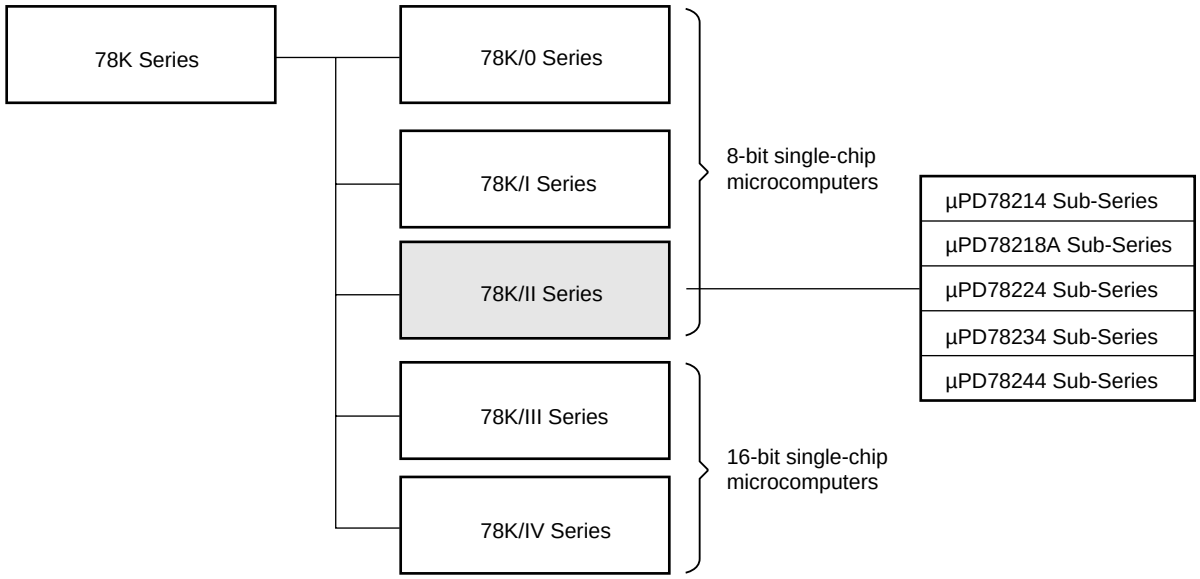
The 78K/II series is one of these 5 series, and comprises general-purpose type products with an on-chip 8-bits CPU.

These products have an instruction system and high-performance interrupt controller suited to control applications, and also incorporate a high-performance CPU provided with a 1M-byte data memory space.

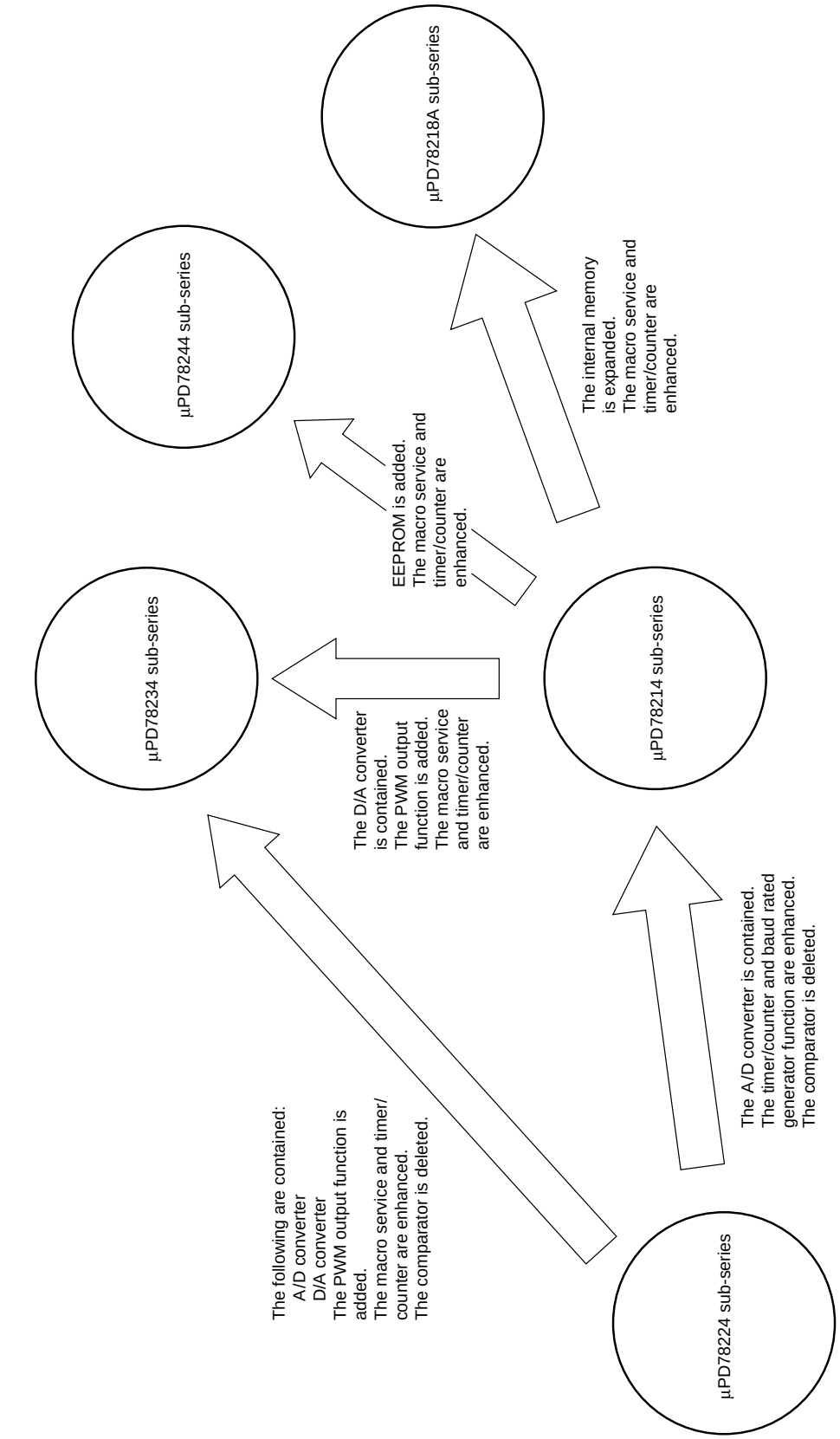
The 78K/II series further comprises 5 sub-series (the μPD78214 sub-series, μPD78218A sub-series, μPD78224 sub-series, μPD78234 sub-series, and μPD78244 sub-series), allowing the most suitable sub-series to be selected for the particular application.

Each sub-series has the same CPU, with differences in the peripheral hardware only; consequently the entire instruction set is shared by all sub-series. The only difference between products within the same sub-series, moreover, is the size of memory.

Figure 1-1. 78K Series and 78K/II Series Composition *



1.1 78K/II SERIES PRODUCT EXPANSION DIAGRAM



1.2 OUTLINE OF μPD78214 SUB-SERIES PRODUCTS (μPD78212, 78213, 78214, 78P214, 78212(A), 78213(A), 78214(A), 78P214(A))

1.2.1 Features

- Instruction cycle : 333 ns (μPD78212, 78214, 78P214)
  500 ns (μPD78213)

- On-chip memory
  - ROM
    Mask ROM : 16K bytes (μPD78214)
    8K bytes (μPD78212)
    Not incorporated (μPD78213)
    PROM : 16K bytes (μPD78P214)
  - RAM : 512 bytes
    384 bytes (μPD78212 only)

- I/O pins : 54
  36 (μPD78213 only)

- On-chip 8-bit A/D converter (8 analog inputs)

- Timer/counters
  - 16 bits x 1
  - 8 bits x 3

- Serial interface
  Independent on-chip UART and CSI

- μPD78212(A), 78213(A), 78214(A), 78P214(A):
  "Special" quality grade products of μPD78212, 78213, 78214, 78P214

1.2.2 Applications

- Standard products : OA equipment including printers, typewriters, PPCs, facsimile, etc., electronic musical instruments, inverters, cameras, etc.

- Special products : Automotive electronic equipment, combustion control, disaster/crime prevention unit *

1.2.3 Ordering Information and Quality Grade

(1) Ordering information

Ordering code	Package	On-chip ROM
µPD78212CW-xxx	64-pin plastic shrink DIP (750 mil)	Mask ROM
µPD78212GC-xxx-AB8	64-pin plastic QFP (14 x 14 mm body)	Mask ROM
µPD78212GJ-xxx-5BJ	74-pin plastic QFP (20 x 20 mm body)	Mask ROM
µPD78213CW	64-pin plastic shrink DIP (750 mil)	None
µPD78213GC-AB8	64-pin plastic QFP (14 x 14 mm body)	None
µPD78213GJ	74-pin plastic QFP (20 x 20 mm body)	None
µPD78213GQ-36	64-pin plastic QUIP	None
µPD78213L	68-pin plastic QFJ (□ 950 mil)	None
µPD78214CW-xxx	64-pin plastic shrink DIP (750 mil)	Mask ROM
µPD78214GC-xxx-AB8	64-pin plastic QFP (14 x 14 mm body)	Mask ROM
µPD78214GJ-xxx-5BJ	74-pin plastic QFP (20 x 20 mm body)	Mask ROM
µPD78214GQ-xxx-36	64-pin plastic QUIP	Mask ROM
µPD78214L-xxx	68-pin plastic QFJ (□ 950 mil)	Mask ROM
µPD78P214CW	64-pin plastic shrink DIP (750 mil)	One-time PROM
µPD78P214GC-AB8	64-pin plastic QFP (14 x 14 mm body)	One-time PROM
µPD78P214GJ	74-pin plastic QFP (20 x 20 mm body)	One-time PROM
µPD78P214GQ-36	64-pin plastic QUIP	One-time PROM
µPD78P214L	68-pin plastic QFJ (□ 950 mil)	One-time PROM
µPD78P214DW	64-pin ceramic shrink DIP with window (750 mil)	EPROM
µPD78212CW(A)-xxx	64-pin plastic shrink DIP (750 mil)	Mask ROM
µPD78212GC(A)-xxx-AB8	64-pin plastic QFP (14 x 14 mm body)	Mask ROM
µPD78213CW(A)	64-pin plastic shrink DIP (750 mil)	None
µPD78213GQ(A)-36	64-pin plastic QUIP	None
µPD78214CW(A)-xxx	64-pin plastic shrink DIP (750 mil)	Mask ROM
µPD78214GC(A)-xxx-AB8	64-pin plastic QFP (14 x 14 mm body)	Mask ROM
µPD78214GJ(A)-xxx-5BJ	74-pin plastic QFP (20 x 20 mm body)	Mask ROM
µPD78214GQ(A)-xxx-36	64-pin plastic QUIP	Mask ROM
µPD78214L(A)-xxx	68-pin plastic QFJ (□ 950 mil)	Mask ROM
µPD78P214CW(A)	64-pin plastic shrink DIP (750 mil)	One-time PROM
µPD78P214GC(A)-AB8	64-pin plastic QFP (14 x 14 mm body)	One-time PROM

Remark xxx is the ROM code number.

(2) Quality grade

Ordering code	Package	Quality grade
μPD78212CW-xxx	64-pin plastic shrink DIP (750 mil)	Standard
μPD78212GC-xxx-AB8	64-pin plastic QFP (14 x 14 mm body)	Standard
μPD78212GJ-xxx-5BJ	74-pin plastic QFP (20 x 20 mm body)	Standard
μPD78213CW	64-pin plastic shrink DIP (750 mil)	Standard
μPD78213GC-AB8	64-pin plastic QFP (14 x 14 mm body)	Standard
μPD78213GJ	74-pin plastic QFP (20 x 20 mm body)	Standard
μPD78213GQ-36	64-pin plastic QUIP	Standard
μPD78213L	68-pin plastic QFJ (□ 950 mil)	Standard
μPD78214CW-xxx	64-pin plastic shrink DIP (750 mil)	Standard
μPD78214GC-xxx-AB8	64-pin plastic QFP (14 x 14 mm body)	Standard
μPD78214GJ-xxx-5BJ	74-pin plastic QFP (20 x 20 mm body)	Standard
μPD78214GQ-xxx-36	64-pin plastic QUIP	Standard
μPD78214L-xxx	68-pin plastic QFJ (□ 950 mil)	Standard
μPD78P214CW	64-pin plastic shrink DIP (750 mil)	Standard
μPD78P214GC-AB8	64-pin plastic QFP (14 x 14 mm body)	Standard
μPD78P214GJ	74-pin plastic QFP (20 x 20 mm body)	Standard
μPD78P214GQ-36	64-pin plastic QUIP	Standard
μPD78P214L	68-pin plastic QFJ (□ 950 mil)	Standard
μPD78P214DW	64-pin ceramic shrink DIP with window (750 mil)	Standard
μPD78212CW(A)-xxx	64-pin plastic shrink DIP (750 mil)	Special
μPD78212GC(A)-xxx-AB8	64-pin plastic QFP (14 x 14 mm body)	Special
μPD78213CW(A)	64-pin plastic shrink DIP (750 mil)	Special
μPD78213GQ(A)-36	64-pin plastic QUIP	Special
μPD78214CW(A)-xxx	64-pin plastic shrink DIP (750 mil)	Special
μPD78214GC(A)-xxx-AB8	64-pin plastic QFP (14 x 14 mm body)	Special
μPD78214GJ(A)-xxx-5BJ	74-pin plastic QFP (20 x 20 mm body)	Special
μPD78214GQ(A)-xxx-36	64-pin plastic QUIP	Special
μPD78214L(A)-xxx	68-pin plastic QFJ (□ 950 mil)	Special
μPD78P214CW(A)	64-pin plastic shrink DIP (750 mil)	Special
μPD78P214GC(A)-AB8	64-pin plastic QFP (14 x 14 mm body)	Special

Remark xxx is the ROM code number.

Please refer to "Quality grade on NEC Semiconductor Devices" (Document number IEI-1209) published by NEC Corporation to know the specification of quality grade on the devices and its recommended applications.

1.2.4 Function Outline

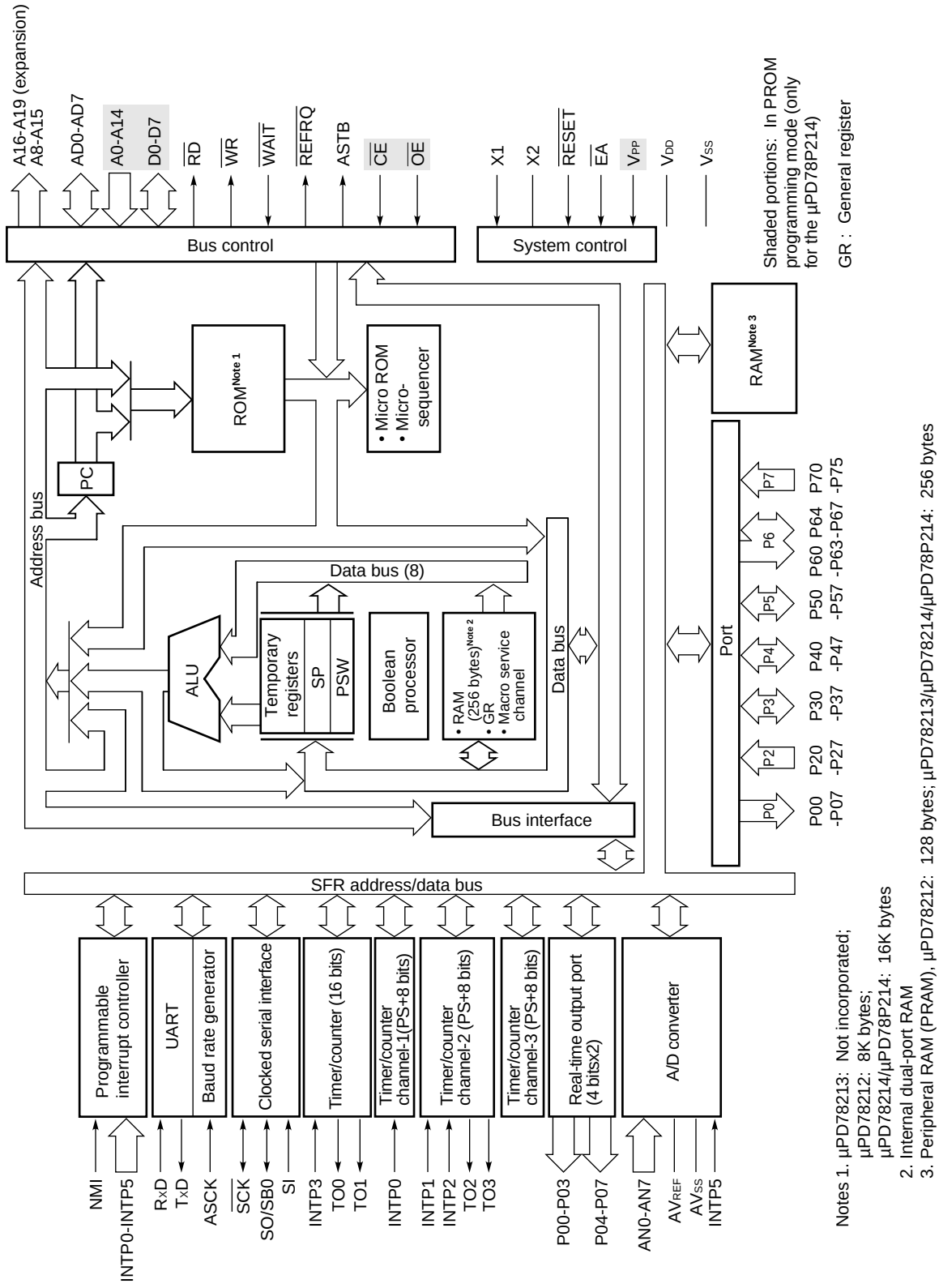
Item \ Product name			μPD78212	μPD78214	μPD78P214	μPD78213
Number of basic instructions (mnemonics)			65			
Minimum instruction execution time (at 12 MHz operation)			333 ns			500 ns
On-chip memory capacity		ROM	8K bytes (Mask ROM)	16K bytes (Mask ROM)	16K bytes (PROM)	ROM-less
		RAM	384 bytes	512 byte		
Memory space			Program: 64K bytes, data: 1M byte			
I/O pins		Input	14			
		Output	12			
		Input/output	28			10
		Total	54			36
Additional function pins[Note]	Pins with pull-up resistor		34			16
	LED direct drive outputs		16			—
	Transistor direct drive outputs		8			
ROM-less mode setting			$\overline{EA}$ pin = low level			ROM-less product
Real-time output ports			4 bits x 2 or 8 bits x 1			
General registers			8 bits x 8 x 4 banks (memory mapped)			
Timer/counters			16-bit timer/counters: Timer register x 1, Capture register x 1, Compare register x 2; Pulse output capability (Toggle output, PWM/PPG output)			
			8-bit timer/counter 1: Timer register x 1, Capture/compare register x 1, Compare register x 1; Pulse output capability (Real-time output: 4 bits x 2)			
			8-bit timer/counter 2: Timer register x 1, Capture register x 1, Compare register x 2; Pulse output capability (Toggle output, PWM/PPG output)			
			8-bit timer/counter 3: Timer register x 1, Compare register x 1			
Serial interface			UART : 1 channel (incorporating dedicated baud rate generator) CSI (3-wire serial I/O, SBI) : 1 channel			

(Continued)

Note Additional function pins are included in the I/O pins.

Item \ Product name	μPD78212	μPD78214	μPD78P214	μPD78213
A/D converter	8-bit resolution x 8 channels			
Interrupts	19 sources (7 external, 12 internal) + BRK instruction 2-level priority (programmable) 2 servicing modes (vectored interrupts, macro service)			
Instruction set	16-bit operation Multiply/divide (8 bits x 8 bits, 16 bits/8 bits) Bit manipulation BCD adjustment, etc.			
Package	64-pin plastic shrink DIP (750 mil) 64-pin plastic QUIP (except μPD78212) 68-pin plastic QFJ (□ 950 mil) (except μPD78212) 64-pin plastic QFP (14 x 14 mm body) 74-pin plastic QFP (20 x 20 mm body) 64-pin ceramic shrink DIP with window (750 mil): μPD78P214 only			

1.2.5 Block Diagram



Notes 1. µPD78213: Not incorporated;
µPD78212: 8K bytes;
µPD78214/µPD78P214: 16K bytes
2. Internal dual-port RAM
3. Peripheral RAM (PRAM), µPD78212: 128 bytes; µPD78213/µPD78214/µPD78P214: 256 bytes

1.3 OUTLINE OF μPD78218A SUB-SERIES PRODUCTS (μPD78217A, 78218A, 78P218A, 78218A(A))

1.3.1 Features

- Instruction cycle : 333 ns (μPD78218A, 78P218A)
  500 ns (μPD78217A)

- On-chip memory
  - ROM
    Mask ROM : 32K bytes (μPD78218A)
    Not incorporated (μPD78217A)
    PROM : 32K bytes (μPD78P218A)
  - RAM : 1024 bytes

- Upward compatible with μPD78214 series
  Enhanced macro service & timer/counters, increased on-chip memory size

- I/O pins : 54
  36 (μPD78217A only)

- On-chip 8-bit A/D converter (8 analog inputs)

- Timer/counters
  - 16 bits x 1
  - 8 bits x 3

- Serial interface
  Independent on-chip UART and CSI

- μPD78218A(A) : "Special" quality grade product of μPD78218A.

1.3.2 Applications

- Standard products: OA equipment including printers, typewriters, PPCs, facsimile, etc., electronic musical instruments, inverters, cameras, etc.
- Special products : Automotive electrical equipment, combustion control, disaster/crime prevention unit *

1.3.3 Ordering Information and Quality Grade

(1) Ordering information

Ordering code	Package	On-chip ROM
µPD78217ACW	64-pin plastic shrink DIP (750 mil)	None
µPD78217AGC-AB8	64-pin plastic QFP (14 x 14 mm body)	None
µPD78218ACW-xxx	64-pin plastic shrink DIP (750 mil)	Mask ROM
µPD78218AGC-xxx-AB8	64-pin plastic QFP (14 x 14 mm body)	Mask ROM
µPD78P218ACW	64-pin plastic shrink DIP (750 mil)	One-time PROM
µPD78P218AGC-AB8	64-pin plastic QFP (14 x 14 mm body)	One-time PROM
µPD78P218ADW	64-pin ceramic shrink DIP with window (750 mil)	EPROM
µPD78218ACW(A)-xxx	64-pin plastic shrink DIP (750 mil)	Mask ROM
µPD78218AGC(A)-xxx-AB8	64-pin plastic QFP (14 x 14 mm body)	Mask ROM

Remark xxx is the ROM code number.

(2) Quality grade

Ordering code	Package	Quality grade
µPD78217ACW	64-pin plastic shrink DIP (750 mil)	Standard
µPD78217AGC-AB8	64-pin plastic QFP (14 x 14 mm body)	Standard
µPD78218ACW-xxx	64-pin plastic shrink DIP (750 mil)	Standard
µPD78218AGC-xxx-AB8	64-pin plastic QFP (14 x 14 mm body)	Standard
µPD78P218ACW	64-pin plastic shrink DIP (750 mil)	Standard
µPD78P218AGC-AB8	64-pin plastic QFP (14 x 14 mm body)	Standard
µPD78P218ADW	64-pin ceramic shrink DIP with window (750 mil)	Standard
µPD78218ACW(A)-xxx	64-pin plastic shrink DIP (750 mil)	Special
µPD78218AGC(A)-xxx-AB8	64-pin plastic QFP (14 x 14 mm body)	Special

Remark xxx is the ROM code number.

Please refer to "Quality grade on NEC Semiconductor Devices" (Document number IEI-1209) published by NEC Corporation to know the specification of quality grade on the devices and its recommended applications.

1.3.4 Function Outline

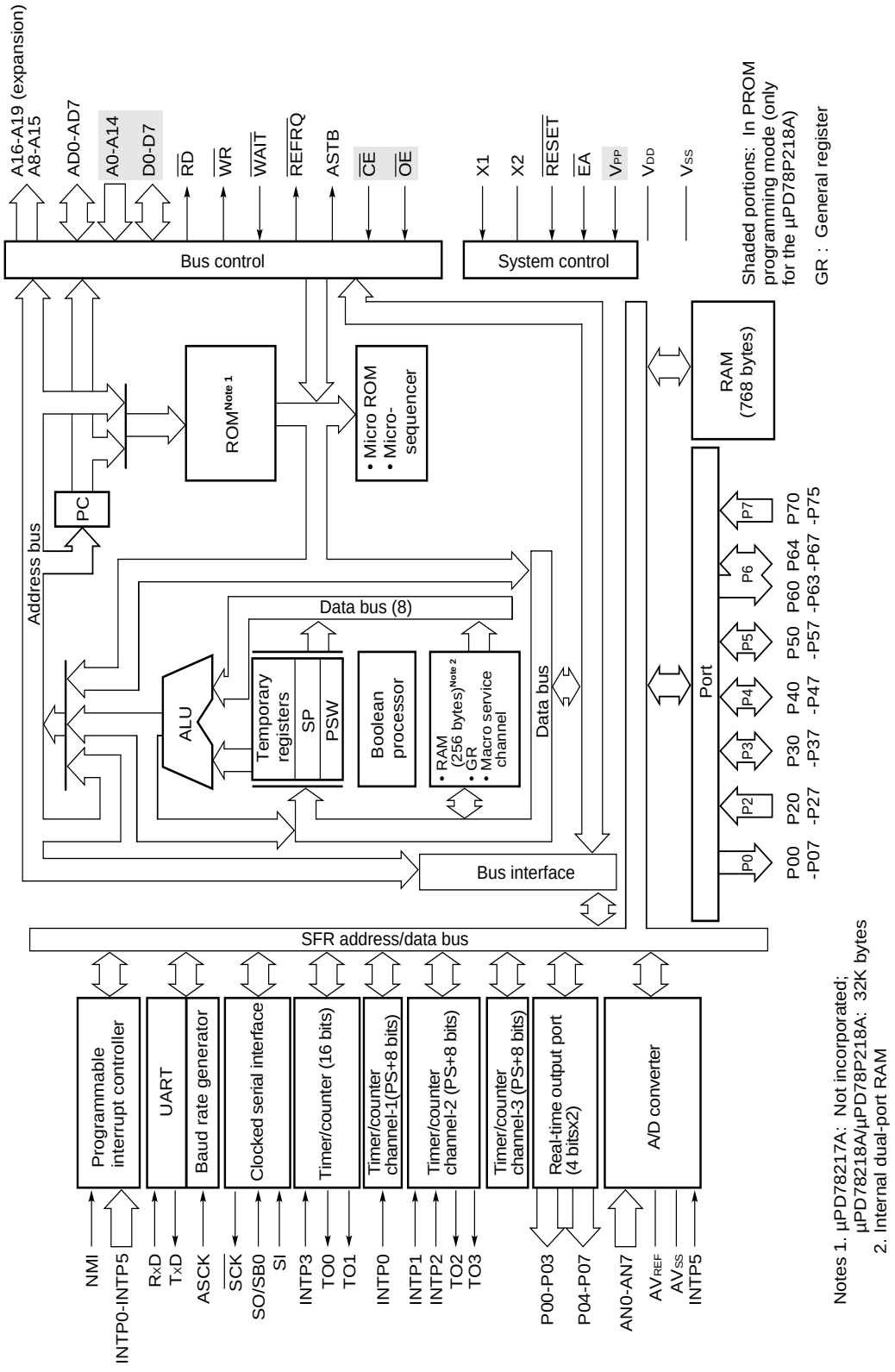
Item \ Product name			μPD78218A	μPD78P218A	μPD78217A
Number of basic instructions (mnemonics)			65		
Minimum instruction execution time (at 12 MHz operation)			333 ns		500 ns
On-chip memory capacity		ROM	32K bytes (Mask ROM)	32K bytes (PROM)	ROM-less
		RAM	1024 byte		
Memory space			Program: 64K bytes, data: 1M byte		
I/O pins		Input	14		
		Output	12		
		Input/output	28		10
		Total	54		36
Additional function pins**Note**	Pins with pull-up resistor		34		16
	LED direct drive outputs		16		—
	Transistor direct drive outputs		8		
ROM-less mode setting			$\overline{EA}$ pin = low level		ROM-less product
Real-time output ports			4 bits x 2 or 8 bits x 1		
General registers			8 bits x 8 x 4 banks (memory mapped)		
Timer/counters			16-bit timer/counters: Timer register x 1, Capture register x 1, Compare register x 2; Pulse output capability (Toggle output, PWM/PPG output, One-shot pulse output)		
			8-bit timer/counter 1: Timer register x 1, Capture/compare register x 1, Compare register x 1; Pulse output capability (Real-time output: 4 bits x 2)		
			8-bit timer/counter 2: Timer register x 1, Capture register x 1, Compare register x 2; Pulse output capability (Toggle output, PWM/PPG output)		
			8-bit timer/counter 3: Timer register x 1, Compare register x 1		
Serial interface			UART : 1 channel (incorporating dedicated baud rate generator) CSI (3-wire serial I/O, SBI) : 1 channel		

(Continued)

Note Additional function pins are included in the I/O pins.

Item \ Product name	μPD78218A	μPD78P218A	μPD78217A
A/D converter	8-bit resolution x 8 channels		
Interrupts	19 sources (7 external, 12 internal) + BRK instruction 2-level priority (programmable) 2 servicing modes (vectored interrupts, macro service)		
Instruction set	16-bit operation Multiply/divide (8 bits x 8 bits, 16 bits/8 bits) Bit manipulation BCD adjustment, etc.		
Package	64-pin plastic shrink DIP (750 mil) 64-pin plastic QFP (14 x 14 mm body) 64-pin ceramic shrink DIP with window (750 mil): μPD78P218A only		

1.3.5 Block Diagram



Notes 1. µPD78217A: Not incorporated;
µPD78218A/µPD78P218A: 32K bytes
2. Internal dual-port RAM

1.4 OUTLINE OF μPD78224 SUB-SERIES PRODUCTS (μPD78220, 78224, 78P224)

1.4.1 Features

- Instruction cycle : 333 ns (μPD78224, 78P224)
  500 ns (μPD78220)

- On-chip memory
  - ROM
    Mask ROM : 16K bytes (μPD78224)
    Not incorporated (μPD78220)
    PROM : 16K bytes (μPD78P224)
  - RAM : 640 bytes

- I/O pins : 71
  53 (μPD78220 only)

- Comparator : 4-bit resolution x 8

- Timer/counters
  - 16 bits x 1
  - 8 bits x 2

- Serial interface
  Independent on-chip UART and CSI

1.4.2 Applications

Areas handling a large amount of data such as kanji character generators, typewriters, hand-held word processors, ECRs, etc.

1.4.3 Ordering Information and Quality Grade

(1) Ordering information

Ordering code	Package	On-chip ROM
µPD78220GJ-5BG	94-pin plastic QFP (20 x 20mm body)	None
µPD78220L	84-pin plastic QFJ (□ 1150 mil)	None
µPD78224GJ-xxx-5BG	94-pin plastic QFJ (20 x 20 mm body)	Mask ROM
µPD78224L-xxx	84-pin plastic QFJ (□ 1150 mil)	Mask ROM
µPD78P224GJ-5BG	94-pin plastic QFP (20 x 20 mm body)	One-time PROM
µPD78P224L	84-pin plastic QFJ (□ 1150 mil)	One-time PROM

Remark xxx is the ROM code number.

(2) Quality grade

Ordering code	Package	Quality grade
µPD78220GJ-5BG	94-pin plastic QFP (20 x 20 mm body)	Standard
µPD78220L	84-pin plastic QFJ (□ 1150 mil)	Standard
µPD78224GJ-xxx-5BG	94-pin plastic QFP (20 x 20 mm body)	Standard
µPD78224L-xxx	84-pin plastic QFJ (□ 1150 mil)	Standard
µPD78P224GJ-5BG	94-pin plastic QFP (20 x 20 mm body)	Standard
µPD78P224L	84-pin plastic QFJ (□ 1150 mil)	Standard

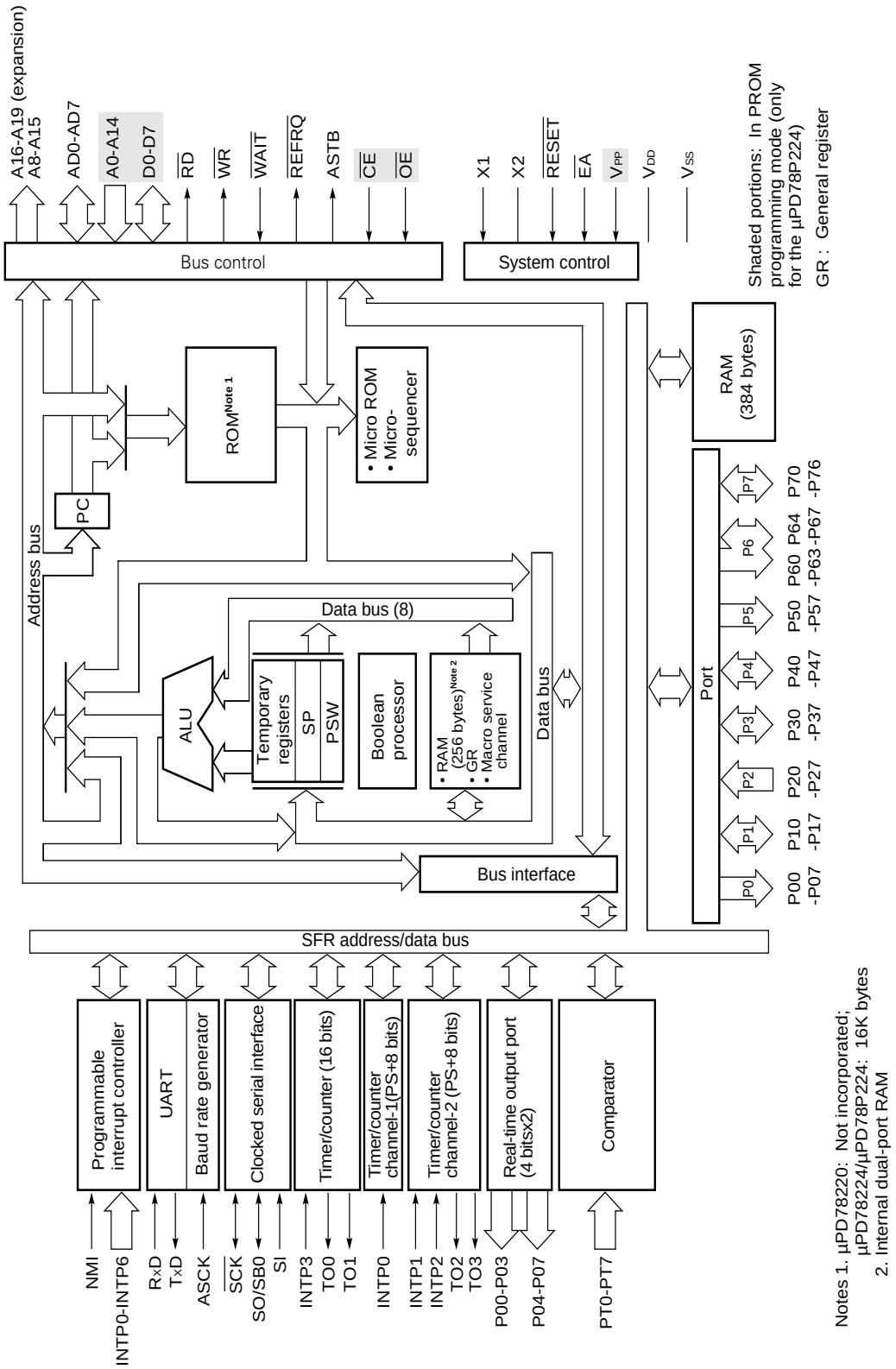
Remark xxx is the ROM code number.

Please refer to "Quality grade on NEC Semiconductor Devices" (Document number IEI-1209) published by NEC Corporation to know the specification of quality grade on the devices and its recommended applications.

