NEC

User's Manual

78K/IV Series

16-Bit Single Chip Microcontroller

Instructions

For all 78K/IV Series

Document No. U10905EJ7V0UM00 (7th edition)
Date Published August 1998 N CP(K)

[MEMO]

NOTES FOR CMOS DEVICES

① **PRECAUTION AGAINST ESD FOR SEMICONDUCTORS**

Note:

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

② **HANDLING OF UNUSED INPUT PINS FOR CMOS**

Note:

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

③ **STATUS BEFORE INITIALIZATION OF MOS DEVICES**

Note:

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

The export of these products from Japan is regulated by the Japanese government. The export of some or all of these products may be prohibited without governmental license. To export or re-export some or all of these products from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

Caution: Purchase of NEC I^2C components conveys a license under the Philips I^2C Patent Rights to use these components in an I^2C system, provided that the system conforms to the I^2C Standard Specification as defined by Philips.

Major Revisions in This Edition

Pages	Contents
Throughout	• Addition of μPD784937, 784955 Subseries. Deletion of μPD784943 Subseries. • The status of following products changed from under development to completed: μPD784031(A), 784035(A), 784036(A), μPD784044(A), 784044(A1), 784044(A2), 784046(A), 784046(A1), 784046(A2), μPD784054(A), 784054(A1), 784054(A2), μPD784214, 784214Y, μPD784915B, 784916B, μPD784927, 78F4928, 784927Y, 78F4928Y • Modification of the package from GC-7EA to GC-8EU in μPD78F4216 and 78F4216Y. • Modification of the power supply voltage in μPD784908 Subseries. Mask ROM version (μPD784907, 784908): V_{DD} = 4.5 to 5.5 V → V_{DD} = 3.5 to 5.5 V PROM version (μPD78P4908): V_{DD} = 4.5 to 5.5 V → V_{DD} = 3.5 to 5.5 V
P. 163	**CHAPTER 6 INSTRUCTION SET** Modification of notes for special instructions (CHKL and CHKLA).
P. 291	**CHAPTER 7 DESCRIPTION OF INSTRUCTIONS** Modification of the operation sequences in the POP instruction. Addition of Caution to the CHKL instruction. Addition of Caution to the CHKLA instruction.
P. 473	**CHAPTER 8 DEVELOPMENT TOOL** Modification of arrangement
P. 479	**CHAPTER 9 SOFTWARE FOR EMBEDDING** Addition of description regarding PC environment.

The mark ★ shows major revised points.

Regional Information

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

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

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

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

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

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

NEC Electronics Italiana s.r.1.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

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

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

NEC Electronics (France) S.A.
Spain Office
Madrid, Spain
Tel: 01-504-2787
Fax: 01-504-2860

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

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

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

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

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

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

J98. 8

INTRODUCTION

Readers : This manual is intended for engineers who understand the functions of 78K/IV Series products and wish to design 78K/IV Series application systems.

- 78K/IV Series products
  - μPD784026 Subseries : μPD784020, 784021, 784025, 784026, 78P4026
  - ★ μPD784038 Subseries : μPD784031, 784035, 784036, 784037, 784038, 78P4038, 784031(A), 784035(A), 784036(A)
  - μPD784038Y Subseries : μPD784031Y, 784035Y, 784036Y, 784037Y, 784038Y, 78P4038Y
  - ★ μPD784046 Subseries : μPD784044, 784046, 784054, 78F4046, 78444(A), (A1), (A2), μPD784046(A), (A1), (A2), 784054(A), (A1), (A2)
  - ★ μPD784216 Subseries : μPD784214, 784215, 784216, 78F4216**Note**
  - ★ μPD784216Y Subseries : μPD784214Y, 784215Y, 784216Y, 78F4216Y**Note**
  - μPD784218 Subseries**Note** : μPD784217, 784218, 78F4218
  - μPD784218Y Subseries**Note** : μPD784217Y, 784218Y, 78F4218Y
  - μPD784225 Subseries**Note** : μPD784224, 784225, 78F4225
  - μPD784225Y Subseries**Note** : μPD784224Y, 784225Y, 78F4225Y
  - μPD784908 Subseries**Note** : μPD784907, 784908, 78P4908
  - ★ μPD784915 Subseries : μPD784915, 784915A, 784916A, 784915B, 784916B, 78P4916
  - ★ μPD784928 Subseries : μPD784927, 78F4928
  - ★ μPD784928Y Subseries : μPD784927Y, 78F4928Y
  - ★ μPD784937 Subseries**Note** : μPD784935, 784936, 784937, 78F4937
  - ★ μPD784955 Subseries**Note** : μPD784953, 784955, 78F4956

Note Under development

Purpose : The purpose of this manual is to explain the various instruction functions of the 78K/IV Series.

Organization : This manual is broadly organized as follows:

- Features of 78K/IV Series products
- CPU functions
- Instruction set
- Instruction descriptions
- Development tools

How to read this manual : Readers require a general understanding of electrical and logic circuits and microcontrollers.

- To check the details of an instruction function when the mnemonic is known:
  → Use **APPENDIX A** and **APPENDIX B INDEX OF INSTRUCTIONS**.

- To check an instruction when you know the general function but do not know the mnemonic:
  → Find the mnemonic in **CHAPTER 6 INSTRUCTION SET**, then check the function in **CHAPTER 7 DESCRIPTION OF INSTRUCTIONS**.

- For a general understanding of the various instruction functions of the 78K/IV Series:
  → Read in accordance with the contents.

- For information on the hardware functions of the 78K/IV Series:
  → Read the separate User's Manual.
  - μPD784026 Subseries User's Manual – Hardware (U10898E)
  - μPD784038/784038Y Subseries User's Manual – Hardware (U R316E)
  - μPD784046 Subseries User's Manual – Hardware (U11515E)
  - μPD784054 User's Manual – Hardware (U11719E)
  - μPD784216/784216Y Subseries User's Manual – Hardware (U12015E)
  - μPD784218/784218Y Subseries User's Manual – Hardware (U12970E)
  - μPD784225/784225Y Subseries User's Manual – Hardware (U12679E)
  - μPD784908 Subseries User's Manual – Hardware (U11787E)
  - μPD784915 Subseries User's Manual – Hardware (U10444E)
  - μPD784928/784928Y Subseries User's Manual – Hardware (U12648E)
  - ★ μPD784937 Subseries User's Manual – Hardware (To be prepared)
  - ★ μPD784955 Subseries User's Manual – Hardware (U12833E)

Conventions :

Significance in data notation	: High-order digit on left, low-order digit on right
Active-low notation	: $\overline{\times\times\times}$ (line above pin or signal name)
Note	: Footnote for item marked with **Note** in the text
Caution	: Information requiring particular attention
Remark	: Supplementary information
Numeric notations	: Binary .... ××××B or ××××
	Decimal .... ××××
	Hexadecimal ..... ××××H

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

- **Documents common to the 78K/IV Series**

Document Name	Document Number	
	Japanese	English
Pamphlet	U10752J	–
User's Manual – Instructions	U10905J	This manual
Instruction Set	U10595J	—
Instruction List	U10594J	—
Application Note – Software Basics	U10095J	U10095E

- **Individual documents**

  - **μPD784026 Subseries**

Document Name	Document Number	
	Japanese	English
μPD784020/784021 Data Sheet	U11514J	U11514E
μPD784025/784026 Data Sheet	U11605J	U11605E
μPD78P4026 Data Sheet	U11609J	U11609E
μPD784026 Subseries User's Manual – Hardware	U10898J	U10898E
μPD784026 Subseries Special Function Register Table	U10593J	—
μPD784026 Subseries Application Note – Hardware Basics	U10573J	U10573E

- **μPD784038/784038Y Subseries**

Document Name	Document Number	
	Japanese	English
μPD784031 Data Sheet	U11507J	U11507E
μPD784035/784036/784037/784038 Data Sheet	U10847J	U10847E
★ μPD784031(A) Data Sheet	U13009J	U13009E
μPD784035(A)/784036(A) Data Sheet	U13010J	U13010E
μPD78P4038 Data Sheet	U10848J	U10848E
μPD784038 Subseries Special Function Register Table	U11090J	—
μPD784031Y Data Sheet	U11504J	U11504E
μPD784035Y/784036Y/784037Y/784038Y Data Sheet	U10741J	U10741E
μPD78P4038Y Data Sheet	U10742J	U10742E
μPD784038Y Subseries Special Function Register Table	U11091J	—
μPD784038/784038Y Subseries User's Manual – Hardware	U11316J	U11316E
★ μPD784038/784038Y Subseries Application Note – Hardware Basics	U13285J	—

• **μPD784046 Subseries**

Document Name	Document Number	
	Japanese	English
μPD784044/784046 Data Sheet	U10951J	U10951E
★ μPD784044(A)/784046(A) Data Sheet	U13121J	U13121E
μPD784054 Data Sheet	U11154J	U11154E
★ μPD784054(A) Data Sheet	U13122J	U13122E
μPD78F4046 Preliminary Product Information	U11447J	U11447E
μPD784046 Subseries Special Function Register Table	U10986J	—
μPD784054 Special Function Register Table	U11113J	—
μPD784046 Subseries User's Manual – Hardware	U11515J	U11515E
μPD784054 User's Manual – Hardware	U11719J	U11719E

• **μPD784216, 784216Y Subseries**

Document Name	Document Number	
	Japanese	English
μPD784214/784215/784216 Preliminary Product Information	U11813J	U11813E
μPD78F4216 Preliminary Product Information	U11825J	U11825E
μPD784216 Subseries Special Function Register Table	U12045J	—
μPD784214Y/784215Y/784216Y Preliminary Product Information	U11725J	U11725E
μPD78F4216Y Preliminary Product Information	U11824J	U11824E
μPD784216Y Subseries Special Function Register Table	U12046J	—
μPD784216/784216Y Subseries User's Manual – Hardware	U12015J	U12015E

• **μPD784218, 784218Y Subseries**

Document Name	Document Number	
	Japanese	English
μPD784217/784218 Preliminary Product Information	U12303J	U12303E
μPD78F4218 Preliminary Product Information	U12439J	U12439E
μPD784218 Subseries Special Function Register Table	Planned	—
μPD784217Y/784218Y Preliminary Product Information	U12304J	U12304E
μPD78F4218Y Preliminary Product Information	U12440J	U12440E
μPD784218Y Subseries Special Function Register Table	U12919J	—
μPD784218/784218Y Subseries User's Manual – Hardware	U12970J	U12970E

• **μPD784225, 784225Y Subseries**

Document Name	Document Number	
	Japanese	English
μPD784224/784225 Preliminary Product Information	U12498J	U12498E
μPD78F4225 Preliminary Product Information	U12499J	U12499E
μPD784225 Subseries Special Function Register Table	U12689J	—
μPD784224Y/784225Y Preliminary Product Information	U12376J	U12376E
μPD78F4225Y Preliminary Product Information	U12377J	U12377E
μPD784225Y Subseries Special Function Register Table	U12699J	—
μPD784225/784225Y Subseries User's Manual – Hardware	U12679J	U12679E

• **μPD784908 Subseries**

Document Name	Document Number	
	Japanese	English
μPD784907/784908 Preliminary Product Information	U11680J	U11680E
μPD78P4908 Preliminary Product Information	U11681J	U11681E
μPD784908 Subseries Special Function Register Table	U11589J	—
μPD784908 Subseries User's Manual – Hardware	U11787J	U11787E

• **μPD784915 Subseries**

Document Name	Document Number	
	Japanese	English
μPD784915 Data Sheet	U11044J	U11044E
μPD784915A/784916A Data Sheet	U11022J	U11022E
μPD784915B/784916B Data Sheet	U13118J	U13118E
μPD78P4916 Data Sheet	U11045J	U11045E
μPD784915 Subseries Special Function Register Table	U10976J	—
μPD784915 Subseries User's Manual – Hardware	U10444J	U10444E
μPD784915 Subseries Application Note – VCR Servo Basics	U11361J	U11361E

★ • **μPD784928, 784928Y Subseries**

Document Name	Document Number	
	Japanese	English
μPD784927 Data Sheet	U12255J	U12255E
μPD78F4928 Preliminary Product Information	U12188J	U12188E
μPD784928 Subseries Special Function Register Table	U11045J	—
μPD784927Y Data Sheet	U12373J	U12373E
μPD78F4928Y Preliminary Product Information	U12271J	U12271E
μPD784928Y Subseries Special Function Register Table	U12719J	—
μPD784928/784928Y Subseries User's Manual – Hardware	U12648J	U12648E

★ • **μPD784937 Subseries**

Document Name	Document Number	
	Japanese	English
μPD784935/784936/784937 Preliminary Product Information	U13572J	To be prepared
μPD78F4937 Preliminary Product Information	U13573J	To be prepared
μPD784937 Subseries Special Function Register Table	To be prepared	—
μPD784937 Subseries User's Manual – Hardware	To be prepared	To be prepared

★ • **μPD784955 Subseries**

Document Name	Document Number	
	Japanese	English
μPD784953/784955 Preliminary Product Information	U12830J	U12830E
μPD78F4956 Preliminary Product Information	U12831J	U12831E
μPD784955 Subseries Special Function Register Table	U12832J	—
μPD784955 Subseries User's Manual – Hardware	U12833J	U12833E

CONTENTS

LIST OF FIGURES

LIST OF TABLES

[MEMO]

CHAPTER 1 FEATURES OF 78K/IV SERIES PRODUCTS

The 78K Series consists of 6 series as shown in Figure 1-1.

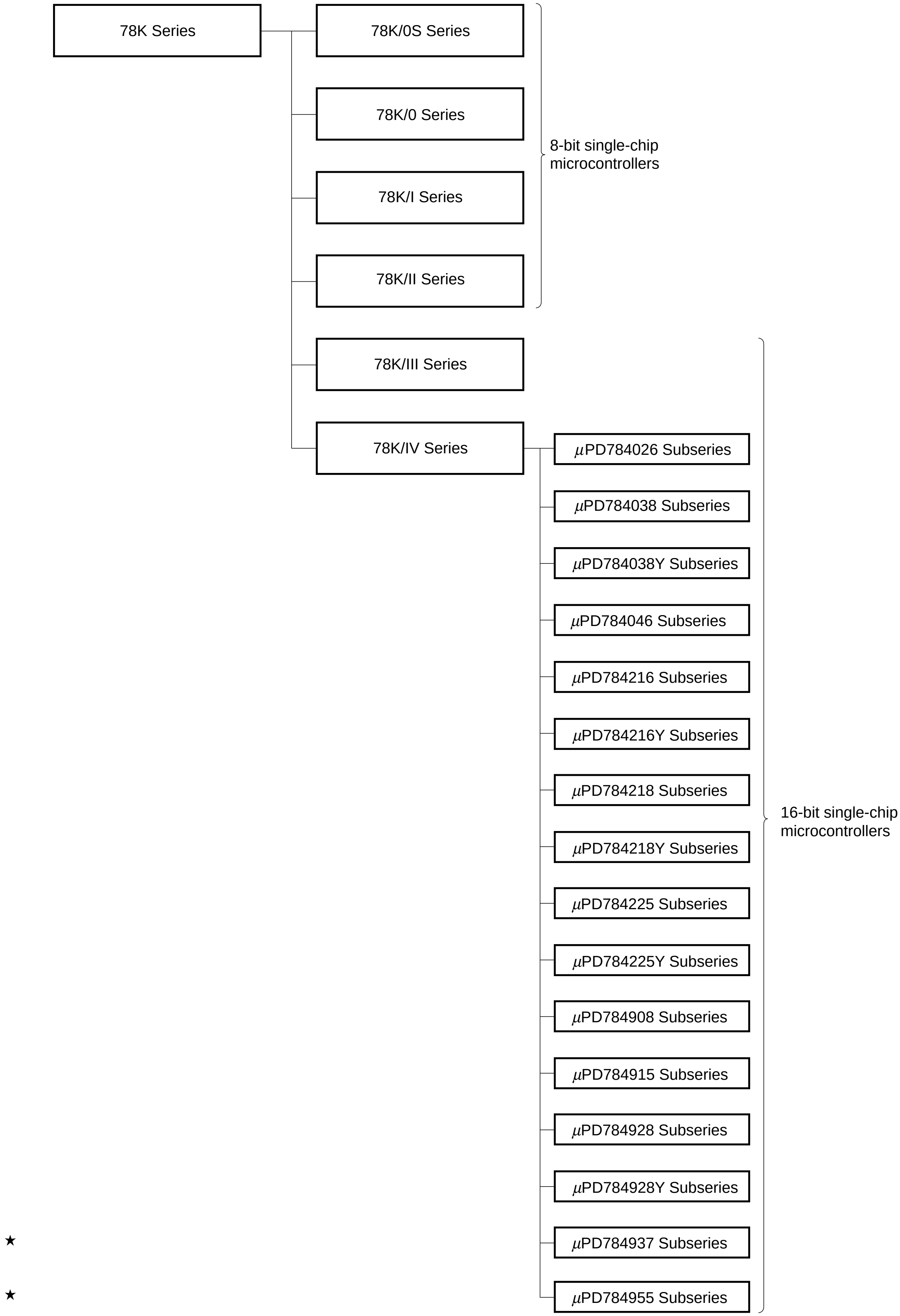
The 78K/IV Series is one of these 6 series, comprising products with an on-chip 16-bit CPU.

These products have an instruction set suitable for control applications, a high-performance interrupt controller, and incorporate a high-performance CPU equipped with a maximum 1-Mbyte program memory space and maximum 16-Mbyte data memory space.

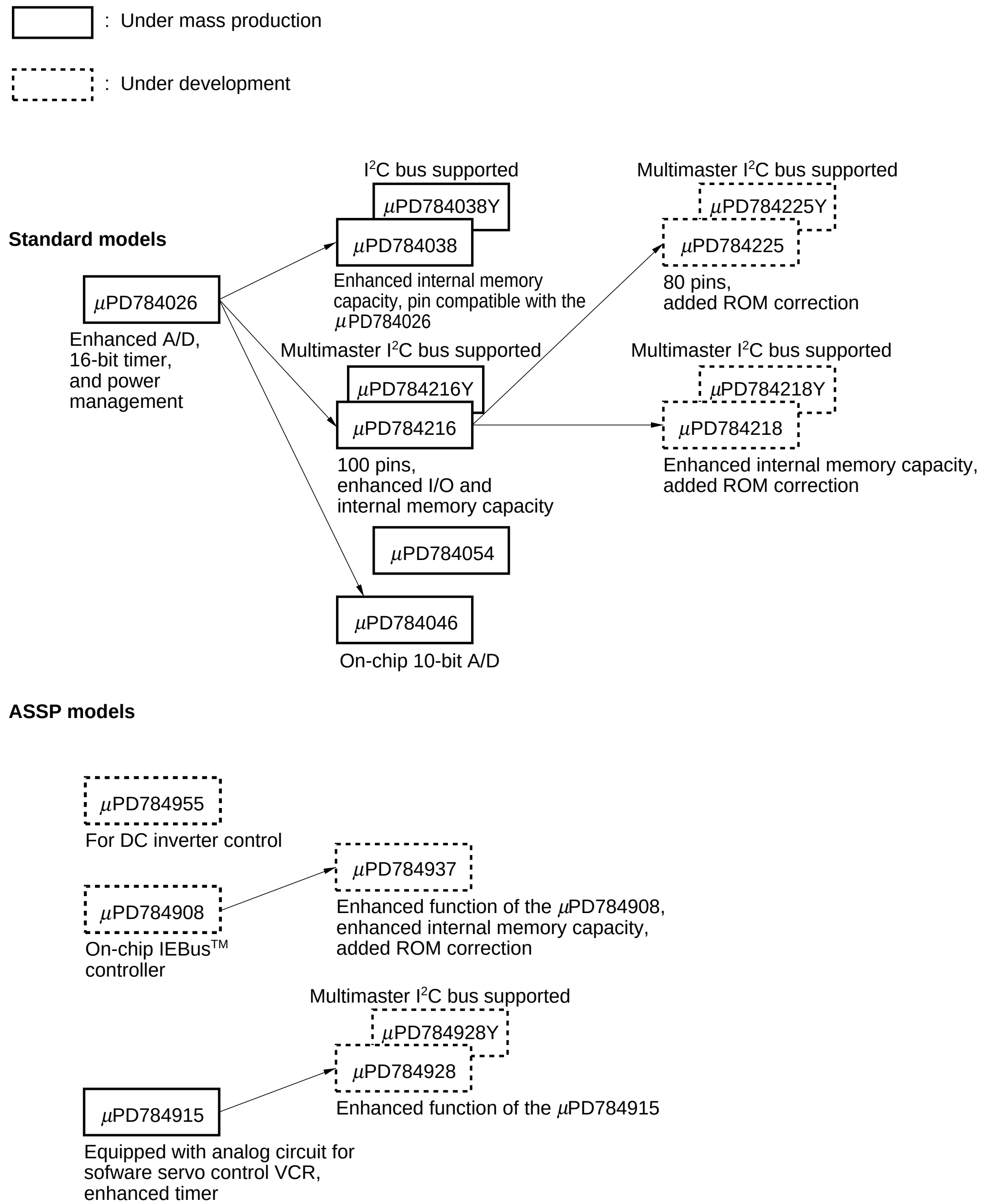
The 78K/IV Series offers a variety of subseries, enabling the most suitable subseries to be selected for a particular application.

All the subseries have the same CPU, and differ only in their peripheral hardware. Consequently, the entire instruction set is common to all subseries. Moreover, individual products within a subseries differ only in the size of on-chip memory.

Figure 1-1. 78K Series and 78K/IV Series Composition



★ 1.1 78K/IV Series Product Development Diagram



1.2 Product Outline of μPD784026 Subseries (μPD784020, 784021, 784025, 784026, 78P4026)

1.2.1 Features

- Pins are compatible with μPD78234 Subseries
- Minimum instruction execution time: 160 ns/320 ns/640 ns/1,280 ns (at 25-MHz operation)
- On-chip memory
  - ROM
    Mask ROM : 48 Kbytes (μPD784025)
    64 Kbytes (μPD784026)
    None (μPD784020, 784021)
    PROM : 64 Kbytes (μPD78P4026)
  - RAM : 2,048 bytes (μPD784021, 784025, 784026)
    512 bytes (μPD784020)
- I/O pins: 64
  46 (μPD784020, 784021 only)
- Timer/counter: 16-bit timer/counter × 3 units
  16-bit timer × 1 unit
- Watchdog timer: 1 channel
- A/D converter: 8-bit resolution × 8 channels
- D/A converter: 8-bit resolution × 2 channels
- Serial interface: 3 channels
  UART/IOE (3-wire serial I/O): 2 channels (on-chip baud rate generator)
  CSI (3-wire serial I/O, SBI): 1 channel
- Interrupt controller (4-level priority)
  Vectored interrupt/macro service/context switching
- Standby function: HALT/STOP/IDLE mode
- Clock output function
  Selectable from f_{CLK}, $f_{CLK}/2$, $f_{CLK}/4$, $f_{CLK}/8$, $f_{CLK}/16$ (except μPD784020, 784021)
- Power supply voltage: V_{DD} = 2.7 to 5.5 V

1.2.2 Applications

Laser beam printers, autofocus cameras, plain paper copiers, printers, electronic typewriters, air conditioners, electronic musical instruments, cellular phones, etc.

1.2.3 Ordering information and quality grade

(1) Ordering information

Part Number	Package	Internal ROM
μPD784020GC-3B9	80-pin plastic QFP (14 × 14 mm)	None
μPD784021GC-3B9	80-pin plastic QFP (14 × 14 mm)	None
μPD784021GK-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	None
μPD784025GC-×××-3B9	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784026GC-×××-3B9	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD78P4026GC-3B9	80-pin plastic QFP (14 × 14 mm)	One-time PROM
μPD78P4026GC-×××-3B9 Note	80-pin plastic QFP (14 × 14 mm)	Preprogramming one-time PROM
μPD78P4026KK-T	80-pin ceramic WQFN (14 × 14 mm)	EPROM

Note QTOP™ microcontroller. "QTOP microcontroller" is a general term for a single-chip microcontroller with on-chip one-time PROM, for which total support is provided by NEC programming service, from programming to marking, screening, and verification.

Remark ××× indicates ROM code suffix.

(2) Quality grades

Part Number	Package	Quality Grade
μPD784020GC-3B9	80-pin plastic QFP (14 × 14 mm)	Standard
μPD784021GC-3B9	80-pin plastic QFP (14 × 14 mm)	Standard
μPD784021GK-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
μPD784025GC-×××-3B9	80-pin plastic QFP (14 × 14 mm)	Standard
μPD784026GC-×××-3B9	80-pin plastic QFP (14 × 14 mm)	Standard
μPD78P4026GC-3B9	80-pin plastic QFP (14 × 14 mm)	Standard
μPD78P4026GC-×××-3B9 Note	80-pin plastic QFP (14 × 14 mm)	Standard
μPD78P4026KK-T	80-pin ceramic WQFN (14 × 14 mm)	Not applicable (for function evaluation)

Note QTOP microcontroller. "QTOP microcontroller" is a general term for a single-chip microcontroller with on-chip one-time PROM, for which total support is provided by NEC programming service from programming to marking, screening, and verification.

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

Caution The EPROM version of the μPD78P4026 does not have a level of reliability intended for volume production of customers' equipment, and should only be used for experimental or preproduction function evaluation.

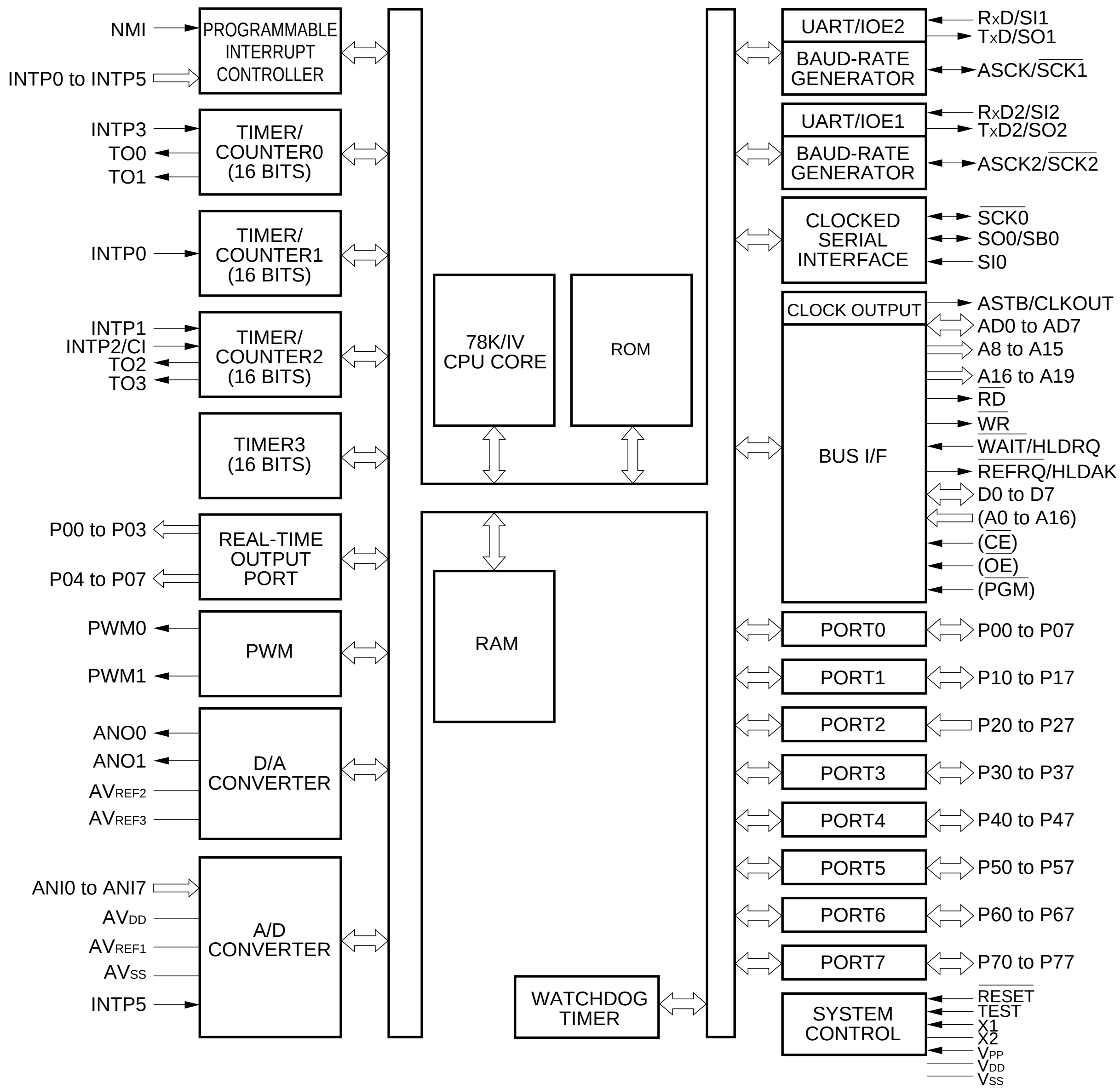
Remark ××× indicates ROM code suffix.

1.2.4 Outline of functions

Item		Product Name: μPD784020	μPD784021	μPD784025	μPD784026	μPD78P4026
Number of basic instructions (mnemonics)		113				
General registers		8 bits × 16 registers × 8 banks or 16 bits × 8 registers × 8 banks (memory mapped)				
Minimum instruction execution time		160 ns/320 ns/640 ns/1,280 ns (at 25-MHz operation)				
Internal memory capacity	ROM	None		48 Kbytes (Mask ROM)	64 Kbytes (Mask ROM)	64 Kbytes (PROM)
	RAM	512 bytes	2,048 bytes			
Memory space		1 Mbyte total both program and data				
I/O port	Total	46		64		
	Input	8		8		
	Input/output	34		56		
	Output	4		0		
Additional function pin Note	Pin with pull-up resistor	32		54		
	LED direct drive output	8		24		
	Transistor direct drive	8				
Real-time output port		4 bits × 2, or 8 bits × 1				
Timer/counter		Timer/counter 0: (16-bit) Timer register × 1 Compare register × 2 Capture register × 1 Pulse output capability • Toggle output • PWM/PPG output • One-shot pulse output				
		Timer/counter 1: (8-/16-bit) Timer register × 1 Compare register × 1 Capture register × 1 Capture/compare register × 1 Pulse output capability • Real-time output: 4 bits × 2				
		Timer/counter 2: (8-/16-bit) Timer register × 1 Compare register × 1 Capture/compare register × 1 Capture register × 1 Pulse output capability • Toggle output • PWM/PPG output				
		Timer 3: (8-/16-bit) Timer register × 1 Compare register × 1				
Watchdog timer		1 channel				
PWM output function		12-bit resolution × 2 channels				
Serial interface		• UART × IOE (3-wire serial I/O): 2 channels (on-chip baud rate generator) • CSI (3-wire serial I/O, SBI) : 1 channel				
A/D converter		8-bit resolution × 8 channels				
D/A converter		8-bit resolution × 2 channels				
Standby function		HALT/STOP/IDLE mode				
Interrupt	Hardware sources	23 (internal: 16, external: 7 (sampling clock variable input: 1))				
	Software sources	BRK instruction, BRKCS instruction, operand error				
	Non-maskable	Internal: 1, external: 1				
	Maskable	Internal: 15, external: 6				
		• 4-level programmable priority • 3 kinds of process mode (vectored interrupt/macro service/context switching)				
Clock output function		—		Selectable from f_{CLK}, $f_{CLK}/2$, $f_{CLK}/4$, $f_{CLK}/8$, $f_{CLK}/16$ (also usable as 1-bit output port)		
Power supply voltage		V_{DD} = 2.7 to 5.5 V				
Package		• 80-pin plastic QFP (14 × 14 mm) • 80-pin plastic TQFP (fine pitch, 12 × 12 mm: μPD784021 only) • 80-pin ceramic WQFN (14 × 14 mm: μPD78P4026 only)				

Note The pins with additional functions are included in the I/O pins.

1.2.5 Block diagram



Remarks 1. Internal ROM and RAM capacities vary depending on the products.

2. V_{PP} applies to the μPD78P4026 only.

3. The pins in parentheses are used in the PROM programming mode.

★ 1.3 Product Outline of μPD784038 Subseries (μPD784031, 784035, 784036, 784037, 784038, 78P4038, 784031(A), 784035(A), 784036(A))

1.3.1 Features

- Pins are compatible with μPD78234 Subseries, μPD784026 Subseries, and μPD784038Y Subseries
- On-chip memory capacity of μPD78234 Subseries and μPD784026 Subseries is expanded.
- Minimum instruction execution time 125 ns/250 ns/500 ns/1,000 ns (at 32-MHz operation)
- On-chip memory
  - ROM
    Mask ROM : None (μPD784031, 784031(A))
    48 Kbytes (μPD784035, 784035(A))
    64 Kbytes (μPD784036, 784036(A))
    96 Kbytes (μPD784037)
    128 Kbytes (μPD784038)
    PROM : 128 Kbytes (μPD78P4038)
  - RAM : 2,048 bytes (μPD784031, 784035, 784036, 784031(A), 784035(A), 784036(A))
    3,584 bytes (μPD784037)
    4,352 bytes (μPD784038)
- I/O port: 64
- Timer/counter: 16-bit timer/counter × 3 units
  16-bit timer × 1 unit
- Watchdog timer: 1 channel
- A/D converter: 8-bit resolution × 8 channels
- D/A converter: 8-bit resolution × 2 channels
- 12-bit PWM output: 2 channels
- Serial interface
  UART/IOE (3-wire serial I/O): 2 channels
  CSI (3-wire serial I/O, 2-wire serial I/O): 1 channel
- Interrupt controller (4-level priority)
  Vectored interrupt/macro service/context switching
- Standby function
  HALT/STOP/IDLE mode
- Clock output function
  Selectable from f_{CLK}, $f_{CLK}/2$, $f_{CLK}/4$, $f_{CLK}/8$, and $f_{CLK}/16$ (except μPD784031)
- Power supply voltage: V_{DD} = 2.7 to 5.5 V

1.3.2 Applications

- Standard-grade devices: Laser beam printers, autofocus cameras, plain paper copiers, printers, electronic typewriters, air conditioners, electronic musical instruments, cellular phones, etc.
- Special-grade devices: Control equipment in automobile electrical system, gas detector and cut off equipment, and various safety equipment.

1.3.3 Ordering information and quality grade

(1) Ordering information

	Part Number	Package	Internal ROM
	μPD784031GC-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	None
	μPD784031GC-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	None
★	μPD784031GC(A)-×××-3E9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	None
	μPD784031GK-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	None
	μPD784035GC-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Mask ROM
	μPD784035GC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Mask ROM
★	μPD784035GC(A)-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Mask ROM
	μPD784035GK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Mask ROM
	μPD784036GC-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Mask ROM
	μPD784036GC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Mask ROM
★	μPD784036GC(A)-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Mask ROM
	μPD784036GK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Mask ROM
	μPD784037GC-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Mask ROM
	μPD784037GC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Mask ROM
	μPD784037GK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Mask ROM
	μPD784038GC-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Mask ROM
	μPD784038GC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Mask ROM
	μPD784038GK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Mask ROM
	μPD78P4038GC-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	One-time PROM
	μPD78P4038GC-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	One-time PROM
	μPD78P4038GC-×××-3B9 Note	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Preprogramming one-time PROM
	μPD78P4038GC-×××-8BT Note	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Preprogramming one-time PROM
	μPD78P4038GK-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	One-time PROM
	μPD78P4038GK-×××-BE9 Note	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Preprogramming one-time PROM
	μPD78P4038KK-T	80-pin ceramic WQFN (14 × 14 mm)	EPROM

Note QTOP microcontrollers. "QTOP microcontroller" is a general term for a single-chip microcontroller with on-chip one-time ROM, for which total support is provided by NEC programming service, from programming to marking, screening, and verification.

Remark ××× indicates ROM code suffix.

(2) Quality grades

	Part Number	Package	Quality Grade
	μPD784031GC-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Standard
	μPD784031GC-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Standard
	μPD784031GC-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
	μPD784035GC-××× -3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Standard
	μPD784035GC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Standard
	μPD784035GK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
	μPD784036GC-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Standard
	μPD784036GC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Standard
	μPD784036GK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
	μPD784037GC-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Standard
	μPD784037GC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Standard
	μPD784037GK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
	μPD784038GC-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Standard
	μPD784038GC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Standard
	μPD784038GK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
	μPD78P4038GC-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Standard
	μPD78P4038GC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Standard
	μPD78P4038GC-×××-3B9 Note	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Standard
	μPD78P4038GC-×××-8BT Note	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Standard
	μPD78P4038GK-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
	μPD78P4038GK-×××-BE9 Note	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
★	μPD784031GC(A)-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Special
★	μPD784035GC(A)-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Special
★	μPD784036GC(A)-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Special
	μPD78P4038KK-T	80-pin ceramic WQFN (14 × 14 mm)	Not applicable (for function evaluation)

Note QTOP microcontrollers. "QTOP microcontroller" is a general term for a single-chip microcontroller with on-chip one-time ROM, for which total support is provided by NEC programming service, from programming to marking, screening, and verification.

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

Remark ××× indicates ROM code suffix.

Caution The EPROM version of the μPD78P4028 does not have a level of reliability intended for volume production of customer's equipment, and should only be used for experimental or preproduction function evaluation.

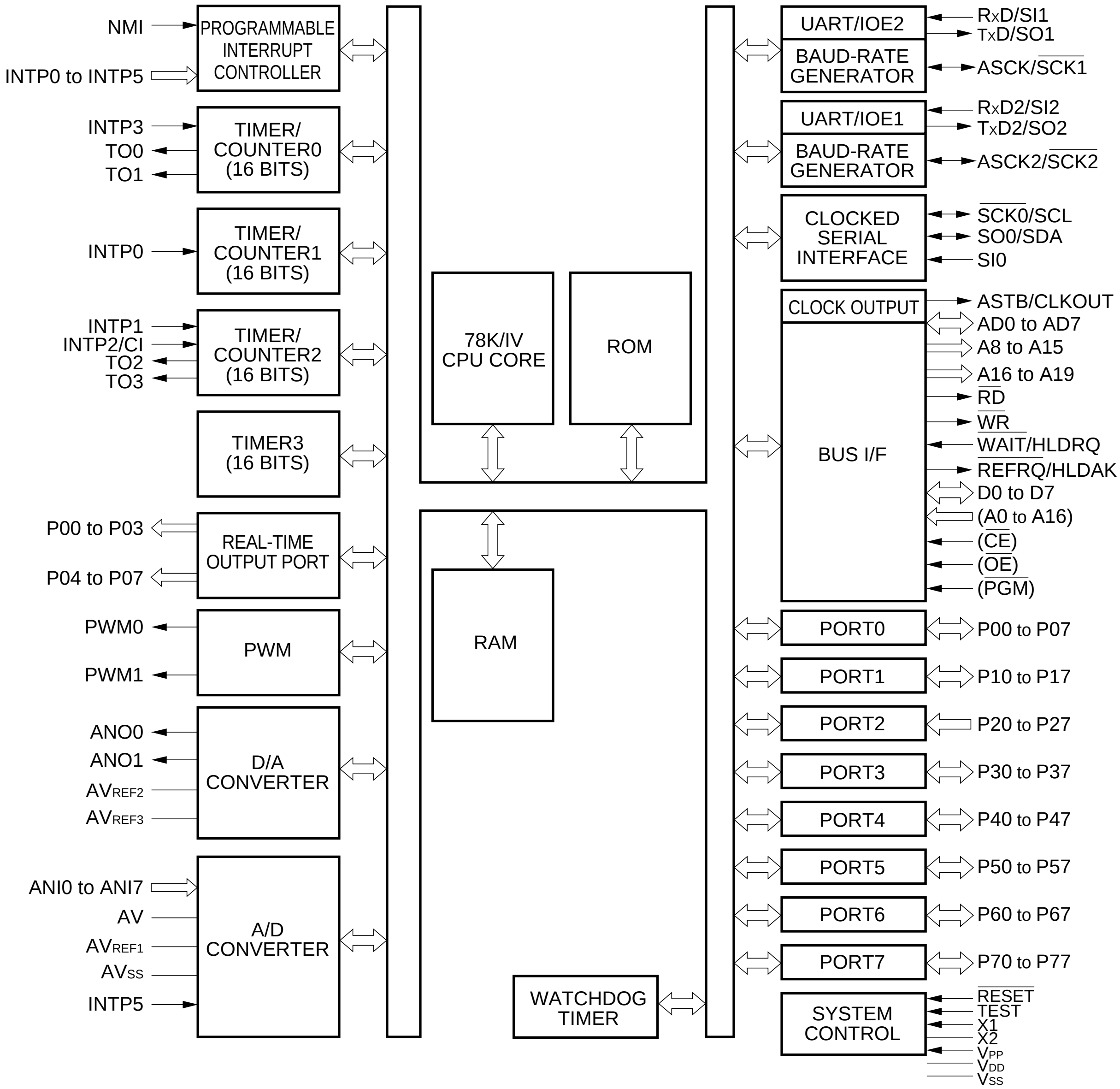
1.3.4 Outline of functions

★

Item		Product Name: μPD784031, 784031(A)	μPD784035, 784035(A)	μPD784036, 784036(A)	μPD784037	μPD784038	μPD78P4038
Number of basic instructions (mnemonics)		113					
General registers		8 bits × 16 registers × 8 banks or 16 bits × 8 registers × 8 banks (memory mapped)					
Minimum instruction execution time		125 ns/250 ns/500 ns/1,000 ns (at 32-MHz operation)					
Internal memory capacity	ROM	None	48 Kbytes (Mask ROM)	64 Kbytes (Mask ROM)	96 Kbytes (Mask ROM)	128 Kbytes (Mask ROM)	128 Kbytes (One-time PROM or EPROM)
	RAM	2,048 bytes			3,584 bytes	4,352 bytes	
Memory space		1 Mbyte total both programs and data					
I/O port	Total	64					
	Input	8					
	Input/Output	56					
Additional function pin Note	Pin with pull-up resistor	54					
	LED direct drive output	24					
	Transistor direct drive	8					
Real-time output port		4 bits × 2, or 8 bits × 1					
Timer/counter		Timer/counter 0: (16-bit)	Timer register × 1 Capture register × 1 Compare register × 2	Pulse output capability • Toggle output • PWM/PPG output • One-shot pulse output			
		Timer/counter 1: (8/16-bit)	Timer register × 1 Capture register × 1 Capture/compare register × 1 Compare register × 1	Pulse output capability • Real-time output (4 bits × 2)			
		Timer/counter 2: (8/16-bit)	Timer register × 1 Capture register × 1 Capture/compare register × 1 Compare register × 1	Pulse output capability • Toggle output • PWM/PPG output			
		Timer 3: (8/16-bit)	Timer register × 1 Compare register × 1				
PWM output		12-bit resolution × 2 channels					
Serial interface		UART/IOE (3-wire serial I/O) : 2 channels (on-chip baud rate generator) CSI (3-wire serial I/O, 2-wire serial I/O): 1 channel					
A/D converter		8-bit resolution × 8 channels					
D/A converter		8-bit resolution × 2 channels					
Clock output		—	Selectable from f_{CLK}, $f_{CLK}/2$, $f_{CLK}/4$, $f_{CLK}/8$, and $f_{CLK}/16$ (also usable as 1-bit output port)				
Watchdog timer		1 channel					
Standby function		HALT/STOP/IDLE mode					
Interrupt	Hardware sources	23 (internal: 16, external: 7 (sampling clock variable input: 1))					
	Software sources	BRK instruction, BRKCS instruction, operand error					
	Non-maskable	Internal: 1, external: 1					
	Maskable	Internal: 15, external: 6					
		• 4-level programmable priority • 3 processing modes (vectored interrupt, macro service, context switching)					
Power supply voltage		V_{DD} = 2.7 to 5.5 V					
Package		• 80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm) • 80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm) • 80-pin plastic TQFP (fine pitch) (12 × 12 mm) • 80-pin ceramic WQFN (14 × 14 mm): μPD78P4038 only					

Note The pins with additional functions are included in the I/O pins.

1.3.5 Block diagram



Remarks 1. Internal ROM and RAM capacities vary depending on the products.