1.4.4 Function Outline

Item \ Product name			μPD78224	μPD78P224	μPD78220
Number of basic instructions (mnemonics)			65		
Minimum instruction execution time (at 12 MHz operation)			333 ns		500 ns
On-chip memory capacity		ROM	16K bytes (Mask ROM)	16K bytes (PROM)	ROM-less
		RAM	640 bytes		
Memory space			Program: 64K bytes, data: 1M byte		
I/O pins	Input		8		
	Output		20 LED drive capability: 8		12 LED drive capability: 8
	Input/output		35		25
	Total		63		45
ROM-less mode setting			$\overline{EA}$ pin = low level		ROM-less product
Real-time output ports			4 bits x 2 or 8 bits x 1		
General registers			8 bits x 8 x 4 banks (memory mapped)		
Timer/counters			16-bit timer/counters: Timer register x 1, Capture register x 1, Compare register x 2 — Pulse output capability (Toggle output)		
			8-bit timer/counter 1: Timer register x 1, Capture/compare register x 1, Compare register x 1 — Pulse output capability (Real-time output: 4 bits x 2)		
			8-bit timer/counter 2: Timer register x 1, Capture register x 1, Compare register x 2 — Pulse output capability (Toggle output)		
Serial interface			UART : 1 channel CSI (3-wire serial I/O, SBI) : 1 channel		
Comparator			4-bit resolution x 8		
Interrupts			17 sources (8 external, 9 internal) + BRK instruction 2-level priority (programmable) 2 servicing modes (vectored interrupts, macro service)		
Instruction set			16-bit operation Multiply/divide (8 bits x 8 bits, 16 bits/8 bits) Bit manipulation BCD adjustment, etc.		
Package			94-pin plastic QFP (20 x 20 mm body) 84-pin plastic QFJ (□ 1150 mil)		

1.4.5 Block Diagram



Notes 1. µPD78220: Not incorporated;
µPD78224/µPD78P224: 16K bytes
2. Internal dual-port RAM

1.5 OUTLINE OF μPD78234 SUB-SERIES PRODUCTS (μPD78233, 78234, 78237, 78238, 78P238, 78234(A), 78238(A))

1.5.1 Features

- Instruction cycle : 333 ns (μPD78234, 78238, 78P238)
  500 ns (μPD78233, 78237)

- On-chip memory
  - ROM
    Mask ROM : 16K bytes (μPD78234)
    32K bytes (μPD78238)
    Not incorporated (μPD78233, 78237)
    PROM : 32K bytes (μPD78P238)
  - RAM : 640 bytes (μPD78233, 78234)
    1024 bytes (μPD78237, 78238, 78P238)

- I/O pins : 64 (μPD78234, 78238, 78P238)
  46 (μPD78233, 78237)

- On-chip 8-bit A/D converter (8 analog inputs)

- On-chip 8-bit D/A converter (2 analog outputs)

- 12-bit PWM outputs (2 outputs)

- Timer/counters
  - 16 bits x 1
  - 8 bits x 3

- Serial interface
  Independent on-chip UART and CSI

- μPD78234(A), 78238(A) : "Special" quality grade products of μPD78234, 78238

1.5.2 Applications

- Standard products : OA equipment including LBP printers, typewriters, HDDs, FDDs, PPCs, facsimile, etc., electronic musical instruments, inverters, cameras, air conditioners, etc.
- * Special products : Automotive electronic equipment, combustion control, disaster/crime prevention unit

1.5.3 Ordering Information and Quality Grade

(1) Ordering information

Ordering code	Package	On-chip ROM
µPD78233GC-3B9	80-pin plastic QFP (14 x 14 mm body)	None
µPD78233GJ-5BG	94-pin plastic QFP (20 x 20 mm body)	None
µPD78233LQ	84-pin plastic QFJ (□ 1150 mil)	None
µPD78234GC-xxx-3B9	80-pin plastic QFP (14 x 14 mm body)	Mask ROM
µPD78234GJ-xxx-5BG	94-pin plastic QFP (20 x 20 mm body)	Mask ROM
µPD78234LQ-xxx	84-pin plastic QFJ (□ 1150 mil)	Mask ROM
µPD78237GC-3B9	80-pin plastic QFP (14 x 14 mm body)	None
µPD78237GJ-5BG	94-pin plastic QFP (20 x 20 mm body)	None
µPD78237LQ	84-pin plastic QFJ (□ 1150 mil)	None
µPD78238GC-3B9	80-pin plastic QFP (14 x 14 mm body)	Mask ROM
µPD78238GJ-5BG	94-pin plastic QFP (20 x 20 mm body)	Mask ROM
µPD78238LQ	84-pin plastic QFJ (□ 1150 mil)	Mask ROM
µPD78P238GC-3B9	80-pin plastic QFP (14 x 14 mm body)	One-time PROM
µPD78P238GJ-5BG	94-pin plastic QFP (20 x 20 mm body)	One-time PROM
µPD78P238LQ	84-pin plastic QFJ (□ 1150 mil)	One-time PROM
µPD78P238KF	94-pin ceramic WQFN	EPROM
µPD78234GC(A)-xxx-3B9	80-pin plastic QFP (14 x 14 mm body)	Mask ROM
µPD78234GJ(A)-xxx-5BG	94-pin plastic QFP (20 x 20 mm body)	Mask ROM
µPD78238GC(A)-xxx-3B9	80-pin plastic QFP (14 x 14 mm body)	Mask ROM
µPD78238GJ(A)-xxx-5BG	94-pin plastic QFP (20 x 20 mm body)	Mask ROM

Remark xxx is the ROM code number.

(2) Quality grade

Ordering code	Package	Quality grade
µPD78233GC-3B9	80-pin plastic QFP (14 x 14 mm body)	Standard
µPD78233GJ-5BG	94-pin plastic QFP (20 x 20 mm body)	Standard
µPD78233LQ	84-pin plastic QFJ (□ 1150 mil)	Standard
µPD78234GC-xxx-3B9	80-pin plastic QFP (14 x 14 mm body)	Standard
µPD78234GJ-xxx-5BG	94-pin plastic QFP (20 x 20 mm body)	Standard
µPD78234LQ-xxx	84-pin plastic QFJ (□ 1150 mil)	Standard
µPD78237GC-3B9	80-pin plastic QFP (14 x 14 mm body)	Standard
µPD78237GJ-5BG	94-pin plastic QFP (20 x 20 mm body)	Standard
µPD78237LQ	84-pin plastic QFJ (□ 1150 mil)	Standard
µPD78238GC-xxx-3B9	80-pin plastic QFP (14 x 14 mm body)	Standard
µPD78238GJ-xxx-5BG	94-pin plastic QFP (20 x 20 mm body)	Standard
µPD78238LQ-xxx	84-pin plastic QFJ (□ 1150 mil)	Standard
µPD78P238GC-3B9	80-pin plastic QFP (14 x 14 mm body)	Standard
µPD78P238GJ-5BG	94-pin plastic QFP (20 x 20 mm body)	Standard
µPD78P238LQ	84-pin plastic QFJ (□ 1150 mil)	Standard
µPD78P238KF	94-pin ceramic WQFN	Standard
µPD78234GC(A)-xxx-3B9	80-pin plastic QFP (14 x 14 mm body)	Special
µPD78234GJ(A)-xxx-5BG	94-pin plastic QFP (20 x 20 mm body)	Special
µPD78238GC(A)-xxx-3B9	80-pin plastic QFP (14 x 14 mm body)	Special
µPD78238GJ(A)-xxx-5BG	94-pin plastic QFP (20 x 20 mm body)	Special

Remark xxx is the ROM code number.

Please refer to "Quality grade on NEC Semiconductor Devices" (Document number IEI-1209) published by NEC Corporation to know the specification of quality grade on the devices and its recommended applications.

1.5.4 Function Outline

Item \ Product name			μPD78234	μPD78238	μPD78P238	μPD78233	μPD78237
Number of basic instructions (mnemonics)			65				
Minimum instruction execution time (at 12 MHz operation)			333 ns			500 ns	
On-chip memory capacity		ROM	16K bytes (Mask ROM)	32K bytes (Mask ROM)	32/16K bytes[Note 1] (PROM)	ROM-less	
		RAM	640 bytes	1024 bytes	1024/640 bytes[Note 1]	640 bytes	1024 bytes
Memory space			Program: 64K bytes, data: 1M byte				
I/O pins		Input	16				
		Output	12				
		Input/output	36			18	
		Total	64			46	
Additional function pins[Note 2]	Pins with pull-up resistor		42			24	
	LED direct drive outputs		24			8	
	Transistor direct drive outputs		8				
ROM-less mode setting			MODE pin = high level	Not possible		ROM-less product	
Real-time output ports			4 bits x 2 or 8 bits x 1				
General registers			8 bits x 8 x 4 banks (memory mapped)				
Timer/counters			16-bit timer/counters: Timer register x 1, Capture register x 1, Compare register x 2 — Pulse output capability: Toggle output, PWM/PPG output, One-shot pulse output				
			8-bit timer/counter 1: Timer register x 1, Capture/compare register x 1, Compare register x 1 — Pulse output capability: Real-time output: 4 bits x 2				
			8-bit timer/counter 2: Timer register x 1, Capture register x 1, Compare register x 2 — Pulse output capability: Toggle output, PWM/PPG output				
			8-bit timer/counter 3: Timer register x 1, Compare register x 1				

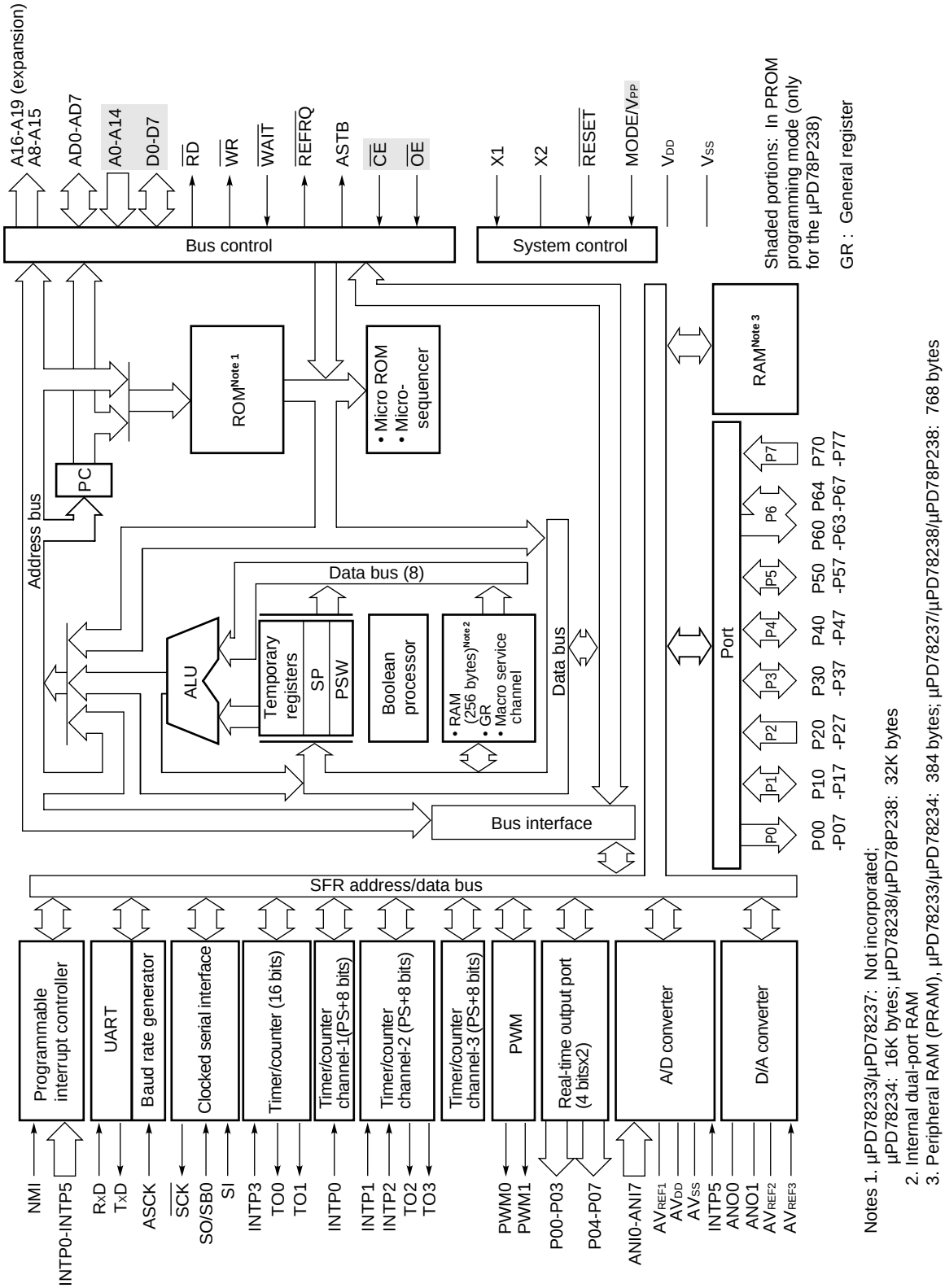
(Continued)

Notes 1. Set by software

2. Additional function pins are included in the I/O pins.

Item \ Product name	μPD78234	μPD78238	μPD78P238	μPD78233	μPD78237
PWM output function	12-bit resolution x 2 channels (PWM frequency = 23.4 kHz)				
Serial interface	UART : 1 channel (incorporating dedicated baud rate generator) CSI (3-wire serial I/O, SBI) : 1 channel				
A/D converter	8-bit resolution x 8 channels				
D/A converter	8-bit resolution x 2 channels				
Interrupts	19 sources (7 external, 12 internal) + BRK instruction 2-level priority (programmable) 2 servicing modes (vectored interrupts, macro service)				
Instruction set	16-bit operation Multiply/divide (8 bits x 8 bits, 16 bits/8 bits) Bit manipulation BCD adjustment, etc.				
Package	80-pin plastic QFP (14 x 14 mm body) 94-pin plastic QFP (20 x 20 mm body) 84-pin plastic QFJ (□ 1150 mil) 94-pin ceramic WQFN (μPD78P238 only)				

1.5.5 Block Diagram



Notes 1. µPD78233/µPD78237: Not incorporated;
µPD78234: 16K bytes; µPD78238/µPD78P238: 32K bytes
2. Internal dual-port RAM
3. Peripheral RAM (PRAM), µPD78233/µPD78234: 384 bytes; µPD78237/µPD78238/µPD78P238: 768 bytes

1.6 OUTLINE OF μPD78244 SUB-SERIES PRODUCTS (μPD78243, 78244)

1.6.1 Features

- Instruction cycle : 333 ns (μPD78244)
  500 ns (μPD78243)

- On-chip memory
  - ROM
    Mask ROM : 16K bytes (μPD78244)
    Not incorporated (μPD78243)
  - RAM : 512 bytes

- On-chip EEPROM**Note**: 512 bytes

Note Electrically erasable/programmable read-only memory. Unlike ordinary data memory (RAM), EEPROM can retain data in the event of a power failure.

- I/O pins : 54 (μPD78244)
  36 (μPD78243)

- On-chip 8-bit A/D converter (8 analog inputs)

- Timer/counters
  - 16 bits x 1
  - 8 bits x 3

- Serial interface
  Independent on-chip UART and CSI

1.6.2 Applications

OA equipment including printers, typewriters, cameras, PPCs, facsimile, etc., adjustment data storage in application set assembly, data retention in case of power failure, etc.

1.6.3 Ordering Information and Quality Grade

(1) Ordering information

Ordering code	Package	On-chip ROM
µPD78243CW	64-pin plastic shrink DIP (750 mil)	None
µPD78243GC-AB8	64-pin plastic QFP (14 x 14 mm body)	None
µPD78243CW-xxx	64-pin plastic shrink DIP (750 mil)	Mask ROM
µPD78243GC-xxx-AB8	64-pin plastic QFP (14 x 14 mm body)	Mask ROM

Remark xxx is the ROM code number.

(2) Quality grade

Ordering code	Package	Quality grade
µPD78243CW	64-pin plastic shrink DIP (750 mil)	Standard
µPD78243GC-AB8	64-pin plastic QFP (14 x 14 mm body)	Standard
µPD78243CW-xxx	64-pin plastic shrink DIP (750 mil)	Standard
µPD78243GC-AB8	64-pin plastic QFP (14 x 14 mm body)	Standard

Remark xxx is the ROM code number.

Please refer to "Quality grade on NEC Semiconductor Devices" (Document number IEI-1209) published by NEC Corporation to know the specification of quality grade on the devices and its recommended applications.

1.6.4 Function Outline

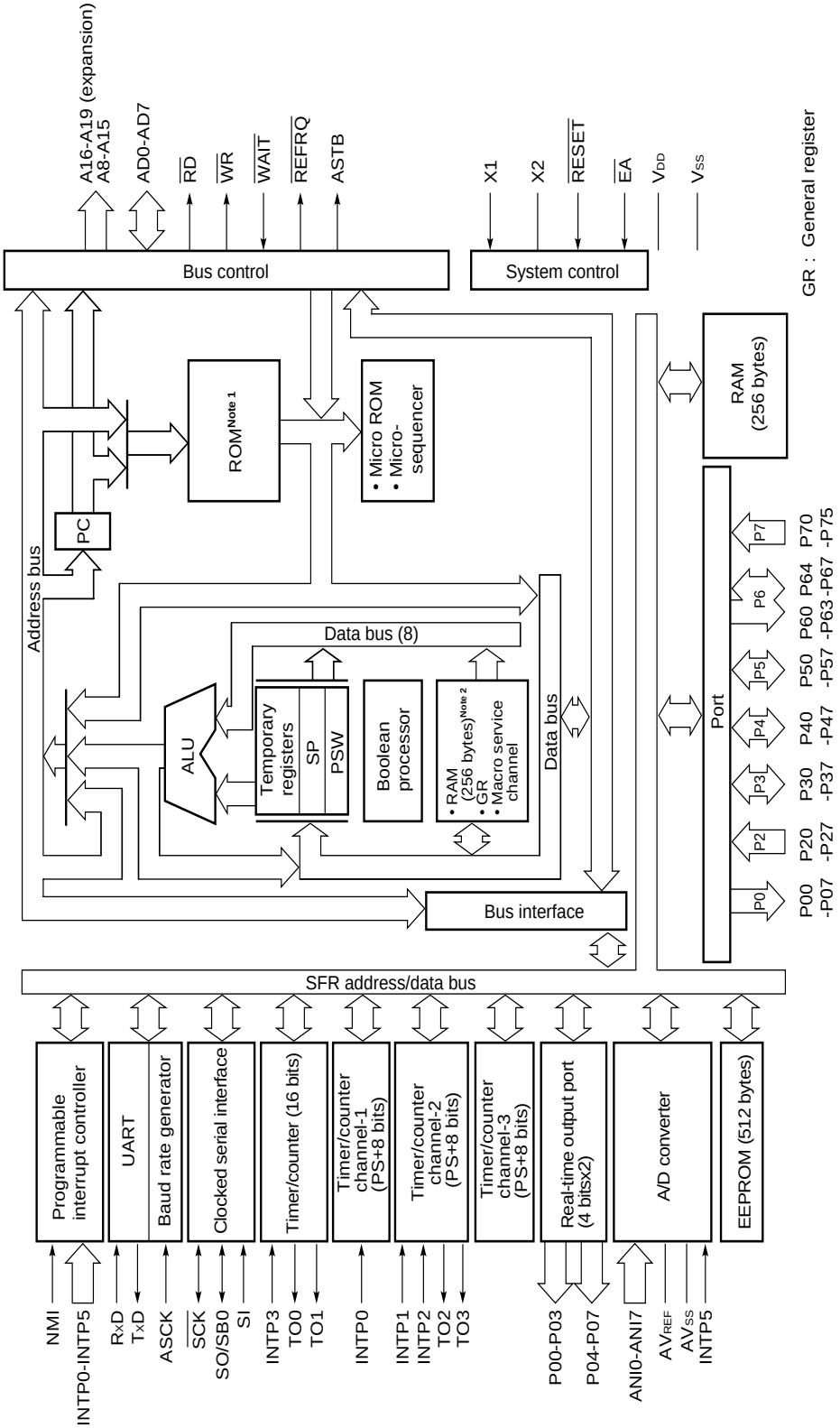
Item \ Product name			μPD78244	μPD78243
Number of basic instructions (mnemonics)			65	
Minimum instruction execution time (at 12 MHz operation)			333 ns	500 ns
On-chip memory capacity		ROM	16K bytes	ROM-less
		EEPROM	512 bytes	
		RAM	512 bytes	
Memory space			Program: 64K bytes, data: 1M byte	
I/O pins		Input	14	
		Output	12	
		Input/output	28	10
		Total	54	36
Additional function pins**Note**	Pins with pull-up resistor		34	16
	LED direct drive outputs		16	—
	Transistor direct drive outputs		8	
ROM-less mode setting			$\overline{EA}$ pin = low level	ROM-less product
Real-time output ports			4 bits x 2 or 8 bits x 1	
General registers			8 bits x 8 x 4 banks (memory mapped)	
Timer/counters			16-bit timer/counters: Timer register x 1, Capture register x 1, Compare register x 2 — Pulse output capability (Toggle output, PWM/PPG output, One-shot pulse output)	
			8-bit timer/counter 1: Timer register x 1, Capture/compare register x 1, Compare register x 1 — Pulse output capability (Real-time output: 4 bits x 2)	
			8-bit timer/counter 2: Timer register x 1, Capture register x 1, Compare register x 2 — Pulse output capability (Toggle output, PWM/PPG output)	
			8-bit timer/counter 3: Timer register x 1, Compare register x 1	

(Continued)

Note Additional function pins are included in the I/O pins.

Item \ Product name	μPD78244	μPD78243
Serial interface	UART : 1 channel (incorporating dedicated baud rate generator) CSI (3-wire serial I/O, SBI) : 1 channel	
A/D converter	8-bit resolution x 8 channels	
Interrupts	21 sources (7 external, 14 internal) + BRK instruction 2-level priority (programmable) 2 servicing modes (vectored interrupts, macro service)	
Instruction set	16-bit operation Multiply/divide (8 bits x 8 bits, 16 bits/8 bits) Bit manipulation BCD adjustment, etc.	
Package	64-pin plastic shrink DIP (750 mil) 64-pin plastic QFP (14 x 14 mm body)	

1.6.5 Block Diagram



Notes 1. µPD78243: Not incorporated;
µPD78244: 16K bytes
2. Internal dual-port RAM

CHAPTER 2 78K/II SERIES PRODUCTS

This chapter shows the functions of the 78K/II series products in tabular form.
For further details, please refer to the relevant User's Manual.

Sub-series name		μPD78214 sub-series			μPD78218A sub-series		μPD78224 sub-series	
Item \ Product name		μPD78212	μPD78213	μPD78214 (μPD78P214)	μPD78217A	μPD78218A (μPD78P218A)	μPD78220	μPD78224 (μPD78P224)
Number of basic instructions		65 (Common to 78K/II series)						
Minimum instruction execution time		333 ns	500 ns	333 ns	500 ns	333 ns	500 ns	333 ns
PUSH and PSW instruction execution time (number of clocks)		When the stack area is internal dual port RAM: 5 or 7 Other than above: 7 or 9			When the stack area is internal dual port RAM: 6 Other than above: 8		When the stack area is internal dual port RAM: 5 or 7	
Operating ambient temperature and supply voltage range		Other than μPD78P218A: T_A = –40 to +85°C, V_{DD} = +5 V ±10% μPD78P218A: T_A = –40 to +85°C, V_{DD} = +5 V± 0.3V					T_A = –40 to +85°C, V_{DD} = +5 V ±5% T_A = –10 to +70°C, V_{DD} = +5 V ±10%	
General registers		8 bits x 8 x 4 banks						
Bank registers		P6 and PM6					P6 only	
On-chip memory	ROM	8K bytes	None	16K bytes	None	32K bytes	None	16K bytes
	EEPROM	None						
	RAM	384 bytes	512 bytes		1024 bytes		640 bytes	
Memory space		Program memory space: 64K bytes; Data memory space: 1M byte						
I/O pins	Input	14					8	
	Output	12					12	20
	Input/output	28	10	28	10	28	25	35
	Total	54	36	54	36	54	45	63
Additional function pins **Note**	Pins with pull-up resistor	34	16	34	16	34	None	
	LED direct drive outputs	16	0	16	0	16	8	
	Transistor direct drive outputs	8					None	
	P0	8-bit output port						
	P1	—					8-bit I/O port	
	P2	8-bit input port						
	P3	8-bit I/O port						
	P4	8-bit I/O port	—	8-bit I/O port	—	8-bit I/O port	—	8-bit I/O port
	P5	8-bit I/O port	—	8-bit I/O port	—	8-bit I/O port	—	8-bit I/O port
	P6	4-bit output port + 4-bit I/O port	4-bit output port + 2-bit I/O port	4-bit output port + 4-bit I/O port	4-bit output port + 2-bit I/O port	4-bit output port + 4-bit I/O port	4-bit output port + 2-bit I/O port	4-bit output port + 4-bit I/O port
	P7	6-bit input port					7-bit I/O port	

Note Additional function pins are included in I/O pins.

(1/3)

μPD78234 sub-series				μPD78244 sub-series	
μPD78233	μPD78234	μPD78237	μPD78238 (μPD78P238)	μPD78243	μPD78244
65 (Common to 78K/II series)					
500 ns	333 ns	500 ns	333 ns	500 ns	333 ns
When the stack area is internal dual port RAM: 6 Other than above: 8					
T_A = –40 to +85°C, V_{DD} = +5 V ±10%				T_A = –10 to +70°C, V_{DD} = +5 V ±10%	
8 bits x 8 x 4 banks					
P6 and PM6					
None	16K bytes	None	32K bytes (32K/16K bytes**Note**)	None	16K bytes
None				512 bytes	
640 bytes		1024 bytes	1024 bytes (1024/640 bytes**Note**)	512 bytes	
Program memory space: 64K bytes; Data memory space: 1M byte					
16				14	
12					
18	36	18	36	10	28
46	64	46	64	36	54
24	42	24	42	16	34
8	24	8	24	0	16
8					
8-bit input/output port					
8-bit input/output port				—	
8-bit input port					
8-bit input/output port					
—	8-bit I/O port	—	8-bit I/O port	—	8-bit I/O port
—	8-bit I/O port	—	8-bit I/O port	—	8-bit I/O port
4-bit output port + 2-bit I/O port	4-bit output port + 4-bit I/O port	4-bit output port + 2-bit I/O port	4-bit output port + 4-bit I/O port	4-bit output port + 2-bit I/O port	4-bit output port + 4-bit I/O port
8-bit input port				6-bit input port	

Note Set by software

Sub-series name		μPD78214 sub-series			μPD78218A sub-series		μPD78224 sub-series	
Item \ Product name		μPD78212	μPD78213	μPD78214 (μPD78P214)	μPD78217A	μPD78218A (μPD78P218A)	μPD78220	μPD78224 μPD78P224
PWM outputs		None						
Comparators		None					4 bits x 8	
A/D converter		8 bits x 8					None	
Conversion time selection		Select according to the operating frequency					—	
AV_{REF} input voltage range		3.4 V to V_{DD}			3.6 V to V_{DD}		—	
Input voltage related restriction		Always pin voltage from 0 V to AV_{REF} for pins selected by bits ANI0 to ANI2 of ADM register only			Pin voltage from 0 V to AV_{REF} for pins subject to A/D conversion only, during A/D conversion		—	
D/A converter		None						
Timer/ counters	16-bit timer counter	1						
	8-bit timer counter	3					2	
	Toggle outputs	4						
	PWM/PPG output	Yes					No	
	One-shot pulse	No			Yes		No	
	Interrupt sources	7					5	
External SFR area		16 bytes 0FFD0H to 0FFDFH					None	
Serial inter-face	UART	1 channel						
	CSI	1 channel (for SBI)						
	BRG timer	Yes (Dual function as timer/counter 3)					Yes	
	Dedicated baud rate generator	Yes					No	
	External baud rate clock input	Yes					No	
Real-time output ports		4 bits x 2 or 8 bits x 1						

(2/3)

μPD78234 sub-series				μPD78244 sub-series	
μPD78233	μPD78234	μPD78237	μPD78238 (μPD78P238)	μPD78243	μPD78244
12 bits x 2				None	
None					
8 bits x 8					
Can be selected arbitrarily.				Select according to the operating frequency.	
3.4 V to V_{DD}				3.6 V to V_{DD}	
Pin voltage from 0 V to AV_{REF} for pins subject to A/D conversion, only during A/D conversion					
8 bits x 2				None	
1					
3					
4					
Yes					
Yes					
7					
16 bytes 0FFD0H to 0FFDFH					
1 channel					
1 channel (for SBI)					
Yes (Dual function as timer/counter 3)					
Yes					
Yes					
4 bits x 2 or 8 bits x 1					

Sub-series name	μPD78214 sub-series			μPD78218A sub-series		μPD78224 sub-series	
Item \ Product name	μPD78212	μPD78213	μPD78214 (μPD78P214)	μPD78217A	μPD78218A (μPD78P218A)	μPD78220	μPD78224 μPD78P224
Interrupts	2 levels (programmable), vectored/macro service						
External	7					8	
Internal	12					9	
Interrupts permitted use of macro service	15					6	
Macro service counter bits	8 bits only			8/16 bits selectable (except Type A)		8 bits only	
Macro service type C MPD/MPT increment	Only low-order 8 bits incremented (high-order unchanged)			16 bits incremented		Only low-order 8 bits incremented (high-order unchanged)	
Macro service execution time	Macro service execution time of the μPD78214 series is the same as that of the μPD78224 series. Macro service execution time of the μPD78218A series is the same as that of the μPD78234 series or that of the μPD78244 series. The execution time varies depending on the mode. Compare these products by referencing their User's Manuals.						
Restriction when data is transferred from macro service type A memory to SFR	Generated when transfer data is D0H to DFH			Generated when transfer source buffer (memory) address is 0FED0H to 0FEDFH		Generated when transfer data is D0H to DFH	
Standby function	HALT/STOP mode						
Oscillation stabilization time after STOP mode release	Fixed			Choice of two times		Fixed	
Pseudo-SRAM refresh function	Yes (Refresh pulse width = $1/f_{CLK}$)					Yes (Refresh pulse width = $1.5/f_{CLK}$	
Memory access restrictions	None					FC80H to FDFFH not accessible when refresh function is used	
ROM-less mode setting	$\overline{EA}$ pin = low level	—	$\overline{EA}$ pin = low level	—	$\overline{EA}$ pin = low level	—	$\overline{EA}$ pin = low level
Package	• 64-pin plastic shrink DIP (750 mil) • 68-pin plastic QFJ (□ 950 mil): Except μPD78212 • 64-pin plastic QFP (14 x 14 mm body) • 74-pin plastic QFP (20 x 20 mm body) • 64-pin plastic QUIP: Except μPD78212 • 64-pin ceramic shrink DIP with window: μPD78P214 only			• 64-pin plastic shrink DIP (750 mil) • 64-pin plastic QFP (14 x 14 mm body) • 64-pin ceramic shrink DIP with window: μPD78P218A only		• 84-pin plastic QFJ (□ 1150 mil) • 94-pin plastic QFP (20 x 20 mm body)	

(3/3)

μPD78234 sub-series				μPD78244 sub-series	
μPD78233	μPD78234	μPD78237	μPD78238 (μPD78P238)	μPD78243	μPD78244
2 levels (programmable), vectored/macro service					
7					
12				14	
15					
8/16 bits selectable (except Type A)					
16 bits incremented					
Macro service execution time of the μPD78214 series is the same as that of the μPD78224 series. Macro service execution time of the μPD78218A series is the same as that of the μPD78234 series or that of the μPD78244 series. The execution time varies depending on the mode. Compare these products by referencing their User's Manuals.					
Generated when transfer data is D0H to DFH				Generated when transfer source buffer (memory) address is 0FED0H to 0FEDFH	
HALT/STOP mode					
Choice of two times					
Yes (Refresh pulse width = $1/f_{CLK}$)					
None					
—	MODE pin = high level	—	MODE pin = high level (not settable)	—	$\overline{EA}$ pin = low level
• 84-pin plastic QFJ (□ 1150 mil) • 80-pin plastic QFP (14 x 14 mm body) • 94-pin plastic QFP (20 x 20 mm body) • 94-pin ceramic WQFN: μPD78P238 only				• 64-pin plastic shrink DIP (750 mil) • 64-pin plastic QFP (14 x 14 mm body)	

[MEMO]

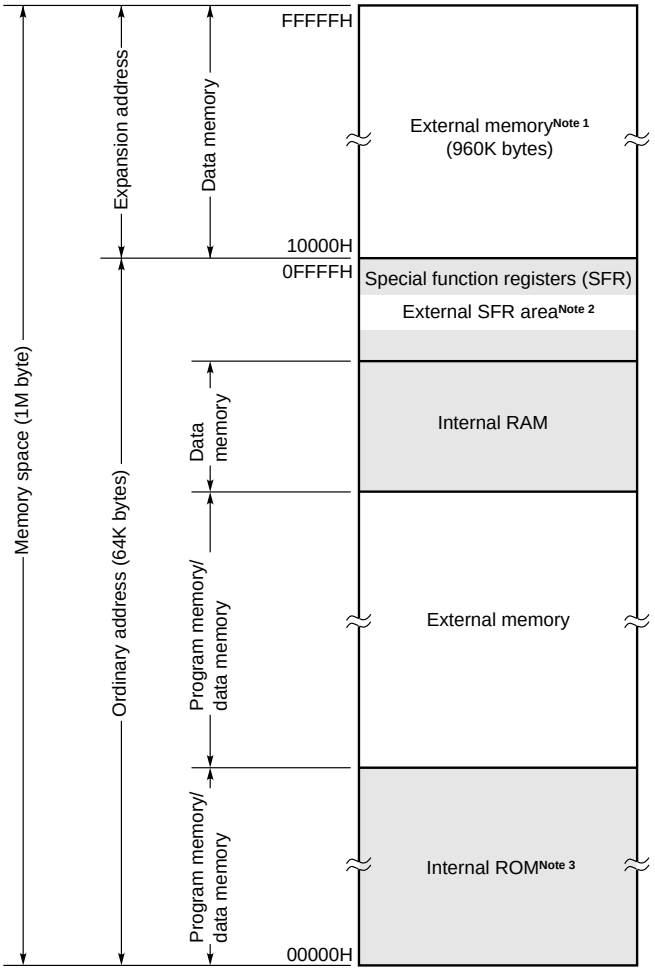
CHAPTER 3 MEMORY SPACE

3.1 MEMORY SPACE

In the 78K/II series, a 1M-byte memory space can be accessed. Program memory mapping differs according to the on-chip memory capacity and pin**Note** statuses. Therefore, see **Section 3.1.1** through **Section 3.1.5** for details of memory map address areas.

Note μPD78214 sub-series, 78218A sub-series, 78224 sub-series,
78224 sub-series, 78244 sub-series : $\overline{EA}$ pin
μPD78234 sub-series : MODE pin

Figure 3-1. Memory Map



Notes 1. Access in 1M-byte expansion mode, ▭ : Internal memory

2. Access in external memory expansion mode (except μPD78224 sub-series)

3. External data memory in μPD78213, 78217A, 78220, 78233, 78237 and 78243 (ROM-less products)

3.1.1 μPD78214 Sub-Series Memory Space

Product name	Program memory		Data memory	
	Internal ROM	External memory[Note 1]	Internal RAM	External memory[Note 2]
μPD78212 ($\overline{EA}$ = L)	—	64896 bytes 00000H to 0FD7FH	384 bytes 0FD80H to 0FEFFH	960K bytes 10000H to FFFFFH
μPD78212 ($\overline{EA}$ = H)	8K bytes 00000H to 01FFFH	56704 bytes 02000H to 0FD7FH		
μPD78213 ($\overline{EA}$ = L)	—	64768 bytes 00000H to 0FCFFH	512 bytes 0FD00H to 0FEFFH	
μPD78214 μPD78P214 ($\overline{EA}$ = L)				
μPD78214 μPD78P214 ($\overline{EA}$ = H)	16K bytes 00000H to 03FFFH	48384 bytes 04000H to 0FCFFH		

Notes 1. Access in external memory expansion mode. Common use with data memory is possible.
2. Access in 1M-byte expansion mode.

3.1.2 μPD78218A Sub-Series Memory Space

Product name	Program memory		Data memory	
	Internal ROM	External memory[Note 1]	Internal RAM	External memory[Note 2]
μPD78217A ($\overline{EA}$ = L)	—	64256 bytes 00000H to 0FAFFH	1024 bytes 0FB00H to 0FEFFH	960K bytes 10000H to FFFFFH
μPD78218A μPD78P218A ($\overline{EA}$ = L)				
μPD78218A μPD78P218 ($\overline{EA}$ = H)	32K bytes 00000H to 07FFFH	31488 bytes 08000H to 0FAFFH		

Notes 1. Access in external memory expansion mode. Common use with data memory is possible.
2. Access in 1M-byte expansion mode.

3.1.3 μPD78224 Sub-Series Memory Space

Product name	Program memory		Data memory	
	Internal ROM	External memory[Note 1]	Internal RAM	External memory[Note 2]
μPD78220 ($\overline{EA}$ = L)	—	64640 bytes 00000H to 0FC7FH	640 bytes 0FC80H to 0FC7FH	960K bytes 10000H to FFFFFH
μPD78224 μPD78P224 ($\overline{EA}$ = L)				
μPD78224 μPD78P224 ($\overline{EA}$ = H)	16K bytes 00000H to 03FFFH	48256 bytes 04000H to 0FC7FH		

Notes 1. Access in external memory expansion mode. Common use with data memory is possible.

2. Access in 1M-byte expansion mode.

3.1.4 μPD78234 Sub-Series Memory Space

Product name	Program memory		Data memory	
	Internal ROM	External memory[Note 1]	Internal RAM	External memory[Note 2]
μPD78233 (MODE = H)	—	64640 bytes 00000H to 0FC7FH	640 bytes 0FC80H to 0FEFFH	960K bytes 10000H to FFFFFH
μPD78234 (MODE = H)				
μPD78234 (MODE = L)	16K bytes 00000H to 03FFFH	48256 bytes 04000H to 0FC7FH		
μPD78237 (MODE = H)	—	64256 bytes 00000H to 0FAFFH	1024 bytes 0FB00H to 0FEFFH	
μPD78238 (MODE = H)				
μPD78238 (MODE = L)	32K bytes 00000H to 07FFFH	31488 bytes 08000H to 0FAFFH		
μPD78P238 (MODE = L)[Note 3]	μPD78234/78238 memory mapping can be selected by the memory size switchover register (IMS)			

Notes 1. Access in external memory expansion mode. Common use with data memory is possible.

2. Access in 1M-byte expansion mode.

3. Cannot be used as MODE = H (ROM-less operation specification) in the μPD78P238.

3.1.5 μPD78244 Sub-Series Memory Space

Product name	Program memory		Data memory		
	Internal ROM	External memory**Note 1**	EEPROM	Internal RAM	External memory**Note 2**
μPD78243 ($\overline{EA}$ = L)	—	64256 bytes 00000H to 0FAFFH	512 bytes 0FB00H to 0FCFFH	512 bytes 0FD00H to 0FEFFH	960K bytes 10000H to FFFFFH
μPD78244 ($\overline{EA}$ = L)					
μPD78244 ($\overline{EA}$ = H)	16K bytes 00000H to 03FFFH	47872 bytes 04000H to 0FAFFH			

Notes 1. Access in external memory expansion mode. Common use with data memory is possible.

2. Access in 1M-byte expansion mode.

3.2 INTERNAL PROGRAM MEMORY AREA (INTERNAL ROM)

ROM is incorporated in 78K/II series products in the address spaces shown below, and can be used to store programs, table data, etc. This area is usually addressed by the program counter (PC).

Product	Address space	Internal ROM
μPD78212	00000H to 01FFFH	8K x 8 bits
μPD78214	00000H to 03FFFH	16K x 8 bits
μPD78P214		
μPD78224		
μPD78P224		
μPD78234		
μPD78244		
μPD78218A	00000H to 07FFFH	32K x 8 bits
μPD78P218A		
μPD78238		
μPD78P238		

Remark In ROM-less products or ROM-less operation in a product with on-chip ROM, this address space functions as external memory.

3.3 VECTOR TABLE AREA

The 64-byte area from 00000H to 0003FH is reserved as a vector table area. The vector table area holds the program start addresses used when a branch is made due to $\overline{\text{RESET}}$ input or generation of an interrupt request. The low-order 8 bits of a 16-bit address are stored in an even address, and the high-order 8 bits in an odd address.

Any part of the area which is not used as a vector table can be used as program memory or data memory.