2. V_{PP} applies to the μPD78P4038 only.

3. The pins in parentheses are used in the PROM programming mode

1.4 Product Outline of μPD784038Y Subseries (μPD784031Y, 784035Y, 784036Y, 784037Y, 784038Y, 78P4038Y)

1.4.1 Features

- I^2C bus control function is added to μPD784038.
- Pins are compatible with μPD78234 Subseries, μPD784026 Subseries, and μPD784038.
- On-chip memory capacity of μPD78234 Subseries and μPD784026 Subseries is expanded.
- Minimum instruction execution time: 125 ns/250 ns/500 ns/1,000 ns (at 32-MHz operation)
- On-chip memory
  - ROM
    - Mask ROM : None (μPD784031Y)
      48 Kbytes (μPD784035Y)
      64 Kbytes (μPD784036Y)
      96 Kbytes (μPD784037Y)
      128 Kbytes (μPD784038Y)
    - PROM : 128 Kbytes (μPD78P4038Y)
  - RAM : 2,048 bytes (μPD784031Y, 784035Y, 784036Y)
    3,584 bytes (μPD784037Y)
    4,352 bytes (μPD784038Y)
- I/O port: 64
- Timer/counter: 16-bit timer/counter × 3 units
  16-bit timer × 1 unit
- Watchdog timer: 1 channel
- A/D converter: 8-bit resolution × 8 channels
- D/A converter: 8-bit resolution × 2 channels
- 12-bit PWM output: 2 channels
- Serial interface
  UART/IOE (3-wire serial I/O): 2 channels
  CSI (3-wire serial I/O, 2-wire serial I/O, I^2C bus): 1 channel
- Interrupt controller (4-level priority)
  Vectored interrupt/macro service/context switching
- Standby function
  HALT/STOP/IDLE modes
- Clock output function
  Selectable from f_{CLK}, $f_{CLK}/2$, $f_{CLK}/4$, $f_{CLK}/8$,and $f_{CLK}/16$ (except μPD784031Y)
- Power supply voltage: V_{DD} = 2.7 to 5.5 V

1.4.2 Applications

Cellular phones, cordless phones, audiovisual equipment, etc.

1.4.3 Ordering information and quality grade

(1) Ordering information

Part Number	Package	Internal ROM
μPD784031YGC-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	None
μPD784031YGC-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	None
μPD784031YGK-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	None
μPD784035YGC-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Mask ROM
μPD784035YGC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Mask ROM
μPD784035YGK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Mask ROM
μPD784036YGC-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Mask ROM
μPD784036YGC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Mask ROM
μPD784036YGK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Mask ROM
μPD784037YGC-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Mask ROM
μPD784037YGC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Mask ROM
μPD784037YGK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Mask ROM
μPD784038YGC-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Mask ROM
μPD784038YGC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Mask ROM
μPD784038YGK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Mask ROM
μPD78P4038YGC-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	One-time PROM
μPD78P4038YGC-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	One-time PROM
μPD78P4038YGC-×××-3B9 Note	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Preprogramming one-time PROM
μPD78P4038YGC-×××-8BT Note	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Preprogramming one-time PROM
μPD78P4038YGK-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	One-time PROM
μPD78P4038YGK-×××-BE9 Note	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Preprogramming one-time PROM
μPD78P4038YKK-T	80-pin ceramic WQFN (14 × 14 mm)	EPROM

Note QTOP microcontrollers. “QTOP microcontroller” is a general term for a single-chip microcontroller with on-chip one-time ROM, for which total support is provided by NEC programming service, from programming to marking, screening, and verification.

Remark ××× indicates ROM code suffix.

Caution μPD784035YGK-×××-BE9 and μPD784036YGK-×××-BE9 are under development.

(2) Quality grades

Part Number	Package	Quality Grade
μPD784031YGC-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Standard
μPD784031YGC-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Standard
μPD784031YGK-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
μPD784035YGC-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Standard
μPD784035YGC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Standard
μPD784035YGK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
μPD784036YGC-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Standard
μPD784036YGC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Standard
μPD784036YGK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
μPD784037YGC-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Standard
μPD784037YGC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Standard
μPD784037YGK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
μPD784038YGC-×××-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Standard
μPD784038YGC-×××-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Standard
μPD784038YGK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
μPD78P4038YGC-3B9	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Standard
μPD78P4038YGC-8BT	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Standard
μPD78P4038YGC-×××-3B9 Note	80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm)	Standard
μPD78P4038YGC-×××-8BT Note	80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm)	Standard
μPD78P4038YGK-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
μPD78P4038YGK-×××-BE9 Note	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
μPD78P4038YKK-T	80-pin ceramic WQFN (14 × 14 mm)	Not applicable (for function evaluation)

Note QTOP microcontrollers. "QTOP microcontroller" is a general term for a single-chip microcontroller with on-chip one-time ROM, for which total support is provided by NEC programming service, from programming to marking, screening, and verification.

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

Remark ××× indicates ROM code suffix.

Cautions 1. The EPROM version of the μPD78P4028 dose not have a level of reliability intended for volume production of customer's equipment, and should only be used for experimental or preproduction function evaluation.

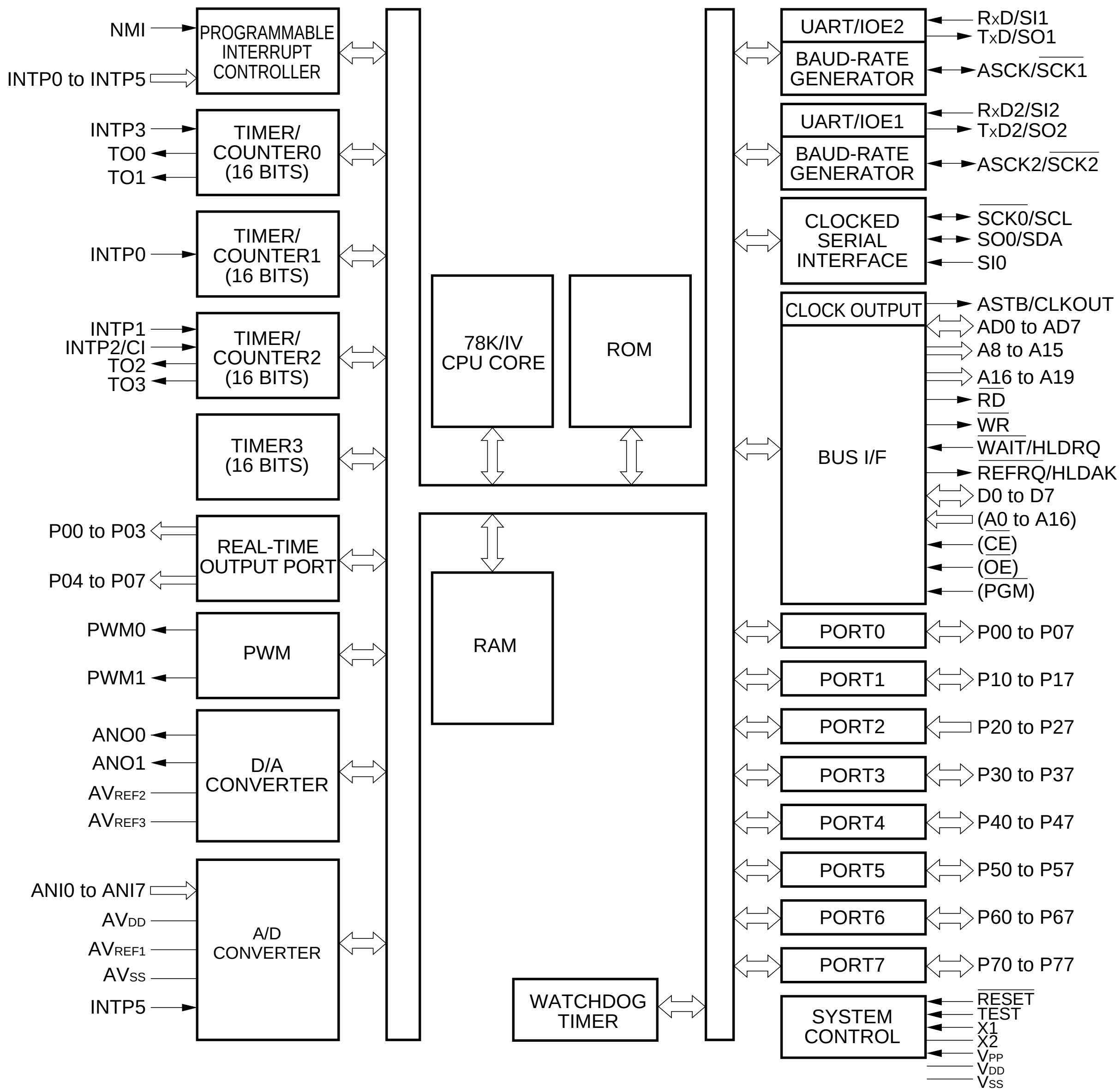
2. μPD784035YGK-×××-BE9 and μPD784036YGK-×××-BE9 are under development.

1.4.4 Outline of functions

Item			Product Name: μPD784031Y	μPD784035Y	μPD784036Y	μPD784037Y	μPD784038Y	μPD78P4038Y
Number of basic instructions (mnemonics)			113					
General registers			8 bits × 16 registers × 8 banks or 16 bits × 8 registers × 8 banks (memory mapped)					
Minimum instruction execution time			125 ns/250 ns/500 ns/1,000 ns (at 32-MHz operation)					
Internal memory capacity	ROM		None	48 Kbytes (Mask ROM)	64 Kbytes (Mask ROM)	96 Kbytes (Mask ROM)	128 Kbytes (Mask ROM)	128 Kbytes (One-time PROM or EPROM)
	RAM		2,048 bytes			3,584 bytes	4,352 bytes	
Memory space			1 Mbyte total both programs and data					
I/O port	Total		64					
	Input		8					
	Input/Output		56					
	Additional function pin Note	Pin with pull-up resistor	54					
		LED direct drive output	24					
		Transistor direct drive	8					
Real-time output port			4 bits × 2, or 8 bits × 1					
Timer/counter			Timer/counter 0: Timer register × 1 Capture register × 1 Compare register × 2 Pulse output capability • Toggle output • PWM/PPG output • One-shot pulse output					
			Timer/counter 1: Timer register × 1 Capture register × 1 Capture/compare register × 1 Compare register × 1 Pulse output capability • Real-time output (4 bits × 2)					
			Timer/counter 2: Timer register × 1 Capture register × 1 Capture/compare register × 1 Compare register × 1 Pulse output capability • Toggle output • PWM/PPG output					
			Timer 3: Timer register × 1 Compare register × 1					
PWM output			12-bit resolution × 2 channels					
Serial interface			UART/IOE (3-wire serial I/O) : 2 channels (on-chip baud rate generator) CSI (3-wire serial I/O, 2-wire serial I/O, I^{2}C bus) : 1 channel					
A/D converter			8-bit resolution × 8 channels					
D/A converter			8-bit resolution × 2 channels					
Clock output			—	Selectable from f_{CLK}, $f_{CLK}/2$, $f_{CLK}/4$, $f_{CLK}/8$,and $f_{CLK}/16$ (also usable as 1-bit output port)				
Watchdog timer			1 channel					
Standby function			HALT/STOP/IDLE mode					
Interrupt	Hardware sources		24 (internal: 17, external: 7 (sampling clock variable input: 1))					
	Software sources		BRK instruction, BRKCS instruction, operand error					
	Non-maskable		Internal: 1, external: 1					
	Maskable		Internal: 16, external: 6					
			• 4-level programmable priority • 3 processing modes (vectored interrupt, macro service, context switching)					
Power supply voltage			V_{DD} = 2.7 to 5.5 V					
Package			• 80-pin plastic QFP (14 × 14 mm, thickness: 1.4 mm) • 80-pin plastic QFP (14 × 14 mm, thickness: 2.7 mm) • 80-pin plastic TQFP (fine pitch) (12 × 12 mm) • 80-pin ceramic WQFN (14 × 14 mm): μPD78P4038Y only					

Note The pins with additional functions are included in the I/O pins.

1.4.5 Block diagram



Remarks 1. Internal ROM and RAM capacities vary depending on the products.

2. V_{PP} applies to the μPD78P4038Y only.

3. The pins in parenthesis are used in the PROM programming mode.

★ 1.5 Product Outline of μPD784046 Subseries (μPD784044, 784054, 784046, 78F4046, 784044(A), 784044(A1), 784044(A2), 784046(A), 784046(A1), 784046(A2), 784054(A), 784054(A1), 784054(A2))

1.5.1 Features

- Minimum instruction execution time:
  125 ns (at internal 16-MHz operation) μPD784044, 784046, 784054, 78F4046
  160 ns (at internal 12.5-MHz operation) .... μPD784044(A), 784046(A), 784054(A)
  200 ns (at internal 10-MHz operation) .... μPD784044(A1), (A2), 784046(A1), (A2), 784054(A1), (A2)
- On-chip memory
  - ROM
    Mask ROM : 64 Kbytes (μPD784046, 784046(A), (A1), (A2))
    : 32 Kbytes (μPD784044, 784044(A), (A1), (A2), 784054, 784054(A), (A1), (A2))
    Flash memory : 64 Kbytes (μPD78F4046)
  - RAM : 2,048 bytes (μPD784046, 784046(A), (A1), (A2), 78F4046)
    1,024 bytes (μPD784044, 784044(A), (A1), (A2), 784054, 784054(A), (A1), (A2))
- I/O port: 65 (64 for only μPD784054 and 784054(A), (A1), (A2))
- Timer/counter: 16-bit timer/counter × 2 units
  16-bit timer × 3 units
  (only 16-bit timer × 3 units for μPD784054 and 784054(A), (A1), (A2))
- Watchdog timer: 1 channel
- A/D converter: 10-bit resolution × 16 channels (V_{DD} = 4.5 to 5.5 V)
- Serial interface
  UART/IOE (3-wire serial I/O): 2 channels (on-chip baud rate generator)
- Interrupt controller (4-level priority)
  Vectored interrupt/macro service/context switching
- Standby function
  HALT/STOP/IDLE mode (/standby invalid function mode … μPD784054 and 784054(A), (A1), (A2) only)
- Power supply voltage: V_{DD} = 4.0 to 5.5 V

1.5.2 Applications

- Standard: Water heaters, vending machines, office automation equipment such as PPCs or printers, and factory automation equipment such as robots or automation machine tools
- Special: Automobile electrical systems, etc.

★ ### 1.5.3 Ordering information and quality grade

(1) Ordering information

Part Number	Package	Internal ROM
μPD784044GC-××× -3B9	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784044GC(A)-××× -3B9	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784044GC(A1)-××× -3B9	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784044GC(A2)-××× -3B9	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784046GC-××× -3B9 Note	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784046GC(A)-××× -3B9 Note	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784046GC(A1)-××× -3B9 Note	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784046GC(A2)-××× -3B9 Note	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784054GC-××× -3B9	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784054GC(A)-××× -3B9	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784054GC(A1)-××× -3B9	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784054GC(A2)-××× -3B9	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD78F4046GC-3B9 Note	80-pin plastic QFP (14 × 14 mm)	Flash Memory

Remark ××× indicates ROM code suffix.

(2) Quality grades

Part Number	Package	Quality Grade
μPD784044GC-××× -3B9	80-pin plastic QFP (14 × 14 mm)	Standard
μPD784046GC-××× -3B9 Note	80-pin plastic QFP (14 × 14 mm)	Standard
μPD784054GC-××× -3B9	80-pin plastic QFP (14 × 14 mm)	Standard
μPD78F4046GC-3B9 Note	80-pin plastic QFP (14 × 14 mm)	Standard
μPD784044GC(A)-××× -3B9	80-pin plastic QFP (14 × 14 mm)	Special
μPD784044GC(A1)-××× -3B9	80-pin plastic QFP (14 × 14 mm)	Special
μPD784044GC(A2)-××× -3B9	80-pin plastic QFP (14 × 14 mm)	Special
μPD784046GC(A)-××× -3B9 Note	80-pin plastic QFP (14 × 14 mm)	Special
μPD784046GC(A1)-××× -3B9 Note	80-pin plastic QFP (14 × 14 mm)	Special
μPD784046GC(A2)-××× -3B9 Note	80-pin plastic QFP (14 × 14 mm)	Special
μPD784054GC(A)-××× -3B9	80-pin plastic QFP (14 × 14 mm)	Special
μPD784054GC(A1)-××× -3B9	80-pin plastic QFP (14 × 14 mm)	Special
μPD784054GC(A2)-××× -3B9	80-pin plastic QFP (14 × 14 mm)	Special

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

Note Under development

Remark ××× indicates ROM code suffix.

1.5.4 Outline of functions

★ **(1) μPD784044, 784044(A), (A1), (A2), 784046, 784046(A), (A1), (A2), 78F4046**

★

Item		Product Name: μPD784044, 784044(A), (A1), (A2)	μPD784046, 784046(A), (A1), (A2)	μPD78F4046
Number of basic instructions (mnemonics)		113		
General registers		8 bits × 16 registers × 8 banks or 16 bits × 8 registers × 8 banks (memory mapped)		
★ Minimum instruction execution time		125 ns (at internal clock 16-MHz operation) ..... μPD784044, 78F4046 160 ns (at internal clock 12.5-MHz operation) .. μPD784044(A), 784046(A) 200 ns (at internal clock 10-MHz operation) ..... μPD784044(A1), (A2), 784046(A1), (A2)		
On-chip memory capacity	ROM	32 Kbytes (Mask ROM)	64 Kbytes (Mask ROM)	64 Kbytes (Flash memory)
	RAM	1,024 bytes	2,048 bytes	
Memory space		1 Mbyte total both programs and data		
I/O port	Total	65		
	Input	17		
	Input/Output	48		
Additional function pin Note	Pin with pull-up resistor	29		
Real-time output port		4 bits × 1		
Timer/counter		Timer 0:	Timer register × 1 Capture/compare register × 4	Pulse output capability • Toggle output • Set/Reset output
		Timer 1:	Timer register × 1 Compare register × 2	Pulse output capability • Toggle output • Set/Reset output
		Timer/counter 2:	Timer register × 1 Compare register × 2	Pulse output capability • Toggle output • PWM/PPG output
		Timer/counter 3:	Timer register × 1 Compare register × 2	Pulse output capability • Toggle output • PWM/PPG output
		Timer 4:	Timer register × 1 Compare register × 2	Pulse output capability • Real-time output (4 bits × 1)
A/D converter		10-bit resolution × 16 channels (AV_{DD} = 4.5 to 5.5 V)		
Serial interface		UART/IOE (3-wire serial I/O): 2 channels (on-chip baud rate generator)		
Watchdog timer		1 channel		
Interrupt	Hardware sources	27 (internal: 23, external: 8 (compatible with internal: 4))		
	Software sources	BRK instruction, BRKCS instruction, operand error		
	Non-maskable	Internal: 1, external: 1		
	Maskable	Internal: 22, external: 7 (compatible with internal: 4)		
		• 4-level programmable priority • 3 processing modes (vectored interrupt, macro service, context switching)		
Bus sizing function		8-bit/16-bit external data bus selectable		
Standby function		HALT/STOP/IDLE mode		
Power supply voltage		V_{DD} = 4.0 to 5.5 V		
Package		80-pin plastic QFP (14 × 14 mm)		

Note The pins with additional functions are included in the I/O pins.

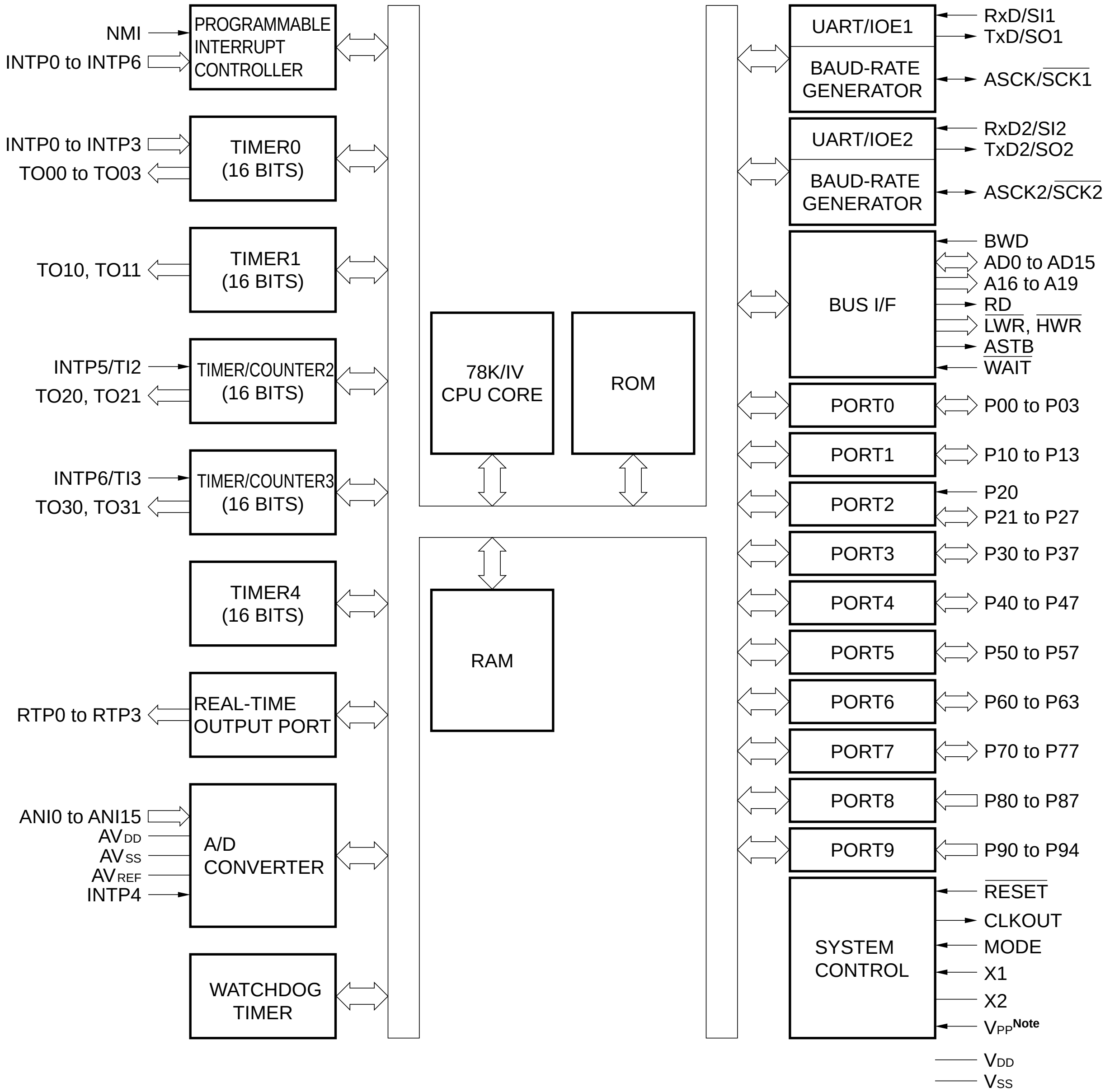
★ **(2) μPD784054, 784054(A), (A1), (A2)**

Item \ Product Name			μPD784054, 784054(A), (A1), (A2)		
Number of basic instructions (mnemonics)			113		
General registers			8 bits × 16 registers × 8 banks or 16 bits × 8 registers × 8 banks (memory mapped)		
★ Minimum instruction execution time			125 ns (at internal clock 16-MHz operation) ...... μPD784054 160 ns (at internal clock 12.5-MHz operation) ... μPD784054(A) 200 ns (at internal clock 10-MHz operation) ...... μPD784054(A1), (A2)		
On-chip memory capacity		ROM	32 Kbytes (Mask ROM)		
		RAM	1,024 bytes		
Memory space			1 Mbyte total both programs and data		
I/O port	Total		64		
	Input		17		
	Input/Output		47		
Additional function pin [Note]	Pin with pull-up resistor		29		
Timer			Timer 0: (16-bit)	Timer register × 1 Capture/compare register × 4	Pulse output capability • Toggle output • Set/Reset output
			Timer 1: (16-bit)	Timer register × 1 Compare register × 2	Pulse output capability • Toggle output • Set/Reset output
			Timer 4: (16-bit)	Timer register × 1 Compare register × 2	
A/D converter			10-bit resolution × 16 channels (AV_{DD} = 4.5 to 5.5 V)		
Serial interface			UART/IOE (3-wire serial I/O): 2 channels (on-chip baud rate generator)		
Watchdog timer			1 channel		
Interrupt	Hardware sources		23 (internal: 19, external: 8 (compatible with internal: 4))		
	Software sources		BRK instruction, BRKCS instruction, operand error		
	Non-maskable		Internal: 1, external: 1		
	Maskable		Internal: 18, external: 7 (compatible with internal: 4)		
			• 4-level programmable priority • 3 processing modes (vectored interrupt, macro service, context switching)		
Bus sizing function			8-bit/16-bit external data bus selectable		
Standby function			HALT/STOP/IDLE mode/standby invalid function mode		
Power supply voltage			V_{DD} = 4.0 to 5.5 V		
Package			80-pin plastic QFP (14 × 14 mm)		

Note The pins with additional functions are included in the I/O pins.

1.5.5 Block diagram

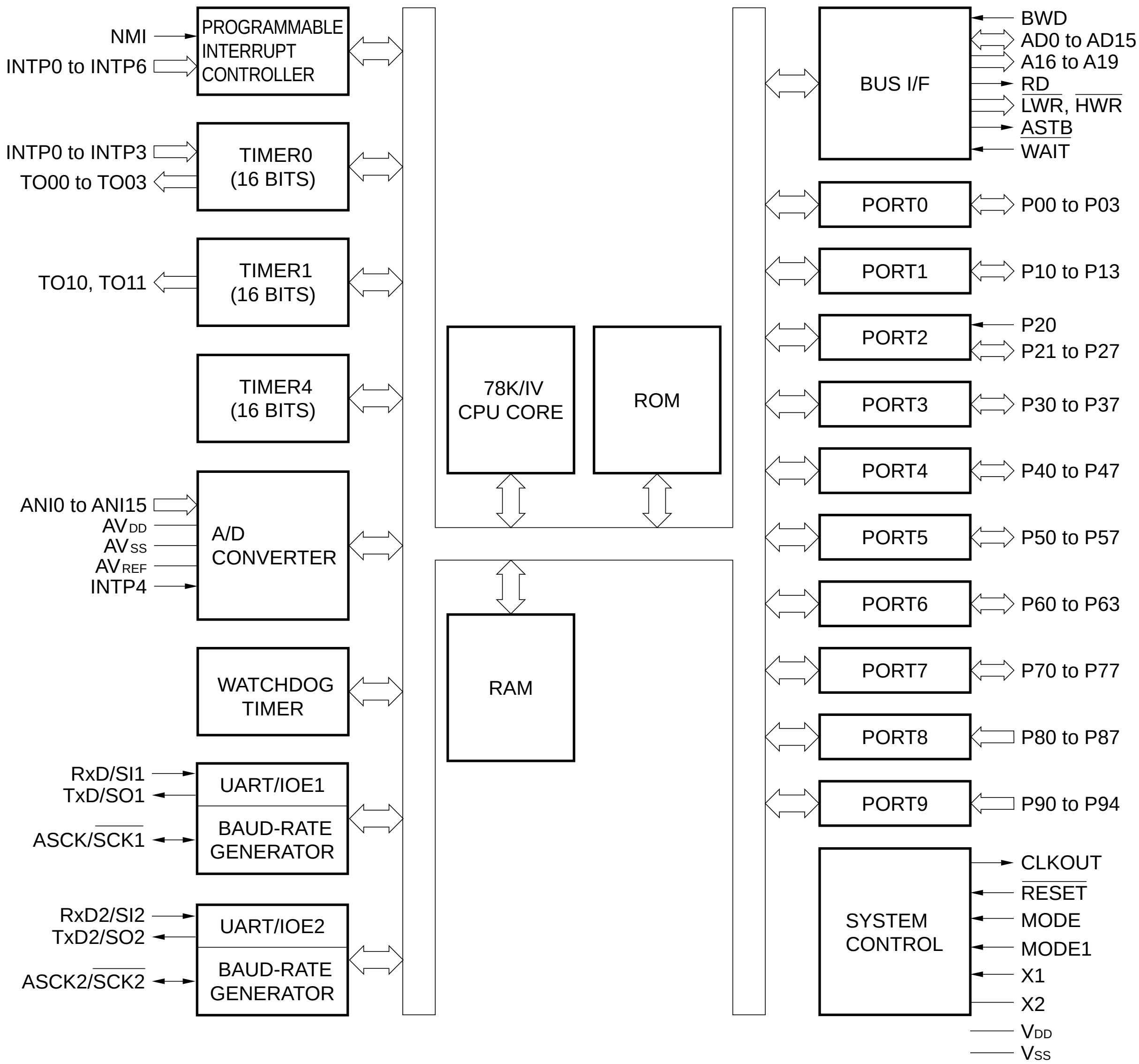
(1) μPD784044, 784044(A), (A1), (A2), 784046, 784046(A), (A1), (A2), 78F4046



Note V_{PP} applies to the μPD78F4046 only.

Remark Internal ROM and RAM capacities vary depending on the products.

(2) μPD784054, 784054(A), (A1), (A2)



1.6 Product Outline of μPD784216 Subseries (μPD784214, 784215, 784216, 78F4216)

1.6.1 Features

- Peripheral functions of μPD78078 are inherited
- Minimum instruction execution time: 160 ns (at 12.5-MHz main system clock operation)
  61 μs (at 32.768-kHz subsystem clock operation)
- On-chip memory
  - ROM
    Mask ROM : 96 Kbytes (μPD784214)
    128 Kbytes (μPD784215, 784216)
    Flash memory : 128 Kbytes (μPD78F4216)
  - RAM : 3,584 bytes (μPD784214)
    : 5,120 bytes (μPD784215)
    4,352 bytes (μPD784216, 78F4216)
- I/O port : 86
- Timer/counter: 16-bit timer/counter × 1 unit
  8-bit timer/counter × 6 units
- Watch timer: 1 channel
- Watchdog timer: 1 channel
- A/D converter: 8-bit resolution × 8 channels
- D/A converter: 8-bit resolution × 2 channels
- Serial interface: 3 channels
  UART/IOE (3-wire serial I/O): 2 channels (on-chip baud rate generator)
  CSI (3-wire serial I/O): 1 channel
- Interrupt controller (4-level priority)
  Vectored interrupt/macro service/context switching
- Clock output function
  Selectable from f_{XX}, $f_{XX}/2$, $f_{XX}/2^2$, $f_{XX}/2^3$, $f_{XX}/2^4$, $f_{XX}/2^5$, $f_{XX}/2^6$, $f_{XX}/2^7$, f_{XT}
- Buzzer output function
  Selectable from $f_{XX}/2^{10}$, $f_{XX}/2^{11}$, $f_{XX}/2^{12}$, $f_{XX}/2^{13}$
- Standby function
  HALT/STOP/IDLE mode
  Low power consumption mode: HALT/IDLE mode (subsystem clock operation)
- Power supply voltage: V_{DD} = 1.8 to 5.5 V

1.6.2 Applications

Cellular phones, PHS, cordless phones, CD-ROMs, audiovisual equipment, etc.

1.6.3 Ordering information and quality grade

(1) Ordering information

	Part Number	Package	Internal ROM
	μPD784214GC-×××-8EU	100-pin plastic LQFP (fine pitch) (14 × 14 mm)	Mask ROM
	μPD784214GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
	μPD784215GC-×××-8EU	100-pin plastic LQFP (fine pitch) (14 × 14 mm)	Mask ROM
	μPD784215GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
	μPD784216GC-×××-8EU	100-pin plastic LQFP (fine pitch) (14 × 14 mm)	Mask ROM
	μPD784216GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
★	μPD78F4216GC-8EU	100-pin plastic LQFP (fine pitch) (14 × 14 mm)	Flash memory
	μPD78F4216GF-3BA	100-pin plastic QFP (14 × 20 mm)	Flash memory

Remark ××× indicates ROM code suffix.

★ **Caution μPD78F4216GC-8EU and μPD78F4216GF-3BA are under development.**

(2) Quality grades

	Part Number	Package	Quality Grade
	μPD784214GC-×××-8EU	100-pin plastic LQFP (fine pitch) (14 × 14 mm)	Standard
	μPD784214GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
	μPD784215GC-×××-8EU	100-pin plastic LQFP (fine pitch) (14 × 14 mm)	Standard
	μPD784215GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
	μPD784216GC-×××-8EU	100-pin plastic LQFP (fine pitch) (14 × 14 mm)	Standard
	μPD784216GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
★	μPD78F4216GC-8EU	100-pin plastic LQFP (fine pitch) (14 × 14 mm)	Standard
	μPD78F4216GF-3BA	100-pin plastic QFP (14 × 20 mm)	Standard

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

Remark ××× indicates ROM code suffix.

★ **Caution μPD78F4216GC-8EU and μPD78F4216GF-3BA are under development.**

1.6.4 Outline of functions

(1/2)

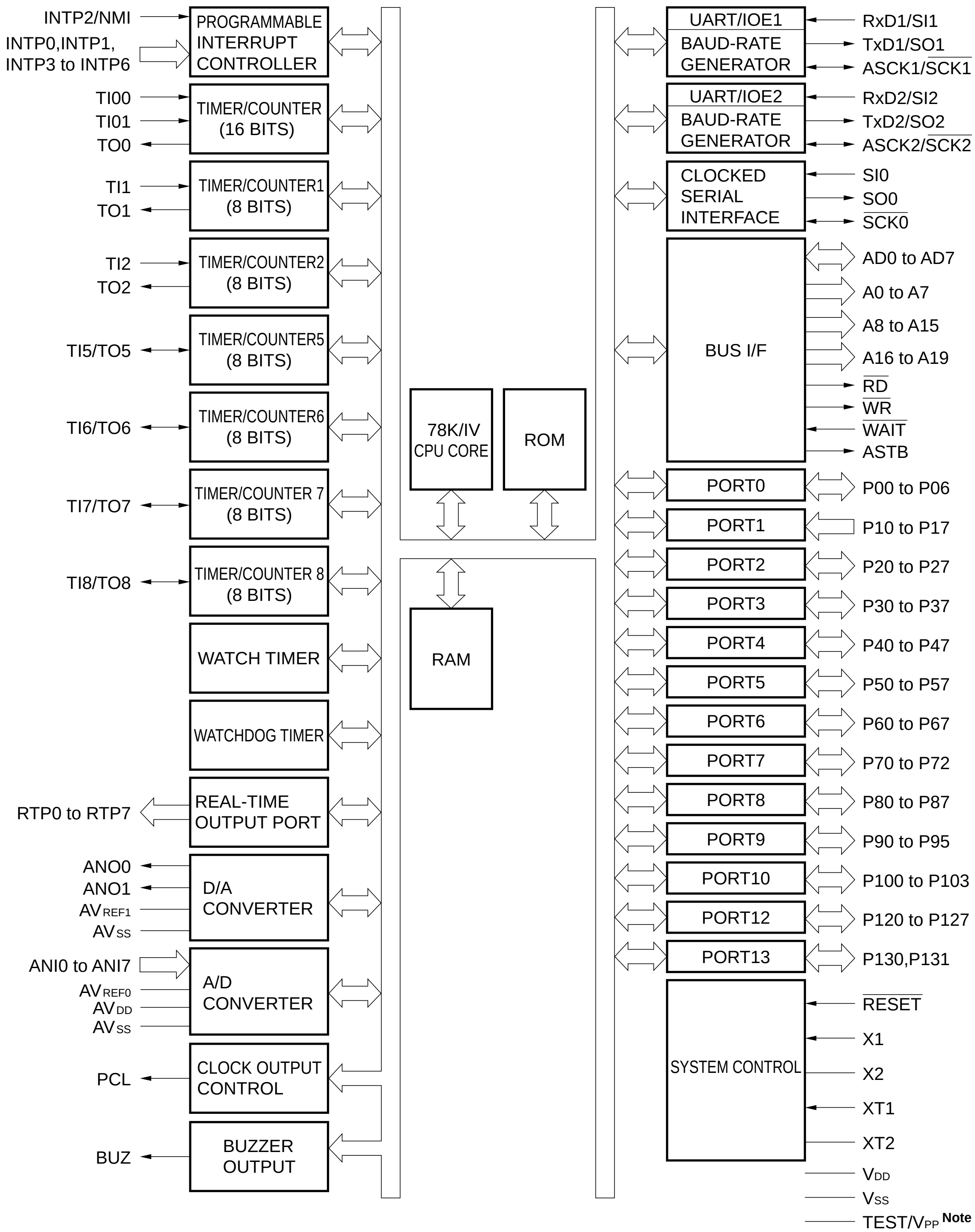
Item		Product Name: μPD784214	μPD784215	μPD784216	μPD78F4216
Number of basic instructions (mnemonics)		113			
General registers		8 bits × 16 registers × 8 banks or 16 bits × 8 registers × 8 banks (memory mapped)			
Minimum instruction execution time	When main system clock is selected	160 n/320 ns/640 ns/1,280 ns/2,560 ns (at 12.5-MHz operation)			
	When subsystem clock is selected	61 μs (at 32.768-kHz operation)			
Internal memory capacity	ROM	96 Kbytes (Mask ROM)	128 Kbytes (Mask ROM)	(Flash memory)	128 Kbytes
	RAM	3,584 bytes	5,120 bytes	8,192 bytes	
Memory space		1 Mbyte total both programs and data			
I/O port	Total	86			
	CMOS input	2			
	CMOS I/O	72			
	N-ch open-drain I/O	6			
Additional function pin Note	Pin with pull-up resistor	70			
	LED direct drive output	22			
	Medium voltage resistance pin	6			
Real-time output port		4 bits × 2, or 8 bits × 1			
Timer/counter		16-bit timer/counter: Timer register × 1, Capture/compare register × 2	Pulse output capability • PWM/PPG output • Square wave output • One-shot pulse output		
		8-bit timer/counter 1: Timer register × 1, Compare register × 1	Pulse output capability • PWM output • Square wave output		
		8-bit timer/counter 2: Timer register × 1, Compare register × 1	Pulse output capability • PWM output • Square wave output		
		8-bit timer/counter 5: Timer register × 1, Compare register × 1	Pulse output capability • PWM output • Square wave output		
		8-bit timer/counter 6: Timer register × 1, Compare register × 1	Pulse output capability • PWM output • Square wave output		
		8-bit timer/counter 7: Timer register × 1, Compare register × 1	Pulse output capability • PWM output • Square wave output		
		8-bit timer/counter 8: Timer register × 1, Compare register × 1	Pulse output capability • PWM output • Square wave output		

Note The pins with additional functions are included in the I/O pins.

(2/2)

Item		μPD784214	μPD784215	μPD784216	μPD78F4216
A/D converter		8-bit resolution × 8 channels			
D/A converter		8-bit resolution × 2 channels			
Serial interface		UART/IOE (3-wire serial I/O): 2 channels (baud rate generator on-chip) CSI (3-wire serial I/O): 1 channel			
Clock output		Selectable from f_{XX}, $f_{XX}/2$, $f_{XX}/2^2$, $f_{XX}/2^3$, $f_{XX}/2^4$, $f_{XX}/2^5$, $f_{XX}/2^6$, $f_{XX}/2^7$, f_{XT}			
Buzzer output		Selectable from $f_{XX}/2^{10}$, $f_{XX}/2^{11}$, $f_{XX}/2^{12}$, $f_{XX}/2^{13}$			
Watch timer		1 channel			
Watchdog timer		1 channel			
Interrupt	Hardware sources	29 (internal: 20, external: 9)			
	Software sources	BRK instruction, BRKCS instructions, operand error			
	Non-maskable	Internal: 1, external: 1			
	Maskable	Internal: 19, external: 8			
		• 4-level programmable priority • 3 processing modes: vectored interrupt, macro service, context switching			
Standby function		• HALT/STOP/IDLE mode • Low power consumption mode (CPU can operate on subsystem clock): HALT/IDLE mode			
Power supply voltage		V_{DD} = 1.8 to 5.5 V			
★ Package		• 100-pin plastic LQFP (fine pitch) (14 × 14 mm) • 100-pin plastic QFP (14 × 20 mm)			

1.6.5 Block diagram



Note V_{PP} applies to the μPD78F4216 only.

Remark Internal ROM and RAM capacities vary depending on the products.

1.7 Product Outline of μPD784216Y Subseries (μPD784214Y, 784215Y, 784216Y, 78F4216Y)

1.7.1 Features

- I^2C bus interface is added to μPD784216 Subseries
- Minimum instruction execution time: 160 ns (main system clock: at 12.5-MHz operation)
  61 μs (subsystem clock: at 32.768-kHz operation)
- On-chip memory
  - ROM
    Mask ROM : 96 Kbytes (μPD784214Y)
    128 Kbytes (μPD784215Y, 784216Y)
    Flash Memory : 128 Kbytes (μPD78F4216Y)
  - RAM : 3,584 bytes (μPD784214Y)
    5,120 bytes (μPD784215Y)
    8,192 bytes (μPD784216Y, 78F4216Y)
- I/O port: 86
- Timer/counter: 16-bit timer/counter × 1 unit
  8-bit timer/counter × 6 units
- Watch timer: 1 channel
- Watchdog timer: 1 channel
- A/D converter: 8-bit resolution × 8 channels
- D/A converter: 8-bit resolution × 2 channels
- Serial interface: 3 channels
  UART/IOE (3-wire serial I/O): 2 channels (on-chip baud rate generator)
  CSI (3-wire serial I/O, multimaster supported I^2C bus): 1 channel
- Interrupt controller (4-level priority)
  Vectored interrupt/macro service/context switching
- Clock output functions
  Selectable from f_{XX}, $f_{XX}/2$, $f_{XX}/2^2$, $f_{XX}/2^3$, $f_{XX}/2^4$, $f_{XX}/2^5$, $f_{XX}/2^6$, $f_{XX}/2^7$, f_{XT}
- Buzzer output functions
  Selectable from $f_{XX}/2^{10}$, $f_{XX}/2^{11}$, $f_{XX}/2^{12}$, $f_{XX}/2^{13}$
- Standby function
  HALT/STOP/IDLE mode
  Low power consumption mode: HALT/IDLE mode (subsystem clock operation)
- Power supply voltage: V_{DD} = 1.8 to 5.5 V

1.7.2 Applications

Cellular phones, PHS, cordless phones, CD-ROMs, audiovisual equipment, etc.

1.7.3 Ordering information and quality grade

(1) Ordering information

	Part Number	Package	Internal ROM
	μPD784214YGC-×××-8EU	100-pin plastic LQFP (fine pitch) (14 × 14 mm)	Mask ROM
	μPD784214YGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
	μPD784215YGC-×××-8EU	100-pin plastic LQFP (fine pitch) (14 × 14 mm)	Mask ROM
	μPD784215YGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
	μPD784216YGC-×××-8EU	100-pin plastic LQFP (fine pitch) (14 × 14 mm)	Mask ROM
	μPD784216YGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
★	μPD78F4216YGC-8EU	100-pin plastic LQFP (fine pitch) (14 × 14 mm)	Flash memory
	μPD78F4216YGF-3BA	100-pin plastic QFP (14 × 20 mm)	Flash memory

Remark ××× indicates ROM code suffix.

★ **Caution μPD78F4216YGC-8EU and μPD78F4216YGF-3BA are under development.**

(2) Quality grades

	Part Number	Package	Quality Grade
	μPD784214YGC-×××-8EU	100-pin plastic LQFP (fine pitch) (14 × 14 mm)	Standard
	μPD784214YGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
	μPD784215YGC-×××-8EU	100-pin plastic LQFP (fine pitch) (14 × 14 mm)	Standard
	μPD784215YGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
	μPD784216YGC-×××-8EU	100-pin plastic LQFP (fine pitch) (14 × 14 mm)	Standard
	μPD784216YGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
★	μPD78F4216YGC-8EU	100-pin plastic LQFP (fine pitch) (14 × 14 mm)	Standard
	μPD78F4216YGF-3BA	100-pin plastic QFP (14 × 20 mm)	Standard

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

Remark ××× indicates ROM code suffix.