Table 3-1. Vector Table

Vector table address	Interrupts
00000H	Reset ($\overline{\text{RESET}}$ input)
00002H	NMI
00006H	INTP0
00008H	INTP1
0000AH	INTP2
0000CH	INTP3
0000EH	INTP4/INTC30**Note 1**
00010H	INTP5/INTAD**Note 1**
00012H	INTC20**Note 1**/INTP6**Note 2**
00014H	INTC00
00016H	INTC01
00018H	INTC10
0001AH	INTC11
0001CH	INTC21
00020H	INTSER
00022H	INTSR
00024H	INTST
00026H	INTCSI
00028H	INTEER**Note 3**
0002AH	INTEPW**Note 3**
0003EH	BRK

Notes 1. Except μPD78224 sub-series

2. μPD78224 sub-series only

3. μPD78244 sub-series only

3.4 CALLT INSTRUCTION TABLE AREA

The 64-byte area from 00040H to 0007FH is used to store 1-byte call instruction (CALLT) subroutine entry addresses.

In the CALLT instruction, this table is referenced and a branch is made to the address written in the table as a subroutine. As the CALLT instruction is one byte in length, the program object size can be compressed by using the CALLT instruction for subroutine calls which appear numerous times within a program. As a maximum of 32 subroutine entry addresses can be written in the table, it is recommended that they be registered in order to frequency of use.

If this area is not used as the CALLT instruction table, it can be used as ordinary program memory or data memory.

3.5 CALLF INSTRUCTION ENTRY TABLE

The area from 00800H to 00FFFH can be accessed by a direct subroutine call by means of a 2-byte call instruction (CALLF).

Since CALLF is a 2-byte call instruction, the object size can be compressed compared with use of the direct subroutine call instruction CALL (3 bytes). Speed is also increased when operating with external ROM.

When high speed is required, writing direct subroutines in this area is an effective solution.

When it is wished to reduce the object size, this can be achieved by writing an unconditional branch (BR) instruction in this area and locating the subroutine itself outside this area for subroutines which are called from 4 or more places. In this case, only the 3 bytes of the BR instruction take up space in the CALLF entry area, and thus the object size can be reduced with many subroutines.

3.6 INTERNAL RAM AREA

78K/II series products incorporate general-purpose static RAM.

This area is configured as follows:

- Internal RAM area
  - Peripheral RAM (PRAM)
  - Internal dual-port RAM (IRAM)

Table 3-2. Internal RAM Area in 78K/II Series Products

Internal RAM / Product name	Internal RAM area	Peripheral RAM: PRAM	Internal dual-port RAM: IRAM
μPD78212	384 bytes (0FD80H to 0FEFFH)	128 bytes (0FD80H to 0FDFFH)	256 bytes (0FE00H to 0FEFFH)
μPD78213	512 bytes (0FD00H to 0FEFFH)	256 bytes (0FD00H to 0FDFFH)	
μPD78214 μPD78P214			
μPD78217A	1024 bytes (0FB00H to 0FEFFH)	768 bytes (0FB00H to 0FDFFH)	
μPD78218A μPD78P218A			
μPD78220	640 bytes (0FC80H to 0FEFFH)	384 bytes (0FC80H to 0FDFFH)	
μPD78224 μPD78P224			
μPD78233			
μPD78234			
μPD78237	1024 bytes (0FB00H to 0FEFFH)	768 bytes (0FB00H to 0FDFFH)	
μPD78238 μPD78P238			
μPD78243	512 bytes (0FD00H to 0FEFFH)	256 bytes (0FD00H to 0FDFFH)	
μPD78244			

Internal dual-port RAM (IRAM) allows high-speed access. In particular, the area from 0FE20H to 0FEFFH can be used in short direct addressing mode for high-speed access.

This area is mapped as shown below.

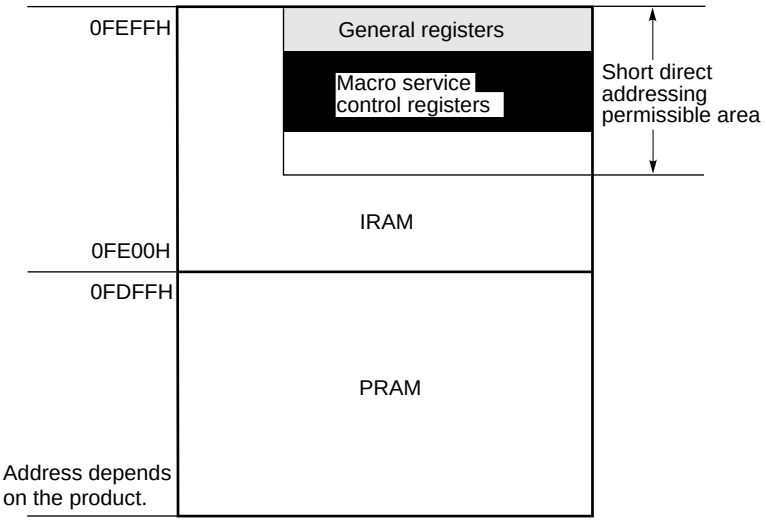
- General register (0FEE0H to 0FEFFH) : (8 registers x 8 bits x 4 banks)
- Macro service control word (0FEC2H (0FED4H**Note**) to 0FEDFH) : (30 (12**Note**) x 8 bits)

Peripheral RAM (PRAM) is used as ordinary data memory.

Note μPD78224 sub-series

Internal RAM mapping is shown in Figure 3-2.

Figure 3-2. Internal RAM Mapping



Cautions 1. Program fetching is not possible from the internal RAM area.

2. In the μPD78224 sub-series, peripheral RAM (PRAM) cannot be used when the refresh function is used.

Remark It is effective to locate frequency accessed data, work areas, status flags, etc., in the area from 0FE20H to 0FEC1H (0FED3H**Note**). Also, using the area from 0FE00H to 0FE1FH for the stack area or the area for macro service channel and macro service data transfers allows fast accesses, and is thus effective in improving system throughput. (Short direct addressing cannot be used on this area: It is manipulated in the same way as the rest of the memory space. However, since this area can be accessed at a higher speed than the rest of the memory space, from an overall viewpoint its use is recommended in the applications mentioned above.)

Note μPD78224 sub-series

3.7 EEPROM AREA (μPD78244 SUB-SERIES ONLY)

The 512-byte area from 0FB00H to 0FCFFH has EEPROM mapped onto it. EEPROM is memory which can be written to or read by a program, and which unlike internal RAM, retains data in the event of a power failure.

Caution The following instructions cannot be used on this area:

- **MOVW meml, AX** **MOVW & meml, AX**
- **ROR4 meml** **ROR4 & meml**
- **ROL4 meml** **ROL4 & meml**

3.8 SPECIAL FUNCTION REGISTER (SFR) AREA

The area from 0FF00H to 0FFFFH has on-chip peripheral hardware special function registers (SFRs) mapped onto it (see **documentation for individual products**).

With the exception of μPD78224 sub-series products, the area from 0FFD0H to 0FFDFH is mapped as the external SFR area, and allows access to externally connected peripheral I/Os etc. in a ROM-less product or in external memory expansion mode (set by the memory expansion mode register (MM)) in a product with on-chip ROM.

Caution Addresses in this area which are not mapped as SFRs should not be accessed. An illegal access may result in CPU deadlock. A deadlock state can be released by reset input only.

3.9 EXTERNAL SFR AREA (EXCEPT μPD78224 SUB-SERIES)

The 16-byte area from 0FFD0H to 0FFDFH within the SFR area is mapped as the external SFR area. In the case of a ROM-less product or use of external memory expansion mode (set by the memory expansion mode register (MM)) in a product with on-chip ROM, externally connected peripheral I/Os etc. can be accessed using the address bus and address/data bus.

As the external SFR area can be accessed by SFR addressing, it has special characteristics such as allowing easy peripheral I/O manipulation, etc., enabling the object size to be reduced, and so forth.

Bus operations when accessing the external SFR area are the same as for ordinary memory accesses.

Caution There is no external SFR area in the μPD78224 sub-series.

3.10 EXTERNAL MEMORY SPACE

External memory space is memory space which can be accessed in accordance with the setting of the memory expansion mode register (MM). It can be used for storage of programs, table data, etc., and allocation of peripheral I/O devices.

Table 3-3. External Memory Space in 78K/II Series Products

Product name	External memory space
μPD78212	64896 bytes (02000H to 0FD7FH)
μPD78213**Note**	64768 bytes (00000H to 0FCFFH)
μPD78214 μPD78P214	48384 bytes (04000H to 0FCFFH)
μPD78217A**Note**	64256 bytes (00000H to 0FAFFH)
μPD78218A μPD78P218A	31488 bytes (08000H to 0FAFFH)
μPD78220**Note**	64640 bytes (00000H to 0FC7FH)
μPD78224 μPD78P224	48256 bytes (04000H to 0FC7FH)
μPD78233**Note**	64640 bytes (00000H to 0FC7FH)
μPD78234	48256 bytes (04000H to 0FC7FH)
μPD78237**Note**	64256 bytes (00000H to 0FAFFH)
μPD78238 μPD78P238	31488 bytes (08000H to 0FAFFH)
μPD78243**Note**	64256 bytes (00000H to 0FAFFH)
μPD78244	47872 bytes (04000H to 0FAFFH)

Note ROM-less product

3.11 EXTERNAL EXPANSION DATA MEMORY SPACE

The area from 10000H to FFFFFH is space which can be accessed when the 1M-byte expansion mode has been specified by means of the memory expansion mode register (MM). In this case, pins P60 to P63 of port 6 function as the 4-bit expansion address bus (A16 to A19). The data memory space is handled as sixteen 64K-byte banks, with the P6 register and the low-order 4 bits of the PM6 register functioning as the bank registers used to select the bank.

This space is useful when handling large amounts of data, as in the case of a kanji character generator, for example.

This space can only be accessed when the bank to be used (4-bit address information on pins A16 to A19) has been set beforehand in the bank register (P60 to P63 of the P6 register or PM60 to PM63 of the PM6 register), and during execution of an instruction which has extended addressing capability.

As there are two bank registers (P6 and PM6), two banks can normally be used as banks on which operations are to be performed. One or other of the bank registers is selected according to whether or not "&" is affixed to the instruction operand. The P6 register is selected as the bank register when "&" is used, and the PM6 register is selected when "&" is not used.

This function facilitates processing in which, for example, data read from data ROM (e.g. kanji data for printing) is expanded or compressed and stored in RAM if a RAM area is specified with one of the two banks used as the main data bank, while a data ROM area is specified with the other bank used as the subsidiary bank.

Remark The operation code and execution time of instructions which use PM6 register as the bank register are shorter than those of instructions which use the P6 register. Also, instructions which manipulate the P6 register are shorter and have a shorter execution time than instructions which manipulate the PM6 register.
It is therefore efficient to use the PM6 register as the main bank register which specifies frequently accessed banks, and to use the P6 register as the subsidiary bank register for which the specified bank frequently changes.

Cautions 1. In the μPD78224 sub-series the low-order 4 bits (PM60 to PM63) of the PM6 register should always be set to "0" before use. Since the low-order 4 bits of the PM6 register are used as "0", bank 0 is always accessed in an addressing mode without "&".

2. When the external expansion data memory space is not used, ensure to set "0" to the low-order 4 bits of the PM6 register (PM60 to PM63). A normal emulation cannot be performed by an in-circuit emulator if "0" is not set.

Example 1. μPD78224 sub-series inter-bank data transfer

- To select bank 5 as the expansion bank and transfer bank 5 data to bank 0.

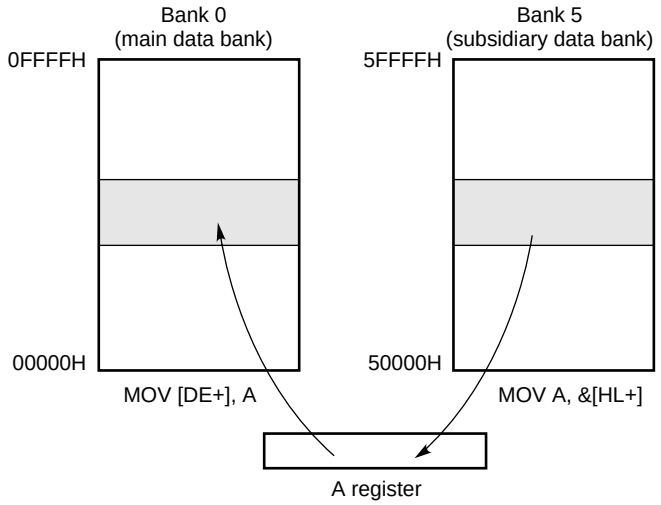
```
        MOV     MM, #47H    ; Set memory expansion mode
        MOV     PM6, #0H    ; Low-order 4 bits of PM6 always set to "0"
        MOV     P6, #5H     ; Subsidiary bank register (P6) setting
        MOV     B, #0FFH    ; Loop counter setting

LOOP:           ⋮

        MOV     A, &[HL+]   ; Read from bank 5 (P6 register contents added
                            as maximum address information)
        MOV     [DE+], A    ; Store data in bank 0
                              (instruction without "&" accesses bank 0)
                ⋮

        DBNZ    B, $LOOP    ; Repeat processing
```

Figure 3-3. Example of Inter-Bank Data Transfer



Remark Both banks of MOV [DE+], and MOV A &[HL+] are stored in 0.

Example 2. Inter-bank data transfer in non-μPD78224 sub-series product

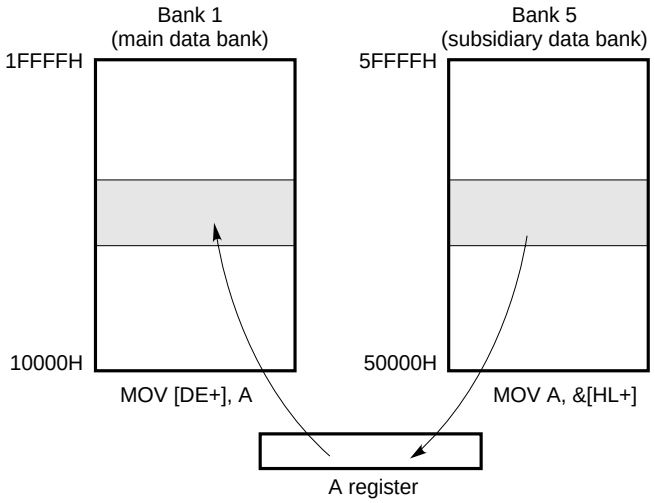
- To select bank 1 as the main bank and bank 5 as the subsidiary bank, and transfer bank 5 data to bank 1.

```
        MOV     MM, #47H    ; Set memory expansion mode
        MOV     PM6, #1H    ; Main bank register (PM6) setting
        MOV     P6, #5H     ; Subsidiary bank register (P6) setting
        MOV     B, #0FFH    ; Loop counter setting

LOOP:   ⋮

        MOV     A, &[HL+]   ; Read from bank 5 (P6 register contents added
                            as maximum address information)
        MOV     [DE+], A    ; Store data in bank 1 (PM6 register contents
                            added as maximum address information)
                ⋮
        DBNZ    B, $LOOP    ; Repeat processing
```

Figure 3-4. Example of Inter-Bank Data Transfer



Remark Both banks of MOV [DE+], A and MOV A, &[HL+] are stored in 0.

CHAPTER 4 REGISTERS

4.1 CONTROL REGISTERS

Control registers comprise the program counter (PC), program status word (PSW), and stack pointer (SP).

4.1.1 Program Counter (PC)

The program counter is a 16-bit binary counter which holds program memory address information (see **Figure 4-1**).

The program counter is normally incremented automatically by the number of instruction bytes fetched. When an instruction associated with a branch is executed, immediate data or register contents are set in the PC.

When a $\overline{\text{RESET}}$ signal is input, the contents of address 00000H of internal ROM (or external memory in the μPD78213, 78217A, 78220, 78233, 78237 and 78243) are set in the low-order 8 bits of the PC, and the contents of address 00001H in the high-order 8 bits.

Figure 4-1. Program Counter Configuration

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PC	PC15	PC14	PC13	PC12	PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0

4.1.2 Program Status Word (PSW)

The program status word (PSW) is an 8-bit register consisting of various flags which are set or reset depending on the result of executing an instruction (see **Figure 4-2**).

In addition to being written to or read as an 8-bit unit, flags can be manipulated individually by means of bit manipulation instructions. The PSW is saved to the stack when a vectored interrupt request is acknowledged or when a BRK or PUSH PSW instruction is executed, and restored when a RETI, RETB or POP PSW instruction is executed.

The PSW is set to 02H by $\overline{\text{RESET}}$ input (interrupt acknowledgment disabled state).

Figure 4-2. Program Status Word Configuration

	7	6	5	4	3	2	1	0
PSW	IE	Z	RBS1	AC	RBS0	0	ISP	CY

(1) Carry flag (CY)

The carry flag stores overflow or underflow during execution of an addition or subtraction instruction. This flag also stores the shifted-out value when a shift/rotate instruction is executed, and functions as a one-bit accumulator when a bit manipulation instruction is executed.

(2) Interrupt priority status flag (ISP)

This flag controls the priority of currently acknowledgeable maskable vectored interrupts. When this flag is "0" low-order vectored interrupts specified by the priority specification flag register (PR0) are acknowledgment-disabled, and when "1" acknowledgment is enabled regardless of priority. Actual acknowledgment is controlled by the status of the IE flag.

ISP contents are updated each time a maskable vectored interrupt is acknowledged.

(3) Register bank selection flag (RBS0, RBS1)

This is a 2-bit flag which selects one of four register banks (see **Table 4-1**).

2-bit information indicating the selected register bank is stored by execution of the SEL RBn instruction.

Table 4-1. Register Bank Selection

RBS1	RBS0	Specified Register Bank
0	0	Register bank 0
0	1	Register bank 1
1	0	Register bank 2
1	1	Register bank 3

(4) Auxiliary carry flag (AC)

This flag is set (1) in the event of a carry out of bit 3 or a borrow into bit 3 as the result of an operation, and reset (0) otherwise.

The AC flag is used when a BCD adjustment instruction is executed.

(5) Zero flag (Z)

This flag is set (1) when the result of an operation is zero, and reset (0) otherwise.

(6) Interrupt request enable flag (IE)

This flag controls CPU interrupt request acknowledgment operations.

When "0", the interrupt-disabled state is set and only acknowledgment of nonmaskable interrupts and macro service with masking released is possible; other interrupts are disabled.

When "1", the interrupt-enabled state is set, and enabling of interrupt request acknowledgment is controlled by the ISP flag, the interrupt mask flag corresponding to the particular interrupt request, and the priority specification flag.

The IE flag is set (1) by execution of the EI instruction, and reset (0) by execution of the DI instruction or interrupt acknowledgment.

4.1.3 Stack Pointer (SP)

This is a 16-bit register which holds the start address of the stack area (LIFO method: 00000H to 0FFFFH) (see **Figure 4-3**). The SP is used for addressing the stack area during subroutine or interrupt servicing.

SP contents are decremented before being written to the stack area, and incremented after being read from the stack area (see **Figures 4-4** and **4-5**).

The SP is accessed by dedicated instructions.

Since SP contents are undefined after $\overline{\text{RESET}}$ input, the SP should always be initialized by an initialization program immediately after reset release (before issuing a subroutine call or acknowledging an interrupt).

Example SP initialization

```
MOVW SP, #0FEE0H; SP <- 0FEE0H (when used from FEDFH)
```

Figure 4-3. Stack Pointer Configuration

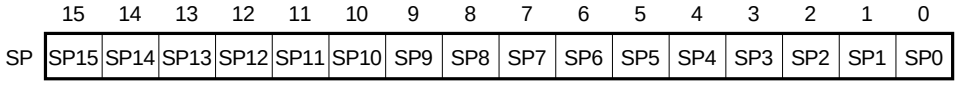


Figure 4-4. Data Saved to Stack Area

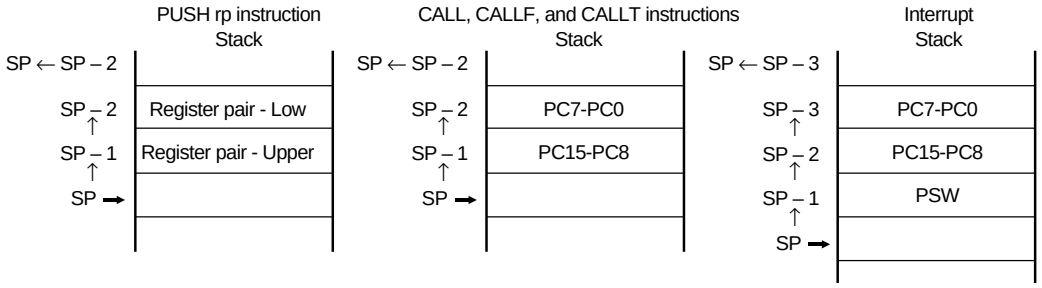
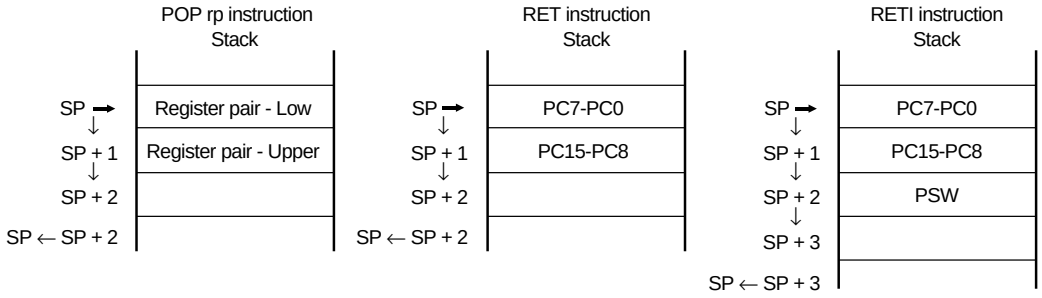


Figure 4-5. Data Restored from Stack Area



Cautions 1. In stack addressing, the entire 64K-byte space can be accessed, but no stack area can be secured in the SFR area and internal ROM area.

2. $\overline{\text{RESET}}$ input makes the SP undefined. A non-maskable interrupt is also acknowledgeable immediately after reset release. Therefore, if a non-maskable interrupt request is generated with the unstable SP immediately after reset release, it may perform unexpected operation. In order to minimize this risk, be sure to perform the SP initialization immediately after reset release.

4.2 GENERAL REGISTERS

4.2.1 Configuration

General registers are configured as four banks each containing eight 8-bit registers (X, A, C, B, E, D, L, H) mapped onto a specific address area (0FEE0H to 0FEFFH) (see **Figure 4-6**).

Figure 4-6. General Register Configuration

	(8-bit processing)			(16-bit processing)
0FEE0H	A E1H	X E0H	↑	AX E0H
	B E3H	C E2H	Register bank 3 (RBS1, 0 = 11)	BC E2H
	D E5H	E E4H		DE E4H
	H E7H	L E6H	↓	HL E6H
	A E9H	X E8H	↑	AX E8H
	B EBH	C EAH	Register bank 2 (RBS1, 0 = 10)	BC EAH
	D EDH	E ECH		DE ECH
	H EFH	L EEH	↓	HL EEH
	A F1H	X F0H	↑	AX F0H
	B F3H	C F2H	Register bank 1 (RBS1, 0 = 01)	BC F2H
	D F5H	E F4H		DE F4H
	H F7H	L F6H	↓	HL F6H
	A F9H	X F8H	↑	AX F8H
	B FBH	C FAH	Register bank 0 (RBS1, 0 = 00)	BC FAH
	D FDH	E FCH		DE FCH
0FEFFH	H FFH	L FEH	↓	HL FEH

The register bank used when an instruction is executed is specified by a CPU control instruction (SEL RBn).

When $\overline{\text{RESET}}$ is input, register bank 0 is specified.

The register bank in use during instruction execution can be checked by reading the register bank selection flag (RBS0, 1) in the PSW.

The area 0FEE0H to 0FEFFH can be addressed or accessed as a normal data memory irrespective of whether or not it is used as a general register.

Remark If it is necessary to return to the original register bank when the register bank is changed, the SEL RBn instruction should be executed after saving the PSW to the stack using the PUSH PSW instruction. If the stack location has not changed, the POP PSW instruction can be used to return to the original register bank.

When the register bank is changed by an interrupt service program, the PSW is saved to the stack automatically when the interrupt is acknowledged, and is restored by the RETI or RETB instruction. Therefore, when only one register bank is used by the interrupt service routine, only the SEL RBn instruction need be executed, and execution of the PUSH PSW and POP PSW instructions is not necessary.

Example 1. When register bank is changed by a normal program register

```
Specifying register bank 2
  ⋮
PUSH    PSW
SEL     RB2      ┐
  ⋮              │
                 ├ Operated by register bank 2
POP     PSW      ┘
  ⋮                Operated by original register bank
```

2. When register bank is changed by an interrupt servicing program

```
Specifying register bank 1

SEL  RB1         ┐
  ⋮              │
                 ├ Operated by register bank 1
RETI             ┘ Restored automatically to original register bank when
                   restoring to interrupt service program
```

4.2.2 Functions

In addition to being manipulated as 8-bit units, general registers can also be manipulated as 16-bit units by pairing two 8-bit registers (AX, BC, DE, HL).

These registers can be used for general purposes such as temporary storage of operation results or as operands in inter-register operation instructions.

The area from 0FEE0H to 0FEFFH can be addressed and accessed as ordinary data memory regardless of whether or not it is used as a general register area.

As 78K/II series is provided with 4 register banks, efficient programs can be written by using different register banks for normal processing and interrupt servicing.

The individual registers have the specific functions described below.

A (R1):

Mainly used for 8-bit data transfers and operation processing. Can also be used for bit data storage.
In addition, this register can also be used to hold the offset value when indexed addressing is used.

AX (RP0):

Mainly used for 16-bit data transfers and operation processing.

X (R0):

Can store bit data.

B (R3):

Has a loop counter function and can be used by the DBNZ instruction.
In addition, this register can also be used to hold the offset value when indexed addressing is used.

C (R2):

Has a loop counter function and can be used by the DBNZ instruction.

DE (RP2), HL (RP3):

These register pairs have a pointer function, and operate as the register which specifies the base address when register indirect addressing or based addressing is used.
In addition, these register pairs also operate as the register which holds the offset value in indexed addressing.
The registers can be described by their absolute names (R0 to R7, RP0 to RP3) as well as by their functional names (X, A, C, B, E, D, L, H, AX, BC, DE, HL) which indicate their specified functions. The correspondence between these names is shown in **Table 4-2**.

Table 4-2. Correspondence Between Function Names and Absolute Names

(1) 8-bit operations

Functional name	Absolute name
X	R0
A	R1
C	R2
B	R3
E	R4
D	R5
L	R6
H	R7

(2) 16-bit opeations

Functional name	Absolute name
AX	RP0
BC	RP1
DE	RP2
HL	RP3

4.3 SPECIAL FUNCTION REGISTERS (SFR)

These are registers to which special functions are allocated, such as on-chip peripheral hardware mode registers, control registers, etc., and are mapped onto the 256-byte space from 0FF00H to 0FFFFH. Please refer to individual product documentation for details of the special function registers.

Caution Addresses in this area which are not allocated to SFRs should not be accessed. An illegal access may result in CPU deadlock. A deadlock state can be released by reset input only.

[MEMO]

CHAPTER 5 INTERRUPT FUNCTIONS

The 78K/II series is provided with two modes for servicing interrupt requests. These two servicing modes can be set as required by the program.

In addition, multiprocessing control using two priority levels can easily be performed for maskable vectored interrupts.

Table 5-1. Interrupt Request Servicing Modes

Interrupt request servicing mode	Servicing by	PC/PSW contents	Servicing mode
Vectored interrupts	Software	According to save/restore operation	Executed after branching to desired service program
Macro service	Hardware (Firmware)	Retained	Execution of preset processing such as memory-I/O data transfer, etc.

Remark There are some interrupt request sources for which macro service cannot be used. Please refer to individual product documentation for details.

5.1 INTERRUPT REQUESTS

There are three kinds of interrupt request:

- Software interrupt requests
- Nonmaskable interrupt requests
- Maskable interrupt requests

5.1.1 Software Interrupt Requests

An interrupt request by software is generated by execution of a BRK instruction (vectored interrupt).

An interrupt request generated by the BRK instruction can be acknowledged even in the interrupt disabled (DI) state. These interrupts are not subject to interrupt priority control. Therefore, when the BRK instruction is executed, the vector table contents are placed in the PC and a branch is performed unconditionally.

Nesting in its own routine is also possible by executing the BRK instruction in a BRK instruction service routine.

An RETB instruction is executed to return from the BRK instruction service routine.

5.1.2 Nonmaskable Interrupt Requests

A nonmaskable interrupt request is generated when a valid edge as specified by bit 0 (ESNMI) of external interrupt mode register 0 (INTM0) is input to the NMI pin.

A nonmaskable interrupt request is unconditionally acknowledged even in the interrupt disable (DI) state. This kind of interrupt request is not subject to interrupt priority control, and takes priority over all other interrupts.

5.1.3 Maskable Interrupt Requests

Maskable interrupt requests are controlled by the setting of the interrupt mask register (MK0). In addition, acknowledgment can be specified as enabled/disabled for maskable interrupts as a whole by means of the IE flag in the PSW.

The order of priority when multiple maskable interrupt requests with the same priority are generated simultaneously is fixed (default priority). It is also possible to perform multiprocessing control by dividing interrupt priorities into a high-priority group and a low-priority group by means of the priority specification flag register (PR0). However, macro servicing acknowledgment is performed without regard to priority control to the IE flag.

5.2 MACRO SERVICE FUNCTION

In macro service, when an interrupt is acknowledged, CPU execution is temporarily suspended and the service set by firmware is executed. As macro service is performed without CPU intervention, it is not necessary to save/restore CPU status information such as that held in the PC and PSW. This method is thus effective in improving the CPU service time.

There are three types of macro service: A, B and C.

(1) Type A

One byte of data is transferred between a special function register (SFR) and memory each time an interrupt request is generated, and when the specified number of data transfers have been performed, a vectored interrupt request is generated.

The SFR involved in the transfer is fixed for each interrupt request, and memory is limited to addresses 0FE00H through 0FEFFH in internal RAM.

The specification method is simple, making this type suitable for small-volume high-speed data transfers.

(2) Type B

As with type A, one byte of data is transferred between a special function register (SFR) and memory each time an interrupt request is generated, and when the specified number of data transfers have been performed, a vectored interrupt request is generated.

The SFR and memory involved in the transfer are specified by the macro service channel (memory is limited to the 64K-byte space from 00000H to 0FEFFH).

This is a general-purpose version of type A, suitable for use with a large volume of transfer data.

(3) Type C

Each time an interrupt request is generated, data is transferred one byte at a time from memory to the real-time output port and the 8-bit timer/counter 1 compare register. When the specified number of data transfers have been performed, a vectored interrupt is generated.

In addition to performing data transfers to two locations in response to a single interrupt request, the type C macro service can be used with the addition of output data ring control (a function for repeated transfers of a series of data) and a function for automatic addition of the compare register and data, etc.

The type C macro service can only be used with INTC10 and INTC11 interrupts, and only certain SFRs can be used in the transfers. The 64K-byte memory space from 00000H to 0FEFFH can be used.

Type C is suitable for stepping motor control, etc., by means of real-time output port macro service.

[MEMO]

CHAPTER 6 ADDRESSING

6.1 INSTRUCTION ADDRESS ADDRESSING

The instruction address is determined by the contents of the program counter (PC), and is normally incremented automatically (+1 for each byte) in accordance with the number of instruction bytes fetched each time an instruction is executed. However, when an instruction involving a branch is executed, the branch destination address is set in the PC in accordance with the addressing methods shown below, and then the program branches to that address.

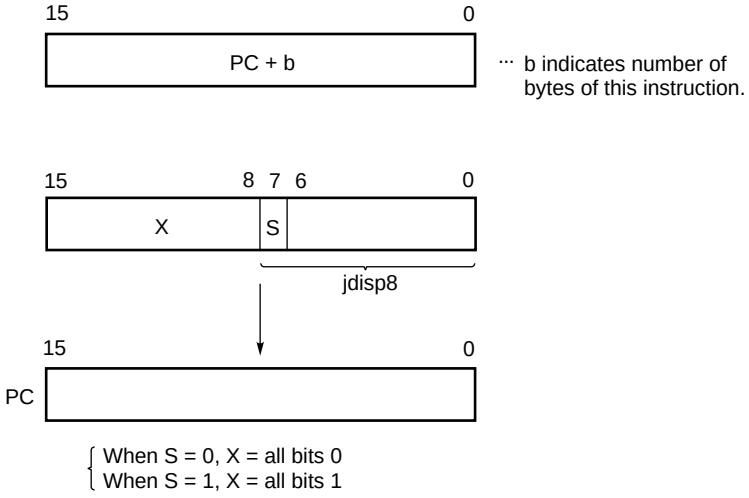
6.1.1 Relative Addressing

[Function]

The value obtained by adding the 8-bit immediate data (displacement value: jdisp8) of the operation code to the start address of the next instruction is transferred to the program counter (PC), then a branch is performed. The displacement value is treated as signed two's complement data (–128 to +127), with bit 7 as the sign bit.

This is performed when the BR $addr16 instruction or a conditional branch instruction is executed.

[Illustration]



6.1.2 Immediate Addressing

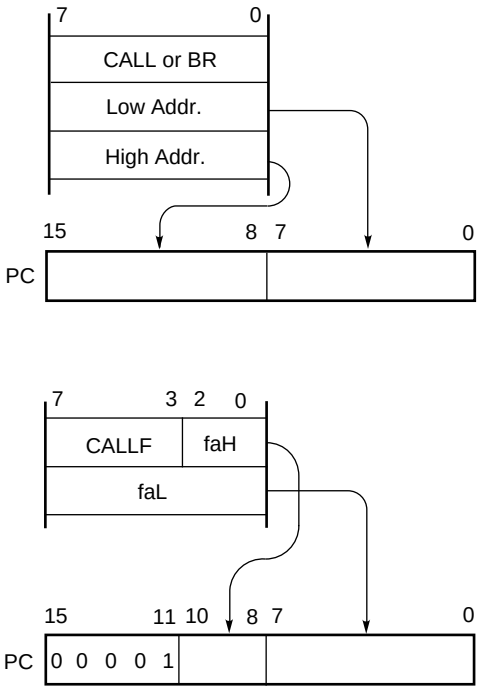
[Function]

The immediate data in the instruction word is transferred to the program counter (PC) and a branch is performed.

This is performed when the CALL !addr16, BR !addr16, or CALLF !addr11 instruction is executed.

In the case of the CALLF !addr11 instruction, the high-order 5-bit address is fixed at 00001.

[Illustration]



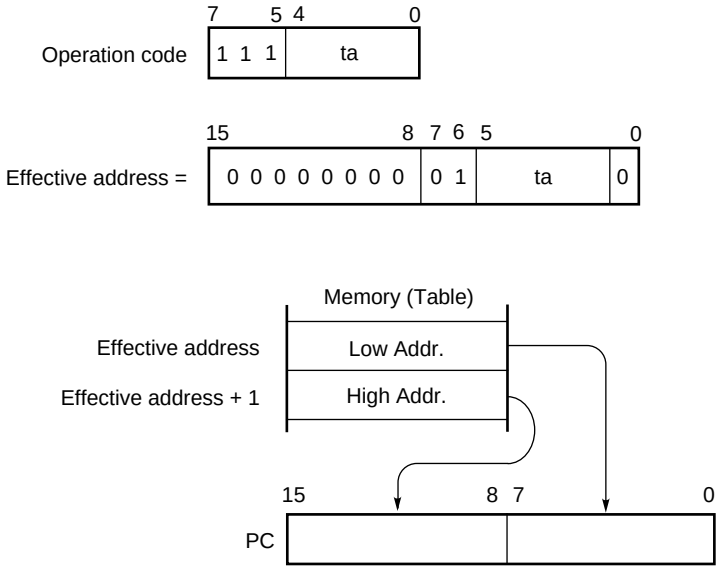
6.1.3 Table Indirect Addressing

[Function]

The contents (branch destination address) of the table in the specific location addressed by the immediate data in the low-order 5 bits of the operation code are transferred to the program counter (PC) and a branch is performed.

This is performed when the CALLT [addr5] instruction is executed.

[Illustration]



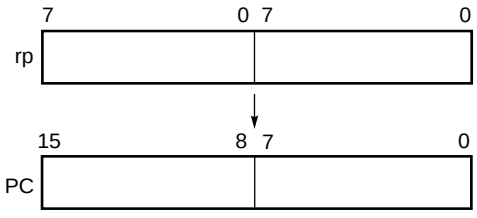
6.1.4 Register Addressing

[Function]

The contents of the register pair (RP0 to RP3) specified by the instruction word are transferred to the program counter (PC) and a branch is performed.

This is performed when a BR rp or CALL rp instruction is executed.

[Illustration]



6.2 OPERAND ADDRESS ADDRESSING

There are a number of methods (addressing methods) as described below for specifying the registers, memory, etc. to be manipulated when an instruction is executed.

6.2.1 Implied Addressing

[Function]

This addressing automatically addresses a register functioning as an accumulator (A, AX) in the general register area.

78K/II series instructions which use implied addressing in the instruction word are shown below.

Instruction	Register specified by implied addressing
MULU	A register as multiplicand, AX register as register storing product
DIVUW	AX register as register storing dividend and quotient
ADJBA/ADJBS	A register as register storing number subject to decimal adjustment
ROR4/ROL4	A register as register storing digit data subject to digit rotation (only low-order 4 bits used)

[Operand format]

As use is determined automatically according to the instruction, there is no specific operand format.

[Coding example]

MULU r ; In an 8-bit x 8-bit multiplication instruction, the product of the A register and r register is stored in AX. The A and AX registers are specified by implied addressing.

6.2.2 Register Addressing

[Function]

This addressing method accesses as an operand the general register specified by the register specification code (Rn, Pn) in the instruction word in the register bank specified by the register bank selection flag (RBS1, RBS0).

Register addressing is performed when an instruction with the operand format shown below is executed; when an 8-bit register is specified, one of eight registers is specified by 3 bits in the operation code, or one of two registers by 1 bit.

When a 16-bit register pair is specified, one of four register pairs is specified by 2 bits in the operation code.

[Operand format]

Identifier	Description
r	X, A, C, B, E, D, L, H
rl	C, B
rp	AX, BC, DE, HL

In addition to the functional names (X, A, C, B, E, D, L, H, AX, BC, DE, HL), absolute names (R0 to R7, RP0 to RP3) can be specified for r, r1 and rp.