★ **Caution μPD78F4216YGC-8EU and μPD78F4216YGF-3BA are under development.**

1.7.4 Outline of functions

(1/2)

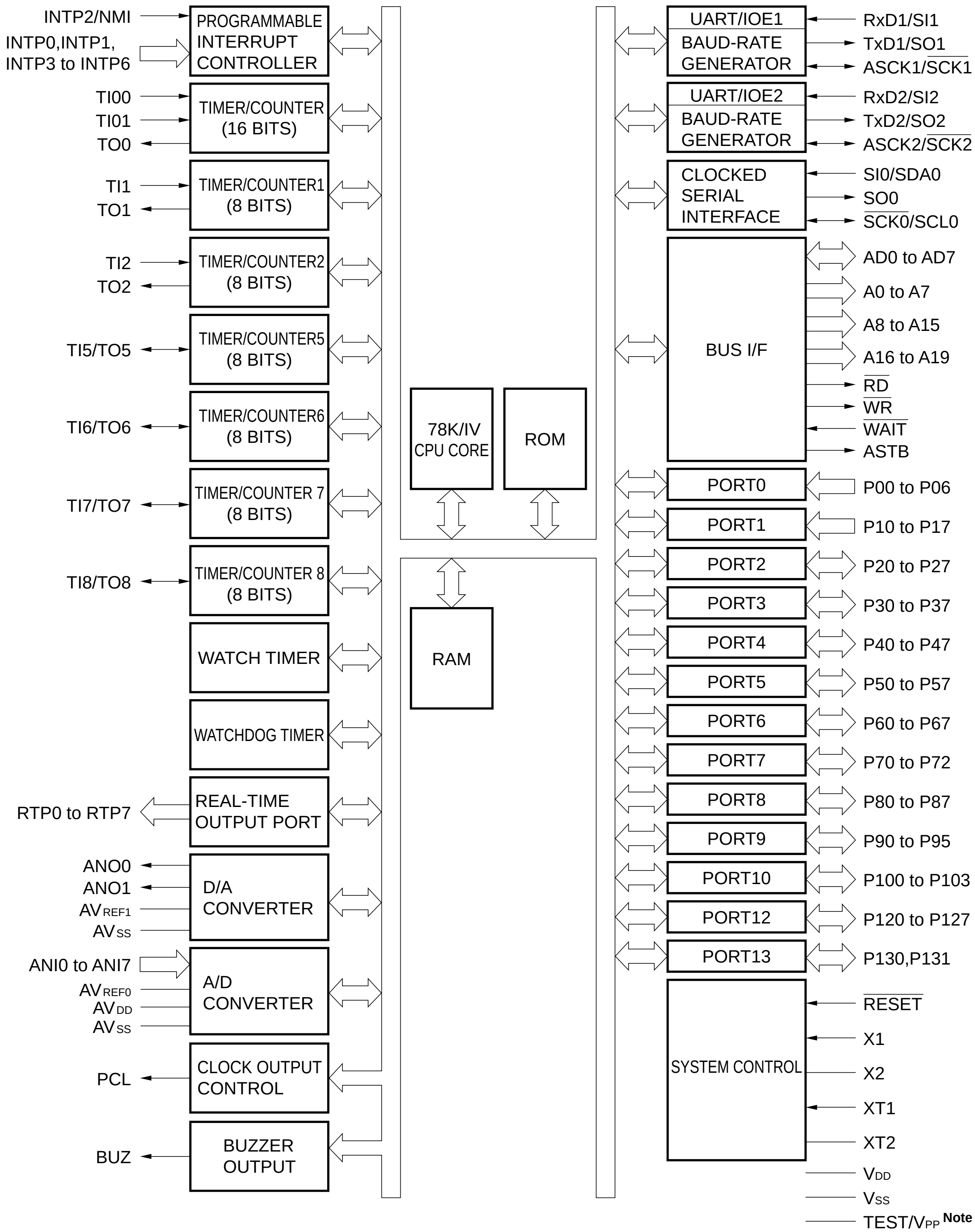
Item			μPD784214Y	μPD784215Y	μPD784216Y	μPD78F4216Y
Number of basic instructions (mnemonics)			113			
General registers			8 bits × 16 registers × 8 banks or 16 bits × 8 registers × 8 banks (memory mapped)			
Minimum instruction execution time	When main system clock is selected		160 ns/320 ns/640 ns/1,280 ns/2,560 ns (12.5-MHz operation)			
	When subsystem clock is selected		61 μs (32.768-kHz)			
Internal memory capacity	ROM (Mask ROM)		96 Kbytes (Mask ROM)	128 Kbytes (Flash memory)		128 Kbytes
	RAM		3,584 bytes	5,120 bytes	8,192 bytes	
Memory space			1 Mbyte total both programs and data			
I/O port	Total		86			
	CMOS input		2			
	CMOS I/O		72			
	N-ch open-drain I/O		6			
	Additional function pin Note	Pin with pull-up resistor	70			
		LED direct drive output	22			
		Medium voltage resistance pin	6			
Real-time output port			4 bits × 2, or 8 bits × 1			
Timer/counter			16-bit timer/counter:	Timer register × 1 Capture/compare register × 2	Pulse output capability • PWM/PPG output • Square wave output • One-shot pulse output	
			8-bit timer/counter 1:	Timer register × 1 Compare register × 1	Pulse output capability • PWM output • Square wave output	
			8-bit timer/counter 2:	Timer register × 1 Compare register × 1	Pulse output capability • PWM output • Square wave output	
			8-bit timer/counter 5:	Timer register × 1 Compare register × 1	Pulse output capability • PWM output • Square wave output	
			8-bit timer/counter 6:	Timer register × 1 Compare register × 1	Pulse output capability • PWM output • Square wave output	
			8-bit timer/counter 7:	Timer register × 1 Compare register × 1	Pulse output capability • PWM output • Square wave output	
			8-bit timer/counter 8:	Timer register × 1 Compare register × 1	Pulse output capability • PWM output • Square wave output	
A/D converter			8-bit resolution × 8 channels			
D/A converter			8-bit resolution × 2 channels			

Note The pins with additional functions are included in the I/O pins.

(2/2)

Item \ Product Name		μPD784214Y	μPD784215Y	μPD784216Y	μPD78F4216Y
Serial interface		• UART/IOE (3-wire serial I/O): 2 channels (on-chip baud rate generator) • CSI (3-wire serial I/O, multimaster supported I^2C bus): 1 channel			
Clock output		Selectable from f_{XX}, $f_{XX}/2$, $f_{XX}/2^2$, $f_{XX}/2^3$, $f_{XX}/2^4$, $f_{XX}/2^5$, $f_{XX}/2^6$, $f_{XX}/2^7$, f_{XT}			
Buzzer output		Selectable from $f_{XX}/2^{10}$, $f_{XX}/2^{11}$, $f_{XX}/2^{12}$, $f_{XX}/2^{13}$			
Watch timer		1 channel			
Watchdog timer		1 channel			
Interrupt	Hardware sources	29 (internal: 20, external: 9)			
	Software sources	BRK instruction, BRKCS instruction, operand error			
	Non-maskable	Internal: 1, external: 1			
	Maskable	Internal: 19, external: 8			
		• 4-level programmable priority • 3 processing modes: vectored interrupt, macro service, context switching			
Standby function		• HALT/STOP/IDLE mode • Low power consumption mode (CPU can operate on subsystem clock): HALT/IDLE mode			
Power supply voltage		V_{DD} = 1.8 to 5.5 V			
Package		• 100-pin plastic LQFP (fine pitch) (14 × 14 mm) • 100-pin plastic QFP (14 × 20 mm)			

1.7.5 Block diagram



Note V_{PP} applies to the μPD78F4216Y only.

Remark Internal ROM and RAM capacities vary depending on the products.

1.8 Product Outline of μPD784218 Subseries (μPD784217, 784218, 78F4218)

1.8.1 Features

- Internal ROM correction
- Inherits the peripheral functions of the μPD78078 Subseries
- Minimum instruction execution time
  - 160 ns (main system clock: f_{XX} = 12.5-MHz operation)
  - 61 μs (subsystem clock: f_{XT} = 32.768-kHz operation)
- Instruction set suited for control applications
- Interrupt controller (4-level priority)
  - Vectored interrupt servicing/macro service/context switching
- Standby function
  - HALT/STOP/IDLE mode
  - In the low power consumption mode: HALT/IDLE mode (subsystem clock operation)
- On-chip memory:

Mask ROM	256 Kbytes (μPD784218)	
	192 Kbytes (μPD784217)	
Flash memory	256 Kbytes (μPD78F4218)	
RAM	12,800 bytes	

- I/O pins: 86
  - Software programmable pull-up resistors: 70 inputs
  - LED direct drive possible: 22 outputs
  - Transistor direct drive possible: 6 outputs
- Timer/counter: 16-bit timer/counter × 1 unit
  8-bit timer/counter × 6 units
- Watch timer: 1 channel
- Watchdog timer: 1 channel
- Serial interfaces
  - UART/IOE (3-wire serial I/O): 2 channels (on-chip baud rate generator)
  - CSI (3-wire serial I/O): 1 channel
- A/D converter: 8-bit resolution × 8 channels
- D/A converter: 8-bit resolution × 2 channels
- Real-time output port (by combining with the timer/counter, two systems of stepping motors can be independently controlled.)
- Clock frequency dividing function
- Clock output function: Selectable from f_{XX}, $f_{XX}/2$, $f_{XX}/2^2$, $f_{XX}/2^3$, $f_{XX}/2^4$, $f_{XX}/2^5$, $f_{XX}/2^6$, $f_{XX}/2^7$, f_{XT}
- Buzzer output function: Selectable from $f_{XX}/2^{10}$, $f_{XX}/2^{11}$, $f_{XX}/2^{12}$, $f_{XX}/2^{13}$
- External access status function
- Power supply voltage: V_{DD} = 1.8 to 5.5 V

1.8.2 Applications

Cellular phones, PHS, cordless phones, CD-ROM, audiovisual equipment, etc.

1.8.3 Ordering information and quality grade

(1) Ordering information

Part Number	Package	Internal ROM
μPD784217GC-×××-7EA	100-pin plastic QFP (fine pitch) (14 × 14 mm)	Mask ROM
μPD784217GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD784218GC-×××-7EA	100-pin plastic QFP (fine pitch) (14 × 14 mm)	Mask ROM
μPD784218GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD78F4218GC-7EA	100-pin plastic QFP (fine pitch) (14 × 14 mm)	Flash memory
μPD78F4218GF-3BA	100-pin plastic QFP (14 × 20 mm)	Flash memory

Remark ××× indicates ROM code suffix.

(2) Quality grade

Part Number	Package	Quality Grade
μPD784217GC-×××-7EA	100-pin plastic QFP (fine pitch) (14 × 14 mm)	Standard
μPD784217GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
μPD784218GC-×××-7EA	100-pin plastic QFP (fine pitch) (14 × 14 mm)	Standard
μPD784218GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
μPD78F4218GC-7EA	100-pin plastic QFP (fine pitch) (14 × 14 mm)	Standard
μPD78F4218GF-3BA	100-pin plastic QFP (14 × 20 mm)	Standard

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

Remark ××× indicates ROM code suffix.

Caution The μPD784218 Subseries is under development.

1.8.4 Outline of functions

(1/2)

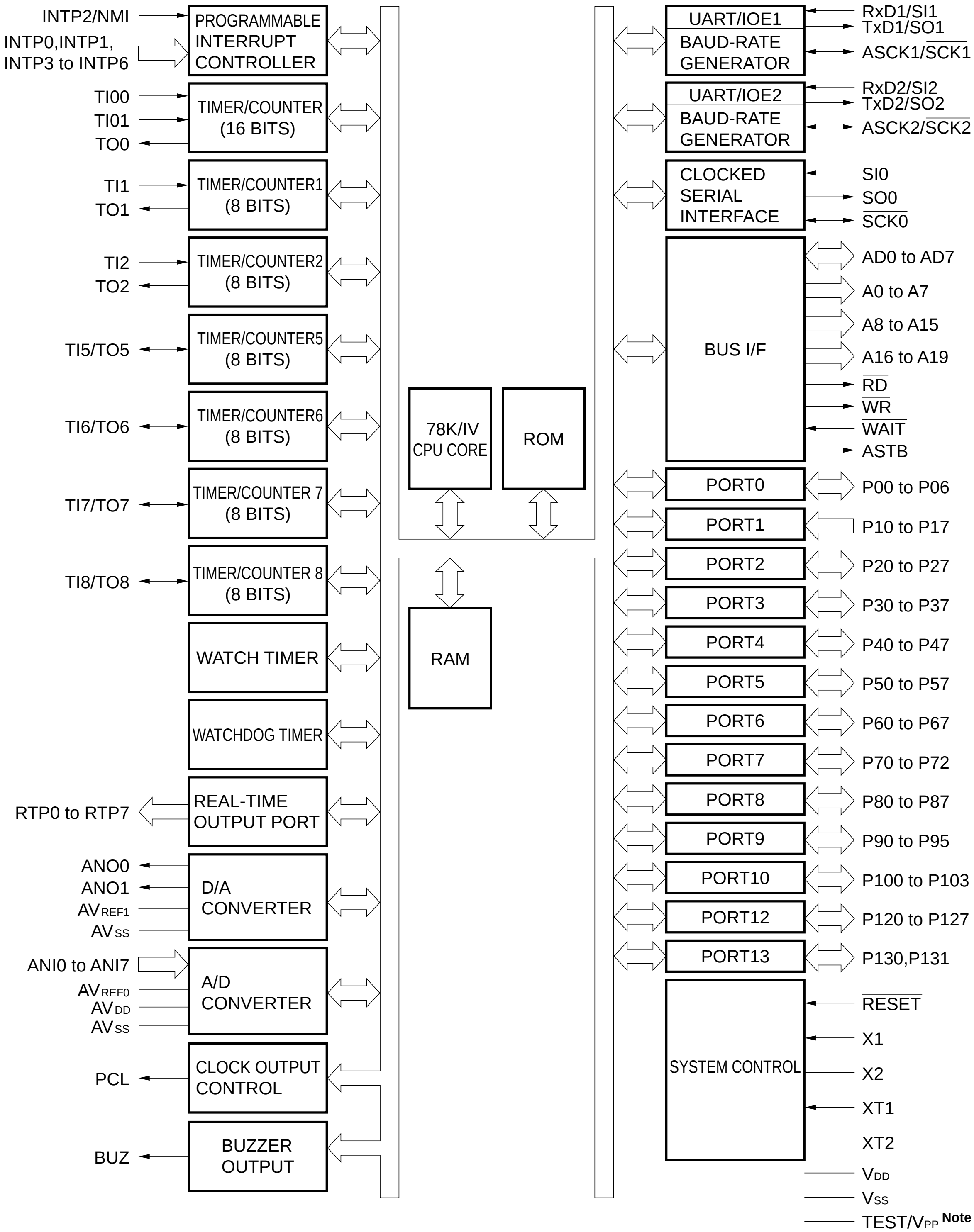
Item		Product Name: μPD784217	μPD784218	μPD78F4218
Number of basic instructions (mnemonics)		113		
General registers		8 bits × 16 registers × 8 banks or 16 bits × 8 registers × 8 banks (memory mapping)		
Minimum instruction execution time		• 160 ns/320 ns/640 ns/1,280 ns/2,560 ns (main system clock: at 12.5-MHz operation) • 61 μs (subsystem clock: at 32.768-kHz operation)		
Internal memory capacity	ROM	192 Kbytes (Mask ROM)	256 Kbytes (Mask ROM)	256 Kbytes (Flash memory)
	RAM	12,800 bytes		
Memory space		1 Mbyte in total of program and data		
I/O ports	Total	86		
	CMOS inputs	8		
	CMOS I/O	72		
	N-ch open-drain I/O	6		
Pins with added functions[Note]	Pins with pull-up resistors	70		
	LED direct drive outputs	22		
	Medium voltage pins	6		
Real-time output ports		4 bits × 2, or 8 bits × 1		
Timer/counters		Timer/counter: (16 bits)	Timer register × 1 Capture/compare register × 2	Pulse output possible • PWM/PPG output • Square wave output • One-shot pulse output
		Timer/counter 1: (8 bits)	Timer register × 1 Compare register × 1	Pulse output possible • PWM output • Square wave output
		Timer/counter 2: (8 bits)	Timer register × 1 Compare register × 1	Pulse output possible • PWM output • Square wave output
		Timer/counter 5: (8 bits)	Timer register × 1 Compare register × 1	Pulse output possible • PWM output • Square wave output
		Timer/counter 6: (8 bits)	Timer register × 1 Compare register × 1	Pulse output possible • PWM output • Square wave output
		Timer/counter 7: (8 bits)	Timer register × 1 Compare register × 1	Pulse output possible • PWM output • Square wave output
		Timer/counter 8: (8 bits)	Timer register × 1 Compare register × 1	Pulse output possible • PWM output • Square wave output

Note The pins with additional functions are included in the I/O pins.

(2/2)

Item \ Product Name		μPD784217	μPD784218	μPD78F4218
Serial interfaces		• UART/IOE (3-wire serial I/O) : 2 channels (on-chip baud rate generator) • CSI (3-wire serial I/O): 1 channel		
A/D converter		8-bit resolution × 8 channels		
D/A converter		8-bit resolution × 2 channels		
Clock output		Selectable from f_{XX}, $f_{XX}/2$, $f_{XX}/2^2$, $f_{XX}/2^3$, $f_{XX}/2^4$, $f_{XX}/2^5$, $f_{XX}/2^6$, $f_{XX}/2^7$, f_{XT}		
Buzzer output		Selectable from $f_{XX}/2^{10}$, $f_{XX}/2^{11}$, $f_{XX}/2^{12}$, $f_{XX}/2^{13}$		
Watch timer		1 channel		
Watchdog timer		1 channel		
Standby function		• HALT/STOP/IDLE mode • In the low power consumption mode (CPU operation by subsystem clock): HALT/IDLE mode		
Interrupts	Hardware sources	29 (internal: 20, external: 9)		
	Software sources	BRK instruction, BRKCS instruction, operand error		
	Non-maskable	Internal: 1, external: 1		
	Maskable	Internal: 19, external: 8		
		• 4-level programmable priority • Three processing formats: Vectored interrupt, macro service, context switching		
Power supply voltage		V_{DD} = 1.8 to 5.5 V		
Package		• 100-pin plastic QFP (fine pitch) (14 × 14 mm) • 100-pin plastic QFP (14 × 20 mm)		

1.8.5 Block diagram



Note The V_{PP} pin applies to the μPD78F4218 only.

Remark Internal ROM capacity varies depending on the products.

1.9 Product Outline of μPD784218Y Subseries (μPD784217Y, 784218Y, 78F4218Y)

1.9.1 Features

- Adds the I^{2}C bus interface to the μPD784218 Subseries.
- Internal ROM correction
- Inherits the peripheral functions of the μPD78078Y Subseries
- Minimum instruction execution time
  - 160 ns (main system clock: f_{XX} = 12.5-MHz operation)
  - 61 μs (subsystem clock: f_{XT} = 32.768-kHz operation)
- Instruction set suited for control applications
- Interrupt controller (4-level priority)
  - Vectored interrupt servicing/macro service/context switching
- Standby function
  - HALT/STOP/IDLE mode
  - In the low power consumption mode: HALT/IDLE mode (subsystem clock operation)
- On-chip memory: Mask ROM 256 Kbytes (μPD784218Y)
  192 Kbytes (μPD784217Y)
  Flash memory 256 Kbytes (μPD78F4218Y)
  RAM 12,800 bytes
- I/O pins: 86
  - Software programmable pull-up resistors: 70 inputs
  - LED direct drive possible: 22 outputs
  - Transistor direct drive possible: 6 outputs
- Timer/counter: 16-bit timer/counter × 1 unit
  8-bit timer/counter × 6 units
- Watch timer: 1 channel
- Watchdog timer: 1 channel
- Serial interfaces
  - UART/IOE (3-wire serial I/O): 2 channels (on-chip baud rate generator)
  - CSI (3-wire serial I/O, multimaster supported I^{2}C bus): 1 channel
- A/D converter: 8-bit resolution × 8 channels
- D/A converter: 8-bit resolution × 2 channels
- Real-time output port (by combining with the timer/counter, two systems of stepping motors can be independently controlled.)
- Clock frequency dividing function
- Clock output function: Selectable from f_{XX}, $f_{XX}/2$, $f_{XX}/2^2$, $f_{XX}/2^3$, $f_{XX}/2^4$, $f_{XX}/2^5$, $f_{XX}/2^6$, $f_{XX}/2^7$, f_{XT}
- Buzzer output function: Selectable from $f_{XX}/2^{10}$, $f_{XX}/2^{11}$, $f_{XX}/2^{12}$, $f_{XX}/2^{13}$
- External access status function
- Power supply voltage: V_{DD} = 1.8 to 5.5 V

1.9.2 Applications

Cellular phones, PHS, cordless phones, CD-ROM, audiovisual equipment, etc.

1.9.3 Ordering information and quality grade

(1) Ordering information

Part Number	Package	Internal ROM
μPD784217YGC-×××-7EA	100-pin plastic QFP (fine pitch) (14 × 14 mm)	Mask ROM
μPD784217YGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD784218YGC-×××-7EA	100-pin plastic QFP (fine pitch) (14 × 14 mm)	Mask ROM
μPD784218YGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD78F4218YGC-7EA	100-pin plastic QFP (fine pitch) (14 × 14 mm)	Flash memory
μPD78F4218YGF-3BA	100-pin plastic QFP (14 × 20 mm)	Flash memory

Remark ××× indicates ROM code suffix.

(2) Quality grade

Part Number	Package	Quality Grade
μPD784217YGC-×××-7EA	100-pin plastic QFP (fine pitch) (14 × 14 mm)	Standard
μPD784217YGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
μPD784218YGC-×××-7EA	100-pin plastic QFP (fine pitch) (14 × 14 mm)	Standard
μPD784218YGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
μPD78F4218YGC-7EA	100-pin plastic QFP (fine pitch) (14 × 14 mm)	Standard
μPD78F4218YGF-3BA	100-pin plastic QFP (14 × 20 mm)	Standard

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

Remark ××× indicates ROM code suffix.

Caution The μPD784218Y Subseries is under development.

1.9.4 Outline of functions

(1/2)

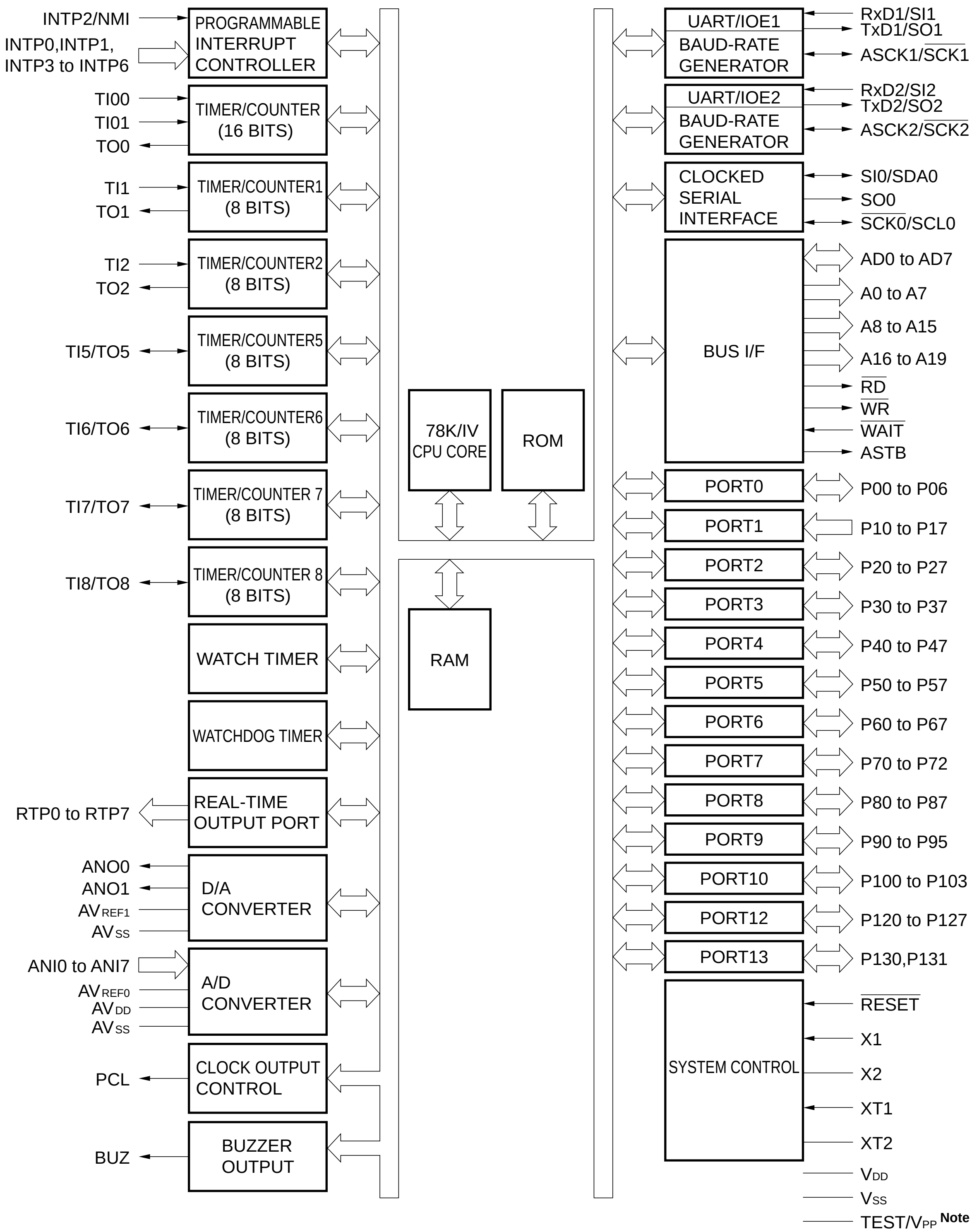
Item		μPD784217Y	μPD784218Y	μPD78F4218Y
Number of basic instructions (mnemonics)		113		
General registers		8 bits × 16 registers × 8 banks or 16 bits × 8 registers × 8 banks (memory mapping)		
Minimum instruction execution time		• 160 ns/320 ns/640 ns/1,280 ns/2,560 ns (main system clock: at 12.5-MHz operation) • 61 μs (subsystem clock: at 32.768-kHz operation)		
Internal memory capacity	ROM	192 Kbytes (Mask ROM)	250 Kbytes (Mask ROM)	256 Kbytes (Flash memory)
	RAM	12,800 bytes		
Memory space		1 Mbyte in total of program and data		
I/O ports	Total	86		
	CMOS inputs	8		
	CMOS I/O	72		
	N-ch open-drain I/O	6		
Pins with added functions[Note]	Pins with pull-up resistors	70		
	LED direct drive outputs	22		
	Medium voltage pins	6		
Real-time output ports		4 bits × 2, or 8 bits × 1		
Timer/counters		Timer/counter: (16 bits)	Timer register × 1 Capture/compare register × 2	Pulse output possible • PWM/PPG output • Square wave output • One-shot pulse output
		Timer/counter 1: (8 bits)	Timer register × 1 Compare register × 1	Pulse output possible • PWM output • Square wave output
		Timer/counter 2: (8 bits)	Timer register × 1 Compare register × 1	Pulse output possible • PWM output • Square wave output
		Timer/counter 5: (8 bits)	Timer register × 1 Compare register × 1	Pulse output possible • PWM output • Square wave output
		Timer/counter 6: (8 bits)	Timer register × 1 Compare register × 1	Pulse output possible • PWM output • Square wave output
		Timer/counter 7: (8 bits)	Timer register × 1 Compare register × 1	Pulse output possible • PWM output • Square wave output
		Timer/counter 8: (8 bits)	Timer register × 1 Compare register × 1	Pulse output possible • PWM output • Square wave output

Note The pins with additional functions are included in the I/O pins.

(2/2)

Item \ Product Name		μPD784217Y	μPD784218Y	μPD78F4218Y
Serial interfaces		• UART/IOE (3-wire serial I/O) : 2 channels (on-chip baud rate generator) • CSI (3-wire serial I/O, multimaster supported I^2C bus): 1 channel		
A/D converter		8-bit resolution × 8 channels		
D/A converter		8-bit resolution × 2 channels		
Clock output		Selectable from f_{XX}, $f_{XX}/2$, $f_{XX}/2^2$, $f_{XX}/2^3$, $f_{XX}/2^4$, $f_{XX}/2^5$, $f_{XX}/2^6$, $f_{XX}/2^7$, f_{XT}		
Buzzer output		Selectable from $f_{XX}/2^{10}$, $f_{XX}/2^{11}$, $f_{XX}/2^{12}$, $f_{XX}/2^{13}$		
Watch timer		1 channel		
Watchdog timer		1 channel		
Standby function		• HALT/STOP/IDLE mode • In the low power consumption mode (CPU operation by subsystem clock): HALT/IDLE mode		
Interrupts	Hardware sources	29 (internal: 20, external: 9)		
	Software sources	BRK instruction, BRKCS instruction, operand error		
	Non-maskable	Internal: 1, external: 1		
	Maskable	Internal: 19, external: 8		
		• 4-level programmable priority • Three processing formats: Vectored interrupt, macro service, context switching		
Power supply voltage		V_{DD} = 1.8 to 5.5 V		
Package		• 100-pin plastic QFP (fine pitch) (14 × 14 mm) • 100-pin plastic QFP (14 × 20 mm)		

1.9.5 Block diagram



Note The V_{PP} pin applies to the μPD78F4218Y only.

Remark Internal ROM capacity varies depending on the products.

1.10 Product Outline of μPD784225 Subseries (μPD784224, 784225, 78F4225)

1.10.1 Features

- Inherits the peripheral functions of the μPD780058 Subseries
- Minimum instruction execution time
  - 160 ns (main system clock: f_{XX} = 12.5-MHz operation)
  - 61 μs (subsystem clock: f_{XT} = 32.768-kHz operation)
- Instruction set suited for control applications
- Interrupt controller (4-level priority)
  - Vectored interrupt servicing/macro service/context switching
- Standby function
  - HALT/STOP/IDLE mode
  - In the low power consumption mode: HALT/IDLE mode (subsystem clock operation)
- On-chip memory:

Mask ROM	128 Kbytes (μPD784225)
	96 Kbytes (μPD784224)
Flash memory	128 Kbytes (μPD78F4225)
RAM	4,352 bytes (μPD784225, 78F4225)
	3,584 bytes (μPD784224)

- I/O pins: 67
  - Software programmable pull-up resistors: 50 inputs
  - LED direct drive possible: 16 outputs
- Timer/counter: 16-bit timer/counter × 1 unit
  8-bit timer/counter × 4 units
- Watch timer: 1 channel
- Watchdog timer: 1 channel
- Serial interfaces
  - UART/IOE (3-wire serial I/O): 2 channels (on-chip baud rate generator)
  - CSI (3-wire serial I/O): 1 channel
- A/D converter: 8-bit resolution × 8 channels
- D/A converter: 8-bit resolution × 2 channels
- Real-time output port (by combining with the timer/counter, two systems of stepping motors can be independently controlled.)
- Clock frequency division function
- Clock output function: Selectable from f_{XX}, $f_{XX}/2$, $f_{XX}/2^2$, $f_{XX}/2^3$, $f_{XX}/2^4$, $f_{XX}/2^5$, $f_{XX}/2^6$, $f_{XX}/2^7$, f_{XT}
- Buzzer output function: Selectable from $f_{XX}/2^{10}$, $f_{XX}/2^{11}$, $f_{XX}/2^{12}$, $f_{XX}/2^{13}$
- Power supply voltage: V_{DD} = 1.8 to 5.5 V

1.10.2 Applications

Car audio, portable audio, air conditioner, telephone, etc.

1.10.3 Ordering information and quality grade

(1) Ordering information

Part Number	Package	Internal ROM
μPD784224GC-×××-8BT	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784224GK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Mask ROM
μPD784225GC-×××-8BT	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784225GK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Mask ROM
μPD78F4225GC-8BT	80-pin plastic QFP (14 × 14 mm)	Flash memory
μPD78F4225GK-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Flash memory

Remark ××× indicates ROM code suffix.

(2) Quality grade

Part Number	Package	Quality Grade
μPD784224GC-×××-8BT	80-pin plastic QFP (14 × 14 mm)	Standard
μPD784224GK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
μPD784225GC-×××-8BT	80-pin plastic QFP (14 × 14 mm)	Standard
μPD784225GK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
μPD78F4225GC-8BT	80-pin plastic QFP (14 × 14 mm)	Standard
μPD78F4225GK-BE9	80-pin plastic TQFP (fine pitch) (12 × 20 mm)	Standard

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

Remark ××× indicates ROM code suffix.

Caution The μPD784225 Subseries is under development.

1.10.4 Outline of functions

(1/2)

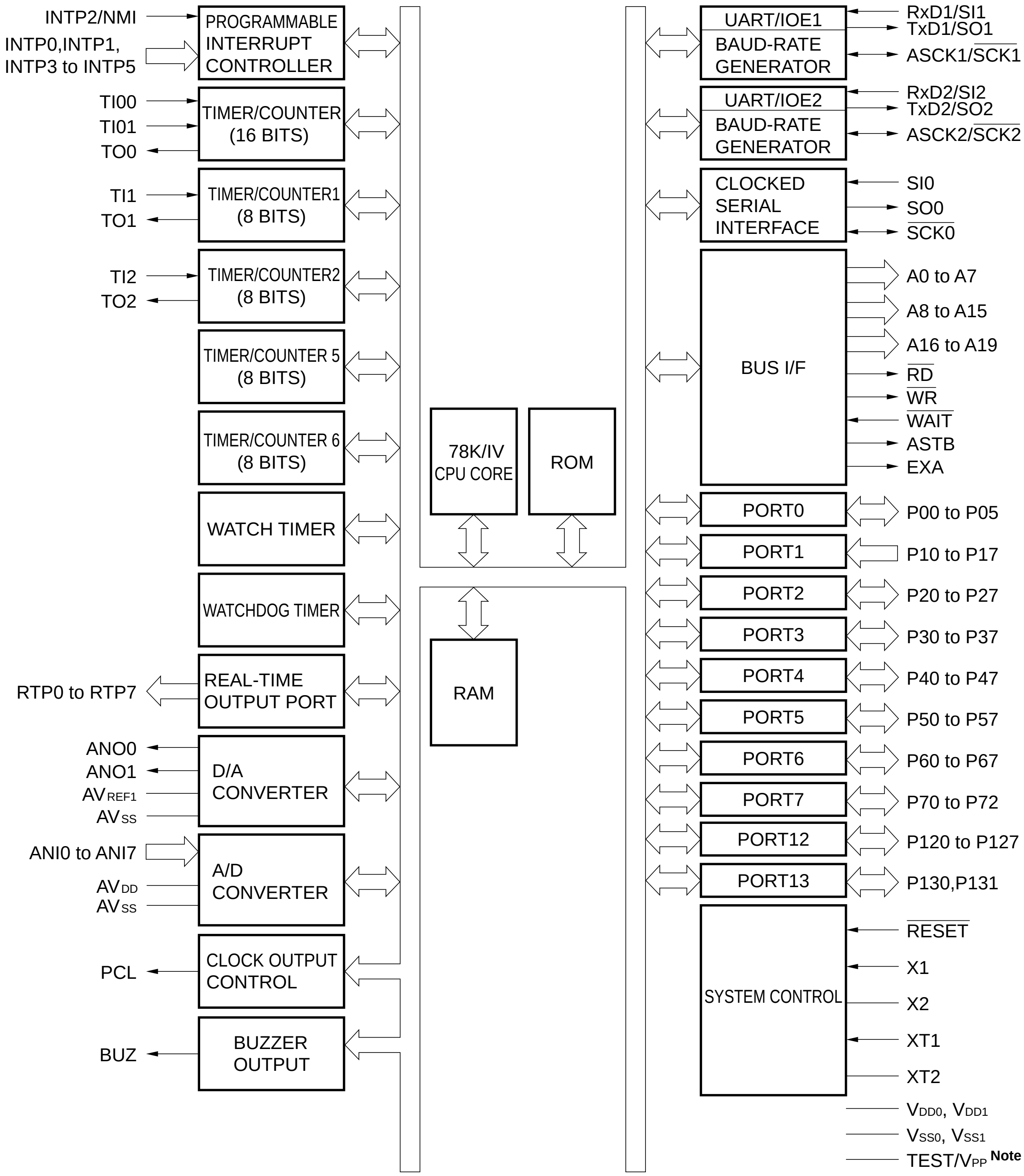
Item		μPD784224	μPD784225	μPD78F4225
Number of basic instructions (mnemonics)		113		
General registers		8 bits × 16 registers × 8 banks or 16 bits × 8 registers × 8 banks (memory mapping)		
Minimum instruction execution time		• 160 ns/320 ns/640 ns/1,280 ns/2,560 ns (main system clock: at 12.5-MHz operation) • 61 μs (subsystem clock: at 32.768-kHz operation)		
Internal memory capacity	ROM	96 Kbytes (Mask ROM)	128 Kbytes (Mask ROM)	128 Kbytes (Flash memory)
	RAM	3,584 bytes	4,352 bytes	
Memory space		1 Mbyte in total of program and data		
I/O ports	Total	67		
	CMOS inputs	8		
	CMOS I/O	59		
Pins with added functions[Note]	Pins with pull-up resistors	57		
	LED direct drive outputs	16		
Real-time output ports		4 bits × 2, or 8 bits × 1		
Timer/counters		Timer/counter: (16 bits) — Timer register × 1, Capture/compare register × 2 — Pulse output possible • PWM/PPG output • Square wave output • One-shot pulse output		
		Timer/counter 1: (8 bits) — Timer register × 1, Compare register × 1 — Pulse output possible • PWM output • Square wave output		
		Timer/counter 2: (8 bits) — Timer register × 1, Compare register × 1 — Pulse output possible • PWM output • Square wave output		
		Timer/counter 5: (8 bits) — Timer register × 1, Compare register × 1		
		Timer/counter 6: (8 bits) — Timer register × 1, Compare register × 1		
Serial interfaces		• UART/IOE (3-wire serial I/O) : 2 channels (on-chip baud rate generator) • CSI (3-wire serial I/O): 1 channel		
A/D converter		8-bit resolution × 8 channels		
D/A converter		8-bit resolution × 2 channels		
Clock output		Selectable from f_{XX}, $f_{XX}/2$, $f_{XX}/2^2$, $f_{XX}/2^3$, $f_{XX}/2^4$, $f_{XX}/2^5$, $f_{XX}/2^6$, $f_{XX}/2^7$, f_{XT}		
Buzzer output		Selectable from $f_{XX}/2^{10}$, $f_{XX}/2^{11}$, $f_{XX}/2^{12}$, $f_{XX}/2^{13}$		
Watch timer		1 channel		
Watchdog timer		1 channel		

Note The pins with additional functions are included in the I/O pins.

(2/2)

Item \ Product Name		μPD784224	μPD784225	μPD78F4225
Standby function		• HALT/STOP/IDLE mode • In the low power consumption mode (CPU operation by subsystem clock): HALT/IDLE mode		
Interrupts	Hardware sources	25 (internal: 18, external: 7)		
	Software sources	BRK instruction, BRKCS instruction, operand error		
	Non-maskable	Internal: 1, external: 1		
	Maskable	Internal: 17, external: 6		
		• 4-level programmable priority • Three processing formats: Vectored interrupt, macro service, context switching		
Power supply voltage		V_{DD} = 1.8 to 5.5 V		
Package		• 80-pin plastic TQFP (fine pitch) (12 × 12 mm) • 80-pin plastic QFP (14 × 14 mm)		

1.10.5 Block diagram



Note The V_{PP} pin applies to the μPD78F4225 only.

Remark Internal ROM and RAM capacities vary depending on the products.

1.11 Product Outline of μPD784225Y Subseries (μPD784224Y, 784225Y, 78F4225Y)

1.11.1 Features

- Adds the I^{2}C bus interface to the μPD784225 Subseries.
- Inherits the peripheral functions of the μPD780058Y Subseries
- Minimum instruction execution time
  - 160 ns (main system clock: f_{XX} = 12.5-MHz operation)
  - 61 μs (subsystem clock: f_{XT} = 32.768-kHz operation)
- Instruction set suited for control applications
- Interrupt controller (4-level priority)
  - Vectored interrupt servicing/macro service/context switching
- Standby function
  - HALT/STOP/IDLE mode
  - In the low power consumption mode: HALT/IDLE mode (subsystem clock operation)
- On-chip memory: Mask ROM 128 Kbytes (μPD784225Y)
  96 Kbytes (μPD784224Y)
  Flash memory 128 Kbytes (μPD78F4225Y)
  RAM 4,352 bytes (μPD784225Y, 78F4225Y)
  3,584 bytes (μPD784224Y)
- I/O pins: 67
  - Software programmable pull-up resistors: 50 inputs
  - LED direct drive possible: 16 outputs
- Timer/counter: 16-bit timer/counter × 1 unit
  8-bit timer/counter × 4 units
- Watch timer: 1 channel
- Watchdog timer: 1 channel
- Serial interfaces
  - UART/IOE (3-wire serial I/O): 2 channels (on-chip baud rate generator)
  - CSI (3-wire serial I/O, multimaster supported I^{2}C bus): 1 channel
- A/D converter: 8-bit resolution × 8 channels
- D/A converter: 8-bit resolution × 2 channels
- Real-time output port (by combining with the timer/counter, two stepping motors can be independently controlled.)
- Clock frequency dividing function
- Clock output function: Selectable from f_{XX}, $f_{XX}/2$, $f_{XX}/2^2$, $f_{XX}/2^3$, $f_{XX}/2^4$, $f_{XX}/2^5$, $f_{XX}/2^6$, $f_{XX}/2^7$, f_{XT}
- Buzzer output function: Selectable from $f_{XX}/2^{10}$, $f_{XX}/2^{11}$, $f_{XX}/2^{12}$, $f_{XX}/2^{13}$
- External access status function
- Power supply voltage: V_{DD} = 1.8 to 5.5 V

1.11.2 Applications

Car audios, portable audios, air conditioners, telephones, etc.

1.11.3 Ordering information and quality grade

(1) Ordering information

Part Number	Package	Internal ROM
μPD784224YGC-×××-8BT	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784224YGK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Mask ROM
μPD784225YGC-×××-8BT	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784225YGK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Mask ROM
μPD78F4225YGC-8BT	80-pin plastic QFP (14 × 14 mm)	Flash memory
μPD78F4225YGK-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Flash memory

Remark ××× indicates ROM code suffix.

(2) Quality grade

Part Number	Package	Quality Grade
μPD784224YGC-×××-8BT	80-pin plastic QFP (14 × 14 mm)	Standard
μPD784224YGK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
μPD784225YGC-×××-8BT	80-pin plastic QFP (14 × 14 mm)	Standard
μPD784225YGK-×××-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard
μPD78F4225YGC-8BT	80-pin plastic QFP (14 × 14 mm)	Standard
μPD78F4225YGK-BE9	80-pin plastic TQFP (fine pitch) (12 × 12 mm)	Standard

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

Remark ××× indicates ROM code suffix.

Caution The μPD784225Y Subseries is under development.