[Coding example 1]

- General example

  MOV A, r

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

- Specific example

  MOV A, C ; When C register is selected as r

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

[Coding example 2]

- General example

  INCW rp

  Operation code

0	1	0	0	0	1	P_1	P_0

- Specific example

  INCW DE; When DE register pair is selected as rp

  Operation code

0	1	0	0	0	1	1	0

6.2.3 Immediate Addressing

[Function]

In this addressing method, 8-bit data and 16-bit data to be manipulated are included in the operation code.

[Operand format]

Immediate addressing is used when executing instructions with the operands shown below.

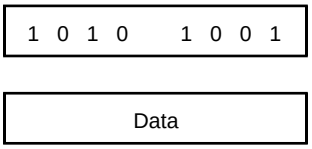
Identifier	Description
byte	Label or 8-bit immediate data
word	Label or 16-bit immediate data

[Coding example]

- General example

  ADD A, #byte

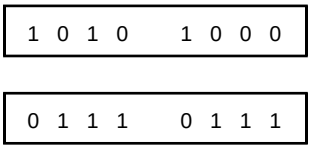
  Operation code

- Specific example

  ADD A, #77H ; When 77H is used for byte

  Operation code

6.2.4 Short Direct Addressing

[Function]

This addressing method uses 8-bit immediate data in the instruction word to directly address the memory to be manipulated in a fixed space. This kind of addressing can be used with most instructions, and allows various kinds of data to be manipulated using a small number of bytes and clock cycles.

This addressing method is used on the 256-byte space from 0FE20H to 0FF1FH. Internal RAM is mapped onto addresses 0FE20H to 0FEFFH, and special function registers (SFR) onto addresses FF00H to FF1FH.

The SFR area (0FF00H to 0FF1FH) on which short direct addressing is used has mapped onto it the ports, timer/counter unit compare registers and capture registers frequently used by the program. These SFRs can be manipulated using a small number of bytes and clock cycles.

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

[Operand format]

This type of addressing is used when executing instructions which include saddr or saddrp in their operands. In an instruction using saddrp, two bytes of data can be accessed: The memory addressed by the effective address as the lower byte, and the memory in the next address as the higher byte.

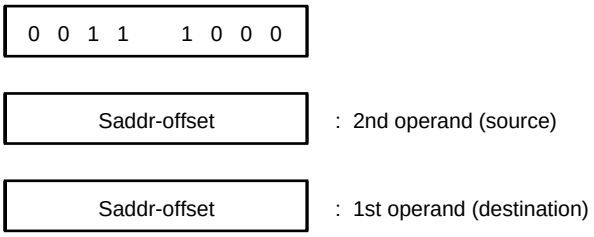
Identifier	Description
saddr	Label or immediate data between FE20H and FF1FH
saddrp	Label or immediate data between FE20H and FF1EH

[Coding example]

- General example

MOV saddr, saddr

Operation code



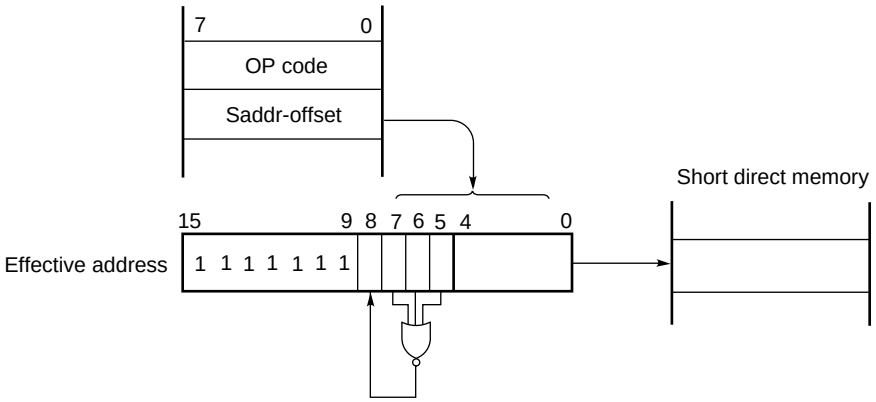
- Specific example

MOV 0FE30H, 0FE50H

Operation code

0 0 1 1 1 0 0 0	: 2nd operand (source)
0 1 0 1 0 0 0 0	: 1st operand (destination)
0 0 1 1 0 0 0 0	

[Illustration]



6.2.5 Special Function Register (SFR) Addressing

[Function]

This addressing method uses 8-bit immediate data in the instruction word to address a memory-mapped special function register (SFR).

The SFR-mapped space on which this type of addressing is used is the 256-byte space from 0FF00H to 0FFFFH. However, SFRs mapped onto addresses 0FF00H to 0FF1FH can be also accessed by short direct addressing.

Remark With NEC's assembler package (RA78K/II), instructions on SFRs mapped onto addresses 0FF00H to 0FF1FH use short direct addressing automatically (forcibly).

[Operand format]

SFR addressing is used when executing instructions with the operand formats shown below.

Identifier	Description
sfr	Special function register name
sfrp	16-bit manipulable special function register name

[Coding example]

- General example

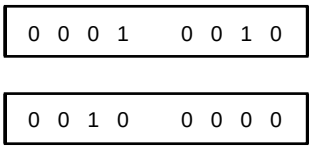
MOV sfr, A

Operation code

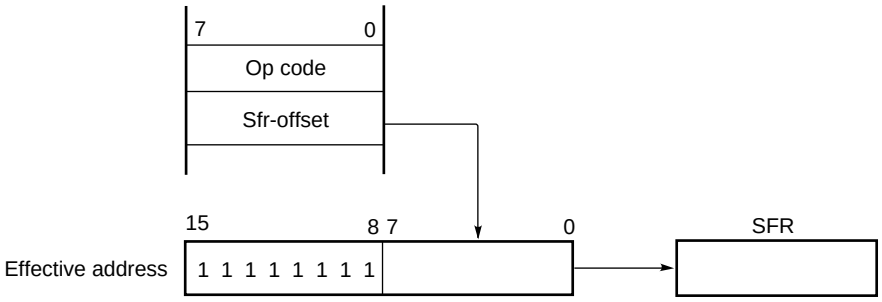
- Specific example

MOV PM0, A ; When PM0 is specified as sfr

Operation code



[Illustration]



6.2.6 Stack Addressing

[Function]

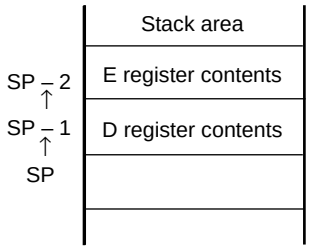
In this addressing method, the 64K-byte stack area is indirectly addressed by the contents of the stack pointer (SP).

This type of addressing is used automatically when a PUSH or POP instruction is executed, when registers are saved/restored due to generation of an interrupt request, and when a subroutine call or return instruction is executed.

[Coding example]

PUSH DE ; Executing this instruction when the contents of the DE register are saved to the stack area using a PUSH instruction automatically decrements (–2) the SP before storing the DE register contents in the stack area.

[Illustration]



Caution Stack addressing can be used for the entire 64K-byte space, but the SFR area and internal ROM area cannot be reserved as a stack area.

6.3 1M-BYTE EXPANSION SPACE ADDRESSING

In 78K/II series products, the following addressing modes can be used to access the 1M-byte expansion data memory area:

- Direct addressing
- Register indirect addressing
- Based addressing
- Indexed addressing

Data manipulation on the 1M-byte expansion data memory area is executed after first loading the bank register (PM6 or P6) with 4-bit bank data which specifies the memory bank on which the operation is to be performed.

Bank data set in bank register PM6 is output as the extension address (A16 to A19) only during execution of a memory manipulation instruction using one of the four addressing modes above.

Bank data set in bank register P6 is output as the extension address (A16 to A19) only during execution of a memory manipulation instruction with the symbol "&" affixed.

Once bank data is set, it is retained until next overwritten by the program.

Cautions 1. When the 1M-byte expansion space is not used, the low-order 4 bits of the PM6 register must be set to "0".

2. In the μPD78224 sub-series, the low-order 4 bits of the PM6 register must be set to "0".

6.3.1 Direct Addressing

[Function]

This addressing method addresses memory directly by means of the 16-bit address data in the instruction word. This addressing method can be used on the entire memory space including the 1M-byte expansion data memory area.

[Operand format]

Direct addressing is used when executing instructions with the operand formats shown below.

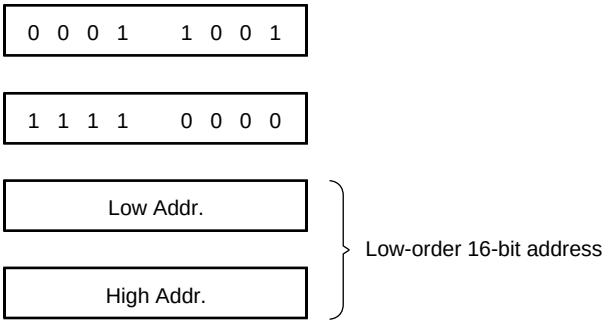
Identifier	Description
!addr16	Label or 16-bit immediate data
&!addr16	Label or 16-bit immediate data

[Coding example]

- General example 1

  MOV A, !addr16

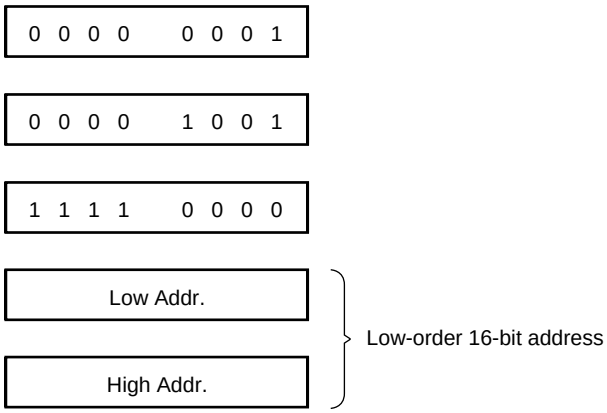
  Operation code



- General example 2

  MOV A, &!addr16

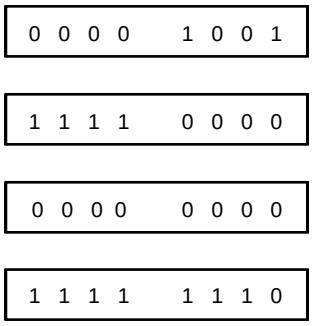
  Operation code



- Specific example

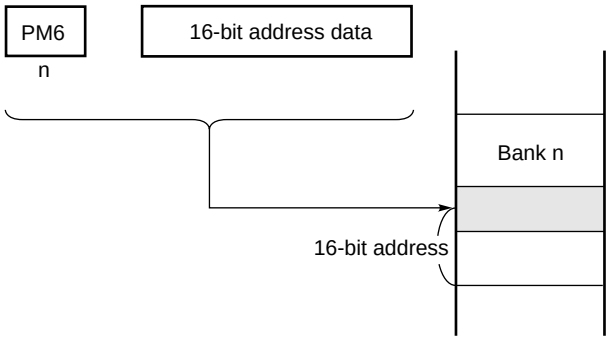
  MOV A, !0FE00H ; When 0FE00H is used as addr16

  Operation code



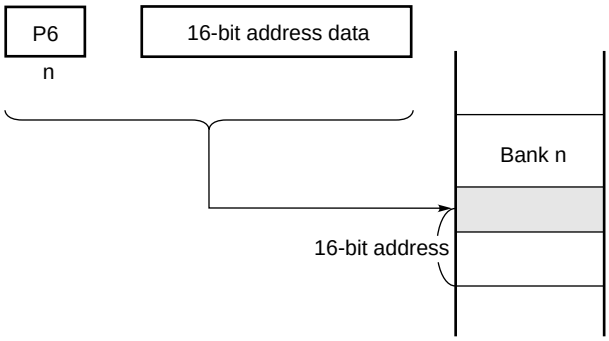
[Illustrations]

(1) 1M-byte addressing without "&" symbol



Remark In the μPD78224 sub-series, the low-order 4 bits of the PM6 register are fixed at 0. Therefore, only bank 0 can be accessed when the "&" symbol is not used.

(2) 1M-byte addressing with "&" symbol



6.3.2 Register Indirect Addressing

[Function]

This addressing method addresses the memory subject to manipulation whose output address is the register pair specified by the register pair specification code in the instruction word, in the register bank specified by the register bank selection flag (RBS1, RBS0). This addressing method can be used on the entire memory space, including the 1M-byte expansion data memory area.

In addition, register indirect addressing with auto-increment and register indirect addressing with auto-decrement are provided: The former increments and the latter decrements the addressed register pair (DE, HL) by 1 after execution of the instruction.

These addressing methods are ideal for processing of multiple consecutive data items, as in block data transfers, etc.

The entire memory space including the 1M-byte expansion data memory area can be addressed.

[Operand format]

Register indirect addressing is used when executing instructions with the operand formats shown below.

Identifier	Description
mem	[DE], [HL], [DE+], [HL+], [DE–], [HL–]
&mem	&[DE], &[HL], &[DE+], &[HL+], &[DE–], &[HL–]
mem1	[DE], [HL]
&mem1	&[DE], &[HL]

Remark "+" after the register name indicates auto-increment, and "–" indicates auto-decrement.

[Coding example]

- General example 1

  MOV A, mem ; When [DE], [HL], [DE+], [HL+], [DE–], or [HL–] is specified for mem in register indirect mode.

  Operation code

0	1	0	1	1	mem

- Specific example 1

  ADD A, mem ; When register indirect mode is specified

  Operation code

0	0	0	1	0	1	1	0

0	mem	1	0	0	0

- General example 2

  XOR A, &mem

  Operation code

0	0	0	0	0	0	0	1

0	0	0	1	0	1	1	0

0	mem	1	1	0	1

- Specific example 2

  MOV A, [DE] ; When [DE] is specified for mem

  Operation code

0	1	0	1	1	1	0	0

- Specific example 3

  ADD A, &[HL+] ; When &[HL+] is specified for mem

  Operation code

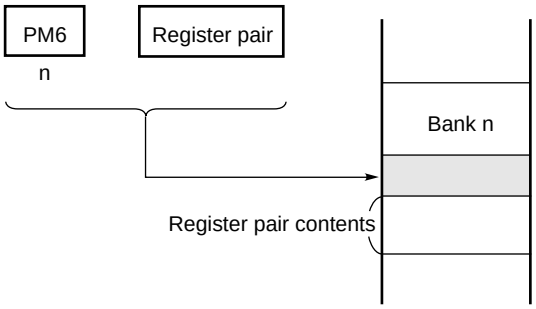
0	0	0	0	0	0	0	1

0	0	0	1	0	1	1	0

0	0	0	1	1	0	0	0

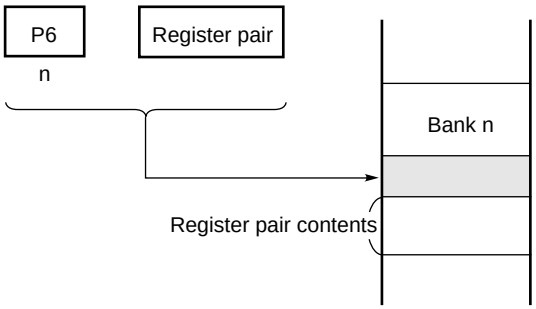
[Illustrations]

(1) 1M-byte addressing without "&" symbol



Remark In the μPD78224 sub-series, the low-order 4 bits of the PM6 register are fixed at 0. Therefore, only bank 0 can be accessed when the "&" symbol is not used.

(2) 1M-byte addressing with "&" symbol



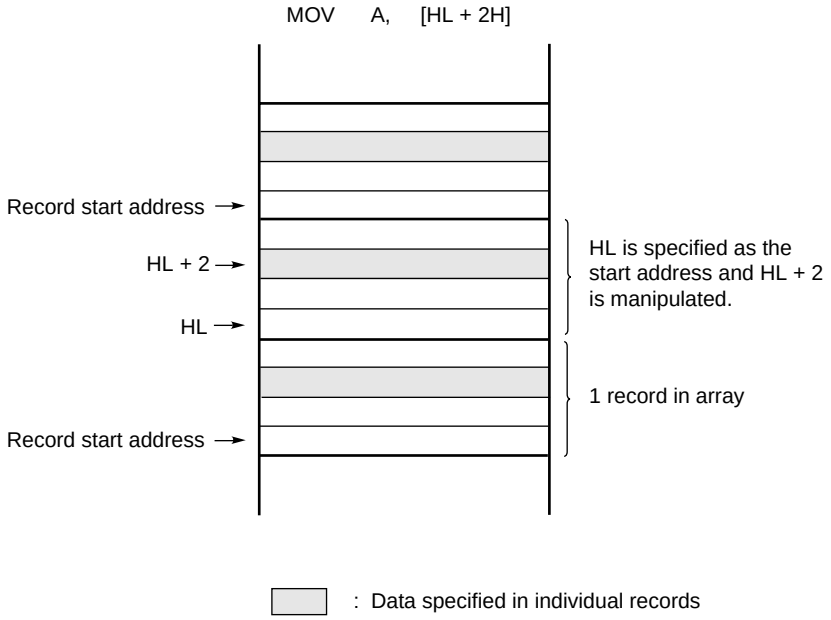
6.3.3 Based Addressing

[Function]

This addressing method addresses memory by using as the base register the register pair specified by the register pair specification code in the instruction word, in the register bank specified by the register bank selection flag (RBS1, RBS0), and adding 8-bit immediate data to these contents as offset data. The addition is performed with the offset data extended to 16 bits as a positive number. A carry out of the 16 the bit is ignored.

The entire memory space including the 1M-byte expansion data memory area can be addressed.

This kind of addressing is used for specifying data in an array of records which are composed of multiple bytes of data.



[Operand format]

Based addressing is used when executing an instruction with the operand formats shown below.

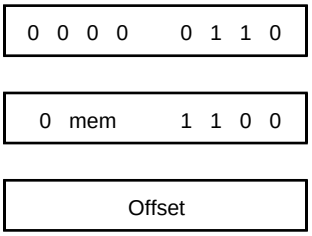
Identifier	Description
mem	[DE+byte], [HL+byte], [SP+byte]
&mem	&[DE+byte], &[HL+byte], &[SP+byte]

[Coding examples]

- General example 1

  AND A, mem

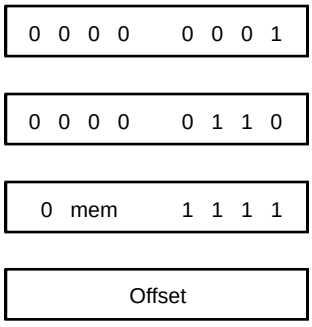
  Operation code

- General example 2

  CMP A, &mem

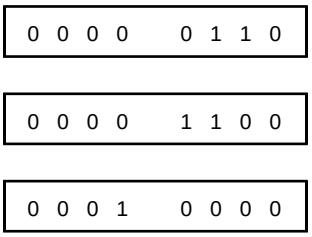
  Operation code

- Specific example

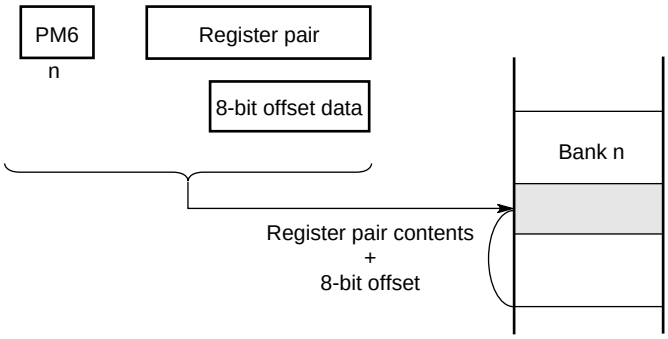
  AND A, [DE+10H] ; When based addressing using the sum of register pair DE and 10H as mem is selected.

  Operation code

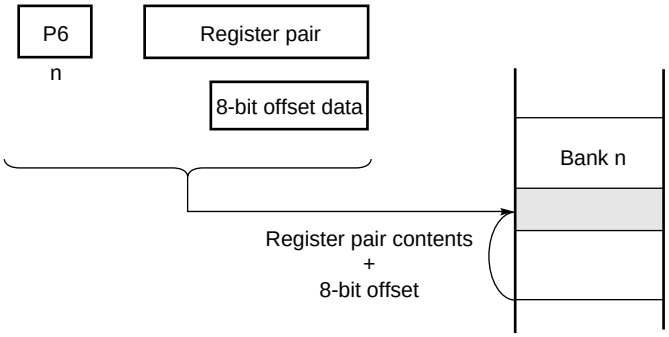
[Illustrations]

(1) 1M-byte addressing without "&" symbol.



Remark In the μPD78224 sub-series, the low-order 4 bits of the PM6 register are fixed at 0. Therefore, only bank 0 can be accessed when the "&" symbol is not used.

(2) 1M-byte addressing with "&" symbol



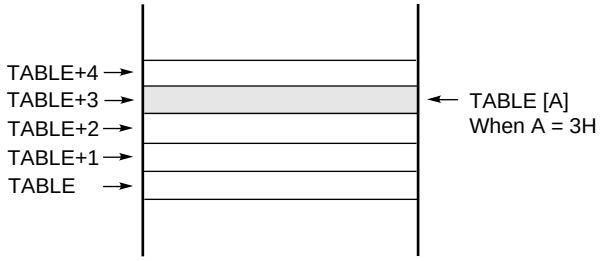
6.3.4 Indexed Addressing

[Function]

This addressing method addresses memory by using the 16-bit address data specified by the operand in the instruction word as the index, and adding to this value the contents of the register specified in the instruction word in the register bank specified by the register bank selection flag (RBS1, RBS0). The addition is performed as addition of two 16-bit positive numbers (if the register is 8 bits in length, the contents of that register are extended to 16 bits as a positive number before the addition). A carry out of the 16th bit is ignored.

The entire memory space including the 1M-byte expansion data memory area can be addressed.

The kind of addressing is used for reading table data etc.



Manipulate 4th data in the table using TABLE [A].

[Operand format]

Indexed addressing is used when executing an instruction with the operand formats shown below.

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

[Coding example]

- General example 1

  ADDC A, mem

  Operation code

0 0 0 0 1 0 1 0
0 mem 1 0 0 1
Low Offset
High Offset

- General example 2

  SUBC A, &mem

  Operation code

0 0 0 0 0 0 0 1
0 0 0 0 1 0 1 0
0 mem 1 1 1 1
Low Offset
High Offset

- Specific example

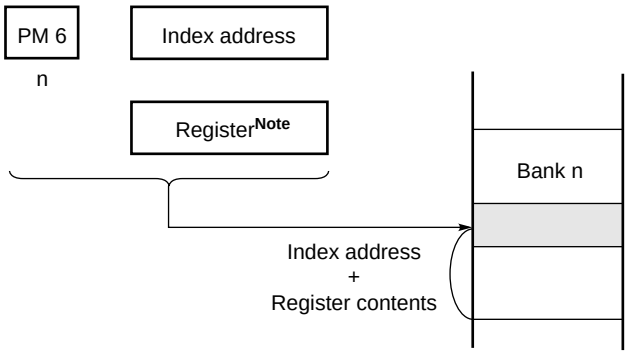
  ADDC A, 4010H [DE] ; When indexed addressing using the sum of register pair DE and 04010H as mem is selected

  Operation code

0 0 0 0 1 0 1 0
0 0 0 0 1 0 0 1
0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0

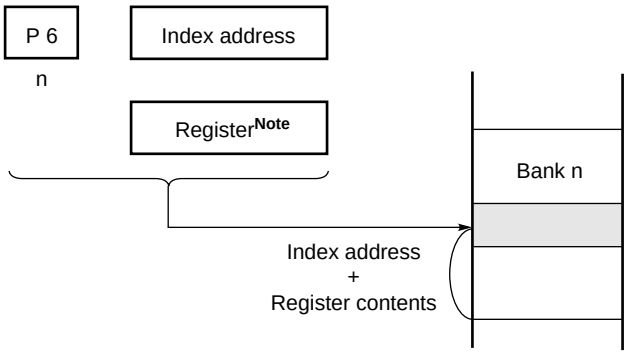
[Illustrations]

(1) 1M-byte addressing without "&" symbol



Remark In the μPD78224 sub-series, the low-order 4 bits of the PM6 register are fixed at 0. Therefore, only bank 0 can be accessed when the "&" symbol is not used.

(2) 1M-byte addressing with "&" symbol



Note 8-bit register or 16-bit register (8-bit register pair)

[MEMO]

CHAPTER 7 INSTRUCTION SET

This chapter lists the 78K/II series instruction set.

The same instructions are used on all 78K/II series products.

7.1 OPERATIONS

7.1.1 Operand Representation Format and Description Method

Code operands in the operand field for each instruction, using the specified operand representation format (for details, refer to the relevant assembler specifications). When several coding forms are presented, any one can be used. Since uppercase letters and symbols +, –, #, !, $, /, [ ], and & are keywords, write any symbols as is.

Do not omit symbols +, –, #, !, $, /, [ ], and & when writing immediate data with labels. r and rp can be written with any functional and absolute names.

+ : Auto increment
– : Auto decrement
: Immediate data
! : Absolute addressing
$: Relative addressing
/ : Bit inversion
[ ] : Indirect addressing
& : Sub-bank specification
r,r' : Registers;
Functional name : X, A, C, B, E, D, L, H
Absolute name : R0-R7
r1 : Register group 1;
B, C
rp, rp' : Register pairs;
Functional name : AX, BC, DE, HL
Absolute name : RP0-RP3
sfr : Special function registers;
A special function register name is specified.
Refer to User's Manual, Hardware of relevant product for details.
sfrp : Special function register pairs;
A special function register pair name is specified.
Refer to User's Manual, Hardware of relevant product for details.
mem : Memory address indicated in indirect addressing mode;
Register indirect mode : [DE], [HL], [DE+], [HL+], [DE–], [HL–]
Base mode : [DE+byte], [HL+byte], [SP+byte]
Indexed mode : word[A], word[B], word[DE], word[HL]
mem1 : Memory address indicated in indirect addressing group 1 mode;
[DE], [HL]

saddr, saddr' : Memory address indicated in short direct addressing mode;
FE20H-FF1FH immediate data or label
saddrp : Memory address indicated in short direct addressing pair mode;
FE20H-FF1EH immediate data or label
addr16 : 16-bit address;
0000H-FEFFH immediate data or label
addr11 : 11-bit address;
800H-FFFH immediate data or label
addr5 : 5-bit address;
40H-7EH immediate data or label (even number only)
word : 16-bit data;
16-bit immediate data or label
byte : 8-bit data;
8-bit immediate data or label
bit : 3-bit data;
3-bit immediate data or label
n : Number of shift bits;
3-bit immediate data (0-7)
RBn : Register bank;
RB0-RB3

7.1.2 Operation Field

A : Register A; 8-bit accumulator
X : Register X
B : Register B
C : Register C
D : Register D
E : Register E
H : Register H
L : Register L
R0-R7 : Register 0 to register 7 (absolute name)
AX : Register pair (AX); 16-bit accumulator
BC : Register pair (BC)
DE : Register pair (DE)
HL : Register pair (HL)
RP0-RP3 : Register pair 0 to register pair 3 (absolute name)
PC : Program counter
SP : Stack pointer
PSW : Program status word
CY : Carry flag
AC : Auxiliary carry flag
Z : Zero flag
RBS1-RBS0 : Register bank selection flag
IE : Interrupt request enable flag

STBC : Standby control register
jdisp8 : Signed 8-bit data (displacement: –128 to +127)
( ) : Contents at address enclosed in parentheses or at address indicated in register enclosed in parentheses
xxH : Hexadecimal number
x_H, x_L : Eight high-order bits and eight low-order bits of 16-bit register pair

7.1.3 Flag Field

Blank : No change
0 : Cleared to zero.
1 : Set to 1.
x : Set or cleared according to the result.
R : Saved values are restored.

7.1.4 List of Basic Instruction Operations

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

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
MOV	r, #byte	2	r <– byte			
	saddr, #byte	3	(saddr) <– byte			
	sfr, #byte	3	sfr <– byte			
	r, r'	2	r <– r'			
	A, r	1	A <– r			
	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)			
	A, &mem	2-5	A <– (&mem)			
	mem, A	1-4	(mem) <– A			
	&mem, A	2-5	(&mem) <– A			
	A, !addr16	4	A <– (!addr16)			
	A, &!addr16	5	A <– (&!addr16)			
	!addr16, A	4	(!addr16) <– A			
	&!addr16, A	5	(&!addr16) <– A			
	PSW, #byte	3	PSW <– byte	x	x	x
	PSW, A	2	PSW <– A	x	x	x
	A, PSW	2	A <– PSW			
XCH	A, r	1	A <–> r			
	r, r'	2	r <–> r'			
	A, mem	2-4	A <–> (mem)			
	A, &mem	3-5	A <–> (&mem)			
	A, saddr	2	A <–> (saddr)			
	A, sfr	3	A <–> sfr			
	saddr, saddr'	3	(saddr) <–> (saddr')			

(2) 16-bit data transfer instructions: MOVW

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
MOVW	rp, #word	3	rp <– word			
	saddrp, #word	4	(saddrp) <– word			
	sfrp, #word	4	sfrp <– word			
	rp, rp'	2	rp <– rp'			
	AX, saddrp	2	AX <– (saddrp)			
	saddrp, AX	2	(saddrp) <– AX			
	AX, sfrp	2	AX <– sfrp			
	sfrp, AX	2	sfrp <– AX			
	AX, mem1	2	AX <– (mem1)			
	AX, &mem1	3	AX <– (&mem1)			
	mem1, AX**Note**	2	(mem1) <– AX			
	&mem1, AX**Note**	3	(&mem1) <– AX			

Note Cannot be used on μPD78244 sub-series EEPROM area.

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

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
ADD	A, #byte	2	A, CY <– A + byte	x	x	x
	saddr, #byte	3	(saddr), CY <– (saddr) + byte	x	x	x
	sfr, #byte	4	sfr, CY <– sfr + byte	x	x	x
	r, r'	2	r, CY <– r + r'	x	x	x
	A, saddr	2	A, CY <– A + (saddr)	x	x	x
	A, sfr	3	A, CY <– A + sfr	x	x	x
	saddr, saddr'	3	(saddr), CY <– (saddr) + (saddr')	x	x	x
	A, mem	2-4	A, CY <– A + (mem)	x	x	x
	A, &mem	3-5	A, CY <– A + (&mem)	x	x	x

(Continued)

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
ADDC	A, #byte	2	A, CY <– A + byte + CY	x	x	x
	saddr, #byte	3	(saddr), CY <– (saddr) + byte + CY	x	x	x
	sfr, #byte	4	sfr, CY <– sfr + byte + CY	x	x	x
	r, r'	2	r, CY <– r + r' + CY	x	x	x
	A, saddr	2	A, CY <– A + (saddr) + CY	x	x	x
	A, sfr	3	A, CY <– A + sfr + CY	x	x	x
	saddr, saddr'	3	(saddr), CY <– (saddr) + (saddr') + CY	x	x	x
	A, mem	2-4	A, CY <– A + (mem) + CY	x	x	x
	A, &mem	3-5	A, CY <– A + (&mem) + CY	x	x	x
SUB	A, #byte	2	A, CY <– A – byte	x	x	x
	saddr, #byte	3	(saddr), CY <– (saddr) – byte	x	x	x
	sfr, #byte	4	sfr, CY <– sfr – byte	x	x	x
	r, r'	2	r, CY <– r – r'	x	x	x
	A, saddr	2	A, CY <– A – (saddr)	x	x	x
	A, sfr	3	A, CY <– A – sfr	x	x	x
	saddr, saddr'	3	(saddr), CY <– (saddr) – (saddr')	x	x	x
	A, mem	2-4	A, CY <– A – (mem)	x	x	x
	A, &mem	3-5	A, CY <– A – (&mem)	x	x	x
SUBC	A, #byte	2	A, CY <– A – byte – CY	x	x	x
	saddr, #byte	3	(saddr), CY <– (saddr) – byte – CY	x	x	x
	sfr, #byte	4	sfr, CY <– sfr – byte – CY	x	x	x
	r, r'	2	r, CY <– r – r' – CY	x	x	x
	A, saddr	2	A, CY <– A – (saddr) – CY	x	x	x
	A, sfr	3	A, CY <– A – sfr – CY	x	x	x
	saddr, saddr'	3	(saddr), CY <– (saddr) – (saddr') – CY	x	x	x
	A, mem	2-4	A, CY <– A – (mem) – CY	x	x	x
	A, &mem	3-5	A, CY <– A – (&mem) – CY	x	x	x

(Continued)

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
AND	A, #byte	2	A <– A ∧ byte	x		
	saddr, #byte	3	(saddr) <– (saddr) ∧ byte	x		
	sfr, #byte	4	sfr <– sfr ∧ byte	x		
	r, r'	2	r <– r ∧ r'	x		
	A, saddr	2	A <– A ∧ (saddr)	x		
	A, sfr	3	A <– A ∧ sfr	x		
	saddr, saddr'	3	(saddr) <– (saddr) ∧ (saddr')	x		
	A, mem	2-4	A <– A ∧ (mem)	x		
	A, &mem	3-5	A <– A ∧ (&mem)	x		
OR	A, #byte	2	A <– A ∨ byte	x		
	saddr, #byte	3	(saddr) <– (saddr) ∨ byte	x		
	sfr, #byte	4	sfr <– sfr ∨ byte	x		
	r, r'	2	r <– r ∨ r'	x		
	A, saddr	2	A <– A ∨ (saddr)	x		
	A, sfr	3	A <– A ∨ sfr	x		
	saddr, saddr'	3	(saddr) <– (saddr) ∨ (saddr')	x		
	A, mem	2-4	A <– A ∨ (mem)	x		
	A, &mem	3-5	A <– A ∨ (&mem)	x		
XOR	A, #byte	2	A <– A ⩛ byte	x		
	saddr, #byte	3	(saddr) <– (saddr) ⩛ byte	x		
	sfr, #byte	4	sfr <– sfr ⩛ byte	x		
	r, r'	2	r <– r ⩛ r'	x		
	A, saddr	2	A <– A ⩛ (saddr)	x		
	A, sfr	3	A <– A ⩛ sfr	x		
	saddr, saddr'	3	(saddr) <– (saddr) ⩛ (saddr')	x		
	A, mem	2-4	A <– A ⩛ (mem)	x		
	A, &mem	3-5	A <– A ⩛ (&mem)	x		

(Continued)

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
CMP	A, #byte	2	A – byte	x	x	x
	saddr, #byte	3	(saddr) – byte	x	x	x
	sfr, #byte	4	sfr – byte	x	x	x
	r, r'	2	r – r'	x	x	x
	A, saddr	2	A – (saddr)	x	x	x
	A, sfr	3	A – sfr	x	x	x
	saddr, saddr'	3	(saddr) – (saddr')	x	x	x
	A, mem	2-4	A – (mem)	x	x	x
	A, &mem	3-5	A – (&mem)	x	x	x

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

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
ADDW	AX, #word	3	AX, CY <– AX + word	x	x	x
	AX, rp	2	AX, CY <– AX + rp	x	x	x
	AX, saddrp	2	AX, CY <– AX + (saddrp)	x	x	x
	AX, sfrp	3	AX, CY <– AX + sfrp	x	x	x
SUBW	AX, #word	3	AX, CY <– AX – word	x	x	x
	AX, rp	2	AX, CY <– AX – rp	x	x	x
	AX, saddrp	2	AX, CY <– AX – (saddrp)	x	x	x
	AX, sfrp	3	AX, CY <– AX – sfrp	x	x	x
CMPW	AX, #word	3	AX – word	x	x	x
	AX, rp	2	AX – rp	x	x	x
	AX, saddrp	2	AX – (saddrp)	x	x	x
	AX, sfrp	3	AX – sfrp	x	x	x

(5) Multiply/divide instructions: MULU, DIVUW

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
MULU	r	2	AX <– A x r			
DIVUW	r	2	AX (quotient), r (remainder) <– AX/r When r = 0, r <– X, AX <– 0FFFFH			

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

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
INC	r	1	r <– r + 1	x	x	
	saddr	2	(saddr) <– (saddr) + 1	x	x	
DEC	r	1	r <– r – 1	x	x	
	saddr	2	(saddr) <– (saddr) – 1	x	x	
INCW	rp	1	rp <– rp + 1			
DECW	rp	1	rp <– rp – 1			

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

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
ROR	r, n	2	(CY, r_7 <– r_0, r_{m-1} <– r_m) x n times n=0 to 7			x
ROL	r, n	2	(CY, r_0 <– r_7, r_{m+1} <– r_m) x n times n=0 to 7			x
RORC	r, n	2	(CY <– r_0, r_7 <– CY, r_{m-1} <– r_m) x n times n=0 to 7			x
ROLC	r, n	2	(CY <– r_7, r_0 <– CY, r_{m-1} <– r_m) x n times n=0 to 7			x
SHR	r, n	2	(CY <– r_0, r_7 <– 0, r_{m-1} <– r_m) x n times n=0 to 7	x	0	x
SHL	r, n	2	(CY <– r_7, r_0 <– 0, r_{m-1} <– r_m) x n times n=0 to 7	x	0	x
SHRW	rp, n	2	(CY <– rp_0, rp_{15} <– 0, rp_{m-1} <– rp_m) x n times n=0 to 7	x	0	x
SHLW	rp, n	2	(CY <– rp_{15}, rp_0 <– 0, rp_{m-1} <– rp_m) x n times n=0 to 7	x	0	x
ROR4[Note]	mem1	2	A_{3-0} <– $(mem1)_{3-0}$, $(mem1)_{7-4}$ <– A_{3-0}, $(mem1)_{3-0}$ <– $(mem1)_{7-4}$			
	&mem1	3	A_{3-0} <– $(\&mem1)_{3-0}$, $(\&mem1)_{7-4}$ <– A_{3-0}, $(\&mem1)_{3-0}$ <– $(\&mem1)_{7-4}$			
ROL4[Note]	mem1	2	A_{3-0} <– $(mem1)_{7-4}$, $(mem1)_{3-0}$ <– A_{3-0}, $(mem1)_{7-4}$ <– $(mem1)_{3-0}$			
	&mem1	3	A_{3-0} <– $(\&mem1)_{7-4}$, $(\&mem1)_{3-0}$ <– A_{3-0}, $(\&mem1)_{7-4}$ <– $(\&mem1)_{3-0}$			

Note Cannot be used on μPD78244 sub-series EEPROM area.