1.11.4 Outline of functions

(1/2)

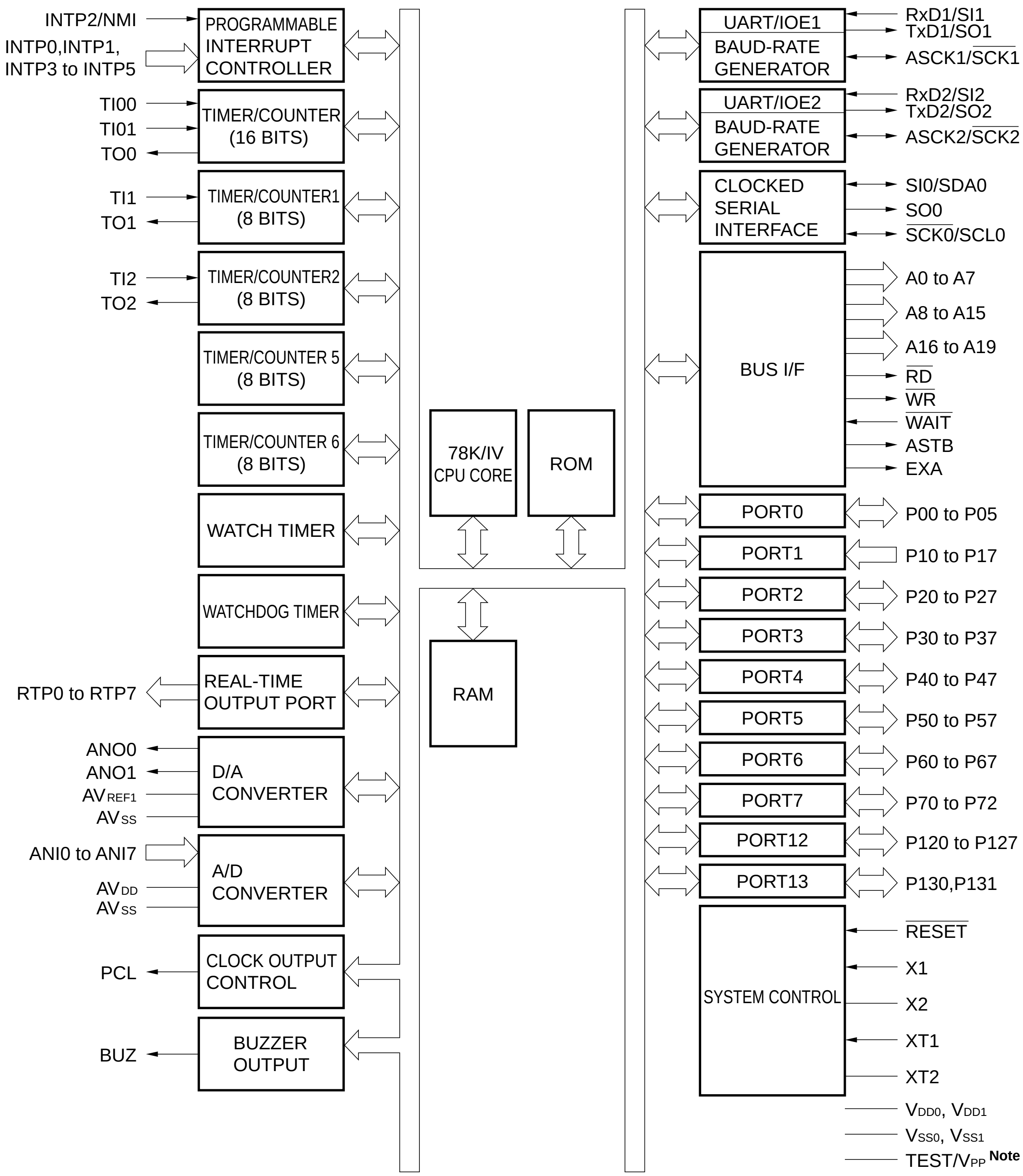
Item		Product Name: μPD784224Y	μPD784225Y	μPD78F4225Y
Number of basic instructions (mnemonics)		113		
General registers		8 bits × 16 registers × 8 banks or 16 bits × 8 registers × 8 banks (memory mapping)		
Minimum instruction execution time		• 160 ns/320 ns/640 ns/1,280 ns/2,560 ns (main system clock: at 12.5-MHz operation) • 61 μs (subsystem clock: at 32.768-kHz operation)		
Internal memory capacity	ROM	96 Kbytes (Mask ROM)	128 Kbytes (Mask ROM)	128 Kbytes (Flash memory)
	RAM	3,584 bytes	4,352 bytes	
Memory space		1 Mbyte in total of program and data		
• I/O ports	Total	67		
	• CMOS inputs	8		
	• CMOS I/O	59		
Pins with added	• Pins with pull-up functions[Note]	57		
	• LED direct drive outputs	16		
• Real-time output ports		4 bits × 2, or 8 bits × 1		
Timer/counters		Timer/counter: (16 bits) — Timer register × 1, Capture/compare register × 2 — Pulse output possible • PWM/PPG output • Square wave output • One-shot pulse output		
		Timer/counter 1: (8 bits) — Timer register × 1, Compare register × 1 — Pulse output possible • PWM output • Square wave output		
		Timer/counter 2: (8 bits) — Timer register × 1, Compare register × 1 — Pulse output possible • PWM output • Square wave output		
		Timer/counter 5: (8 bits) — Timer register × 1, Compare register × 1		
		Timer/counter 6: (8 bits) — Timer register × 1, Compare register × 1		
Serial interfaces		• UART/IOE (3-wire serial I/O) : 2 channels (on-chip baud rate generator) • CSI (3-wire serial I/O, multimaster supported I^2C bus): 1 channel		
A/D converter		8-bit resolution × 8 channels		
D/A converter		8-bit resolution × 2 channels		
Clock output		Selectable from f_{XX}, $f_{XX}/2$, $f_{XX}/2^2$, $f_{XX}/2^3$, $f_{XX}/2^4$, $f_{XX}/2^5$, $f_{XX}/2^6$, $f_{XX}/2^7$, f_{XT}		
Buzzer output		Selectable from $f_{XX}/2^{10}$, $f_{XX}/2^{11}$, $f_{XX}/2^{12}$, $f_{XX}/2^{13}$		
Watch timer		1 channel		
Watchdog timer		1 channel		

Note The pins with additional functions are included in the I/O pins.

(2/2)

Item \ Product Name		μPD784224Y	μPD784225Y	μPD78F4225Y
Standby function		• HALT/STOP/IDLE mode • In the low power consumption mode (CPU operation by subsystem clock): HALT/IDLE mode		
Interrupts	Hardware sources	25 (internal: 18, external: 7)		
	Software sources	BRK instruction, BRKCS instruction, operand error		
	Non-maskable	Internal: 1, external: 1		
	Maskable	Internal: 17, external: 6		
		• 4-level programmable priority • Three processing formats: Vectored interrupt, macro service, context switching		
Power supply voltage		V_{DD} = 1.8 to 5.5 V		
Package		• 80-pin plastic TQFP (fine pitch) (12 × 12 mm) • 80-pin plastic QFP (14 × 14 mm)		

1.11.5 Block diagram



Note The V_{PP} pin applies to the μPD78F4225Y only.

Remark Internal ROM and RAM capacities vary depending on the products.

1.12 Product Outline of μPD784908 Subseries (μPD784907, 784908, 78P4908)

1.12.1 Features

★ • Minimum instruction execution time: 160 ns (at 12.58-MHz operation)
• On-chip memory
  • Mask ROM : 96 Kbytes (μPD784907)
    128 Kbytes (μPD784908)
  PROM : 128 Kbytes (μPD78P4908)
★ • RAM : 3,584 bytes (μPD784907)
    4,352 bytes (μPD784908, 78P4908)
• I/O port: 80
• Timer/counter: 16-bit timer/counter × 3 units
  16-bit timer × 1 unit
• Watch timer: 1 channel
• Watchdog timer: 1 channel
• Serial interfaces: 4 channels
  • UART/IOE (3-wire serial I/O): 2 channels
  • CSI (3-wire serial I/O): 2 channels
• Standby function
• HALT/STOP/IDLE mode
• Clock frequency dividing function
• Clock output function: Selectable from f_{CLK}, $f_{CLK}/2$, $f_{CLK}/4$, $f_{CLK}/8$, $f_{CLK}/16$
• A/D converter: 8-bit resolution × 8 channels
• Internal IEBus controller
• Low power consumption
★ • Power supply voltage: V_{DD} = 3.5 to 5.5 V (Mask ROM version)
  V_{DD} = 4.0 to 5.5 V (PROM version)

1.12.2 Applications

Car audios, etc.

1.12.3 Ordering information and quality grade

(1) Ordering information

Part Number	Package	Internal ROM
μPD784907GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD784908GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD78P4908GF-3BA	100-pin plastic QFP (14 × 20 mm)	One-time PROM

Remark ××× indicates ROM code suffix.

(2) Quality grade

Part Number	Package	Quality Grade
μPD784907GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
μPD784908GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
μPD78P4908GF-3BA	100-pin plastic QFP (14 × 20 mm)	Standard

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

Remark ××× indicates ROM code suffix.

Caution The μPD784908 Subseries is under development.

1.12.4 Outline of functions

(1/2)

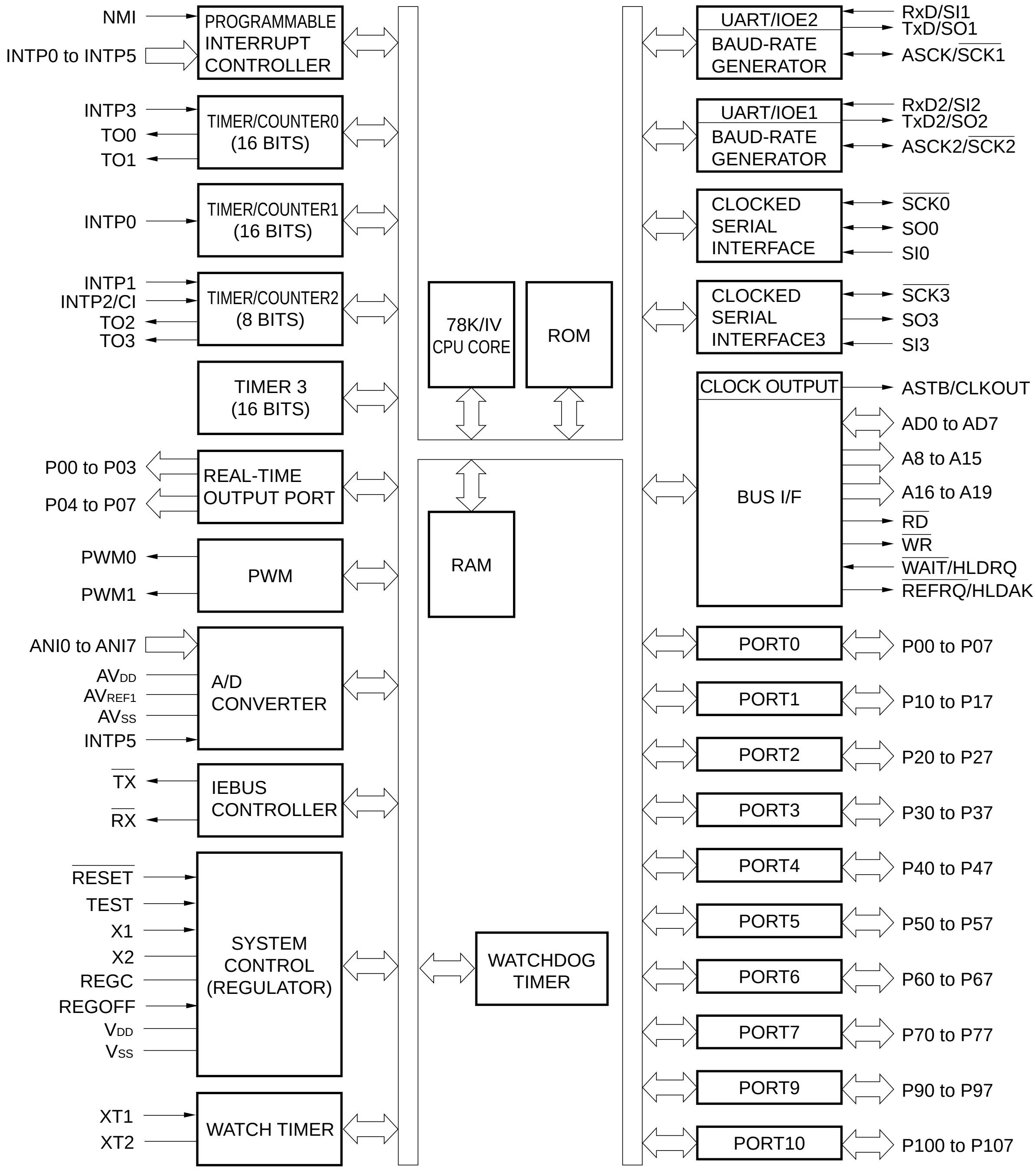
Item \ Product Name		μPD784907	μPD784908	μPD78P4908
Number of basic instructions (mnemonics)		113		
General registers		8 bits × 16 registers × 8 banks or 16 bits × 8 registers × 8 banks (memory mapping)		
★ Minimum instruction execution time		160 ns/320 ns/636 ns/1.27 μs (at 12.58-kHz operation)		
Internal memory capacity	ROM	92 Kbytes (Mask ROM)	128 Kbytes (Mask ROM)	128 Kbytes (PROM)
	RAM	3,584 bytes	4,352 bytes	
Memory space		1 Mbyte in total of program and data		
I/O ports	Total	80		
	Inputs	8		
	I/O	72		
Pins with added functions[Note]	LED direct drive outputs	24		
	Transistor direct drive	8		
	N-ch open-drain	4		
Real-time output ports		4 bits × 2, or 8 bits × 1		
IEBus controller		Internal (simplify)		
Timer/counters		Timer/counter 0: (16 bits)	Timer register × 1 Capture register × 1 Compare register × 2	Pulse output capability • Toggle output • PWM/PPG output • One-shot pulse output
		Timer/counter 1: (16 bits)	Timer register × 1 Capture register × 1 Capture/compare register × 1 Compare register × 1	Real-time output port
		Timer/counter 2:	Timer register × 1 Capture register × 1 Capture/compare register × 1 Compare register × 1	Pulse output capability • Toggle output • PWM/PPG output
		Timer 3:	Timer register × 1 Compare register × 1	
Watch timer		Interrupt occurs at an interval of 0.5 sec. (Has an internal clock oscillator.) The input clock can be selected from among the main clock ★ (12.58 MHz) or clock (32.7 kHz).		
Clock output		Selectable from f_{CLK}, $f_{CLK}/2$, $f_{CLK}/4$, $f_{CLK}/8$, $f_{CLK}/16$, (also usable as output port)		
PWM output		12-bit resolution × 2 channels		
Serial interfaces		UART/IOE (3-wire serial I/O): 2 channels (on-chip baud rate generator) CSI (3-wire serial I/O): 2 channels		
A/D converter		8-bit resolution × 8 channels		
Watchdog timer		1 channel		
Standby function		HALT/STOP/IDLE mode		

Note The pins with additional functions are included in the I/O pins.

(2/2)

Item \ Product Name		μPD784907	μPD784908	μPD78F4908
Interrupts	Hardware sources	27 (internal: 20, external: 7 (sampling clock variable input: 1))		
	Software sources	BRK instruction, BRKCS instruction, operand error		
	Non-maskable	Internal: 1, external: 1		
	Maskable	Internal: 19, external: 6		
		4-level programmable priority		
		Three processing formats: Vectored interrupt, macro service, context switching		
★ Power supply voltage		V_{DD} = 3.5 to 5.5 V		V_{DD} = 4.0 to 5.5 V
Package		100-pin plastic QFP (14 × 20 mm)		

1.12.5 Block diagram



Remark Internal ROM and RAM capacities vary depending on the products.

1.13 Product Outline of μPD784915 Subseries (μPD784915, 784915A, 784916A, 784915B, 784916B, 78P4916)

1.13.1 Features

- 78K/IV Series (16-bit CPU core employed): Minimum instruction execution time: 250 ns (at 8-MHz internal clock)
- Internal timer unit for VCR servo control (super timer unit)
- Internal analog circuit for VHS type VCR
  - CTL amplifier
  - RECCTL driver (supports rewriting)
  - DPFG separation circuit (ternary separation circuit)
  - DFG amplifier, DPG comparator, CFG amplifier
  - Reel FG comparator (2 channels), CSYNC comparator
- I/O port: 54
- Serial interface: 2 channels (3-wire serial I/O)
- A/D converter: 12 channels (conversion time: 10 μs)
- PWM output: 16-bit resolution × 3 channels, 8-bit resolution × 3 channels
- Interrupt function
  - Vectored interrupt function
  - Macro service function
  - Context switching function
- Low-frequency oscillation mode supported: Main system clock frequency = internal clock frequency
- Low-power consumption mode: CPU can operate on subsystem clock.
- Hardware watch function: Watch operation on low voltage (V_{DD} = 2.5 V (MIN.)) and with low current consumption
- Package for high-density mounting: 100-pin plastic QFP (0.65-mm pitch, 14 × 20 mm)

1.13.2 Applications

For controlling system/servo/timer of VCR (stationary type and camcorder type)

1.13.3 Ordering information and quality grade

(1) Ordering information

	Part Number	Package	Internal ROM
	μPD784915GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
	μPD784915AGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
	μPD784916AGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
★	μPD784915BGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
★	μPD784916BGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
	μPD78P4916GF-3BA	100-pin plastic QFP (14 × 20 mm)	One-time PROM

Remark ××× indicates ROM code suffix.

(2) Quality grades

	Part Number	Package	Quality Grade
	μPD784915GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
	μPD784915AGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
	μPD784916AGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
★	μPD784915BGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
★	μPD784916BGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
	μPD78P4916GF-3BA	100-pin plastic QFP (14 × 20 mm)	Standard

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

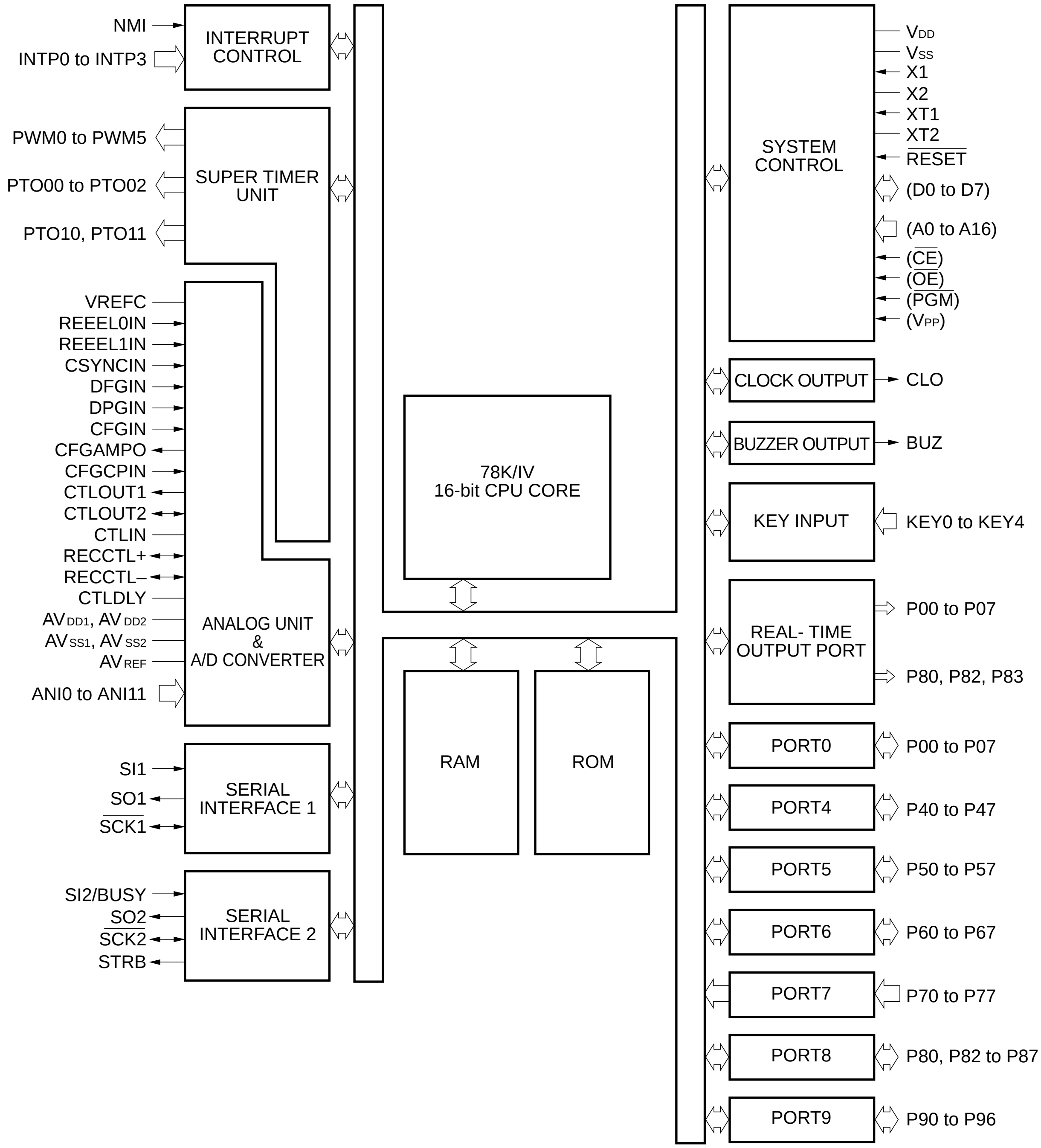
Remark ××× indicates ROM code suffix.

1.13.4 Outline of functions

★

<table>
<tr><th colspan="2">Product Name
Item</th><th>μPD784915, 784915A</th><th>μPD784915B</th><th>μPD784916A</th><th>μPD784916B</th><th>μPD78P4916</th></tr>
<tr><td colspan="2">Number of instructions</td><td colspan="5">113</td></tr>
<tr><td colspan="2">Minimum instruction execution time</td><td colspan="5">250 ns (8-MHz internal clock operation)</td></tr>
<tr><td rowspan="2">Internal memory capacity</td><td>ROM</td><td colspan="2">48 Kbytes
(Mask ROM)</td><td colspan="2">62 Kbytes
(Mask ROM)</td><td>62 Kbytes
(One-time PROM)</td></tr>
<tr><td>RAM</td><td colspan="4">1,280 bytes</td><td>2,048 bytes</td></tr>
<tr><td colspan="2">Interrupt</td><td colspan="5">4 levels (programmable), vector interrupt, macro service, context switching</td></tr>
<tr><td colspan="2">External source</td><td colspan="5">9 (including NMI)</td></tr>
<tr><td colspan="2">Internal source</td><td colspan="5">19</td></tr>
<tr><td colspan="2">Number of interrupts that can use macro service</td><td colspan="5">25</td></tr>
<tr><td colspan="2">Types of macro services</td><td colspan="5">4 types, 10 macro services</td></tr>
<tr><td colspan="2">I/O port</td><td colspan="5">Input: 8, I/O: 46</td></tr>
<tr><td colspan="2">Time base counter</td><td colspan="5">• 22-bit FRC
• Resolution: 125 ns, Maximum count time: 524 ms</td></tr>
<tr><td colspan="2">Capture register</td><td colspan="5"><table>
<tr><th>Input signal</th><th>Number of bits</th><th>Measurement cycle</th><th>Operating edge</th></tr>
<tr><td>CFG</td><td>22</td><td>125 ns to 524 ms</td><td>↑ ↓</td></tr>
<tr><td>DFG</td><td>22</td><td>125 ns to 524 ms</td><td>↑</td></tr>
<tr><td>HSW</td><td>16</td><td>1 μs to 65.5 ms</td><td>↑ ↓</td></tr>
<tr><td>V_{SYNC}</td><td>22</td><td>125 ns to 524 ms</td><td>↑</td></tr>
<tr><td>CTL</td><td>16</td><td>1 μs to 65.5 ms</td><td>↑ ↓</td></tr>
<tr><td>TREEL</td><td>22</td><td>125 ns to 524 ms</td><td>↑ ↓</td></tr>
<tr><td>SREEL</td><td>22</td><td>125 ns to 524 ms</td><td>↑ ↓</td></tr>
</table></td></tr>
<tr><td colspan="2">General-purpose timer</td><td colspan="5">16-bit timer × 3</td></tr>
<tr><td colspan="2">PBCTL duty identification</td><td colspan="5">• Duty of playback control signal
• VISS detection, wide aspect detection</td></tr>
<tr><td colspan="2">Linear time counter</td><td colspan="5">5-bit UDC for counting CTL signal</td></tr>
<tr><td colspan="2">Real-time output port</td><td colspan="5">11</td></tr>
<tr><td colspan="2">Serial interface</td><td colspan="5">Clocked (3-wire): 2 channels</td></tr>
<tr><td colspan="2">A/D converter</td><td colspan="5">8-bit resolution × 12 channels, conversion time: 10 μs</td></tr>
<tr><td colspan="2">PWM output</td><td colspan="5">• 16-bit resolution × 3 channels, 8-bit resolution × 3 channels
• Carrier frequency: 62.5 kHz</td></tr>
<tr><td colspan="2">Watch function</td><td colspan="5">0.5-sec measurement, low-voltage operation</td></tr>
<tr><td colspan="2">Standby function</td><td colspan="5">HALT mode/STOP mode</td></tr>
<tr><td colspan="2">Analog circuit</td><td colspan="5">• CTL amplifier
• RECCTL driver (supports rewriting)
• DPFG separation circuit (ternary separation circuit)
• DFG amplifier, DPG comparator, CFG amplifier
• Reel FG comparator
• CSYNC comparator</td></tr>
<tr><td colspan="2">Power supply voltage</td><td colspan="5">V_{DD} = 2.7 to 5.5 V</td></tr>
<tr><td colspan="2">Package</td><td colspan="5">100-pin plastic QFP (14 × 20 mm)</td></tr>
</table>

1.13.5 Block diagram



Remarks 1. Internal ROM and RAM capacities vary depending on the products.