(8) BCD conversion instructions: ADJBA, ADJBS

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
ADJBA		1	Use the decimal adjust accumulator after addition.	x	x	x
ADJBS		1	Use the decimal adjust accumulator after subtraction.	x	x	x

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

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
MOV1	CY, saddr.bit	3	CY <– (saddr.bit)			x
	CY, sfr.bit	3	CY <– sfr.bit			x
	CY, A.bit	2	CY <– A.bit			x
	CY, X.bit	2	CY <– X.bit			x
	CY, PSW.bit	2	CY <– PSW.bit			x
	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			
	PSW.bit, CY	2	PSW.bit <– CY	x	x	
AND1	CY, saddr.bit	3	CY <– CY ∧ (saddr.bit)			x
	CY, /saddr.bit	3	CY <– CY ∧ ($\overline{\text{saddr.bit}}$)			x
	CY, sfr.bit	3	CY <– CY ∧ sfr.bit			x
	CY, /sfr.bit	3	CY <– CY ∧ $\overline{\text{sfr.bit}}$			x
	CY, A.bit	2	CY <– CY ∧ A.bit			x
	CY, /A.bit	2	CY <– CY ∧ $\overline{\text{A.bit}}$			x
	CY, X.bit	2	CY <– CY ∧ X.bit			x
	CY, /X.bit	2	CY <– CY ∧ $\overline{\text{X.bit}}$			x
	CY, PSW.bit	2	CY <– CY ∧ PSW.bit			x
	CY, /PSW.bit	2	CY <– CY ∧ $\overline{\text{PSW.bit}}$			x

(Continued)

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
OR1	CY, saddr.bit	3	CY <– CY ∨ (saddr.bit)			x
	CY, /saddr.bit	3	CY <– CY ∨ ($\overline{\text{saddr.bit}}$)			x
	CY, sfr.bit	3	CY <– CY ∨ sfr.bit			x
	CY, /sfr.bit	3	CY <– CY ∨ $\overline{\text{sfr.bit}}$			x
	CY, A.bit	2	CY <– CY ∨ A.bit			x
	CY, /A.bit	2	CY <– CY ∨ $\overline{\text{A.bit}}$			x
	CY, X.bit	2	CY <– CY ∨ X.bit			x
	CY, /X.bit	2	CY <– CY ∨ $\overline{\text{X.bit}}$			x
	CY, PSW.bit	2	CY <– CY ∨ PSW.bit			x
	CY, /PSW.bit	2	CY <– CY ∨ $\overline{\text{PSW.bit}}$			x
XOR1	CY, saddr.bit	3	CY <– CY ⩛ (saddr.bit)			x
	CY, sfr.bit	3	CY <– CY ⩛ sfr.bit			x
	CY, A.bit	2	CY <– CY ⩛ A.bit			x
	CY, X.bit	2	CY <– CY ⩛ X.bit			x
	CY, PSW.bit	2	CY <– CY ⩛ PSW.bit			x
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			
	PSW.bit	2	PSW.bit <– 1	x	x	x
	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			
	PSW.bit	2	PSW.bit <– 0	x	x	x
	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 <– $\overline{\text{X.bit}}$			
	PSW.bit	2	PSW.bit <– $\overline{\text{PSW.bit}}$	x	x	x
	CY	1	CY <– $\overline{\text{CY}}$			x

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

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
CALL	!addr16	3	$(SP - 1) \leftarrow (PC + 3)_H$, $(SP - 2) \leftarrow (PC + 3)_L$, $PC \leftarrow$ addr16, $SP \leftarrow SP - 2$			
	rp	2	$(SP - 1) \leftarrow (PC + 2)_H$, $(SP - 2) \leftarrow (PC + 2)_L$, $PC_H \leftarrow rp_H$, $PC_L \leftarrow rp_L$, $SP \leftarrow SP - 2$			
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, $SP \leftarrow SP - 2$			
CALLT	[addr5]	1	$(SP - 1) \leftarrow (PC + 1)_H$, $(SP - 2) \leftarrow (PC + 1)_L$, $PC_H \leftarrow$ (0000000001, addr5 + 1), $PC_L \leftarrow$ (0000000001, addr5), $SP \leftarrow SP - 2$			
BRK		1	$(SP - 1) \leftarrow$ PSW, $(SP - 2) \leftarrow (PC + 1)_H$ $(SP - 3) \leftarrow (PC + 1)_L$, $PC_L \leftarrow$ (003EH), $PC_H \leftarrow$ (003FH), $SP \leftarrow SP - 3$, IE $\leftarrow$ 0			
RET		1	$PC_L \leftarrow$ (SP), $PC_H \leftarrow$ (SP + 1), $SP \leftarrow SP + 2$			
RETI		1	$PC_L \leftarrow$ (SP), $PC_H \leftarrow$ (SP + 1), PSW $\leftarrow$ (SP + 2), $SP \leftarrow$ (SP + 3), NMIS $\leftarrow$ 0	R	R	R
RETB		1	$PC_L \leftarrow$ (SP), $PC_H \leftarrow$ (SP + 1), PSW $\leftarrow$ (SP + 2), $SP \leftarrow$ (SP + 3)	R	R	R

(11) Stack manipulation instructions: PUSH, POP, MOVW, INCW, DECW

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
PUSH	PSW	1	$(SP - 1) \leftarrow$ PSW, $SP \leftarrow SP - 1$			
	sfr	2	$(SP - 1) \leftarrow$ sfr, $SP \leftarrow SP - 1$			
	rp	1	$(SP - 1) \leftarrow rp_H$, $(SP - 2) \leftarrow rp_L$, $SP \leftarrow SP - 2$			
POP	PSW	1	PSW $\leftarrow$ (SP), $SP \leftarrow SP + 1$	R	R	R
	sfr	2	sfr $\leftarrow$ (SP), $SP \leftarrow SP + 1$			
	rp	1	$rp_L \leftarrow$ (SP), $rp_H \leftarrow$ (SP + 1), $SP \leftarrow SP + 2$			
MOVW	SP, #word	4	$SP \leftarrow$ word			
	SP, AX	2	$SP \leftarrow$ AX			
	AX, SP	2	AX $\leftarrow$ SP			
INCW	SP	2	$SP \leftarrow SP + 1$			
DECW	SP	2	$SP \leftarrow SP - 1$			

(12) Unconditional branch instruction: BR

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
BR	!addr16	3	PC <– addr16			
	rp1	2	PC_H <– rp_H, PC_L <– rp_L			
	$addr16	2	PC <– PC + 2 + jdisp8			

(13) Conditional branch instructions: BC, BL, BNC, BNL, BZ, BE, BNZ, BNE, BT, BF, BTCLR, DBNZ

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	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						
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			
	PSW.bit, $addr16	3	PC <– PC + 3 + jdisp8 if PSW.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			
	PSW.bit, $addr16	3	PC <– PC + 3 + jdisp8 if PSW.bit = 0			
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			
	PSW.bit, $addr16	3	PC <– PC + 3 + jdisp8 if PSW.bit = 1 then reset PSW. bit	x	x	x
DBNZ	r1, $addr16	2	r1 <– r1 – 1, then PC <– PC + 2 + jdisp8 if r1 ≠ 0			
	saddr, $addr16	3	(saddr) <– (saddr) – 1, then PC <– PC + 3 + jdisp8 if (saddr) ≠ 0			

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

Mnemonic	Operand	No. of bytes	Operation	Flags		
				Z	AC	CY
MOV	STBC, #byte	4	STBC <– byte			
SEL	RBn	2	RBS1 – 0 <– n, n = 0 - 3			
NOP		1	No operation			
EI		1	IE <– 1 (Enable interrupts)			
DI		1	IE <– 0 (Disable interrupts)			

7.1.5 Instruction Lists for Each Addressing Type

(1) 8-bit instructions

MOV, XCH, ADD, ADDC, SUB, SUBC, AND, OR, XOR, CMP, MULU, DIVUW, INC, DEC, ROR, ROL, RORC, ROLC, SHR, SHL, ROR4, ROL4, DBNZ, PUSH, and POP

Table 7-1. 8-Bit Instructions for Each Addressing Type

First operand \ Second operand	#byte	A	r r'	saddr saddr'	sfr	mem	&mem	!addr16	&!addr16	PSW	n	None[Note 2]
A	ADD[Note 1]		MOV XCH	MOV XCH ADD[Note 1]	MOV XCH ADD[Note 1]	MOV XCH ADD[Note 1]	MOV XCH ADD[Note 1]	MOV	MOV	MOV		
r	MOV		MOV XCH ADD[Note 1]								ROL ROLC ROR RORC SHR SHL	MULU DIVUW INC DEC
r1												DBNZ
saddr	MOV ADD[Note 1]	MOV		MOV XCH ADD[Note 1]								INC DBNZ DEC
sfr	MOV ADD[Note 1]	MOV										PUSH POP
mem &mem		MOV										
mem1 &mem1												ROR4[Note 3] ROL4[Note 3]
!addr16		MOV										
&!addr16		MOV										
PSW	MOV	MOV										PUSH POP
STBC	MOV											

Notes 1. ADDC, SUB, SUBC, AND, OR, XOR, and CMP are the same as ADD.

2. The second operand does not exist or is not an operand address.

3. Cannot be used for EEPROM area of the μPD78244 sub-series.

(2) 16-bit instructions

MOVW, ADDW, SUBW, CMPW, INCW, DECW, SHRW, SHLW, PUSH, and POP

Table 7-2. 16-Bit Instructions for Each Addressing Type

First operand \ Second operand	#word	AX	rp rp'	saddrp	sfrp	mem1	&mem1	SP	n	None
AX	ADDW SUBW CMPW		ADDW SUBW CMPW	MOVW ADDW SUBW CMPW	MOVW ADDW SUBW CMPW	MOVW	MOVW	MOVW		
rp	MOVW		MOVW						SHLW SHRW	INCW DECW PUSH POP
saddrp	MOVW	MOVW								
sfrp	MOVW	MOVW								
mem1 &mem1		MOVW[Note]								
SP	MOVW	MOVW								INCW DECW

Note Cannot be used for EEPROM area of the μPD78244 sub-series.

(3) Bit manipulation instructions

MOV1, AND1, OR1, XOR1, SET1, CLR1, NOT1, BT, BF, and BTCLR

Table 7-3. Bit Manipulation Instructions for Each Addressing Type

First operand \ Second operand	CY	A.bit	/A.bit	X.bit	/X.bit	saddr.bit	/saddr.bit	sfr.bit	/sfr.bit	PSW.bit	/PSW.bit	None[Note]
CY		MOV1 AND1 OR1 XOR1	AND1 OR1	MOV1 AND1 OR1 XOR1	AND1 OR1	MOV1 AND1 OR1 XOR1	AND1 OR1	MOV1 AND1 OR1 XOR1	AND OR1	MOV1 AND1 OR1 XOR1	AND1 OR1	SET1 CLR1 NOT1
A.bit	MOV1											SET1 CLR1 NOT1 BT BF BTCLR
X.bit	MOV1											SET1 CLR1 NOT1 BT BF BTCLR
saddr.bit	MOV1											SET1 CLR1 NOT1 BT BF BTCLR
sfr.bit	MOV1											SET1 CLR1 NOT1 BT BF BTCLR
PSW.bit	MOV1											SET1 CLR1 NOT1 BT BF BTCLR

Note The second operand does not exist or is not an operand address.

(4) Call instructions and branch instructions

CALL, CALLF, CALLT, BR, BC, BT, BF, BTCLR, DBNZ, BL, BNC, BNL, BZ, BE, BNZ, and BNE

Table 7-4. Call Instructions and Branch Instructions for Each Addressing Type

Instruction addressing operand	$addr16	!addr16	rp	!addr11	[addr5]
Basic instruction	BR BC[Note]	CALL BR	CALL BR	CALLF	CALLT
Composite instruction	BT BF BTCLR DBNZ				

Note BL, BNC, BNL, BZ, BE, BNZ, and BNE are the same as BC.

(5) Other instructions

ADJBA, ADJBS, BRK, RET, RETI, RETB, NOP, EI, DI, and SEL

7.2 OPERATION CODES

7.2.1 Operation Code Symbols

r, r'

R_2	R_1	R_0	r, r'	
R_6	R_5	R_4	r	
0	0	0	R0	X
0	0	1	R1	A
0	1	0	R2	C
0	1	1	R3	B
1	0	0	R4	E
1	0	1	R5	D
1	1	0	R6	L
1	1	1	R7	H

rl

R_0	reg
0	C
1	B

rp, rp'

P_1	P_0	rp	
P_2	P_1	rp, rp'	
P_6	P_5	rp	
0	0	PR0	AX
0	1	PR1	BC
1	0	PR2	DE
1	1	PR3	HL

Bn : Immediate data corresponding to bit
Nn : Immediate data corresponding to n
Data : 8-bit immediate data corresponding to byte
Low/High Byte : 16-bit immediate data corresponding to word
Saddr-offset : Low-order 8-bit offset data of 16-bit address corresponding to saddr or saddrp
Saddr'-offset : Low-order 8-bit offset data of 16-bit address corresponding to saddr'
Sfr-offset : Special function register (sfr, sfrp) 16-bit address low-order 8-bit offset data
Low/High Offset : 16-bit offset data corresponding to word in indexed addressing
Low/High Addr. : 16-bit immediate data corresponding to addr16
jdisp8 : Signed two's complement data (8-bit) for relative address distance between start address of next instruction and branch address
fa : Low-order 11 bits of immediate data corresponding to addr11
ta : Low-order 5 bits of immediate data corresponding to (addr5 x dis)

Caution The code when the 1st and 2nd operands in the operand field are both registers or a register pair is as described below.
The high-order 4 bits of the register specification byte comprise the 1st operand specification code and the low-order 4 bits comprise the 2nd operand specification code.

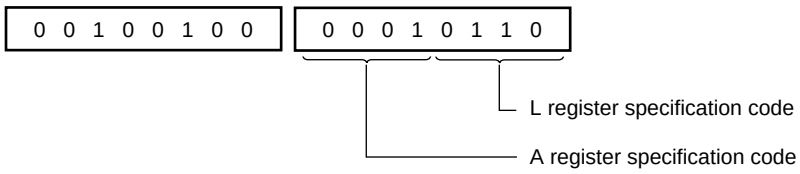
Example: MOV r, r'

Operation code | 0 0 1 0 0 1 0 0 | 0 R_6 R_5 R_4 0 R_2 R_1 R_0 |

When the A register is specified as the 1st operand and the L register as the 2nd operand, the instruction is written as follows:

MOV A, L (Transfer L register contents to A register)

The operation code for this instruction is as follows:



7.2.2 Operation Code When mem, &mem, mem1 or &mem1 Is Specified as Operand

(1) Operation codes when mem or &mem is written in operand field

The codes assigned to mod and mem in the operation code field fixed for the contents written in mem or &mem in the operand field are shown in Table 7-5.

Table 7-5. Operation Codes for mem, &mem

mem, &mem addressing mode		Operation mode							
Addressing mode name	Description	mod					mem		
Register indirect mode	[DE+]	1	0	1	1	0	0	0	0
	[HL+]	1	0	1	1	0	0	0	1
	[DE–]	1	0	1	1	0	0	1	0
	[HL–]	1	0	1	1	0	0	1	1
	[DE]	1	0	1	1	0	1	0	0
	[HL]	1	0	1	1	0	1	0	1
Base mode	[DE+byte]	0	0	1	1	0	0	0	0
	[SP+byte]	0	0	1	1	0	0	0	1
	[HL+byte]	0	0	1	1	0	0	1	0
Indexed mode	word [DE]	0	1	0	1	0	0	0	0
	word [A]	0	1	0	1	0	0	0	1
	word [HL]	0	1	0	1	0	0	1	0
	word [B]	0	1	0	1	0	0	1	1

(2) Operation codes when mem1 or &mem1 is written in operand field

The table below shows R_0 or R_1 in the operation code field determined according to the contents written for mem1 or &mem1 in the operand field.

Operand field	Operation code field
mem1, &mem1	R_0 or R_1
[DE]	0
[HL]	1

7.2.3 List of Operation Codes

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

Mnemonic	Operands	Operation code		
		B1	B2	B3
		B4	B5	
MOV	r, #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, r’	0 0 1 0 0 1 0 0	0 R_6 R_5 R_4 0 R_2 R_1 R_0	
	A, r	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 –>	
	A, sfr	0 0 0 1 0 0 0 0	<– Sfr-offset –>	
	sfr, A	0 0 0 1 0 0 1 0	<– Sfr-offset –>	
	saddr, saddr’	0 0 1 1 1 0 0 0	<– Saddr-offset –>	<– Saddr-offset –>
	A, mem **Note**	0 1 0 1 1 mem		
		0 0 0 mod	0 mem 0 0 0 0	<– Low Offset –>
		<– High Offset –>		
	A, &mem **Note**	0 0 0 0 0 0 0 1	0 1 0 1 1 mem	
		0 0 0 0 0 0 0 1	0 0 0 mod	0 mem 0 0 0 0
		<– Low Offset –>	<– High Offset –>	
	mem, A **Note**	0 1 0 1 0 mem		
		0 0 0 mod	1 mem 0 0 0 0	<– Low Offset –>
		<– High Offset –>		
	&mem, A **Note**	0 0 0 0 0 0 0 1	0 1 0 1 0 mem	
		0 0 0 0 0 0 0 1	0 0 0 mod	1 mem 0 0 0 0
		<– Low Offset –>	<– High Offset –>	
	A, !addr16	0 0 0 0 1 0 0 1	1 1 1 1 0 0 0 0	<– Low Addr. –>
		<– High Addr. –>	<–	
	A, &!addr16	0 0 0 0 0 0 0 1	0 0 0 0 1 0 0 1	1 1 1 1 0 0 0 0
		<– Low Addr. –>	<– High Addr. –>	

(Continued)

Note When [DE], [HL], [DE+], [DE–], [HL+] or [HL–] is written in mem or &mem, this comprises a special 1-byte code or 2-byte code respectively.

Mnemonic	Operands	Operation code		
		B1	B2	B3
		B4	B5	
MOV	!addr16, A	0 0 0 0 1 0 0 1	1 1 1 1 0 0 0 1	<– Low Addr. –>
		<– High Addr. –>		
	&!addr16, A	0 0 0 0 0 0 0 1	0 0 0 0 1 0 0 1	1 1 1 1 0 0 0 1
		<– Low Addr. –>	<– High Addr. –>	
	PSW, #byte	0 0 1 0 1 0 1 1	1 1 1 1 1 1 1 0	<– Data –>
	PSW, A	0 0 0 1 0 0 1 0	1 1 1 1 1 1 1 0	
	A, PSW	0 0 0 1 0 0 0 0	1 1 1 1 1 1 1 0	
XCH	r, r'	0 0 1 0 0 1 0 1	0 R_6 R_5 R_4 0 R_2 R_1 R_0	
	A, r	1 1 0 1 1 R_2 R_1 R_0		
	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 –>
	saddr, saddr'	0 0 1 1 1 0 0 1	<– Saddr-offset –>	<– Saddr-offset –>
	A, mem	0 0 0 mod	0 mem 0 1 0 0	<– Low Offset –>
		<– High Offset –>		
	A, &mem	0 0 0 0 0 0 0 1	0 0 0 mod	0 mod 0 1 0 0
		<– Low Offset –>	<– High Offset –>	

(2) 16-bit data transfer instructions: MOVW

Mnemonic	Operands	Operation code		
		B1	B2	B3
		B4	B5	
MOVW	rp, #word	0 1 1 0 0 P_2 P_1 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, rp'	0 0 1 0 0 1 0 0	0 P_6 P_5 0 1 P_2 P_1 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 –>	
	AX, sfrp	0 0 0 1 0 0 0 1	<– Sfr-offset –>	
	sfrp, AX	0 0 0 1 0 0 1 1	<– Sfr-offset –>	
	AX, mem1	0 0 0 0 0 1 0 1	1 1 1 0 0 0 1 R_0	
	mem1, AX	0 0 0 0 0 1 0 1	1 1 1 0 0 1 1 R_0	
	AX, &mem1**Note**	0 0 0 0 0 0 0 1	0 0 0 0 0 1 0 1	1 1 1 0 0 0 1 R_0
	&mem1, AX**Note**	0 0 0 0 0 0 0 1	0 0 0 0 0 1 0 1	1 1 1 0 0 1 1 R_0

Note Cannot be used on μPD78244 sub-series EEPROM area.

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

Mnemonic	Operands	Operation 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, r'	1 0 0 0 1 0 0 0	0 R_6 R_5 R_4 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 –>		
	A, &mem	0 0 0 0 0 0 0 1	0 0 0 mod	0 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, r'	1 0 0 0 1 0 0 1	0 R_6 R_5 R_4 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 –>		
	A, &mem	0 0 0 0 0 0 0 1	0 0 0 mod	0 mem 1 0 0 1
		<– Low Offset –>	<– High Offset –>	

(Continued)

Mnemonic	Operands	Operation 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, r’	1 0 0 0 1 0 1 0	0 R_6 R_5 R_4 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 –>		
	A, &mem	0 0 0 0 0 0 0 1	0 0 0 mod	0 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, r’	1 0 0 0 1 0 1 1	0 R_6 R_5 R_4 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 –>		
	A, &mem	0 0 0 0 0 0 0 1	0 0 0 mod	0 mem 1 0 1 1
		<– Low Offset –>	<– High Offset –>	

(Continued)

Mnemonic	Operands	Operation code		
		B1	B2	B3
		B4	B5	
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, r'	1 0 0 0 1 1 0 0	0 R_6 R_5 R_4 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 –>
	A, mem	0 0 0 mod	0 mem 1 1 0 0	<– Low Offset –>
		<– High Offset –>		
	A, &mem	0 0 0 0 0 0 0 1	0 0 0 mod	0 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, r'	1 0 0 0 1 1 1 0	0 R_6 R_5 R_4 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 –>		
	A, &mem	0 0 0 0 0 0 0 1	0 0 0 mod	0 mem 1 1 1 0
		<– Low Offset –>	<– High Offset –>	

(Continued)

Mnemonic	Operands	Operation code		
		B1	B2	B3
		B4	B5	
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, r’	1 0 0 0 1 1 0 1	0 R_6 R_5 R_4 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 –>		
	A, &mem	0 0 0 0 0 0 0 1	0 0 0 mod	0 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	<– Saddr-offset –>
		<– Data –>		
	r, r’	1 0 0 0 1 1 1 1	0 R_6 R_5 R_4 0 R_2 R_1 R_0	
	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 –>		
	A, &mem	0 0 0 0 0 0 0 1	0 0 0 mod	0 mem 1 1 1 1
		<– Low Offset –>	<– High Offset –>	

(Continued)

(4) 16-bit operation instructions: ADDW, SUBW, CMPW

Mnemonic	Operands	Operation code		
		B1	B2	B3
		B4	B5	
ADDW	AX, #word	0 0 1 0 1 1 0 1	<– Low Byte –>	<– High Byte –>
	AX, rp	1 0 0 0 1 0 0 0	0 0 0 0 1 P_2 P_1 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 –>
SUBW	AX, #word	0 0 1 0 1 1 1 0	<– Low Byte –>	<– High Byte –>
	AX, rp	1 0 0 0 1 0 1 0	0 0 0 0 1 P_2 P_1 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 –>
CMPW	AX, #word	0 0 1 0 1 1 1 1	<– Low Byte –>	<– High Byte –>
	AX, rp	1 0 0 0 1 1 1 1	0 0 0 0 1 P_2 P_1 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 –>

(5) Multiplication/division instructions: MULU, DIVUW

Mnemonic	Operands	Operation code		
		B1	B2	B3
		B4	B5	
MULU	r	0 0 0 0 0 1 0 1	0 0 0 0 1 R_2 R_1 R_0	
DIVUW	r	0 0 0 0 0 1 0 1	0 0 0 1 1 R_2 R_1 R_0	

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

Mnemonic	Operands	Operation code		
		B1	B2	B3
		B4	B5	
INC	r	1 1 0 0 0 R_2 R_1 R_0		
	saddr	0 0 1 0 0 1 1 0	<– Saddr-offset –>	
DEC	r	1 1 0 0 1 R_2 R_1 R_0		
	saddr	0 0 1 0 0 1 1 1	<– Saddr-offset –>	
INCW	rp	0 1 0 0 0 1 P_1 P_0		
DECW	rp	0 1 0 0 1 1 P_1 P_0		

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

Mnemonic	Operands	Operation code		
		B1	B2	B3
		B4	B5	
ROR	r, n	0 0 1 1 0 0 0 0	0 1 N_2 N_1 N_0 R_2 R_1 R_0	
ROL	r, n	0 0 1 1 0 0 0 1	0 1 N_2 N_1 N_0 R_2 R_1 R_0	
RORC	r, n	0 0 1 1 0 0 0 0	0 0 N_2 N_1 N_0 R_2 R_1 R_0	
ROLC	r, n	0 0 1 1 0 0 0 1	0 0 N_2 N_1 N_0 R_2 R_1 R_0	
SHR	r, n	0 0 1 1 0 0 0 0	1 0 N_2 N_1 N_0 R_2 R_1 R_0	
SHL	r. n	0 0 1 1 0 0 0 1	1 0 N_2 N_1 N_0 R_2 R_1 R_0	
SHRW	rp, n	0 0 1 1 0 0 0 0	1 1 N_2 N_1 N_0 P_2 P_1 0	
SHLW	rp, n	0 0 1 1 0 0 0 1	1 1 N_2 N_1 N_0 P_2 P_1 0	
ROR4[Note]	mem1	0 0 0 0 0 1 0 1	1 0 0 0 1 1 R_1 0	
	&mem1	0 0 0 0 0 0 0 1	0 0 0 0 0 1 0 1	1 0 0 0 1 1 R_1 0
ROL4[Note]	mem1	0 0 0 0 0 1 0 1	1 0 0 1 1 1 R_1 0	
	&mem1	0 0 0 0 0 0 0 1	0 0 0 0 0 1 0 1	1 0 0 1 1 1 R_1 0

Note Cannot be used on μPD78244 sub-series EEPROM area.

(8) BCD adjustment instructions: ADJBA, ADJBS

Mnemonic	Operands	Operation code		
		B1	B2	B3
		B4	B5	
ADJBA		0 0 0 0 1 1 1 0		
ADJBS		0 0 0 0 1 1 1 1		

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

Mnemonic	Operands	Operation 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	
	CY, X.bit	0 0 0 0 0 0 1 1	0 0 0 0 0 B_2 B_1 B_0	
	CY, PSW.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	
	PSW.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, PSW.bit	0 0 0 0 0 0 1 0	0 0 1 0 0 B_2 B_1 B_0	
	CY, /PSW.bit	0 0 0 0 0 0 1 0	0 0 1 1 0 B_2 B_1 B_0	

(Continued)

Mnemonic	Operands	Operation code		
		B1	B2	B3
		B4	B5	
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, PSW.bit	0 0 0 0 0 0 1 0	0 1 0 0 0 B_2 B_1 B_0	
	CY, /PSW.bit	0 0 0 0 0 0 1 0	0 1 0 1 0 B_2 B_1 B_0	
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, PSW.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	
	PSW.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	
	PSW.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	
	PSW.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		

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

Mnemonic	Operands	Operation code		
		B1	B2	B3
		B4	B5	
CALL	!addr16	0 0 1 0 1 0 0 0	<– Low Addr. –>	<– High Addr. –>
	rp	0 0 0 0 0 1 0 1	0 1 0 1 1 P_2 P_1 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		
RETI		0 1 0 1 0 1 1 1		
RETB		0 1 0 1 1 1 1 1		

(11) Stack manipulation instructions: PUSH, POP, MOVW, INCW, DECW

Mnemonic	Operands	Operation code		
		B1	B2	B3
		B4	B5	
PUSH	rp	0 0 1 1 1 1 P_1 P_0		
	PSW	0 1 0 0 1 0 0 1		
	sfr	0 0 1 0 1 0 0 1	<– Sfr-offset –>	
POP	rp	0 0 1 1 0 1 P_1 P_0		
	PSW	0 1 0 0 1 0 0 0		
	sfr	0 1 0 0 0 0 1 1	<– Sfr-offset –>	
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	

(12) Unconditional branch instructions: BR

Mnemonic	Operands	Operation code		
		B1	B2	B3
		B4	B5	
BR	!addr16	0 0 1 0 1 1 0 0	<– Low Addr. –>	<– High Addr. –>
	rp	0 0 0 0 0 1 0 1	0 1 0 0 1 P_2 P_1 0	
	$addr16	0 0 0 1 0 1 0 0	<– jdisp –>	

(13) Conditional branch instructions: BC, BL, BNC, BNL, BZ, BE, BNZ, BNE, BT, BF, BTCLR, DBNZ

Mnemonic	Operands	Operation code		
		B1	B2	B3
		B4	B5	
BC **BL**	$addr16	1 0 0 0 0 0 1 1	<– jdisp –>	
BNC **BNL**	$addr16	1 0 0 0 0 0 1 0	<– jdisp –>	
BZ **BE**	$addr16	1 0 0 0 0 0 0 1	<– jdisp –>	
BNZ **BNE**	$addr16	1 0 0 0 0 0 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 –>
	PSW.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 –>
	PSW.bit, $addr16	0 0 0 0 0 0 1 0	1 0 1 0 0 B_2 B_1 B_0	<– jdisp –>

(Continued)

Mnemonic	Operands	Operation code		
		B1	B2	B3
		B4	B5	
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 –>
	PSW.bit, $addr16	0 0 0 0 0 0 1 0	1 1 0 1 0 B_2 B_1 B_0	<– jdisp –>
DBNZ	r1, $addr16	0 0 1 1 0 0 1 R_0	<– jdisp –>	
	saddr, $addr16	0 0 1 1 1 0 1 1	<– Saddr-offset –>	<– jdisp –>

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

Mnemonic	Operands	Operation 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 –>		
SEL	RBn	0 0 0 0 0 1 0 1	1 0 1 0 1 0 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		

7.3 INSTRUCTION CLOCK CYCLES

7.3.1 Clock Cycles Column

This column outlines the number of clock cycles in instruction execution. One clock cycle is 1/f_{CLK} (167 ns when f_{CLK} = 6 MHz).

(1) Clock field classification

The number of instruction clock cycles varies according to the instruction fetch method and the memory accessed or branched to by the instruction.

<1> Instruction fetch method

- For internal ROM high-speed fetch : Shows the number of clock cycles when an internal ROM program is executed with MM register IFCH = 1. When IFCH = 0, same as for external ROM fetch.
- For external ROM fetch : Shows the number of clock cycles in execution using external programmable memory.

<2> Memory area accessed or branched to by instruction

- Internal ROM: Internal ROM fetch
- IRAM : Access to internal dual-port RAM (0FE00H to 0FEFFH)
- SFR : Special function register access
- PRAM : Access to non-IRAM area of internal RAM
- EMEM : External memory access

(2) Use of n in clock cycles column

n in the clock cycles column of a shift/rotate instruction indicates the number of bits shifted.

(3) Use of "/" in clock cycles column

- "/": a/b –> a or b

7.3.2 List of Clock Cycles

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

Mnemonic	Operands	Bytes	Clock cycles									
			Internal ROM high-speed fetch					External ROM fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
MOV	r, #byte	2	—	2	—	—	—	—	6	—	—	—
	saddr, #byte	3		3		5			9		9	
	sfr, #byte	3		—			5		—			12
	r, r'	2		2		—	—		6		—	—
	A, r	1							3			
	A, saddr	2				4			6		6	
	saddr, A	2		3		5					8	
	saddr, saddr'	3		3 Note 1		7 Note 2			9 Note1			
	A, sfr	2		—		4	4		—		6	9
	sfr, A	2				5	5					
	A, mem	1-4		See **Table 7-6 (1/4, 2/4)** for details.								
	A,&mem	2-5										
	mem, A	1-4										
	&mem, A	2-5										
	A, !addr16	4	7	6	8	8	8	15	14	16		16
	A,&!addr16	5	9	8	10	10	10	18	17	19		19
	!addr16, A	4	—	6	8	8	8	—	14	16		17
	&!addr16, A	5		8	10	10	10		17	19		20
	PSW, #byte	3		—	—	5	—		—	—	9	—
	PSW, A	2									6	
	A, PSW	2				4						

(Continued)

Notes 1. IRAM for both saddr and saddr'

2. SFR for both saddr and saddr'

Mnemonic	Operands	Bytes	Clock cycles									
			Internal ROM high-speed fetch					External ROM fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
XCH	A, r	1	—	4	—	—	—	—	4	—	—	—
	r, r'	2		3					6			
	A, mem	2-4	See **Table 7-6 (3/4)** for details.									
	A, &mem	3-5										
	A, saddr	2	—	4	—	8	—	—	6	—		—
	A, sfr	3		—		10	10		—		13	
	saddr, saddr''	3		6**Note 1**		14**Note 2**	—		10**Note 1**			—

Notes 1. IRAM for both saddr and saddr'

2. SFR for both saddr and saddr'

(2) 16-bit data transfer instructions: MOVW

Mnemonic	Operands	Bytes	Clock cycles									
			Internal ROM high-speed fetch					External ROM fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
MOVW	rp, #word	3	—	3	—	—	—	—	9	—	—	—
	saddrp, #word	4		4		8			12		12	
	sfrp, #word	4		—								
	rp, rp'	2		4		—			6		—	
	AX, saddrp	2		6		10			8		12	
	saddrp, AX	2		5		9			7			
	AX, sfrp	2		—		10			—			
	sfrp, AX	2				9						
	AX, mem1	2	11	9	13	13	13	14	12	16	10	16
	AX, &mem1	3	13	11	15	15	15	17	15	19	19	19
	mem1, AX**Note**	2	—	8	12	12	12	—	11	15	15	15
	&mem1, AX**Note**	3		10	14	14	14		14	18	18	18

Note Cannot be used on μPD78244 sub-series EEPROM area.

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

Mnemonic	Operands	Bytes	Clock cycles									
			Internal ROM high-speed fetch					External ROM fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
ADD	A, #byte	2	—	2	—	—	—	—	6	—	—	—
	saddr, #byte	3		4		8			9		11	
	sfr, #byte	4		—		10	10		—		14	18
	r, r'	2		3		—	—		7		—	—
	A, saddr	2				5			6		7	
	A, sfr	3		—		7	7		—		10	11
	saddr, saddr'	3		4 Note 1		10 Note 2	—		9 Note 1		11 Note 2	—
	A, mem	2-4	See **Table 7-6 (4/4)** for details.									
	A, &mem	3-5										
ADDC	A, #byte	2	—	2	—	—	—	—	6	—	—	—
	saddr, #byte	3		4		8			9		11	
	sfr, #byte	4		—		10	10		—		14	18
	r, r'	2		3		—	—		7		—	—
	A, saddr	2				5			6		7	
	A, sfr	3		—		7	7		—		10	11
	saddr, saddr'	3		4 Note 1		10 Note 2	—		9 Note 1		11 Note 2	—
	A, mem	2-4	See **Table 7-6 (4/4)** for details.									
	A, &mem	3-5										
SUB	A, #byte	2	—	2	—	—	—	—	6	—	—	—
	saddr, #byte	3		4		8			9		11	
	sfr, #byte	4		—		10	10		—		14	18
	r, r'	2		3		—	—		7		—	—
	A, saddr	2				5			6		7	
	A, sfr	3		—		7	7		—		10	11
	saddr, saddr'	3		4 Note 1		10 Note 2	—		9 Note 1		11 Note 2	—
	A, mem	2-4	See **Table 7-6 (4/4)** for details.									
	A, &mem	3-5										

(Continued)

Notes 1. IRAM for both saddr and saddr'

2. SFR for both saddr and saddr'

Mnemonic	Operands	Bytes	Clock cycles									
			Internal ROM high-speed fetch					External ROM fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
SUBC	A, #byte	2	—	2	—	—	—	—	6	—	—	—
	saddr, #byte	3		4		8			9		11	
	sfr, #byte	4		—		10	10		—		14	18
	r, r'	2		3		—	—		7		—	—
	A, saddr	2				5			6		7	
	A, sfr	3		—		7	7		—		10	11
	saddr, saddr'	3		4 Note 1		10 Note 2	—		9 Note 1		11 Note 2	—
	A, mem	2-4	See **Table 7-6 (4/4)** for details.									
	A, &mem	3-5										
AND	A, #byte	2	—	2	—	—	—	—	6	—	—	—
	saddr, #byte	3		4		8			9		11	
	sfr, #byte	4		—		10	10		—		14	18
	r, r'	2		3		—	—		7		—	—
	A, saddr	2				5			6		7	
	A, sfr	3		—		7	7		—		10	11
	saddr, saddr'	3		4 Note 1		10 Note 2	—		9 Note 1		11 Note 2	—
	A, mem	2-4	See **Table 7-6 (4/4)** for details.									
	A, &mem	3-5										
OR	A, #byte	2	—	2	—	—	—	—	6	—	—	—
	saddr, #byte	3		4		8			9		11	
	sfr, #byte	4		—		10	10		—		14	18
	r, r'	2		3		—	—		7		—	—
	A, saddr	2				5			6		7	
	A, sfr	3		—		7	7		—		10	11
	saddr, saddr'	3		4 Note 1		10 Note 2	—		9 Note 1		11 Note 2	—
	A, mem	2-4	See **Table 7-6 (4/4)** for details.									
	A, &mem	3-5										

(Continued)

Notes 1. IRAM for both saddr and saddr'

2. SFR for both saddr and saddr'

Mnemonic	Operands	Bytes	Clock cycles									
			Internal ROM high-speed fetch					External ROM fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
XOR	A, #byte	2	—	2	—	—	—	—	6	—	—	—
	saddr, #byte	3		4		8			9		11	
	sfr, #byte	4		—		10	10		—		14	18
	r, r'	2		3		—	—		7		—	—
	A, saddr	2				5			6		7	
	A, sfr	3		—		7	7		—		10	11
	saddr, saddr'	3		4 **Note 1**		10 **Note 2**	—		9 **Note 1**		11 **Note 2**	—
	A, mem	2-4	See **Table 7-6 (4/4)** for details.									
	A, &mem	3-5										
CMP	A, #byte	2	—	2	—	—	—	—	6	—	—	—
	saddr, #byte	3		3		5			9		11	
	sfr, #byte	4		—		7	7		—		14	15
	r, r'	2		3		—	—		7		—	—
	A, saddr	2				5			6		7	
	A, sfr	3		—		7	7		—		10	11
	saddr, saddr'	3		3 **Note 1**		7 **Note 2**	—		9 **Note 1**		11 **Note 2**	—
	A, mem	2-4	See **Table 7-6 (4/4)** for details.									
	A, &mem	3-5										

Notes 1. IRAM for both saddr and saddr'

2. SFR for both saddr and saddr'

(4) 16-bit operation instructions: ADDW, SUBW, CMPW

Mnemonic	Operands	Bytes	Clock cycles									
			Internal ROM high-speed fetch					External ROM fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
ADDW	AX, #word	3	—	4	—	—	—	—	9	—	—	—
	AX, rp	2		6					8			
	AX, saddrp	2		7		11			9		13	
	AX, sfrp	3		—		13			—		16	
SUBW	AX, #word	3	—	4	—	—	—	—	9	—	—	—
	AX, rp	2		6					8			
	AX, saddrp	2		7		11			9		13	
	AX, sfrp	3		—		13			—		16	
CMPW	AX, #word	3	—	3	—	—	—	—	9	—	—	—
	AX, rp	2		5					7			
	AX, saddrp	2		6		10			8		12	
	AX, sfrp	3		—		12			—		15	

(5) Multiplication/division instructions: MULU, DIVUW

Mnemonic	Operands	Bytes	Clock cycles									
			Internal ROM high-speed fetch					External ROM fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
MULU	r	2	—	22	—	—	—	—	24	—	—	—
DIVUW	r	2	—	74	—	—	—	—	76	—	—	—

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

Mnemonic	Operands	Bytes	Clock cycles									
			Internal ROM high-speed fetch					External ROM fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
INC	r	1	—	2	—	—	—	—	3	—	—	—
	saddr	2		3		7			6		7	—
DEC	r	1	—	2	—	—	—	—	3	—	—	—
	saddr	2		3		7			6		7	
INCW	rp	1	—	3	—	—	—	—	3	—	—	—
DECW	rp	1	—	3	—	—	—	—	3	—	—	—

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

Mnemonic	Operands	Bytes	Clock cycles									
			Internal ROM high-speed fetch					External ROM fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
ROR	r, n	2	—	3+2n	—	—	—	—	5+2n	—	—	—
ROL	r, n	2	—	3+2n	—	—	—	—	5+2n	—	—	—
RORC	r, n	2	—	3+2n	—	—	—	—	5+2n	—	—	—
ROLC	r, n	2	—	3+2n	—	—	—	—	5+2n	—	—	—
SHR	r, n	2	—	3+2n	—	—	—	—	5+2n	—	—	—
SHL	r, n	2	—	3+2n	—	—	—	—	5+2n	—	—	—
SHRW	rp, n	2	—	3+3n	—	—	—	—	5+3n	—	—	—
SHLW	rp, n	2	—	3+3n	—	—	—	—	5+3n	—	—	—
ROR4[Note]	mem1	2	—	24	32	32	32	—	26	34	34	34
	&mem1	3		26	34	34	34		29	37	37	37
ROL4[Note]	mem1	2	—	25	33	33	33	—	27	35	35	35
	&mem1	3		27	35	35	35		30	38	38	38

Note Cannot be used on μPD78244 sub-series EEPROM area.

(8) BCD adjustment instructions: ADJBA, ADJBS

Mnemonic	Operands	Bytes	Clock cycles									
			Internal ROM high-speed fetch					External ROM fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
ADJBA		1	—	3	—	—	—	—	3	—	—	—
ADJBS		1	—	3	—	—	—	—	3	—	—	—

Caution Internal ROM

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

Mnemonic	Operands	Bytes	Clock cycles									
			Internal ROM high-speed fetch					External ROM fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
MOV1	CY, saddr.bit	3	—	5	—	7	—	—	9	—	9	—
	CY, sfr.bit	3		—			7		—			11
	CY, A.bit	2		5		—	—		7		—	—
	CY, X.bit	2										
	CY, PSW.bit	2		—		5			—		7	
	saddr.bit, CY	3		8		12			12		14	
	sfr.bit, CY	3		—			12		—			14
	A.bit, CY	2		8		—	—		10		—	—
	X.bit, CY	2										
	PSW.bit, CY	2		—		7			—		9	
AND1	CY, saddr.bit	3	—	5	—	7	—	—	9	—	11	—
	CY, /saddr.bit	3										
	CY, sfr.bit	3		—			7		—			12
	CY, /sfr.bit	3										
	CY, A.bit	2		5		—	—		7		—	—
	CY, /A.bit	2										
	CY, X.bit	2										
	CY, /X.bit	2										
	CY, PSW.bit	2		—		5			—		7	
	CY, /PSW.bit	2										

(Continued)

Mnemonic	Operands	Bytes	Clock cycles									
			Internal ROM high-speed fetch					External ROM fetch				
			Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
OR1	CY, saddr.bit	3	—	5	—	7	—	—	9	—	11	—
	CY, /saddr.bit	3										
	CY, sfr.bit	3		—			7		—			12
	CY, /sfr.bit	3										
	CY, A.bit	2		5		—	—		7		—	—
	CY, /A.bit	2										
	CY, X.bit	2										
	CY, /X.bit	2										
	CY, PSW.bit	2		—		5			—		7	
	CY, /PSW.bit	2										
XOR1	CY, saddr.bit	3	—	5	—	7	—	—	9	—	11	—
	CY, sfr.bit	3		—			7		—			12
	CY, A.bit	2		5		—	—		7		—	—
	CY, X.bit	2										
	CY, PSW.bit	2		—		5			—		7	
SET1	saddr.bit	2	—	3	—	7	—	—	6	—	11	—
	sfr.bit	3		—		10	10		—		14	16
	A.bit	2		6		—	—		8		—	—
	X.bit	2										
	PSW.bit	2		—		5			—		7	
	CY	1				2					3	
CLR1	saddr.bit	2	—	3	—	7	—	—	6	—	11	—
	sfr.bit	3		—		10	10		—		14	16
	A.bit	2		6		—	—		8		—	—
	X.bit	2										
	PSW.bit	2		—		5			—		7	
	CY	1				2					3	
NOT1	saddr.bit	2	—	6	—	10	—	—	10	—	14	—
	sfr.bit	3		—			10		—			16
	A.bit	2		6		—	—		8		—	—
	X.bit	2										
	PSW.bit	2		—		5			—		7	
	CY	1				2					3	

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

Mnemonic	Operands	Bytes	Clock cycles											
			Internal ROM high-speed fetch						External ROM fetch					
			Internal ROM –> Internal ROM			Internal ROM –> External ROM			External ROM –> Internal ROM			External ROM –> External ROM		
			IRAM	PRAM	EMEM	IRAM	PRAM	EMEM	IRAM	PRAM	EMEM	IRAM	PRAM	EMEM
CALL	!addr16	3	11/12	15/16	15/16	12/13	16/17	16/17	15/16	19/20	19/20	17/18	21/22	21/22
	rp	2	12/13	16/17	16/17	14/15	18/19	18/19	13/14	17/18	17/18	15/16	19/20	19/20
CALLF	!addr11	2	11/12	15/16	15/16	—	—	—	12/13	16/17	16/17	14/15	18/19	18/19
CALLT	[addr5]	1	14/15	18/19	18/19	16/17	20/21	20/21	14/15	18	18	20/21	24/25	24/25
BRK		1	16/17 /18	22/23 /24	22/23 /24	18/19 /20	24/25 /26	24/25 /26	17/18 /19	23/24 /25	23/24 /25	22/24	28/30	28/30
RET		1	10/11	14/15	14/15	11/12	15/16	15/16	10/11	14/15	14/15	11/12	15/16	15/16
RETI		1	15/16	21/22	21/22	15/16	21/22	21/22	15/16	21/22	21/22	15/16	21/22	21/22
RETB		1	14/15	20/21	20/21	14/15	20/21	20/21	14/15	20/21	20/21	14/15	20/21	20/21

(11) Stack manipulation instructions: PUSH, POP, MOVW, INCW, DECW

Mnemonic	Operands	Bytes	Clock cycles					
			Internal ROM high-speed fetch			External ROM fetch		
			IRAM	PRAM	EMEM	IRAM	PRAM	EMEM
PUSH	PSW	1	**Note**	**Note**	**Note**	**Note**	**Note**	**Note**
	sfr	2	7	9	9	9	12	12
	rp	1	8/9	12/13	12/13	8/9	12/13	12/13
POP	PSW	1	6	8	8	6	8	8
	sfr	2	9	11	11	9	12	12
	rp	1	11/12	15/16	15/16	11/12	15/16	15/16
MOVW	SP, #word	4	8	8	8	12	12	12
	SP, AX	2	9	9	9	11	11	11
	AX, SP	2	10	10	10	12	12	12
INCW	SP	2	5	5	5	7	7	7
DECW	SP	2	5	5	5	7	7	7

Note μPD78214 sub-series and 78224 sub-series: 5/7, 7/9, 7/9, 5/7, 7/9, 7/9
μPD78218A sub-series, 78234 sub-series and 78244 sub-series: 6, 8, 8, 6, 8, 8

(12) Unconditional branch instructions: BR

Mnemonic	Operands	Bytes	Clock cycles					
			Internal ROM high-speed fetch			External ROM fetch		
			No branch	Branch		No branch	Branch	
				Internal ROM –> Internal ROM	Internal ROM –> External ROM		External ROM –> Internal ROM	External ROM –> External ROM
BR	!addr16	3	—	5	7	—	9	11
	rp	2		6	8		8	10
	$addr16	2		4	6		7	9

(13) Conditional branch instructions: BC, BL, BNC, BNL, BZ, BE, BNZ, BNE, BT, BF, BTCLR, DBNZ

Mnemonic	Operands	Bytes	Clock cycles					
			Internal ROM high-speed fetch			External ROM fetch		
			No branch	Branch		No branch	Branch	
				Internal ROM –> Internal ROM	Internal ROM –> External ROM		External ROM –> Internal ROM	External ROM –> External ROM
BC / **BL**	$addr16	2	2	4	6	6	7	9
BNC / **BNL**	$addr16	2	2	4	6	6	7	9
BZ / **BE**	$addr16	2	2	4	6	6	7	9
BNZ / **BNE**	$addr16	2	2	4	6	6	7	9
BT	saddr.bit, $addr16	3	4	6	8	9	10	12
	sfr.bit, $adddr16	4	7	9	11	13	15	16
	A.bit, $addr16	3	5	7	8	9	10	12
	X.bit, $addr16	3						
	PSW.bit, $addr16	3						

(Continued)

Mnemonic	Operands	Bytes	Clock cycles					
			Internal ROM high-speed fetch			External ROM fetch		
			No branch	Branch		No branch	Branch	
				Internal ROM –> Internal ROM	Internal ROM –> External ROM		External ROM –> Internal ROM	External ROM –> External ROM
BF	saddr.bit, $addr16	4	5	7	9	12	13	15
	sfr.bit, $adddr16	4	7	9	11	13	15	16
	A.bit, $addr16	3	5	7	8	9	10	12
	X.bit, $addr16	3						
	PSW.bit, $addr16	3						
BTCLR	saddr.bit, $addr16	4	5	9	9	12	15	15
	sfr.bit, $addr16	4	7	13	13	13	18	18
	A.bit, $addr16	3	5	9	9	9	12	12
	X.bit, $addr16	3						
	PSW.bit, $addr16	3		8			11	
DBNZ	r1, $addr16	2	3	5	7	6	7	9
	saddr, $addr16	3	4	6	8	9	10	12

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

Mnemonic	Operands	Bytes	Clock cycles	
			Internal ROM high-speed fetch	External ROM fetch
MOV	STBC, #byte	4	9	15
SEL	RBn	2	2	6
NOP		1	2	3
EI		1	2	3
DI		1	2	3

Table 7-6. Table of Instruction Execution Cycles (1/4)

Instruction group	Mnemonic	Operands	Bytes	Internal ROM high-speed fetch					External ROM fetch				
				Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
8-bit data transfer instructions	MOV	A, [DE] A, [HL]	1	6	5	7	7	7	7	6	8	8	8
		A, &[DE] A, &[HL]	2	8	7	9	9	9	10	9	11	11	11
	Note	A, [DE+] A, [HL+] A, [DE–] A, [HL–]	1	9	8/9	10	10	10	10	9/10	11	11	11
		A, &[DE+] A, &[HL+] A, &[DE–] A, &[HL–]	2	11	10/11	12	12	12	13	12/13	14	14	14
		A, [DE+byte] A, [HL+byte]	3	8	7/8	9	9	9	12	11/12	13	13	13
		A, &[DE+byte] A, &[HL+byte]	4	10	9/10	11	11	11	15	14/15	16	16	16
		A, [SP+byte]	3	9	8/9	10	10	10	13	12/13	14	14	14
		A, &[SP+byte]	4	11	10/11	12	12	12	16	15/16	17	17	17
		A, word[A] A, word[B] A, word[DE] A, word[HL]	4	8	7/8	9	9	9	15	14	16	16	16
		A, &word[A] A, &word[B] A, &word[DE] A, &word[HL]	5	10	9/10	11	11	11	18	17	19	19	19

Note The above figures apply when instructions with a short word length are used.
When one-byte word length is long : Internal ROM fetch : +1 cycle
External ROM fetch : +3 cycles

Table 7-6. Table of Instruction Execution Cycles (2/4)

Instruction group	Mnemonic	Operands	Bytes	Internal ROM high-speed fetch					External ROM fetch				
				Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
8-bit data transfer instructions	MOV	[DE], A [HL], A	1	—	5	7	7	7	—	6	8	8	8
		&[DE], A &[HL], A	2	—	7	9	9	9	—	9	11	11	11
	Note	[DE+], A [HL+], A [DE–], A [HL–], A	2	—	8/9	10	10	10	—	9/10	11	11	11
		&[DE+], A &[HL+], A &[DE–], A &[HL–], A	3	—	10/11	12	12	12	—	12/13	14	14	14
		[DE+byte], A [HL+byte], A	3	—	7/8	9	9	9	—	11/12	13	13	13
		&[DE+byte], A &[HL+byte], A	4	—	9/10	11	11	11	—	14/15	16	16	16
		[SP+byte], A	3	—	8/9	10	10	10	—	12/13	14	14	14
		&[SP+byte], A	4	—	10/11	12	12	12	—	15/16	17	17	17
		word[A], A word[B], A word[DE], A word[HL], A	4	—	7/8	9	9	9	—	14	16	16	16
		&word[A], A &word[B], A &word[DE], A word[HL], A	5	—	9/10	11	11	11	—	17	19	19	19

Note The above figures apply when instructions with a short word length are used.
When one-byte word length is long : Internal ROM fetch : +1 cycle
External ROM fetch : +3 cycles

Table 7-6. Table of Instruction Execution Cycles (3/4)

Instruction group	Mnemonic	Operands	Bytes	Internal ROM high-speed fetch					External ROM fetch				
				Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
8-bit data transfer instruc-tions	XCH	A, [DE] A, [HL]	2	—	9	13	13	13	—	12	16	16	16
		A, &[DE] A, &[HL]	3	—	11	15	15	15	—	15	19	19	19
		A, [DE+] A, [HL+] A, [DE–] A, [HL–]	2	—	11/12	15	15	15	—	14/15	18	18	18
		A, &[DE+] A, &[HL+] A, &[DE–] A, &[HL–]	3	—	13/14	17	17	17	—	17/18	21	21	21
		A, [DE+byte] A, [HL+byte]	3	—	9/10	13	13	13	—	13/14	17	17	17
		A, &[DE+byte] A, &[HL+byte]	4	—	11/12	15	15	15	—	16/17	20	20	20
		A, [SP+byte]	3	—	10/11	14	14	14	—	14/15	18	18	18
		A, &[SP+byte]	4	—	12/13	16	16	16	—	17/18	21	21	21
		A, word[A] A, word[B] A, word[DE] A, word[HL]	4	—	9/10	13	13	13	—	16	20	20	20
		A, &word[A] A, &word[B] A, &word[DE] A, &word[HL]	5	—	11/12	15	15	15	—	19	23	23	23

Table 7-6. Table of Instruction Execution Cycles (4/4)

Instruction group	Mnemonic	Operands	Bytes	Internal ROM high-speed fetch					External ROM fetch				
				Internal ROM	IRAM	PRAM	SFR	EMEM	Internal ROM	IRAM	PRAM	SFR	EMEM
8-bit data arithmetic & logical operation instruc-tions	ADD ADDC SUB SUBC AND OR XOR CMP	A, [DE] A, [HL]	2	9	8	10	10	10	12	11	13	13	13
		A, &[DE] A, &[HL]	3	11	10	12	12	12	15	14	16	16	16
		A, [DE+] A, [HL+] A, [DE–] A, [HL–]	2	11	10/11	12	12	12	14	13	15	15	15
		A, &[DE+] A, &[HL+] A, &[DE–] A, &[HL–]	3	13	12/13	14	14	14	17	16	18	18	18
		A, [DE+byte] A, [HL+byte]	3	9	8/9	10	10	10	13	12	14	14	14
		A, &[DE+byte] A, &[HL+byte]	4	11	10/11	12	12	12	16	15	17	17	17
		A, [SP+byte]	3	10	9/10	11	11	11	14	13	15	15	15
		A, &[SP+byte]	4	12	11/12	13	13	13	17	16	18	18	18
		A, word[A] A, word[B] A, word[DE] A, word[HL]	4	9	8/9	10	10	10	14	15	17	17	17
		A, &word[A] A, &word[B] A, &word[DE] A, &word[HL]	5	11	10/11	12	12	12	17	18	20	20	20

[MEMO]

CHAPTER 8 INSTRUCTION DESCRIPTIONS

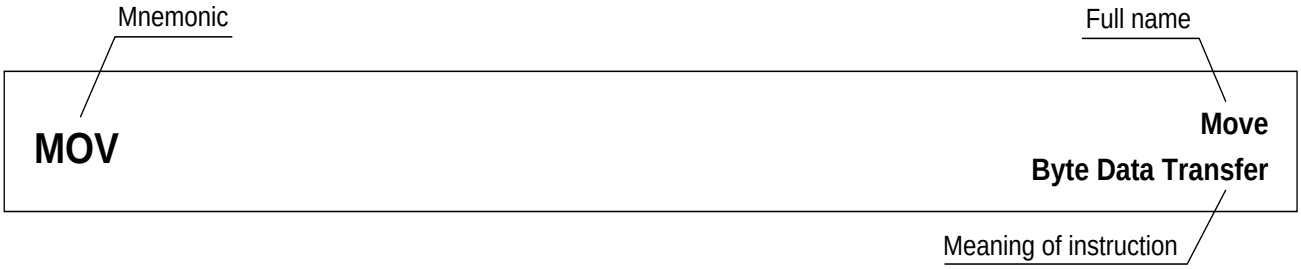
This chapter describes the instructions of the 78K/II series products. Instructions with different operands are grouped by mnemonic.

The basic organization of the instruction descriptions is shown on the next page.

Please refer to **Chapter 7** "Instruction Set" for the number of bytes and operation codes of the instructions.

The same instructions are used on all 78K/II series products.

Description Example



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

[Operation] **dst <- src**: Shows the instruction operation using symbols.

[Operands] Shows the operands which can be specified with this instruction. See **Chapter 7** for the symbols used for the various operands.

Mnemonic	Operands
MOV	r, #byte
	saddr, #byte
	A, saddr
	saddr, A
	A, mem

Mnemonic	Operands
MOV	A, &mem
	A, !addr16
	A, &!addr16
	PSW, A
	A, PSW

[Flags] Shows the operation of flags which are changed by execution of the instruction. The symbols used for flag operations are shown below.

Z	AC	CY

Legend

Symbol	Meaning
Blank	No change
0	Cleared to 0
1	Set to 1
x	Set or cleared depending on result
R	Previously saved value is restored

[Description] Describes the operation of the instruction in detail.

- Transfers the contents of the source operand (src) specified by the 2nd operand to the destination operand (dst) specified by the 1st operand.

[Coding example]

MOV A, #4DH ; Transfers 4DH to the A register

8.1 8-BIT DATA TRANSFER INSTRUCTIONS

8-bit data transfer instructions are as follows:

MOV

Move
Byte Data Transfer

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

[Operation] **dst <- src**

[Operands]

Mnemonic	Operands (dst, src)
MOV	r, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, r
	A, saddr
	saddr, A
	saddr, saddr'
	A, sfr
	sfr, A
	A, mem

Mnemonic	Operands (dst, src)
MOV	A, &mem
	mem, A
	&mem, A
	A, !addr16
	A, &!addr16
	!addr16, A
	&!addr16, A
	PSW, #byte
	PSW, A
	A, PSW

[Flags]

In case of PSW, #byte and PSW, A operands

Z	AC	CY
x	x	x

Other than cases at left

Z	AC	CY

[Description]

- Transfers the contents of the source operand (src) specified by the 2nd operand to the destination operand (dst) specified by the 1st operand.
- No interrupts or macro services are acknowledged between MOV PSW, #byte instruction, MOV PSW, A instruction and the next instruction.

[Coding example]

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

XCH

Exchange
Byte Data Exchange

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

[Operation] **dst <-> src**

[Operands]

Mnemonic	Operands (dst, src)
XCH	A, r
	r, r'
	A, mem
	A, &mem
	A, saddr
	A, sfr
	saddr, saddr'

[Flags]

Z	AC	CY

[Description]

- Exchanges the contents of the 1st operand with contents of the 2nd operand.

[Coding example]

XCH A, OFEBCH ; Exchanges the contents of register A with the contents of address OFEBCH.

8.2 16-BIT DATA TRANSFER INSTRUCTIONS

16-bit data transfer instructions are as follows:

MOVW ... 147

MOVW

Move Word
Word Data Transfer

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

[Operation] **dst <- drc**

[Operands]

Mnemonic	Operands (dst, src)
MOVW	rp, #word
	saddrp, #word
	sfrp, #word
	rp, rp'
	AX, saddrp
	saddrp, AX
	AX, sfrp
	sfrp, AX
	AX, mem1
	AX, &mem1
	mem1, AX**Note**
	&mem1, AX**Note**

Note Cannot be used for the EEPROM area of the μPD78244 sub-series.

[Flags]

Z	AC	CY

[Description]

- Transfers the contents of the source operand (src) specified by the 2nd operand to the destination operand (dst) specified by the 1st operand.

[Coding example]

MOVW AX, [HL] ; Transfers the memory contents indicated by the HL register to the AX register.

8.3 8-BIT OPERATION INSTRUCTIONS

8-bit operation instructions are as follows:

ADD

Add
Byte Data Addition

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

[Operation] **dst, CY <- dst + src**

[Operands]

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

[Flags]

Z	AC	CY
x	x	x

[Description]

- Adds the source operand (src) specified by the 2nd operand to the destination operand (dst) specified by the 1st operand, and stores the result in the CY flag and the destination operand (dst).
- If dst is 0 as a result of the addition the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a carry is generated out of bit 7 as a result of the addition the CY flag is set (1), otherwise the CY flag is cleared (0).
- If a carry is generated out of bit 3 into bit 4 as a result of the addition the AC flag is set (1), otherwise the AC flag is cleared (0).

[Coding example]

ADD CR11, #56H ; Adds 56H to the CR11 register and stores the result in the CR11 register.

ADDC

Add with Carry
Byte Data Addition including 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, r'
	A, saddr
	A, sfr
	saddr, saddr'
	A, mem
	A, &mem

[Flags]

Z	AC	CY
x	x	x

[Description]

- Adds the source operand (src) specified by the 2nd operand and the CY flag to the destination operand (dst) specified by the 1st operand, and stores the result in the destination operand (dst) and the CY flag. The CY flag is added to the LSB.
  This instruction is mainly used in multiple-byte additions.
- If dst is 0 as a result of the addition the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a carry is generated out of bit 7 as a result of the addition the CY flag is set (1), otherwise the CY flag is cleared (0).
- If a carry is generated out of bit 3 into bit 4 as a result of the addition the AC flag is set (1), otherwise the AC flag is cleared (0).

[Coding example]

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

SUB

Subtract
Byte Data Subtraction

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

[Operation] **dst, CY <- dst – src**

[Operands]

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

[Flags]

Z	AC	CY
x	x	x

[Description]

- Subtracts the source operand (src) specified by the 2nd operand from the destination operand (dst) specified by the 1st operand, and stores the result in the destination operand (dst) and the CY flag. The destination operand can be cleared to 0 by making the source operand (src) and the destination operand (dst) the same.
- If dst is 0 as a result of the subtraction the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a borrow is generated in bit 7 as a result of the subtraction the CY flag is set (1), otherwise the CY flag is cleared (0).
- If a borrow is generated out of bit 4 into bit 3 as a result of the subtraction the AC flag is set (1), otherwise the AC flag is cleared (0).

[Coding example]

SUB D, L ; Subtracts the L register from the D register and stores the result in the D register.

SUBC

Subtract with Carry
Byte Data Subtraction including 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, r'
	A, saddr
	A, sfr
	saddr, saddr'
	A, mem
	A, &mem

[Flags]

Z	AC	CY
x	x	x

[Description]

- Subtracts the source operand (src) specified by the 2nd operand and the CY flag from the destination operand (dst) specified by the 1st operand, and stores the result in the destination operand (dst). The CY flag is subtracted from the LSB.
  This instruction is mainly used in multiple-byte subtractions.
- If dst is 0 as a result of the subtraction the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a borrow is generated in bit 7 as a result of the subtraction the CY flag is set (1), otherwise the CY flag is cleared (0).
- If a borrow is generated out of bit 4 into bit 3 as a result of the subtraction the AC flag is set (1), otherwise the AC flag is cleared (0).

[Coding example]

SUBC A, [DE+] ; Subtracts the contents of (DE register) address and the CY flag from the A register and stores the result in the A register. (The DE register is incremented after the operation.)

AND

And
Byte Data Logical Product

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

[Operation] **dst <- dst ∧ src**

[Operands]

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

[Flags]

Z	AC	CY
x		

[Description]

- Obtains the bit-by-bit logical product of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand, and stores the result in the destination operand (dst).
- If all bits are 0 as a result of obtaining the logical product, the Z flag is set (1), otherwise the Z flag is cleared (0).

[Coding example]

AND 0FEBAH, #11011100B ; Obtains the bit-by-bit logical product of the contents of 0FEBAH and 11011100B, and stores the result in 0FEBAH.

OR

Or
Byte Data Logical Sum

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

[Operation] **dst <- dst ϖ src**

[Operands]

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

[Flags]

Z	AC	CY
x		

[Description]

- Obtains the bit-by-bit logical sum of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand, and stores the result in the destination operand (dst).
- If all bits are 0 as a result of obtaining the logical sum, the Z flag is set (1), otherwise the Z flag is cleared (0).

[Coding example]

OR A, 0FE98H ; Obtains the bit-by-bit logical sum of the A register and 0FE98H, and stores the result in the A register.

XOR

Exclusive Or
Byte Data Exclusive Logical Sum

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

[Operation] **dst <- dst ⩛ src**

[Operands]

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

[Flags]

Z	AC	CY
x		

[Description]

- Obtains the bit-by-bit exclusive logical sum of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand, and stores the result in the destination operand (dst).
  The logical negation of all bits of the destination operand (dst) can be obtained by selecting #0FFH as the source operand (src) of this instruction.
- If all bits are 0 as a result of obtaining the exclusive logical sum, the Z flag is set (1), otherwise the Z flag is cleared (0).

[Coding example]

XOR A, P2 ; Obtains the bit-by-bit exclusive logical sum of the A register and the P2 register, and stores the result in the A register.

CMP

Compare
Byte Data Comparison

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

[Operation] **dst – src**

[Operands]

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

[Flags]

Z	AC	CY
x	x	x

[Description]

- Subtracts the source operand (src) specified by the 2nd operand from the destination operand (dst) specified by the 1st operand.
  The result of the subtraction is not stored anywhere; only the Z, AC and CY flags are changed.
- If the result of the subtraction is 0 the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a borrow is generated in bit 7 as a result of the subtraction the CY flag is set (1), otherwise the CY flag is cleared (0).
- If a borrow is generated out of bit 4 into bit 3 as a result of the subtraction the AC flag is set (1), otherwise the AC flag is cleared (0).

[Coding example]

```
CMP 0FE38H, 0FED0H  ; Subtracts the contents of address 0FED0H from the contents of address
                      0FE38H, and performs flag modification only (comparison of address 0FE38H
                      contents and address 0FED0H contents).
```

8.4 16-BIT OPERATION INSTRUCTIONS

16-bit operation instructions are as follows:

ADDW

Add Word
Word Data Addition

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

[Operation] **dst, CY <- dst + src**

[Operands]

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

[Flags]

Z	AC	CY
x	x	x

[Description]

- Adds the source operand (src) specified by the 2nd operand to the destination operand (dst) specified by the 1st operand, and stores the result in the destination operand (dst).
- If dst is 0 as a result of the addition the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a carry is generated out of bit 15 as a result of the addition the CY flag is set (1), otherwise the CY flag is cleared (0).
- The AC flag is undefined as a result of the addition.

[Coding example]

ADDW AX, #0AB0DH ; Adds 0ABCDH to the AX register and stores the result in the AX register.

SUBW

Subtract Word
Word Data Subtraction

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

[Operation] **dst, CY <- dst – src**

[Operands]

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

[Flags]

Z	AC	CY
x	x	x

[Description]

- Subtracts the source operand (src) specified by the 2nd operand from the destination operand (dst) specified by the 1st operand, and stores the result in the destination operand (dst) and the CY flag. The destination operand can be cleared to 0 by making the source operand (src) and the destination operand (dst) the same.
- If dst is 0 as a result of the subtraction the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a borrow is generated in bit 15 as a result of the subtraction the CY flag is set (1), otherwise the CY flag is cleared (0).
- The AC flag is undefined as a result of the subtraction.

[Coding example]

SUBW AX, CR01 ; Subtracts the contents of the CR01 register from the contents of the AX register, and stores the result in the AX register.

CMPW

Compare Word
Word Data Comparison

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

[Operation] **dst – src**

[Operands]

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

[Flags]

Z	AC	CY
x	x	x

[Description]

- Subtracts the source operand (src) specified by the 2nd operand from the destination operand (dst) specified by the 1st operand.
  The result of the subtraction is not stored anywhere; only the Z, AC and CY flags are changed.
- If a result of the subtraction is 0 the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a borrow is generated in bit 15 as a result of the subtraction the CY flag is set (1), otherwise the CY flag is cleared (0).
- The AC flag is undefined as a result of the subtraction.

[Coding example]

CMPW AX, 0FE43H ; Subtracts the word data in address 0FE43H from the AX register, and performs flag modification only (comparison of the AX register and address 0FE43H word data).

8.5 MULTIPLICATION/DIVISION INSTRUCTIONS

Multiplication/division instructions are as follows:

MULU

Multiply Unsigned
Unsigned Data Multiplication

[Instruction format] **MULU src**

[Operation] **AX <- AX x src**

[Operands]

Mnemonic	Operands (src)
MULU	r

[Flags]

Z	AC	CY

[Description]

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

[Coding example]

MULU H ; Multiplies the A register contents by the H register contents and stores the result in the AX register.

DIVUW

Divide Unsigned Word
Unsigned Word Data Division

[Instruction format] **DIVUW dst**

[Operation] **AX (quotient), dst (remainder) <- AX/dst**

[Operands]

Mnemonic	Operands (dst)
DIVUW	r

[Flags]

Z	AC	CY

[Description]

- 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 treats the contents of the AX register and the destination operand (dst) as unsigned data.
- Dividing the contents by 0 (dst = 0) results in:
  AX (quotient) = FFFFH
  dst (remainder) = Original value of the X register

[Coding example]

DIVUW E ; Divides the AX register contents by the E register contents, and stores the quotient in the AX register and the remainder in the E register.

8.6 INCREMENT/DECREMENT INSTRUCTIONS

Increment/decrement instructions are as follows:

INC

Increment
Byte Data Increment

[Instruction format] **INC dst**

[Operation] **dst <- dst + 1**

[Operands]

Mnemonic	Operands (dst)
INC	r
	saddr

[Flags]

Z	AC	CY
x	x	

[Description]

- Increments the contents of the destination operand (dst) by 1.
- If the result of the increment is 0 the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a carry is generated out of bit 3 into bit 4 as a result of the increment the AC flag is set (1), otherwise the AC flag is cleared (0).
- Since this instruction is often used to increment a counter for repeated processing or the offset register in indexed addressing, the contents of the CY flag are not changed (in order to retain the CY flag contents in a multiple-byte operation).

[Coding example]

INC B ; Increments the B register.

DEC

Decrement
Byte Data Decrement

[Instruction format] **DEC dst**

[Operation] **dst <- dst – 1**

[Operands]

Mnemonic	Operands (dst)
DEC	r
	saddr

[Flags]

Z	AC	CY
x	x	

[Description]

- Decrements the contents of the destination operand (dst) by 1.
- If the result of the decrement is 0 the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a carry is generated out of bit 4 into bit 3 as a result of the decrement the AC flag is set (1), otherwise the AC flag is cleared (0).
- Since this instruction is often used to decrement a counter for repeated processing or the offset register in indexed addressing, the contents of the CY flag are not changed (in order to retain the CY flag contents in a multiple-byte operation).
- If it is not wished to change the AC and CY flags when dst is the B register, C register or saddr, the DBNZ instruction can be used.

[Coding example]

DEC 0FE92H ; Decrements the contents of address 0FE92H.

INCW

Increment Word
Word Data Increment

[Instruction format] **INCW dst**

[Operation] **dst <- dst + 1**

[Operands]

Mnemonic	Operands (dst)
INCW	rp

[Flags]

Z	AC	CY

[Description]

- Increments the contents of the destination operand (dst) by 1.
- Since this instruction is often used to increment the register (pointer) in addressing which uses a register, the contents of the Z, AC and CY flags are not changed.

[Coding example]

INCW HL ; Increments the HL register.

DECW

Decrement Word
Word Data Decrement

[Instruction format] **DECW dst**

[Operation] **dst <- dst – 1**

[Operands]

Mnemonic	Operands (dst)
DECW	rp

[Flags]

Z	AC	CY

[Description]

- Decrements the contents of the destination operand (dst) by 1.
- Since this instruction is often used to decrement the register (pointer) in addressing which uses a register, the contents of the Z, AC and CY flags are not changed.

[Coding example]

DECW DE ; Decrements the DE register.

8.7 SHIFT/ROTATE INSTRUCTIONS

Shift/rotate instructions are as follows:

ROR

Rotate Right
Byte Data Right Rotation

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

[Operation] **(CY, dst_7 <- dst_0, dst_{m-1} <- dst_m) x cnt times cnt = 0 to 7**

[Operands]

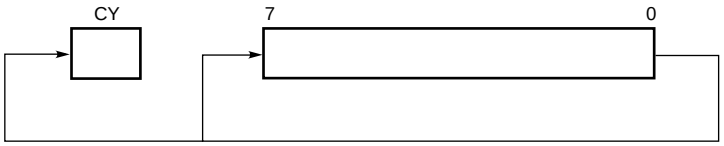
Mnemonic	Operands (dst, cnt)
ROR	r, n

[Flags]

Z	AC	CY
		x

[Description]

- Rotates to the right the contents of the destination operand (dst) specified by the 1st operand cnt times as specified by the 2nd operand.
- The contents of the LSB (bit 0) are rotated into the MSB (bit 7) and at the same time transferred to the CY flag.
- * If the 2nd operand (cnt) is 0, no processing is performed. (Z, AC, and CY flags also do not change.)



[Coding example]

ROR C, 4 ; Rotates the contents of the C register 4 bits to the right.

ROL

Rotate Left
Byte Data Left Rotation

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

[Operation] **(CY, dst_0 <- dst_7, dst_{m+1} <- dst_m) x cnt times cnt = 0 to 7**

[Operands]

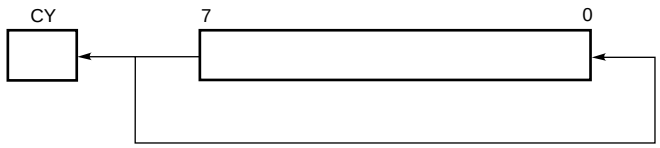
Mnemonic	Operands (dst, cnt)
ROL	r, n

[Flags]

Z	AC	CY
		x

[Description]

- Rotates to the left the contents of the destination operand (dst) specified by the 1st operand cnt times as specified by the 2nd operand.
- The contents of the MSB (bit 7) are rotated into the LSB (bit 0) and at the same time transferred to the CY flag.
- If the 2nd operand (cnt) is 0, no processing is performed.



*

[Coding example]

ROL L, 2 ; Rotates the contents of the L register 2 bits to the left.

RORC

Rotate Right with Carry
Byte Data Right Rotation including Carry

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

[Operation] **(CY <- dst_0, dst_7 <- CY, dst_{m-1} <- dst_m) x cnt times cnt = 0 to 7**

[Operands]

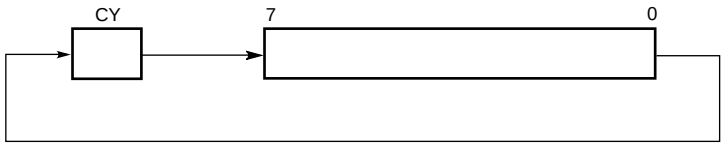
Mnemonic	Operands (dst, cnt)
RORC	r, n

[Flags]

Z	AC	CY
		x

[Description]

- Rotates to the right the contents of the destination operand (dst) specified by the 1st operand, including the CY flag, cnt times as specified by the 2nd operand.
- * If the 2nd operand (cnt) is 0, no processing is performed. (Z, AC, and CY flags also do not change.)



[Coding example]

RORC B, 1 ; Rotates the contents of the B register 1 bit to the right, including the CY flag.

ROLC

Rotate Left with Carry
Byte Data Left Rotation including Carry

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

[Operation] **(CY <- dst_7, dst_0 <- CY, dst_{m+1} <- dst_m) x cnt times cnt = 0 to 7**

[Operands]

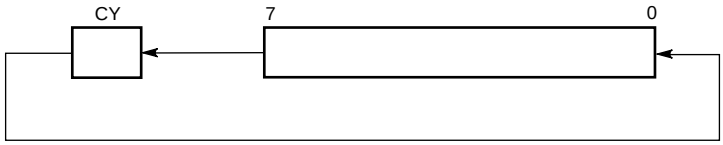
Mnemonic	Operands (dst, cnt)
ROLC	r, n

[Flags]

Z	AC	CY
		x

[Description]

- Rotates to the left the contents of the destination operand (dst) specified by the 1st operand, including the CY flag, cnt times as specified by the 2nd operand.
- If the 2nd operand (cnt) is 0, no processing is performed. (Z, AC, and CY flags also do not change.) *
- To perform a one-bit left rotation, execution time can be reduced by using ADDC r, r.



[Coding example]

ROLC A, 3 ; Rotates the contents of the A register 3 bits to the left, including the CY flag.

SHR

Shift Right (Logical)
Byte Data Logical Right Shift

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

[Operation] **(CY <- dst_0, dst_7 <- 0, dst_{m-1} <- dst_m) x cnt times cnt = 0 to 7**

[Operands]

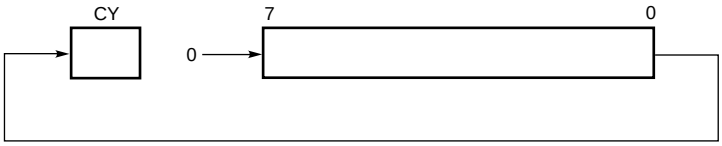
Mnemonic	Operands (dst, cnt)
SHR	r, n

[Flags]

Z	AC	CY
x	0	x

[Description]

- Shifts to the right the contents of the destination operand (dst) specified by the 1st operand cnt times as specified by the 2nd operand.
  0 is shifted into the MSB (bit 7) each time 1 bit is shifted.
- If the result of the shift operation is 0 the Z flag is set (1), otherwise the Z flag is cleared (0).
- The AC flag is always 0 regardless of the result of the shift operation.
- The final data shifted out of the LSB (bit 0) as a result of the shift operation is set in the CY flag.
- * If the second operand (cnt) is 0, no processing is performed. (Z, AC, and CY flags also do not change.)
- This instruction gives the same result as dividing the destination operand (dst) by 2^{cnt}.



[Coding example]

SHR H, 2 ; Shifts the contents of the H register 2 bits to the right.

SHL

Shift Left (Logical)
Byte Data Logical Left Shift

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

[Operation] **(CY <- dst_7, dst_0 <- 0, dst_{m+1} <- dst_m) x cnt times cnt = 0 to 7**

[Operands]

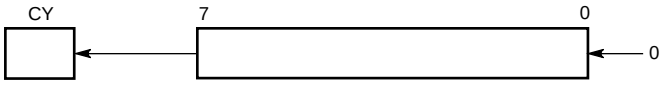
Mnemonic	Operands (dst, cnt)
SHL	r, n

[Flags]

Z	AC	CY
x	0	x

[Description]

- Shifts to the left the contents of the destination operand (dst) specified by the 1st operand cnt times as specified by the 2nd operand.
  0 is shifted into the LSB (bit 0) each time 1 bit is shifted.
- If the result of the shift operation is 0 the Z flag is set (1), otherwise the Z flag is cleared (0).
- The AC flag is always 0 regardless of the result of the shift operation.
- The final data shifted out of the MSB (bit 7) as a result of the shift operation is set in the CY flag.
- If the second operand (cnt) is 0, no processing is performed. (Z, AC, and CY flags also do not change.) *
  To perform a one-bit left shift, execution time can be reduced by using ADD r, r.



[Coding example]

SHL E, 1 ; Shifts the contents of the E register 1 bit to the left.

SHRW

Shift Right (Logical) Word
Word Data Logical Right Shift

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

[Operation] **(CY <- dst_0, dst_{15} <- 0, dst_{m-1} <- dst_m) x cnt times cnt = 0 to 7**

[Operands]

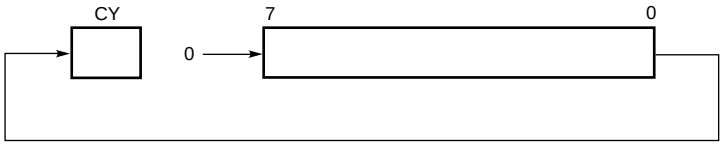
Mnemonic	Operands (dst, cnt)
SHRW	rp, n

[Flags]

Z	AC	CY
x	0	x

[Description]

- Shifts to the right the contents of the destination operand (dst) specified by the 1st operand cnt times as specified by the 2nd operand.
  0 is shifted into the MSB (bit 15) each time 1 bit is shifted.
- If the result of the shift operation is 0 the Z flag is set (1), otherwise the Z flag is cleared (0).
- The AC flag is always 0 regardless of the result of the shift operation.
- The final data shifted out of the LSB (bit 0) as a result of the shift operation is set in the CY flag.
- * If the second operand (cnt) is 0, no processing is performed. (Z, AC, and CY flags also do not change.)
  This instruction gives the same result is dividing the destination operand (dst) by 2^{cnt}.



[Coding example]

SHRW AX, 3 ; Shifts the contents of the AX register 3 bits to the right (divides the AX register contents by 8).

SHLW

Shift Left (Logical) Word
Word Data Logical Left Shift

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

[Operation] **(CY <- dst_{15}, dst_0 <- 0, dst_{m+1} <- dst_m) x cnt times cnt = 0 to 7**

[Operands]

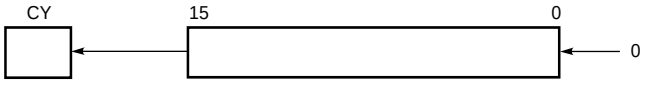
Mnemonic	Operands (dst, cnt)
SHLW	rp, n

[Flags]

Z	AC	CY
x	0	x

[Description]

- Shifts to the left the contents of the destination operand (dst) specified by the 1st operand cnt times as specified by the 2nd operand.
  0 is shifted into the LSB (bit 0) each time 1 bit is shifted.
- If the result of the shift operation is 0 the Z flag is set (1), otherwise the Z flag is cleared (0).
- The AC flag is always 0 regardless of the result of the shift operation.
- The final data shifted out of the MSB (bit 15) as a result of the shift operation is set in the CY flag.
- If the second operand (cnt) is 0, no processing is performed. (Z, AC, and CY flags also do not change.) *



[Coding example]

SHLW E, 1 ; Shifts the contents of the E register 1 bit to the left.

ROR4

Rotate Right Digit
Digit Right Rotation

[Instruction format] **ROR4 dst**

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

[Operands]

Mnemonic	Operands (dst)
ROR4[Note]	mem1
	&mem1

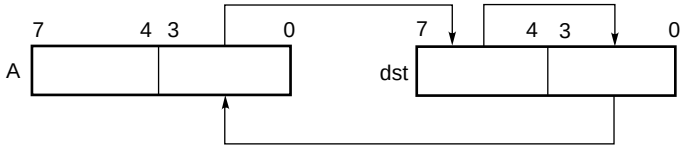
Note Cannot be used on μPD78244 sub-series EEPROM area.

[Flags]

Z	AC	CY

[Description]

- Rotates the low-order 4 bits of the A register and the destination operand (dst) 2-digit digit data (4-bit data) to the right.
  The high-order 4 bits of the A register are not changed.



[Coding example]

ROR4[HL] ; Performs digit rotation to the right of the memory contents specified by the A and HL registers.

	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 / Digit Left Rotation

[Instruction format] **ROL4 dst**

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

[Operands]

Mnemonic	Operands (dst)
ROL4[Note]	mem1
	&mem1

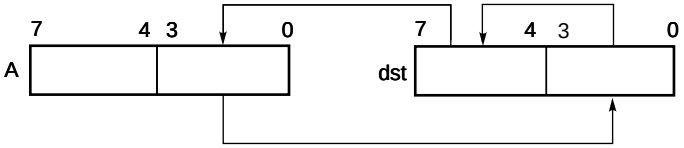
Note Cannot be used on μPD78244 sub-series EEPROM area.

[Flags]

Z	AC	CY

[Description]

- Rotates the low-order 4 bits of the A register and the destination operand (dst) 2-digit data (4-bit data) to the left.
  The high-order 4 bits of the A register are not changed.



[Coding example]

ROL4[DE+54H] ; Performs digit rotation to the left of the A register and the memory contents of address (DE register contents + 54H).

	A		(DE + 54H)	
	7 4 3 0		7 4 3 0	
Before execution	0001	0010	0100	1000
After execution	0001	0100	1000	0010

8.8 BCD ADJUSTMENT INSTRUCTIONS

BCD adjustment instructions are as follows:

ADJBA

Decimal Adjust Register for Addition
Decimal Adjustment of Addition Result

[Instruction format] **ADJBA**

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

[Operands]

None

[Flags]

Z	AC	CY
x	x	x

[Description]

- Performs A register, CY flag and AC flag decimal adjustment from the contents of the A register, CY flag and AC flag. The operation of this instruction is meaningful only when the result is stored in the A register after addition of BCD (binary-coded decimal) format data. (In other case, a meaningless operation is performed.) The adjustment method is shown in the table below.
- If the contents of the A register are 0 as a result of the adjustment the Z flag is set (1), otherwise the Z flag is cleared (0).

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

ADJBS

Decimal Adjust Register for Subtraction
Decimal Adjustment of Subtraction Result

[Instruction format] **ADJBS**

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

[Operands]

None

[Flags]

Z	AC	CY
x	x	x

[Description]

- Performs A register, CY flag and AC flag decimal adjustment from the contents of the A register, CY flag and AC flag. The operation of this instruction is meaningful only when the result is stored in the A register after subtraction of BCD (binary-coded decimal) format data. (In other cases, a meaningless operation is performed.) The adjustment method is shown in the table below.
- If the contents of the A register are 0 as a result of the adjustment the Z flag is set (1), otherwise the Z flag is cleared (0).

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

8.9 BIT MANIPULATION INSTRUCTIONS

Bit manipulation instructions are as follows:

MOV1

Move Single Bit
1-Bit Data Transfer

[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, PSW.bit
	saddr.bit, CY
	sfr.bit, CY
	A.bit, CY
	X.bit, CY
	PSW.bit, CY

[Flags]

dst = CY

Z	AC	CY
		x

PSW.bit

Z	AC	CY
x	x	x

Other than cases at left

Z	AC	CY

[Description]

- Transfers the bit data of the source operand (src) specified by the 2nd operand to the destination operand (dst) specified by the 1st operand.
- If the destination operand (dst) is CY or PSW.bit, only the relevant flag is changed.

[Coding example]

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

AND1

And Single Bit 1-Bit Data Logical Product

[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, sfr.bit
	CY, A.bit
	CY, X.bit
	CY, PSW.bit

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

[Flags]

Z	AC	CY
		x

[Description]

- Obtains the logical product of the bit data of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand, and stores the result in the destination operand (dst).
- If the 2nd operand is prefixed with "/", the logical product after logical negation of the source operand (src) is obtained.
- The result of the operation is stores in the CY flag (since it is the destination operand (dst)).

[Coding example]

AND1, CY, 0FE7FH.3; Obtains the logical product of bit 3 of 0FE7FH and the CY flag, and stores the result in the CY flag.

AND1 CY, /PSW.6 ; Obtains the logical product of the logical negation of PSW bit 6 (the Z flag) and the CY flag, and stores the result in the CY flag.

OR1

Or Single Bit
1-Bit Data Logical Sum

[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, sfr.bit
	CY, A.bit
	CY, X.bit
	CY, PSW.bit

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

[Flags]

Z	AC	CY
		x

[Description]

- Obtains the logical sum of the bit data of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand, and stores the result in the destination operand (dst).
- If the 2nd operand is prefixed with "/", the logical sum after logical negation of the source operand (src) is obtained.
- The result of the operation is stored in the CY flag (since it is the destination operand (dst)).

[Coding example]

OR1, CY, P2.5 ; Obtains the logical sum of bit 5 of port 2 and the CY flag, and stores the result in the CY flag.

OR1 CY, /X.0 ; Obtains the logical sum of the logical negation of bit 0 of the X register and the CY flag, and stores the result in the CY flag.

XOR1

Exclusive Or Single Bit
1-Bit Data Exclusive Logical Sum

[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, PSW.bit

[Flags]

Z	AC	CY
		x

[Description]

- Obtains the exclusive logical sum of the bit data of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand, and stores the result in the destination operand (dst).
- The result of the operation is stored in the CY flag (since it is the destination operand (dst)).

[Coding example]

XOR1, CY, A.7 ; Obtains the exclusive logical sum of bit 7 of the A register and the CY flag, and stores the result in the CY flag.

SET1

Set Single Bit (Carry Flag) 1-Bit Data Setting

[Instruction format] **SET1 dst**

[Operation] **dst <- 1**

[Operands]

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

[Flags]

dst = PSW.bit

Z	AC	CY
x	x	x

dst = CY

Z	AC	CY
		1

Other than cases at left

Z	AC	CY

[Description]

- Sets (1) the destination operand (dst).
- If the destination operand (dst) is CY or PSW.bit, only the relevant flag is set (1).

[Coding example]

SET1 0FE55H.1 ; Sets (1) bit 1 of 0FE55H.

CLR1

Not Single Bit (Carry Flag) 1-Bit Data Clear

[Instruction format] **CLR1 dst**

[Operation] **dst <- 0**

[Operands]

Mnemonic	Operands (dst)
CLR1	saddr.bit
	sfr.bit
	A.bit
	X.bit
	PSW.bit
	CY

[Flags]

dst = PSW.bit

Z	AC	CY
x	x	x

dst = CY

Z	AC	CY
		0

Other than cases at left

Z	AC	CY

[Description]

- Clears (0) the destination operand (dst).
- If the destination operand (dst) is CY or PSW.bit, only the relevant flag is cleared (0).

[Coding example]

CLR1 P3.7 ; Clears (0) bit 7 of port 3.

NOT1

Clear Single Bit (Carry Flag)
1-Bit Data Clear

[Instruction format] **NOT1 dst**

[Operation] **dst <- $\overline{\text{dst}}$**

[Operands]

Mnemonic	Operands (dst)
NOT1	saddr.bit
	sfr.bit
	A.bit
	X.bit
	PSW.bit
	CY

[Flags]

dst = PSW.bit

Z	AC	CY
x	x	x

dst = CY

Z	AC	CY
		x

Other than cases at left

Z	AC	CY

[Description]

- Obtains the logical negation of the bit specified by the destination operand (dst), and stores the result in the destination operand (dst).
- If the destination operand (dst) is CY or PSW.bit, only the relevant flag is changed.

[Coding example]

NOT1 A.2 ; Inverts bit 2 of the A register.

8.10 CALL/RETURN INSTRUCTIONS

Call/return instructions are as follows:

CALL — Call
Subroutine Call (16-Bit Direct/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 instruction bytes

[Operands]

Mnemonic	Operands (target)
CALL	!addr16
	rp

[Flags]

Z	AC	CY

[Description]

- Makes a subroutine call using a 16-bit absolute address or register indirect address.
- The start address of the next instruction (PC + n) is saved to the stack, and a branch is made to the address specified by the target operand (target).

[Coding example]

CALL !3059H ; Makes a subroutine call to 03059H.

CALLF

Call Flag
Subroutine Call (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	Operands (target)
CALLF	!addr11

[Flags]

Z	AC	CY

[Description]

- This subroutine call can branch only to an address in the range 00800H to 00FFFH.
- The start address of the next instruction (PC + 2) is saved to the stack, and a branch is made to an address in the range 00800H to 00FFFH.
- Only the low-order 11 address bits are specified (the high-order 5 bits are fixed at 00001B).
- Locating a subroutine in the area from 00800H to 00FFFH and using this instruction enables the program size to be reduced. The execution time is also reduced when there is a program in external memory.

[Coding example]

CALLF !0C2AH ; 00C2AH subroutine call

CALLT

Call Table
Subroutine Call (Call Table Reference)

[Instruction format] **CALLT [addr5]**

[Operation]

(SP – 1) <- (PC + 1)$_H$,
(SP – 2) <- (PC + 1)$_L$,
SP <- SP – 2,
PC$_H$ <- (00000000001, addr5 + 1)
PC$_L$ <- (00000000001, addr5)

[Operands]

Mnemonic	Operands [addr5]
CALLT	[addr5]

[Flags]

Z	AC	CY

[Description]

- This is a call table reference subroutine call.
- The start address of the next instruction (PC + 1) is saved to the stack, and a branch is made to the address indicated by the word data in the call table (the high-order 10 bits of the address are fixed at 0000000001B, the next 5 bits are specified by addr5, and the LSB is fixed at 0).

[Coding example]

CALLT [60H] ; Subroutine call to the address indicated by the word data in addresses 00060H and 00061H.

BRK

Break
Software Vectored Interrupt

[Instruction format] **BRK**

[Operation]

(SP – 1) <- PSW,
(SP – 2) <- (PC + 1)$_H$,
(SP – 3) <- (PC + 1)$_L$,
IE <- 0,
SP <- SP – 4,
PC$_H$ <- (3FH),
PC$_L$ <- (3EH)

[Operands]

None

[Flags]

Z	AC	CY

[Description]

- The software interrupt instruction.
- The PSW and the address of the next instruction (PC + 1) are saved to the stack, then the IE flag is cleared (0) and a branch is made to the address indicated by the vector address (0003EH) word data.
  As the IE flag is cleared (0), subsequent maskable vectored interrupts are disabled.
- Return from a software vectored interrupt generated by this instruction is performed by the RETB instruction.

RET

Return
Return from Subroutine

[Instruction format] **RET**

[Operation]

PC_L <- (SP),
PC_H <- (SP + 1),
SP <- SP + 2

[Operands]

None

[Flags]

Z	AC	CY

[Description]

- This instruction is used to return from a subroutine called by the CALL, CALLF or CALLT instruction.
- The word data saved to the stack is restored to the PC, and a return is made from the subroutine.

RETI

Return from Interrupt
Return from Hardware Vectored Interrupt

[Instruction format] **RETI**

[Operation]

PC_L <- (SP),
PC_H <- (SP + 1),
PSW <- (SP + 2),
SP <- SP + 3

[Operands]

None

[Flags]

Z	AC	CY
R	R	R

[Description]

- This instruction is used to return from a vectored interrupt.
- The data saved to the stack is restored to the PC and PSW, the NMIS flag in the IST register is cleared (0), and a return is made from the interrupt service routine.
- This instruction cannot be used to return from a software interrupt generated by the BRK instruction.
- No interrupts or macro services are acknowledged between this instruction and the next instruction to be executed.

RETB

Return from Break
Return from Software Vectored Interrupt

[Instruction format] **RETB**

[Operation]

PC_L <- (SP),
PC_H <- (SP + 1),
PSW <- (SP + 2),
SP <- SP + 3

[Operands]

None

[Flags]

Z	AC	CY
R	R	R

[Description]

- This instruction is used to return from a software interrupt generated by the BRK instruction..
- The PC and PSW saved to the stack are restored, and a return is made from the interrupt service routine.
- No interrupts or macro services are acknowledged between this instruction and the next instruction to be executed.

8.11 STACK MANIPULATION INSTRUCTIONS

Stack manipulation instructions are as follows:

PUSH

Push
Push

[Instruction format] **PUSH src**

[Operation]

When src = rp

(SP – 1) <- src_H,
(SP – 2) <- src_L,
SP <- SP – 2

When src = PSW or sfr

(SP – 1) <- src
SP <- SP – 1

[Operands]

Mnemonic	Operands (src)
PUSH	PSW
	sfr
	rp

[Flags]

Z	AC	CY

[Description]

- Saves the data in the register specified by the source operand (src) to the stack.

[Coding example]

PUSH AX ; Saves the contents of the AX register to the stack.

POP

Pop
Pop

[Instruction format] **POP dst**

[Operation]

When dst = rp

dst_L <- (SP),
dst_H <- (SP + 1),
SP <- SP + 2

When dst = PSW or sfr

dst <- (SP)
SP <- SP + 1

[Operands]

Mnemonic	Operands (src)
POP	PSW
	sfr
	rp

[Flags]

src = PSW

Z	AC	CY
R	R	R

Other than cases at left

Z	AC	CY

[Description]

- Restores data from the stack to the register specified by the destination operand (dst).
- If the operand is the PSW, each flag is replaced with stack data.
- No interrupts or macro services are acknowledged between the POP PSW instruction and the next instruction.

[Coding example]

POP IMK0L ; Restores stack data to the IMK0L register.

MOVW SP, src
MOVW AX, SP

Move Word
Stack Pointer/Word Data Transfer

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

[Operation] **dst <- src**

[Operands]

Mnemonic	Operands (dst, src)
MOVW	SP, #word
	SP, AX
	AX, SP

[Flags]

Z	AC	CY

[Description]

- These instructions are used to manipulate the contents of the stack pointer.
- Stores the source operand (src) specified by the 2nd operand in the destination operand (dst) specified by the 1st operand.

[Coding example]

MOVW PS, #0FE1FH ; Stores 0FE1FH in the stack pointer.

INCW SP

Increment Word
Stack Pointer Increment

[Instruction format] **INCW SP**

[Operation] **SP <- SP + 1**

[Operands]

None

[Flags]

Z	AC	CY

[Description]

- Increments the contents of the SP (stack pointer) by 1.

DECW SP

Decrement Word
Stack Pointer Decrement

[Instruction format] **DECW SP**

[Operation] **SP <- SP – 1**

[Operands]

None

[Flags]

Z	AC	CY

[Description]

- Decrements the contents of the SP (stack pointer) by 1.

8.12 UNCONDITIONAL BRANCH INSTRUCTIONS

The unconditional branch instruction is as follows:

```
BR  ...  206
```

BR

Branch
Unconditional Branch

[Instruction format] **BR target**

[Operation] **PC <- target**

[Operands]

Mnemonic	Operands (target)
BR	!addr16
	rp
	$addr16

[Flags]

Z	AC	CY

[Description]

- Performs an unconditional branch.
- Transfers the target address operand (target) word data to the PC, then branches.

[Coding example]

BR AX ; Branches using the contents of the AX register as the branch address.

8.13 CONDITIONAL BRANCH INSTRUCTIONS

Conditional branch instructions are as follows:

BC / BL — Branch if Carry/Less than — Conditional Branch depending on Carry Flag (CY = 1)

[Instruction format] **BC $addr16**
BL $addr16

[Operation] **PC <- PC + 2 + jdisp8 if CY = 1**

[Operands]

Mnemonic	Operands ($addr16)
BC	$addr16
BL	

[Flags]

Z	AC	CY

[Description]

- When CY = 1, branches to the address specified by the operand.
  When CY = 0, no processing is performed and the next instruction is executed.
- The operation of the BC instruction and the BL instruction is the same. Differences in their use are as follows:
  - BC instruction : Checks whether a carry has been generated after an addition instruction. Determines the result of bit manipulation.
  - BL instruction : Checks whether a borrow has been generated after a subtraction instruction. After a compare instruction, checks whether or not the 1st operand of the compare instruction is smaller.

[Coding example]

BC $300H ; Branches to 00300H if CY = 1 (the start of this instruction is in the address range from 0027FH to 0037EH).

BNC / BNL — Branch if Not Carry/Less than — Conditional Branch depending on Carry Flag (CY = 0)

[Instruction format] **BNC $addr16**
BNL $addr16

[Operation] **PC <- PC + 2 + jdisp8 if CY = 0**

[Operands]

Mnemonic	Operands ($addr16)
BNC / **BNL**	$addr16

[Flags]

Z	AC	CY

[Description]

- When CY = 0, branches to the address specified by the operand.
  When CY = 1, no processing is performed and the next instruction is executed.
- The operation of the BNC instruction and the BNL instruction is the same. Differences in their use are as follows:
  - BNC instruction : Checks whether a carry has been generated after an addition instruction. Determines the result of bit manipulation.
  - BNL instruction : Checks whether a borrow has been generated after a subtraction instruction. After a compare instruction, checks whether or not the 1st operand of the compare instruction is smaller.

[Coding examples]

```
CMP A, B      ; Compares the size of the A register contents and B register contents.
BNL #1500H    ; Branches to 01500H if the contents of the A register are not smaller than the contents
              ; of the B register (the start of this instruction is in the address range from 0147FH to
              ; 0157EH).
```

BZ / BE — Branch if Zero/Equal — Conditional Branch depending on Zero Flag (Z = 1)

[Instruction format] **BZ $addr16**
BE $addr16

[Operation] **PC <- PC + 2 + jdisp8 if Z = 1**

[Operands]

Mnemonic	Operands ($addr16)
BZ	$addr16
BE	

[Flags]

Z	AC	CY

[Description]

- When Z = 1, branches to the address specified by the operand.
  When Z = 0, no processing is performed and the next instruction is executed.
- The operation of the BZ instruction and the BE instruction is the same. Differences in their use are as follows:
  - BZ instruction: Checks whether the result of an addition, subtraction or increment/decrement instruction or an 8-bit logical operation is 0.
  - BE instruction: Checks for a match after a compare instruction.

[Coding example]

```
DEC B
BZ $3C5H ; Branches to 003C5H if the B register contents are 0 (the start of this instruction is in the
           address range from 00344H to 00443H).
```

BNZ / BNE — Branch if Not Zero/Not Equal — Conditional Branch depending on Zero Flag (Z = 0)

[Instruction format] **BNZ $addr16**
BNE $addr16

[Operation] **PC <- PC + 2 + jdisp8 if Z = 0**

[Operands]

Mnemonic	Operands ($addr16)
BNZ	$addr16
BNE	

[Flags]

Z	AC	CY

[Description]

- When Z = 0, branches to the address specified by the operand.
  When Z = 1, no processing is performed and the next instruction is executed.
- The operation of the BNZ instruction is the same. Differences in their use are as follows:
  - BNZ instruction: Checks whether the result of an addition, subtraction or increment/decrement instruction or an 8-bit logical operation is 0.
  - BNE instruction: Checks for a match after a compare instruction.

[Coding example]

```
CMP A, #55H
BNE $0A39H ; Branches to 00A39H if the A register contents are not 0055H (the start of this instruction
             is in the address range from 009B8H to 00AB7H).
```

BT

Branch if True
Conditional Branch depending on 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 (Number of bytes)
BT	saddr.bit, $addr16	3
	sfr.bit, $addr16	4
	A.bit, $addr16	3
	X.bit, $addr16	3
	PSW.bit, $addr16	3

[Flags]

Z	AC	CY

[Description]

- If the contents of the 1st operand (bit) are set (1), branches to the address specified by the 2nd operand ($addr16).
  If the contents of the 1st operand (bit) are not set (1), no processing is performed and the next instruction is executed.

[Coding example]

BT 0FE47H.3, $55CH ; Branches to 0055CH if bit 3 of address 0FE47H is 1 (the start of this instruction is in the address range from 004DAH to 005D9H).

BF

Branch if False
Conditional Branch depending on 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 (Number of bytes)
BF	saddr.bit, $addr16	4
	sfr.bit, $addr16	4
	A.bit, $addr16	3
	X.bit, $addr16	3
	PSW.bit, $addr16	3

[Flags]

Z	AC	CY

[Description]

- If the contents of the 1st operand (bit) are cleared (0), branches to the address specified by the 2nd operand ($addr16).
  If the contents of the 1st operand (bit) are not cleared (0), no processing is performed and the next instruction is executed.

[Coding example]

BF P2.2, $1549H ; Branches to 01549H if bit 2 of port 2 is 0 (the start of this instruction is in the address range from 014C6H to 015C5H).

BTCLR

Branch if True and Clear
Conditional Branch and Clear depending on 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 (Number of bytes)
BTCLR	saddr.bit, $addr16	4
	sfr.bit, $addr16	4
	A.bit, $addr16	3
	X.bit, $addr16	3
	PSW.bit, $addr16	3

[Flags]

bit = PSW.bit

Z	AC	CY
x	x	x

Other than cases at left

Z	AC	CY

[Description]

- If the contents of the 1st operand (bit) are set (1), clears (0) the contents of the 1st operand (bit) and branches to the address specified by the 2nd operand.
  If the contents of the 1st operand (bit) are not set (1), no processing is performed and the next instruction is executed.
- If the 1st operand (bit) is PSW.bit, only the contents of the relevant flag are cleared (0).

[Coding example]

BTCLR PSW.0, $356H ; If PSW bit 0 (CY flag) is 1, clears the CY flag and branches to address 00356H (the start of this instruction is in the address range from 002D4H to 003D3H).

DBNZ

Decrement and Branch if Not Zero Conditional Loop (R1 ≠ 0)

[Instruction format] **DBNZ dst, $addr16**

[Operation] **dst <- dst – 1,**
then PC <- PC + b + jdisp16 if dst R1 ≠ 0

[Operands]

Mnemonic	Operands (bit, $addr16)	b (Number of bytes)
DBNZ	r1, ← $addr16	2
	saddr, $addr16	3

[Flags]

Z	AC	CY

[Description]

- Decrements by 1 the contents of the destination operand (dst) specified by the 1st operand, and branches to the destination operand (dst).
- If the result of decrementing the destination operand (dst) by 1 is not 0, the instruction branches to the address indicated by the 2nd operand ($addr16).
  If the result of decrementing the destination operand (dst) by 1 is 0, no processing is performed and the next instruction is executed.
- Flags are not changed.

[Coding example]

DBNZ B, $1215H ; Decrements the contents of the B register, and if the result is not 0, branches to 001215H (the start of this instruction is in the address range from 001194H to 001293H).

8.14 CPU CONTROL INSTRUCTIONS

CPU control instructions are as follows:

MOV STBC, #byte

Move
Standby Mode Setting

[Instruction format] **MOV STBC #byte**

[Operation] **STBC <- byte**

[Operands]

Mnemonic	Operands
MOV	STBC, #byte

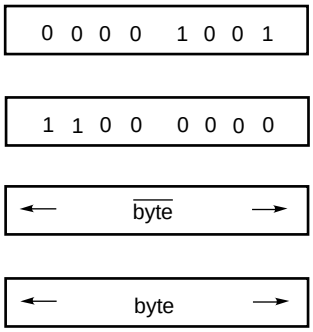
[Flags]

Z	AC	CY

[Description]

- This is a dedicated write instruction for the standby control register (STBC). This instruction writes the immediate data specified by the 2nd operand to STBC.
  Only this instruction can be used to write to the STBC register.
- This instruction has a special format, and in addition to the immediate data used in performing the write, the operation code must also provide data resulting from the logical negation of that value (see figure below).
  (This is generated automatically by NEC assemblers.)

  - Instruction code format



- The CPU checks the data obtained by logical negation of the immediate data to be written, and if correct, performs the write. If the data is incorrect, the write is not performed and the next instruction is executed.

[Coding example]

MOV STBC, #2 ; Writes 2 to STBC (sets STOP mode).

SEL RBn

Select Register Bank
Register Bank Selection

[Instruction format] **SEL RBn**

[Operation] **RBS0, RBS1 <- n; (n = 0 to 3)**

[Operands]

Mnemonic	Operands (RBn)
SEL	RBn

[Flags]

Z	AC	CY

[Description]

- Selects the register bank specified by the operand (RBn) as the register bank to be used by the next and subsequent instructions.
- RBn comprises RB0 to RB3.

[Coding example]

SEL RB2 ; Selects register bank 2 as the register bank to be used from the next instruction onward.

NOP

No Operation
No Operation

[Instruction format] **NOP**

[Operation] **no operation**

[Operands]

None

[Flags]

Z	AC	CY

[Description]

- Expends time without performing any processing.

EI

Enable interrupt
Interrupt Enabling

[Instruction format] **EI**

[Operation] **IE <- 1**

[Operands]

None

[Flags]

Z	AC	CY

[Description]

- Sets the acknowledgment enabled state for maskable interrupts (sets (1) the interrupt enable flag (IE)).
- No interrupts or macro service requests are acknowledged between this instruction and the next instruction.
- Acknowledgment of vectored interrupts from other sources can be disabled even when this instruction is executed. Refer to individual documentation for details.

DI

Disable interrupt
Interrupt Disabling

[Instruction format] **DI**

[Operation] **IE <- 0**

[Operands]

None

[Flags]

Z	AC	CY

[Description]

- Disables acknowledgment of vectored maskable interrupts (clears (0) the interrupt enable flag (IE)).
- No interrupts or macro service requests are acknowledged between this instruction and the next instruction.
- Refer to individual documentation for details of interrupt servicing.

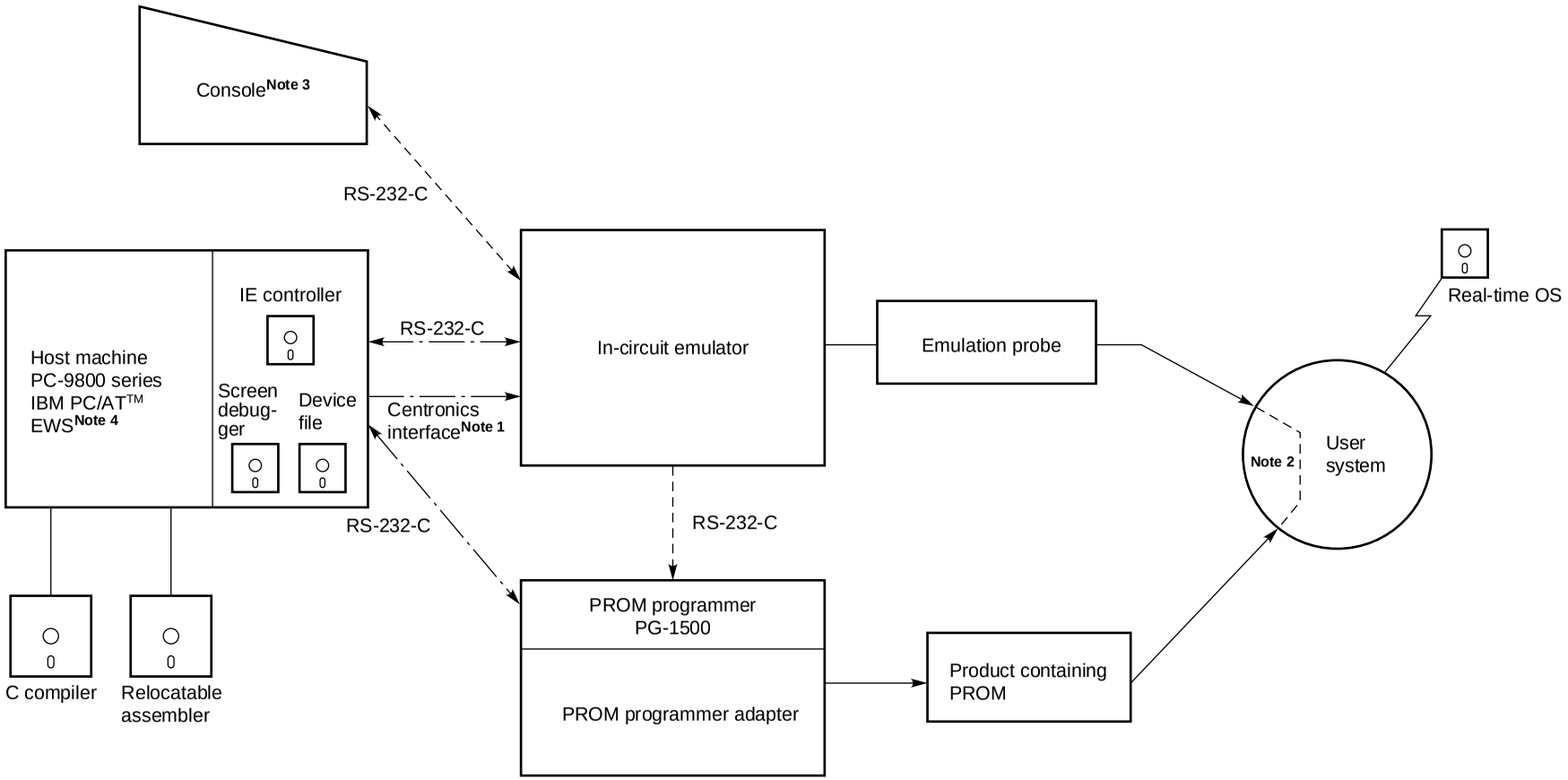
[MEMO]

CHAPTER 9 DEVELOPMENT TOOLS

9.1 DEVELOPMENT TOOLS

The tools required for 78K/II series product development are shown in Figure 9-1 and Tables 9-1 and 9-2.

Figure 9-1. Development Tool Configuration



—·—·—·—: When used connected to host machine

– – – – – – – – –: When IE is used as stand-alone device connected to console

Notes 1. Used for high-speed file transfer (downloading). (Cannot be used with IE-78210-R or IE-78220-R.)

2. Socket for connection of QFP emulation probe

3. Only when using IE-78230-R or IE-78240-R

4. EWS comprises the HP9000 series 700, SUN4/3900, EWS-4800/200 series. EWS cannot be connected with an in-circuit emulator.

[MEMO]

Table 9-1. Development Tools (for Screen Debugger)

Target device	Package	In-circuit emulator	Screen debugger	Device file	Emulation probe
μPD78214 subseries μPD78212Note 1 μPD78213 μPD78214	64SDIP	IE-78240-R-A	SD78K/II	DF78210	EP-78240CW-R
	64QUIP				EP-78240GQ-R
	64QFP (14 x 14 mm body)				EP-78240GC-R
	68QFJ				EP-78240LP-R
	74QFP (20 x 20 mm body)				EP-78240GJ-R
μPD78218A subseries μPD78217A μPD78218A μPD78P218A	64SDIP	IE-78240R-A		DF78210	EP-78240CW-R
	64QFP (14 x 14 mm body)				EP-78240GC-R
μPD78224 subseries μPD78220 μPD78224 μPD78P224	84QFJ	IE-78230-R-A		DF78220	EP-78230LQ-R
	64QFP (20 x 20 mm body)				EP-78230GJ-R
μPD78234 subseries μPD78233 μPD78234 μPD78237 μPD78238 μPD78P238	80QFP (14 x 14 mm body)	IE-78230-R-A		DF78230	EP-78230GC-R
	84QFJ				EP-78230LQ-R
	94QFP (20 x 20 mm body)				EP-78230GJ-R
	94WQFNNote 2				
μPD78244 subseries μPD78243 μPD78244	64SDIP	IE-78240-R-A		DF78240	EP-78240CW-R
	64QFP (14 x 14 mm body)				EP-78240GC-R

Notes 1. The μPD78212 package range comprises 64-pin SDIP, 64-pin QFP, and 74-pin QFP only.

2. μPD78P238 only

Conversion socket[Note]	PROM programmer adapter	Assembler	C compiler	C compiler library source file
–	PA-78P214CW	RA78K/II	CC78K/II	CC78K/II-L
	PA-78P214GQ			
EV-9200GC-64	PA-78P214GC			
–	PA-78P214L			
EV-9200G-74	PA-78P214GJ			
–	PA-78P214CW			
EV-9200GC-64	PA-78P214GC			
–	PA-78P224L			
EV-9200G-94	PA-78P224GJ			
EV-9200GC-80	PA-78P238GC			
–	PA-78P238LQ			
EV-9200G-94	PA-78P238GJ			
	PA-78P238KF			
–	–			
EV-9200GC-64				

Note Socket for connecting a QFP emulation probe to the user system. Mounted on the user system board for use.

Table 9-2. Development Tools (for In-Circuit Emulator Control Program[Note 1])

Target device	Package	In-circuit emulator[Note 1]	In-circuit emulator control program[Note 1]	Emulation probe
μPD78214 subseries μPD78212[Note 2] μPD78213 μPD78214 μPD78P214	64SDIP	IE-78210-R IE-78240-R	IE-78210[Note 4] IE-78240[Note 4]	EP-78210CW[Note 1] EP-78240CW-R
	64QUIP			EP-78210GQ[Note 1] EP-78240GQ-R
	64QFP (14 x 14 mm body)			EP-78210GC[Note 1] EP-78240GC-R
	68QFJ			EP-78210L[Note 1] EP-78240LP-R
	74QFP (20 x 20 mm body)			EP-78210GJ[Notes 1,6] EP-78240GJ-R
μPD78218A subseries μPD78217A μPD78218A μPD78P218A	64SDIP	IE-78240-R	IE-78240	EP-78210CW[Note 1] EP-78240CW-R
	64QFP (14 x 14 mm body)			EP-78210GC[Note 1] EP-78240GC-R
μPD78224 subseries μPD78220 μPD78224 μPD78P224	84QFJ	IE-78220-R IE-78230-R	IE-78220[Note 5] IE-78230[Note 5]	EP-78220L[Note 1] EP-78230LQ-R
	64QFP (20 x 20 mm body)			EP-78220GJ[Notes 1,7] EP-78230GJ-R
μPD78234 subseries μPD78233 μPD78234 μPD78237 μPD78238 μPD78P238	80QFP (14 x 14 mm body)	IE-78230-R	IE-78230	EP-78230GC-R
	84QFJ			EP-78230LQ-R
	94QFP (20 x 20 mm body)			EP-78230GJ-R
	94WQFN[Note 3]			
μPD78244 subseries μPD78243 μPD78244	64SDIP	IE-78240-R	IE-78240	EP-78210CW[Note 1] EP-78240CW-R
	64QFP (14 x 14 mm body)			EP-78210GC[Note 1] EP-78240GC-R

Notes 1. No longer manufactured and not available for purchase.

2. The μPD78212 package range comprises 64-pin SDIP, 64-pin QFP, and 74-pin QFP only.

3. μPD78P238 only

4. IE-78210-R and IE-78240-R require IE-78210 and IE-78240, respectively.

5. IE-78220-R and IE-78230-R require IE-78220 and IE-78230, respectively.

6. Requires EP-78210L or EP-78240LP-R.

7. Requires EP-78220L or EP-78230LQ-R.

Conversion socket[Note]	PROM programmer adapter	Assembler	C compiler	C compiler library source file
–	PA-78P214CW	RA78K/II	CC78K/II	CC78K/II-L
	PA-78P214GQ			
EV-9200GC-64	PA-78P214GC			
–	PA-78P214L			
EV-9200G-74	PA-78P214GJ			
–	PA-78P214CW			
EV-9200GC-64	PA-78P214GC			
–	PA-78P224L			
EV-9200G-94	PA-78P224GJ			
EV-9200GC-80	PA-78P238GC			
–	PA78P238LQ			
EV-9200G-94	PA-78P238GJ			
	PA-78P238KF			
–	–			
EV-9200GC-64				

Note Socket for connecting a QFP emulation probe to the user system. Mounted on the user system board for use.

9.2 OUTLINE OF TOOLS

9.2.1 Hardware

(1) Relevant to in-circuit emulator (1/3)

IE-78240-R-A
IE-78230-R-A

These are function enhanced versions of the previous 78K/II series in-circuit emulator. The target device of each emulator is shown below.
When PC-9800 series or IBM PC/AT is used as a host machine, these can be used.
These require the screen debugger and device file separately sold and allow debugging in the level of source program such as C language or structured assembly language by use in combination with them.
Simultaneous trace of data access and program fetch and C0 coverage function enables efficient debugging and program test.
IE-78210-R or IE-78240-R can be used in the same way as IE-78240-R-A by purchasing the board separately sold (IE-78200-R-BK), and IE-78220-R or IE-78230-R can be used in the same way as IE-78230-R-A by purchasing the board separately sold (IE-78200-R-BK).

In-circuit emulator	Target devices
IE-78240-R-A	μPD78214 sub-series, μPD78218A sub-series, μPD78244 sub-series
IE78230-R-A	μPD78224 sub-series, μPD78234 sub-series

IE-78240-R**Note**
IE-78210-R**Note**
IE-78230-R**Note**
IE-78220-R**Note**

In-circuit emulators for the 78K/II series. The target devices of each emulator are shown below. Purchase of the separately available emulation board allows conversion to a different 78K/II series in-circuit emulator.
The in-circuit emulator is connected to a host machine or console to perform debugging. Connecting the in-circuit emulator to a host machine allows symbolic debugging and object file exchange with the host machine, enabling highly efficient debugging to be performed. Two RS-232-C serial interface channels are incorporated, allowing the connection of a PG-1500 PROM programmer. The IE-78230-R and IE-78240-R also incorporate a Centronics interface, allowing high-speed object file and symbol file downloading.

In-circuit emulator	Target devices
IE-78210-R**Note**	μPD78214 sub-series
IE78220-R**Note**	μPD78224 sub-series
IE-78230-R**Note**	μPD78224 sub-series, μPD78234 sub-series
IE-78240-R**Note**	μPD78214 sub-series, μPD78218A sub-series, μPD78244 sub-series

(Continued)

Note No longer manufactured and not available for purchase.

(1) Relevant to in-circuit emulator (2/3)

IE-78240-R-EM IE-78210-R-EM**Note** IE-78230-R-EM IE-78220-R-EKM**Note** IE-78200-R-EMK**Note** IE-782K00-R-BK	Boards enabling a 75X series or 78K series in-circuit emulator to upgrade to another 78K/II series in-circuit emulator. For details, see **Section 9.3**.

Emulation board	Target devices
IE-78210-R-EM**Note**	μPD78214 sub-series
IE-78220-R-EM**Note**	μPD78224 sub-series
IE-78230-R-EM	μPD78224 sub-series, μPD78234 sub-series
IE-78240-R-EM	μPD78214 sub-series, μPD78218A sub-series, μPD78244 sub-series

EP-78210CW**Note** EP-78240CW-R	Emulation probes for μPD78214 sub-series, μPD78218A sub-series, and μPD78244 sub-series 64-pin shrink DIP. The EP-78240CW-R is a long-cabled version of the EP-78210CW.
EP-78210GC**Note** EP-78240GC-R	Emulation probes for μPD78214 sub-series, μPD78218A sub-series, and μPD78244 sub-series for 64-pin QFP. Used together with the EV-9200GC-64. The EP-78240GC-R is a long-cabled version of the EP-78210GC.
EP-78210GJ**Note**	Socket adapter for μPD78214 sub-series 74-pin QFP. Used together with the EP-78210L or EP-78240LP-R and the EV-9200G-74.
EP-78240GJ-R	Emulation probe for μPD78214 sub-series 74-pin QFP. Used together with the EV-9200G-74. Unlike the EP-78210GJ, this is a stand-alone probe and is easy to handle.
EP-78210GQ**Note** EP-78240GQ-R	Emulation probes for μPD78214 sub-series 64-pin QUIP. The EP-78240GQ-R is a long-cabled version of the EP-78210GQ.
EP-78210L**Note** EP-78240LP-R	Emulation probes for μPD78214 sub-series 64-pin plastic QFJ. The EP-78240LP-R is a long-cabled version of the EP-78210L.
EP-78220GJ	Socket adapter for μPD78224 sub-series 94-pin QFP. Used together with the EP-78220L or EP-78230LQ-R and the EV-9200G-94.
EP-78230GJ-R	Emulation probe for μPD78224 sub-series and μPD78234 sub-series 94-pin QFP. Used together with the EV-9200G-94. Unlike the EP-78220GJ, this is a stand-alone probe and is easy to handle.
EP-78220L**Note**	Emulation probe for μPD78224 sub-series and μPD78234 sub-series 84-pin plastic QFJ. No new orders are currently being accepted. The EP-78230LQ-R should be ordered instead.

(Continued)

Note No longer manufactured and not available for purchase.

(1) Relevant to in-circuit emulator (3/3)

EP-78230LQ-R	Emulation probe for μPD78224 sub-series and μPD78234 sub-series 84-pin plastic QFJ. The EP-78230LQ-R is a long-cabled version of the EP-78220L for use with the μPD78234 sub-series.
EP-78230GC-R	Emulation probe for μPD78234 sub-series 80-pin QFP. Use together with the EV-9200GC-80.
EV-9200GC-74	Socket mounted on user system board for 74-pin QFP use. Used together with EP-78210GJ or EP-78240GJ-R.
EV-9200GC-80	Socket mounted on user system board for 80-pin QFP use. Used together with EP-78230GC-R.
EV-9200G-94	Socket mounted on user system board for 94-pin QFP use. Used together with EP-78220GJ or EP-78230GJ-R.
EV-9200GC-64	Socket mounted on user system board for 64-pin QFP use. Used together with EP-78210GC or EP-78240GC-R.
EV-9900	A jig used to remove the μPD78P238KF from the EV-9200G-94. A pincer may be a substitute. The use of the EV-9900 facilitates the work. The use of two EV-9900 facilitates the work even more.

Remarks 1. One EV-9200G-74 and EV-9200GC-64 are provided with the EP-78210GJ, EP-78210GC, EP-78240GC-R, and EP-78240GJ-R.

2. One EV-9200G-94, and EV-9200GC-80 are provided with the EP-78220GJ, EP-78230GJ-R, and EP-78230GC-R.

3. The EV-9200G-74, EV-9200GC-64, EV-9200G-94, and EV-9200GC-80 are sold in sets of five. (ordered in units of a set.)

(2) PROM write tools

PG-1500	PROM programmer which enables an on-chip PROM single-chip microcomputer to be programmed by stand-alone or manipulation from the host machine connecting with an optional board and a programmer adapter separately sold. And typical PROMs of 256K bits to 4M bits are programmable.
PA-78P214CW	PROM adapter for μPD78P214CW, 78P214DW, 78P218ACW, and 78P218ADW, used in combination with PG-1500, etc.
PA-78P214GC	PROM programmer adapter for μPD78P214GC-AB8 and 78P218AGC-ABE, used in combination with PG-1500, etc.
PA-78P214GJ	PROM programmer adapter for μPD78P214GJ-5BJ, used in combination with PG-1500, etc.
PA-78P214GQ	PROM programmer adapter for μPD78P214GQ-36, used in combination with PG-1500, etc.
PA-78P214L	PROM programmer adapter for μPD78P214L, used in combination with PG-1500, etc.
PA-78P224GJ	PROM programmer adapter for μPD78P224GJ-5BG, used in combination with PG-1500, etc.
PA-78P224L	PROM programmer adapter for μPD78P224L, used in combination with PG-1500, etc.
PA-78P238GC	PROM programmer adapter for μPD78P238GC-3B9, used in combination with PG-1500, etc.
PA78P238GJ	PROM programmer adapter for μPD78P238GJ-5BG, used in combination with PG-1500, etc.
PA-78P238KF	PROM programmer adapter for μPD78P238KF, used in combination with PG-1500, etc.
PA-78P238LQ	PROM programmer adapter for μPD78P238LQ, used in combination with PG-1500, etc.

9.2.2 Software

(1) Language processing software (1/3)

78K/II series relocatable assembler (RA78K/II)	This relocatable assembler can be used for all the 78K/II series products. Its macro functions enhance efficiency in software development. It also includes a structured assembler, which makes the program control structure more comprehensive, thus improving software productivity and maintainability. The relocatable assembler consists of the following programs:

Structured assembler preprocessor (program name: ST78K2)	Converts a source program written in the structured assembler language into a form that can be input to the relocatable assembler.
Relocatable assembler (program name: RA78K2)	Converts a source program written in assembly language into a machine language program, enabling the generation of a relocatable object module file.
Linker (program name: LK78K2)	Links an object module file, generated by the relocatable assembler, with a library file, to determine the absolute address of the program and to generate a load module file.
Object converter (program name: OC78K2)	Converts a load module file, generated by the linker, into a suitable form for downloading to the in-circuit emulator and PROM programmer.
Librarian (program name: LB78K2)	Links object module files, generated by the relocatable assembler, to create a single library file. It also updates the library files.
List converter (program name: LCNV78K2)	Creates an assemble list from the assemble list file output by the relocatable assembler, using the object and load module files. The assemble list contains absolute values.

(1) Language processing software (2/3)

78K/II series relocatable assembler (RA78K/II)	Host machine	OS	Distribution medium	Part number	
	PC-9800 series	MS-DOS™ (Ver. 3.30 to Ver. 5.00A[Note 3])	8-inch 2D[Note 1]	μS5A1RA78K2	
			5.25-inch 2HD	μS5A10RA78K2	
			3.5-inch 2HD	μS5A13RA78K2	
	IBM PC/AT or compatibles	See **(4)**.	5.25-inch 2D[Note 2]	μS7B11RA78K2	
			5.25-inch 2HC	μS7B10RA78K2	*
			3.5-inch 2HC	μS7B13RA78K2	
	HP9000 series 700™	HP-UX™ (rel. 9.01)	DAT	μS3P16RA78K2	*
	SPARCstation™	SunOS™ (rel. 4.1.1)	Cartridge tape (QIC-24)	μS3K15RA78K2	
	EWS-4800 series (RISC)	EWS-UX/V (rel. 4.0)		μS3M15RA78K2	
78K/II series C compiler (CC78K/II)	This C compiler can be used with all 78K/II series products. Its language specifications conform to ANSI standards, and compiled programs can be written into ROM. The compiler offers such features as special function register manipulation, bit manipulation, variables using short direct addressing, and interrupt control. The use of these features ensures effective programming and high object efficiency. It also has a start-up routine sample program and a standard function object library. To use this compiler, the 78K/II series relocatable assembler (RA78K/II) is necessary.				
(CC78K/II)	Host machine	OS	Distribution medium	Part number	
	PC-9800 series	MS-DOS (Ver. 3.30 to Ver. 5.00A[Note 3])	5.25-inch 2HD	μS5A10CC78K2	
			3.5-inch 2HD	μS5A13CC78K2	
	IBM PC/AT or compatibles	See **(4)**.	5.25-inch 2D[Note 2]	μS7B11CC78K2	
			5.25-inch 2HC	μS7B10CC78K2	*
			3.5-inch 2HC	μS7B13CC78K2	
	HP9000 series 700	HP-UX (rel. 9.01)	DAT	μS3P16CC78K2	*
	SPARCstation	SunOS (rel. 4.1.1)	Cartridge tape (QIC-24)	μS3K15CC78K2	
	EWS-4800 series (RISC)	EWS-UX/V (rel. 4.0)		μS3M15CC78K2	

Notes 1. The 8-inch 2D model has been superseded by the 5.25-inch 2HD and 3.5-inch 2HD models. Those users who have already purchased an 8-inch 2D model will be supplied with a 5.25-inch 2HD or 3.5-inch 2HD model when the product is next upgraded.

2. The 5.25-inch 2D model is no longer available. Those users who have already purchased a 5.25-inch 2D model will be supplied with a 5.25-inch 2HC or 3.5-inch 2HC model when the product is next upgraded.

3. Versions 5.00 and 5.00A feature a task swap function. However, the task swap function cannot be used with this software.

(1) Language processing software (3/3)

78K/II series C compiler library source file (CC78K/II-L)	This source program is used to modify the libraries supplied with CC78K/II to satisfy user specifications.			
	Host machine	OS	Distribution medium	Part number
	PC-9800 series	MS-DOS (Ver. 3.30 to Ver. 5.00A**Note**)	5.25-inch 2HD	μS5A10CC78K2-L
			3.5-inch 2HD	μS5A13CC78K2-L
*	IBM PC/AT or compatibles	See **(4)**.	5.25-inch 2HC	μS7B10CC78K2-L
			3.5-inch 2HC	μS7B13CC78K2-L
*	HP9000 series 700	HP-UX (rel. 9.01)	DAT	μS3P16CC78K2-L
	SPARCstation	SunOS (rel. 4.1.1)	Cartridge tape (QIC-24)	μS3K15CC78K2-L
	EWS-4800 series (RISC)	EWS-UX/V (rel. 4.0)		μS3M15CC78K2-L

Note Versions 5.00 and 5.00A feature a task swap function. However, the task swap function cannot be used with this software.

(2) Software for the in-circuit emulator (1/2)

Screen debugger (SD78K/II)

This program controls the in-circuit emulator for the 78K/II series when used together with the device file. It can be used when the in-circuit emulator has been upgraded to IE-78230-R-A or IE-78240-R-A class and a PC-9800 series or IBM PC/AT computer is being used as a host computer. This debugger can debug source programs written in C, structured assembly language, and assembly language. Its split screen function, by which the screen is split into sections to enable the simultaneous display of different information, makes debugging more efficient.

Host machine	OS	Distribution medium	Part number	
PC-9800 series	MS-DOS (Ver. 3.30 to Ver. 5.00A**Note**)	5.25-inch 2HD	μS5A10SD78K2	
		3.5-inch 2HD	μS5A13SD78K2	
IBM PC/AT or compatibles	See **(4)**.	5.25-inch 2HC	μS7B10SD78K2	
		3.5-inch 2HC	μS7B13SD78K2	*

Device file

DF78210
DF78220
DF78230
DF78240

This is used together with the screen debugger (SD78K/II) to debug programs of the μPD78214 sub-series.

Host machine	OS	Distribution medium	Part number	
PC-9800 series	MS-DOS (Ver. 3.30 to Ver. 5.00A**Note**)	5.25-inch 2HD	μS5A10DF78210	
		3.5-inch 2HD	μS5A13DF78210	
IBM PC/AT or compatibles	See **(4)**.	5.25-inch 2HC	μS7B10DF78210	
		3.5-inch 2HC	μS7B13DF78210	*
PC-9800 series	MS-DOS (Ver. 3.30 to Ver. 5.00A**Note**)	5.25-inch 2HD	μS5A10DF78220	
		3.5-inch 2HD	μS5A13DF78220	
IBM PC/AT or compatibles	See **(4)**.	5.25-inch 2HC	μS7B10DF78220	
		3.5-inch 2HC	μS7B13DF78220	*
PC-9800 series	MS-DOS (Ver. 3.30 to Ver. 5.00A**Note**)	5.25-inch 2HD	μS5A10DF78230	
		3.5-inch 2HD	μS5A13DF78230	
IBM PC/AT or compatibles	See **(4)**.	5.25-inch 2HC	μS7B10DF78230	
		3.5-inch 2HC	μS7B13DF78230	*
PC-9800 series	MS-DOS (Ver. 3.30 to Ver. 5.00A**Note**)	5.25-inch 2HD	μS5A10DF78240	
		3.5-inch 2HD	μS5A13DF78240	
IBM PC/AT or compatibles	See **(4)**.	5.25-inch 2HC	μS7B10DF78240	
		3.5-inch 2HC	μS7B13DF78240	*

Note Versions 5.00 and 5.00A feature a task swap function. However, the task swap function cannot be used with this software.

(2) Software for the in-circuit emulator (2/2)

In-circuit emulator control program
(IE78210, IE78220, IE78234, IE78240)

This program enables control of the in-circuit emulator for the 78K/II series from the host machine. It can automatically execute commands, thus enhancing efficiency in debugging. The following programs are available, depending on the type of in-circuit emulator:

Emulator	Host machine	OS	Distribution medium	Part number
IE-78210-R IE-78210-R-EM	PC-9800 series	MS-DOS (Ver. 3.10 to Ver. 5.00A Note 2)	8-inch 2D Note 1	μS5A1IE78210-P01
			5.25-inch 2HD	μS5A10IE78210-P01
			3.5-inch 2HD	μS5A13IE78210
*	IBM PC/AT or compatibles	See **(4)**.	5.25-inch 2D Note 3	μS7B11IE78210-P02
			5.25-inch 2HC	μS7B10IE78210
			3.5-inch 2HC	μS7B13IE78210
IE-78220-R IE-78220-R-EM	PC-9800 series	MS-DOS (Ver. 3.10 to Ver. 5.00A Note 2)	8-inch 2D Note 1	μS5A1IE78220-P01
			5.25-inch 2HD	μS5A10IE78220-P01
			3.5-inch 2HD	μS5A13IE78220
*	IBM PC/AT or compatibles	See **(4)**.	5.25-inch 2D Note 3	μS7B11IE78220-P02
			5.25-inch 2HC	μS7B10IE78220-P02
			3.5-inch 2HC	μS7B13IE78220-P02
IE-78230-R IE-78230-R-EM	PC-9800 series	MS-DOS (Ver. 3.10 to Ver. 5.00A Note 2)	8-inch 2D Note 1	μS5A1IE78230
			5.25-inch 2HD	μS5A10IE78230
			3.5-inch 2HD	μS5A13IE78230
*	IBM PC/AT or compatibles	See **(4)**.	5.25-inch 2D Note 3	μS7B11IE78230
			5.25-inch 2HC	μS7B10IE78230
			3.5-inch 2HC	μS7B13IE78230
IE-78240-R IE-78240-R-EM	PC-9800 series	MS-DOS (Ver. 3.10 to Ver. 5.00A Note 2)	8-inch 2D Note 1	μS5A1IE78240
			5.25-inch 2HD	μS5A10IE78240
			3.5-inch 2HD	μS5A13IE78240
*	IBM PC/AT or compatibles	See **(4)**.	5.25-inch 2D Note 3	μS7B11IE78240
			5.25-inch 2HC	μS7B10IE78240
			3.5-inch 2HC	μS7B13IE78240

Notes 1. The 8-inch 2D model has been superseded by the 5.25-inch 2HD and 3.5-inch 2HD models. Those users who have already purchased an 8-inch 2D model will be supplied with a 5.25-inch 2HD or 3.5-inch 2HD model when the product is next upgraded.

2. Versions 5.00 and 5.00A feature a task swap function. However, the task swap function cannot be used with this software.

3. The 5.25-inch 2D model is no longer available. Those users who have already purchased a 5.25-inch 2D model will be supplied with a 5.25-inch 2HC or 3.5-inch 2HC model when the product is next upgraded.

(3) Software for the PROM Programmer

PG-1500 controller	This program provides the serial and parallel interfaces between PG-1500 and the host machine, enabling the host machine to control the PG-1500.			
	Host machine	OS	Distribution medium	Part number
	PC-9800 series	MS-DOS (Ver. 3.10 to Ver. 5.00A**Note 1**)	5.25-inch 2HD	µS5A10PG1500
			3.5-inch 2HD	µS5A13PG1500
	IBM PC/AT or compatibles	See **(4)**.	5.25-inch 2D**Note 2**	µS7B11PG1500
			5.25-inch 2HC	µS7B10PG1500 *
			3.5-inch 2HC	µS7B13PG1500

Notes 1. Versions 5.00 and 5.00A feature a task swap function. However, the task swap function cannot be used with this software.

2. The 5.25-inch 2D model is no longer available. Those users who have already purchased a 5.25-inch 2D model will be supplied with a 5.25-inch 2HC or 3.5-inch 2HD model when the product is next upgraded.

(4) OS for the IBM PC *

The following OSs are supported for the IBM PC:

OS	Version
PC DOS™	Ver. 3.1 to Ver. 6.3 J6.1/V**Note 2** to J6.3/V**Note 2**
Windows™**Note 1**	Ver. 3.1
MS-DOS	Ver. 5.0 to Ver. 6.2 5.0/V**Note 2** to 6.2/V**Note 2**
IBM DOS™	J5.02/V**Note 2**

Notes 1. PC DOS and Windows are used together for the fuzzy knowledge-data creation tool.

2. Only English-version systems are supported.

Caution Versions 5.00 and later feature a task swap function. However, the task swap function cannot be used with this software.

9.3 UPGRADING OTHER IN-CIRCUIT EMULATORS TO 78K/II SERIES LEVEL

The 78K series and 75X series in-circuit emulators can be upgraded to the level of the 78K/II series by replacing their internal boards with an optional board.

Note that the upgraded in-circuit emulator requires an appropriate new control program.

9.3.1 Upgrading to IE-78240-R-A Level

Emulator	IE group number	Required board	Remarks
IE-78230-R-A IE-78140-R	1	IE-78240-R-EM	—
IE-78240-R**Note 1**	2	IE-78200-R-BK	—
IE-78112-R**Note 1** IE-78210-R**Note 1** IE-78220-R**Note 1** IE-78310-R**Note 1** IE-78310A-R	3	IE-78200-R-BK IE-78240-R-EM**Note 2**	The high-speed download function is not supported. Those users who are also using an in-circuit emulator of IE group 1, 2, or 4, are recommended to upgrade these emulators also. Those users with an in-circuit emulator of IE group 1 do not need to purchase the IE-78200-R-BK (the IE-78200-R-BK board is built into the IE group 1 in-circuit emulator).
IE-75000-R IE-75001-R IE-78000-R IE-78130-R IE-78230-R**Note 1** IE-78320-R**Note 1** IE-78327-R IE-78330-R IE-78350-R IE-78600-R	4	IE-78200-R-BK IE-78240-R-EM	Those users with an in-circuit emulator of IE group 1 do not need to purchase the IE-78200-R-BK (the IE-78200-R-BK board is built into the IE group 1 in-circuit emulator).

Notes 1. This product is no longer produced, and is not available from NEC.

2. This board is used for emulation for the μPD78214 sub-series. Those users who already have the IE-78210-R-EM**Note 1** do not have to purchase this board.

9.3.2 Upgrading to IE-78240-R[Note 1] Level

Emulator	IE group number	Required board	Remarks
IE-78112-R[Note 1] IE-78210-R[Note 1] IE-78220-R[Note 1]	1	IE-78240-R-EM[Note 2]	The high-speed download function is not supported. Those users who are also using an in-circuit emulator of IE group 4, are recommended to upgrade to IE group 4 level.
IE-78130-R IE-78230-R[Note 1]	2	IE-78240-R-EM	—
IE-78310-R[Note 1] IE-78310A-R	3	IE-78200-R-EM[Note 1] IE-78240-R-EM[Note 2]	The high-speed download function is not supported. Those users who have an in-circuit emulator of IE group 1 do not need to purchase the IE-78200-R-EM (the IE-78200-R-EM board is built into the IE group 1 in-circuit emulator).
IE-75000-R IE-75001-R IE-78000-R IE-78320-R[Note 1] IE-78327-R IE-78330-R IE-78350-R IE-78600-R	4	IE-78200-R-EM[Note 1] IE-78240-R-EM	Those users who have an in-circuit emulator of IE group 1 do not need to purchase the IE-78200-R-EM (the IE-78200-R-EM board is built into the IE group 1 in-circuit emulator).
IE-78140-R IE-78230-R-A	5	IE-78200-R-EM[Note 1] IE-78240-R-EM	Upgrading to IE-78240-R-A level is recommended.

Notes 1. This product is no longer produced, and is not available from NEC.

2. This board is used for emulation for the μPD78214 sub-series. Those users who already have the IE-78210-R-EM[Note 1] do not have to purchase this board.

9.3.3 Upgrading to IE-78230-R-A Level

Emulator	IE group number	Required board	Remarks
IE-78240-R-A IE-78140-R	1	IE-78230-R-EM	—
IE-78230-R**Note 1**	2	IE-78200-R-BK	—
IE-78112-R**Note 1** IE-78210-R**Note 1** IE-78220-R**Note 1** IE-78310-R**Note 1** IE-78310A-R	3	IE-78200-R-BK IE-78230-R-EM**Note 2**	The high-speed download function is not supported. Those users who are also using an in-circuit emulator of IE group 1, 2, or 4, are recommended to upgrade these emulators also. Those users with an in-circuit emulator of IE group 1 do not need to purchase the IE-78200-R-BK (the IE-78200-R-BK board is built into the IE group 1 in-circuit emulator).
IE-75000-R IE-75001-R IE-78000-R IE-78130-R IE-78240-R IE-78320-R**Note 1** IE-78327-R IE-78330-R IE-78350-R IE-78600-R	4	IE-78200-R-BK IE-78230-R-EM	Those users with an in-circuit emulator of IE group 1 do not need to purchase the IE-78200-R-BK (the IE-78200-R-BK board is built into the IE group 1 in-circuit emulator).

Notes 1. This product is no longer produced, and is not available from NEC.

2. This board is used for emulation for the μPD78224 sub-series. Those users who already have the IE-78220-R-EM**Note 1** do not have to purchase this board.

9.3.4 Upgrading to IE-78230-R[Note 1] Level

Emulator	IE group number	Required board	Remarks
IE-78112-R[Note 1] IE-78210-R[Note 1] IE-78220-R[Note 1]	1	IE-78230-R-EM[Note 2]	The high-speed download function is not supported. Those users who are also using an in-circuit emulator of IE group 4, are recommended to upgrade to IE group 4 level.
IE-78130-R IE-78240-R[Note 1]	2	IE-78230-R-EM	—
IE-78310-R[Note 1] IE-78310A-R	3	IE-78200-R-EM[Note 1] IE-78230-R-EM[Note 2]	The high-speed download function is not supported. Those users who have an in-circuit emulator of IE group 1 do not need to purchase the IE-78200-R-EM (the IE-78200-R-EM board is built into the IE group 1 in-circuit emulator).
IE-75000-R IE-75001-R IE-78000-R IE-78320-R[Note 1] IE-78327-R IE-78330-R IE-78350-R IE-78600-R	4	IE-78200-R-EM[Note 1] IE-78230-R-EM	Those users who have an in-circuit emulator of IE group 1 do not need to purchase the IE-78200-R-EM (the IE-78200-R-EM board is built into the IE group 1 in-circuit emulator).
IE-78140-R IE-78240-R-A	5	IE-78200-R-EM[Note 1] IE-78230-R-EM	Upgrading to IE-78230-R-A level is recommended.

Notes 1. This product is no longer produced, and is not available from NEC.

2. This board is used for emulation for the μPD78224 sub-series. Those users who already have the IE-78220-R-EM[Note 1] do not have to purchase this board.

9.3.5 Upgrading to IE-78220-R[Note 1] Level

Emulator	IE group number	Required board	Remarks
IE-78112-R[Note 1] IE-78210-R[Note 1]	1	IE-78220-R-EM[Note 2]	—
IE-78310-R[Note 1] IE-78310A-R	2	IE-78200-R-EM IE-78220-R-EM[Note 2]	Those users who have an in-circuit emulator of IE group 1 do not need to purchase the IE-78200-R-EM (the IE-78200-R-EM board is built into the IE group 1 in-circuit emulator).
IE-75000-R IE-75001-R IE-78000-R IE-78130-R IE-78140-R IE-78230-R[Note 1] IE-78230-R-A IE-78240-R[Note 1] IE-78240-R-A IE-78320-R[Note 1] IE-78327-R IE-78330-R IE-78350-R IE-78600-R	3	—	Upgrading to IE-78220-R level is not allowed. Upgrading to IE-78230-R-A is recommended.

Notes 1. This product is no longer produced, and is not available from NEC.

2. This board is no longer produced, and is not available from NEC. Those users who do not have the IE-78220-R-EM, are recommended to upgrade to the IE-78230-R-A level, which includes the functions of IE-78230-R.

9.3.6 Upgrading to IE-78210-R[Note 1] Level

Emulator	IE group number	Required board	Remarks
IE-78112-R[Note 1] IE-78220-R[Note 1]	1	IE-78210-R-EM[Note 2]	—
IE-78310-R[Note 1] IE-78310A-R	2	IE-78200-R-EM[Note 1] IE-78210-R-EM[Note 2]	Those users who have an in-circuit emulator of IE group 1 do not need to purchase the IE-78200-R-EM (the IE-78200-R-EM board is built into the IE group 1 in-circuit emulator).
IE-75000-R IE-75001-R IE-78000-R IE-78130-R IE-78140-R IE-78230-R[Note 1] IE-78230-R-A IE-78240-R[Note 1] IE-78240-R-A IE-78320-R[Note 1] IE-78327-R IE-78330-R IE-78350-R IE-78600-R	3	—	Upgrading to IE-78210-R level is not allowed. Upgrading to IE-78240-R-A is recommended.

Notes 1. This product is no longer produced, and is not available from NEC.

2. This board is no longer produced, and is not available from NEC. Those users who do not have the IE-78210-R-EM, are recommended to upgrade to the IE-78240-R-A level, which includes the functions of IE-78240-R.

[MEMO]

CHAPTER 10 BUILT-IN SOFTWARE

10.1 REAL-TIME OS

Real-time OS (RX78K/II)

The RX78K/II is intended to achieve a multitask environment for control fields which require real-time control. CPU idle time can be assigned to other processing, allowing overall system performance to be improved.
The RX78K/II offers 31 system calls compliant with the μITRON specifications.
The RX78K/II package provides a tool (configurator) to create the RX78K/II nucleus and multiple information tables.
However, the use of this requires RAM of 1K byte or more**Note 1**.

Host machine	OS	Distribution medium	Part number
PC-9800 series	MS-DOS (Ver. 3.30 to Ver. 5.00A**Note 2**)	5.25-inch 2HD	μS5A10RX78217
		3.5-inch 2HD	μS5A13RX78217
IBM PC/AT or compatibles	See **Section 9.2.2 (4)**.	5.25-inch 2HC	μS7B10RX78217
		3.5-inch 2HC	μS7B13RX78217
HP9000 series 700	HP-UX (rel. 9.01)	DAT	μS3P16RX78217
SPARCstation	SunOS (rel. 4.1.1)	Cartridge tape (QIC-24)	μS3K15RX78217
EWS-4800 series (RISC)	EWS-UX/V (rel. 4.0)		μS3M15RX78217

*

*

Notes 1. Target devices: μPD78217A, 78218A, 78P218A, 78237, 78238, 78P238

2. Versions 5.00 and 5.00A feature a task swap function. However, the task swap function cannot be used with this software.

Caution To purchase the RX78K/II, you need to fill in a purchase form and enter into a use authorization contract in advance.

Remark When the RX78K/II real-time OS is used, the RA78K/II assembler package (available separately) is necessary.

10.2 FUZZY INFERENCE DEVELOPMENT SUPPORT SYSTEM

Fuzzy knowledge-data creation tool	Supports input and editing, as well as the evaluation (simulation) of fuzzy knowledge-data (fuzzy rules and membership functions).			
	Host machine	OS	Distribution medium	Part number
	PC-9800 series	MS-DOS (Ver. 3.30 to Ver. 5.00A[Note])	5.25-inch 2HD	μS5A10FE9000
			3.5-inch 2HD	μS5A13FE9000
*	IBM PC/AT or compatibles	See **Section 9.2.2 (4)**.	5.25-inch 2HC	μS7B10FE9200
			3.5-inch 2HC	μS7B13FE9200
Translator	This program converts fuzzy knowledge-data, obtained with the fuzzy knowledge-data creation tool, into an assembler source program for the RA78K/II.			
	Host machine	OS	Distribution medium	Part number
	PC-9800 series	MS-DOS (Ver. 3.30 to Ver. 5.00A[Note])	5.25-inch 2HD	μS5A10FT9080
			3.5-inch 2HD	μS5A13FT9080
*	IBM PC/AT or compatibles	See **Section 9.2.2 (4)**.	5.25-inch 2HC	μS7B10FT9085
			3.5-inch 2HC	μS7B13FT9085
Fuzzy inference module (FI78K/II)	This program performs fuzzy inference by linking with the fuzzy knowledge data converted by the translator.			
	Host machine	OS	Distribution medium	Part number
	PC-9800 series	MS-DOS (Ver. 3.30 to Ver. 5.00A[Note])	5.25-inch 2HD	μS5A10FI78K2
			3.5-inch 2HD	μS5A13FI78K2
*	IBM PC/AT or compatibles	See **Section 9.2.2 (4)**.	5.25-inch 2HC	μS7B10FI78K2
			3.5-inch 2HC	μS7B13FI78K2
Fuzzy inference debugger (FD78K/II)	This program performs fuzzy inference by linking with the fuzzy knowledge-data at the hardware level, using the in-circuit emulator.			
	Host machine	OS	Distribution medium	Part number
	PC-9800 series	MS-DOS (Ver. 3.30 to Ver. 5.00A[Note])	5.25-inch 2HD	μS5A10FD78K2
			3.5-inch 2HD	μS5A13FD78K2
*	IBM PC/AT or compatibles	See **Section 9.2.2 (4)**.	5.25-inch 2HC	μS7B10FD78K2
			3.5-inch 2HC	μS7B13FD78K2

Note Versions 5.00 and 5.00A feature a task swap function. However, the task swap function cannot be used with this software.

APPENDIX A INDEX OF INSTRUCTIONS (MNEMONICS CLASSIFIED BY FUNCTION)

[Stack manipulation instructions]

[Unconditional branch instructions]

[Conditional branch instructions]

[CPU control instructions]

APPENDIX B INDEX OF INSTRUCTIONS (MNEMONICS IN ALPHABETICAL ORDER)

[S]

[X]

APPENDIX C REVISION HISTORY *

A revision history is shown below. Chapters described in the revised-chapter column indicate those for the corresponding edition.

Edition	Major changes	Revised chapter
Fifth	• The 68-pin PLCC has been changed to the 68-pin plastic QFJ. • The 64-pin ceramic LCC with window has been changed to the 64-pin ceramic WQFN. • The IBM PC series has been changed to the IBM PC/AT.	Throughout
	Cautions and remarks have been added to **Section 4.1.3**.	Chapter 4
	The instruction code of BT has been modified in **Section 7.2.3**.	Chapter 7
	• **Table 9-1** has been divided into a table for the screen debugger and a table for the in-circuit emulator control program. • Descriptions related to the EV-9900, HP9000 series 300, SPARCstation, and EWS-4800 series have been added. • Description related to IBM PC/AT has been added to the screen debugger and device file.	Chapter 9
	Chapter 10 has been added.	Chapter 10
Sixth	A disaster/crime prevention unit has been added as a special product in applications of the μPD78214 sub-series, μPD78218A sub-series, and μPD78234 sub-series.	Chapter 1
	(Z, AC, and CY flags also do not change) has been added to the sentence, “If the second operand (cnt) is 0, no processing is performed.” in [Description] of ROR, ROL, RORC, ROLC, SHR, SHL, SHRW, and SHLW.	Chapter 8
	• Description related to the 3.5-inch 2HC has been added to the IBM PC/AT. • The HP9000 series 300 has been changed to the HP9000 series 700. • The screen debugger of IBM PC/AT and the 5.25-inch 2HC of the device file have been already developed. • **(4) OS for the IBM PC** has been added. • The Fuzzy inference debugger (FD78K/II) has been already developed.	Chapter 9
	Appendix C has been added.	Appendix C

[MEMO]