2. V_{PP} applies to the μPD78P4916 only.

3. The pins in parentheses are used in the PROM programming mode.

★ ## 1.14 Product Outline of μPD784928 Subseries (μPD784927, 78F4928)

1.14.1 Features

- 16-bit CPU core: Minimum instruction execution time: 250 ns (with 8-MHz internal clock)
- Internal timer unit (super timer unit) for VCR servo control
- I/O ports: 74
- Internal analog circuits for VHS type VCR
  - CTL amplifier
  - RECCTL driver (supporting rewrite)
  - CFG amplifier
  - DFG amplifier
  - DPG amplifier
  - DPFG separation circuit (ternary separation circuit)
  - Reel FG comparator (2 channels)
  - CSYNC comparator
- Serial interface: 2 channels
  - 3-wire serial I/O: 2 channels
- A/D converter: 12 channels (conversion time: 10 μs)
- PWM output: 16-bit resolution × 3 channels, 8-bit resolution × 3 channels
- Interrupt function
  - Vector interrupt function
  - Macro service function
  - Context switching function
- Low frequency oscillation mode: main system clock frequency = internal clock frequency
- Low power consumption mode: CPU can operate on subsystem clock.
- Power supply voltage: V_{DD} = 2.7 to 5.5 V
- Hardware watch function: Low-voltage (V_{DD} = 2.7 V MIN.), low-current consumption operation

1.14.2 Applications

For stationary type and camcorder type VCRs.

★ 1.14.3 Ordering information

(1) Ordering information

Part Number	Package	Internal ROM
μPD784927GF-××× -3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD78F4928GF-3BA	100-pin plastic QFP (14 × 20 mm)	Flash memory

Remark ××× indicates ROM code suffix.

(2) Quality grade

Part Number	Package	Quality Grade
μPD784927GF-××× -3BA	100-pin plastic QFP (14 × 20 mm)	Standard
μPD78F4928GF-3BA	100-pin plastic QFP (14 × 20 mm)	Standard

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

Remark ××× indicates ROM code suffix.

★ ### 1.14.4 Outline of functions

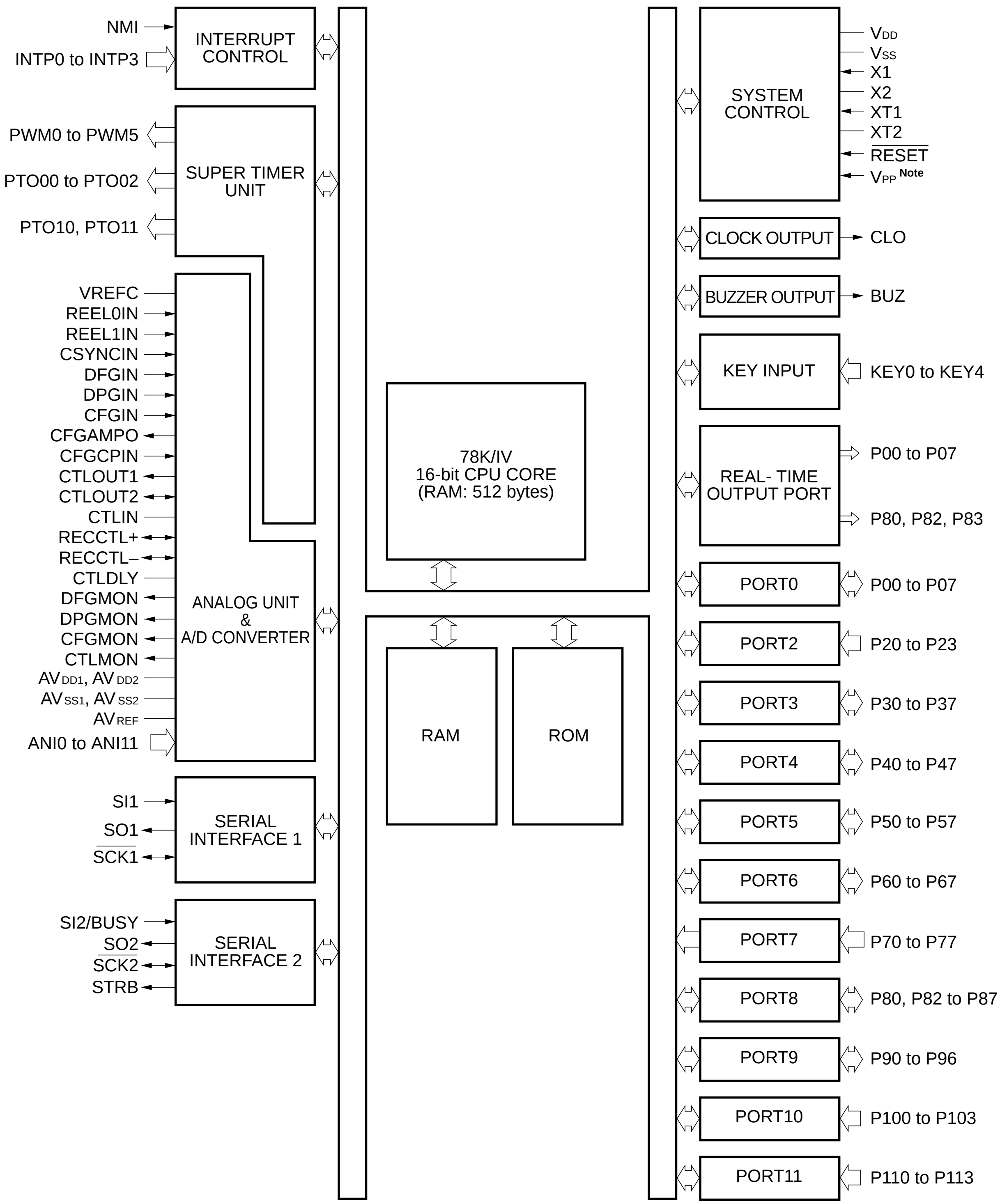
★

Item \ Product Name		μPD784927	μPD78F4928
Number of instructions		113	
Minimum instruction execution time		250 ns (internal clock: 8-MHz operation)	
Internal memory capacity	ROM	96 Kbytes (Mask ROM)	128 Kbytes (Flash memory)
	RAM	2,048 bytes	3,584 bytes
Interrupt sources	External	9 (including NMI)	
	Internal	22 (including software interrupt)	
		• 4 levels programmable priority • 3 types of processing methods Vectored interrupt, macro service, context switching	
I/O ports	Input	20	
	I/O	54 (including LED direct drive ports: 8)	
Time base counter		• 22-bit FRC • Resolution: 125 ns, maximum count time: 524 ms	
Capture registers		see below	
General-purpose timer		16-bit timer × 3	
PBCTL duty identification		• Identifies duty of recording control signal • VISS detection, wide aspect detection	
Linear time counter		5-bit UDC counts CTL signal	
Real-time output port		11	
Serial interface		3-wire serial I/O: 2 channels (including BUSY/STRB control possible: 1 channel)	
Buzzer output function		1.95 kHz, 3.91 kHz, 7.81 kHz, 15.6 kHz (internal: 8-MHz operation) 2.048 kHz, 4.096 kHz, 32.768 kHz (subsystem clock: 32.768-kHz operation)	
A/D converter		8-bit resolution × 12 channels, conversion time: 10 μs	
PWM output		• 16-bit resolution × 3 channels, 8-bit resolution × 3 channels • Carrier frequency: 62.5 kHz	
Watch function		0.5-second measurement, low-voltage operation (V_{DD} = 2.7 V) possible	
Standby function HALT mode		HALT mode/STOP mode/low power consumption mode/low power consumption	
Analog circuits		• CTL amplifier • RECCTL driver (rewriting supported) • CFG amplifier • DFG amplifier	• DPG amplifier • DPFG separation circuit (ternary separation circuit) • Reel FG comparator • CSYNC comparator
Power supply voltage		V_{DD} = +2.7 to 5.5 V	
Package		100-pin plastic QFP (14 × 20 mm)	

Capture registers:

Input signal	Number of bits	Measuring cycle	Operating edge
CFG	22	125 ns to 524 ms	↑ ↓
DFG	22	125 ns to 524 ms	↑
HSW	16	1 μs to 65.5 ms	↑ ↓
V_{SYNC}	22	125 ns to 524 ms	↑
CTL	16	1 μs to 65.5 ms	↑ ↓
T_{REEL}	22	125 ns to 524 ms	↑ ↓
S_{REEL}	22	125 ns to 524 ms	↑ ↓

1.14.5 Block diagram



Note The V_{PP} pin applies to the μPD78F4928 only.

Remark Internal ROM and RAM capacities vary depending on the products.

★ ## 1.15 Product Outline of μPD784928Y Subseries (μPD784927Y, 78F4928Y)

1.15.1 Features

- Add the I^{2}C bus interface to the μPD784928 Subseries.
- 16-bit CPU core: Minimum instruction execution time: 250 ns (at 8-MHz internal clock)
- Internal timer unit (super timer unit) for VCR servo control
- I/O ports: 74
- Internal analog circuits for VHS type VCR
  - CTL amplifier
  - RECCTL driver (supporting rewrite)
  - CFG amplifier
  - DFG amplifier
  - DPG amplifier
  - DPFG separation circuit (ternary separation circuit)
  - Reel FG comparator (2 channels)
  - CSYNC comparator
- Serial interface: 2 channels
  - 3-wire serial I/O: 2 channels
  - I^{2}C bus interface: 1 channel
- A/D converter: 12 channels (conversion time: 10 μs)
- PWM output: 16-bit resolution × 3 channels, 8-bit resolution × 3 channels
- Interrupt function
  - Vector interrupt function
  - Macro service function
  - Context switching function
- Low frequency oscillation mode: main system clock frequency = internal clock frequency
- Low power consumption mode: CPU can operate on subsystem clock.
- Power supply voltage: V_{DD} = 2.7 to 5.5 V
- Hardware watch function: Low-voltage (V_{DD} = 2.7 V MIN.), low-current consumption operation

1.15.2 Applications

For stationary type and camcorder type VCRs.

★ 1.15.3 Ordering information

(1) Ordering information

Part Number	Package	Internal ROM
μPD784927YGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD78F4928YGF-3BA	100-pin plastic QFP (14 × 20 mm)	Flash memory

Remark ××× indicates ROM code suffix.

(2) Quality grade

Part Number	Package	Quality Grade
μPD784927YGF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
μPD78F4928YGF-3BA	100-pin plastic QFP (14 × 20 mm)	Standard

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

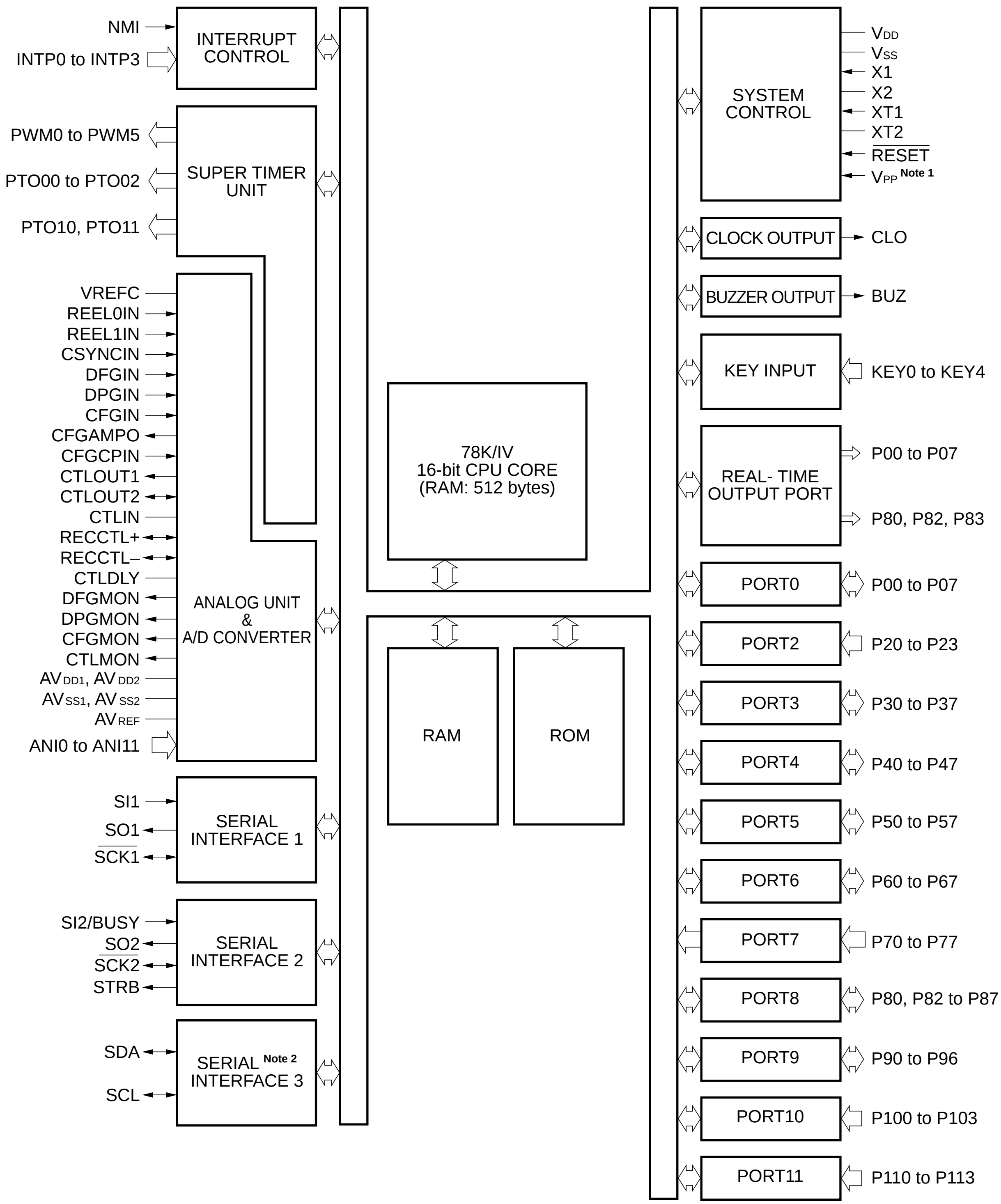
Remark ××× indicates ROM code suffix.

1.15.4 Outline of functions

★

Item \ Product Name		µPD784927Y	µPD78F4928Y
Number of instructions		113	
Minimum instruction execution time		250 ns (internal clock: 8-MHz operation)	
Internal memory capacity	ROM	96 Kbytes (mask ROM)	128 Kbytes (flash memory)
	RAM	2,048 bytes	3,584 bytes
Interrupt sources	External	9 (including NMI)	
	Internal	23 (including software interrupt)	
		• 4 levels programmable priority • 3 types of processing methods Vectored interrupt, macro service, context switching	
I/O ports	Input	20	
	I/O	54 (including LED direct drive ports: 8)	
Time base counter		• 22-bit FRC • Resolution: 125 ns, maximum count time: 524 ms	
Capture registers		Input signal / Number of bits / Measuring cycle / Operating edge CFG / 22 / 125 ns to 524 ms / ↑ ↓ DFG / 22 / 125 ns to 524 ms / ↑ HSW / 16 / 1 µs to 65.5 ms / ↑ ↓ V_{SYNC} / 22 / 125 ns to 524 ms / ↑ CTL / 16 / 1 µs to 65.5 ms / ↑ ↓ T_{REEL} / 22 / 125 ns to 524 ms / ↑ ↓ S_{REEL} / 22 / 125 ns to 524 ms / ↑ ↓	
General-purpose timer		16-bit timer × 3	
PBCTL duty identification		• Identifies duty of recording control signal • VISS detection, wide aspect detection	
Linear time counter		5-bit UDC counts CTL signal	
Real-time output port		11	
Serial interface		• 3-wire serial I/O: 2 channels (including BUSY/STRB control possible: 1 channel) • I^2C bus interface (multimaster supported): 1 channel	
Buzzer output function		1.95 kHz, 3.91 kHz, 7.81 kHz, 15.6 kHz (internal: 8-MHz operation) 2.048 kHz, 4.096 kHz, 32.768 kHz (subsystem clock: 32.768-kHz operation)	
A/D converter		8-bit resolution × 12 channels, conversion time: 10 µs	
PWM output		• 16-bit resolution × 3 channels, 8-bit resolution × 3 channels • Carrier frequency: 62.5 kHz	
Watch function		0.5-second measurement, low-voltage operation (V_{DD} = 2.7 V) possible	
Standby function HALT mode		HALT mode/STOP mode/low power consumption mode/low power consumption	
Analog circuits		• CTL amplifier • RECCTL driver (rewriting supported) • CFG amplifier • DFG amplifier	• DPG amplifier • DPFG separation circuit (ternary separation circuit) • Reel FG comparator • CSYNC comparator
Power supply voltage		V_{DD} = +2.7 to 5.5 V	
Package		100-pin plastic QFP (14 × 20 mm)	

1.15.5 Block diagram



Notes 1. The V_{PP} pin applies to the μPD78F4928Y only.

2. I^2C bus interface supported.

Remark Internal ROM and RAM capacities vary depending on the products.

★

1.16 Product Outline of μPD784937 Subseries (μPD784935, 784936, 78F4937, 78F4937)

1.16.1 Features

- Inherits the peripheral functions of the μPD784908 Subseries
- Minimum instruction execution time: 160 ns (at f_{XX} = 12.5-MHz operation)
- On-chip memory
  - Mask ROM : 96 Kbytes (μPD784935)
    128 Kbytes (μPD784936)
    192 Kbytes (μPD784937)
  - Flash memory : 192 Kbytes (μPD78F4937)
  - RAM : 5,120 bytes (μPD784935)
    : 6,656 bytes (μPD784936)
    : 8,192 bytes (μPD784937, 78F4937)
- I/O port: 80
- Timer/counter: 16-bit timer/counter × 1 unit
  16-bit timer/counter × 2 units
  16-bit timer × 1 unit
- Serial interface: 4 channels
  - UART/IOE (3-wire serial I/O): 2 channels (on-chip baud-rate generator)
  - CSI (3-wire serial I/O): 2 channels
- PWM output: 2 outputs
- Standby function
  HALT/STOP/IDLE mode
- Clock frequency dividing function
- Clock output function: Selectable from f_{XX}, $f_{XX}/2$, $f_{XX}/2^2$, $f_{XX}/2^3$, $f_{XX}/2^4$, $f_{XX}/2^5$
- External expansion function
- Internal ROM correction function
- A/D converter: 8-bit resolution × 8 channels
- Internal IEBus controller
- Watchdog timer: 1 channel
- Low power consumption
- Power supply voltage: V_{DD} = 2.7 to 5.5 V

1.16.2 Applications

Car audios, etc.

1.16.3 Ordering information and quality grade

(1) Ordering information

Part Number	Package	Internal ROM
μPD784935GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD784935GC-×××-8EU	100-pin plastic LQFP (14 × 14 mm)	Mask ROM
μPD784936GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD784936GC-×××-8EU	100-pin plastic LQFP (14 × 14 mm)	Mask ROM
μPD784937GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Mask ROM
μPD784937GC-×××-8EU	100-pin plastic LQFP (14 × 14 mm)	Mask ROM
μPD78F4937GF-3BA	100-pin plastic QFP (14 × 20 mm)	Flash Memory
μPD78F4937GC-8EU	100-pin plastic LQFP (14 × 14 mm)	Flash Memory

Remark ××× indicates ROM code suffix.

(2) Quality grades

Part Number	Package	Quality Grade
μPD784935GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
μPD784935GC-×××-8EU	100-pin plastic LQFP (14 × 14 mm)	Standard
μPD784936GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
μPD784936GC-×××-8EU	100-pin plastic LQFP (14 × 14 mm)	Standard
μPD784937GF-×××-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
μPD784937GC-×××-8EU	100-pin plastic LQFP (14 × 14 mm)	Standard
μPD78F4937GF-3BA	100-pin plastic QFP (14 × 20 mm)	Standard
μPD78F4937GC-8EU	100-pin plastic LQFP (14 × 14 mm)	Standard

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

Remark ××× indicates ROM code suffix.

Caution The μPD784937 Subseries is under development.

1.16.4 Outline of functions

(1/2)

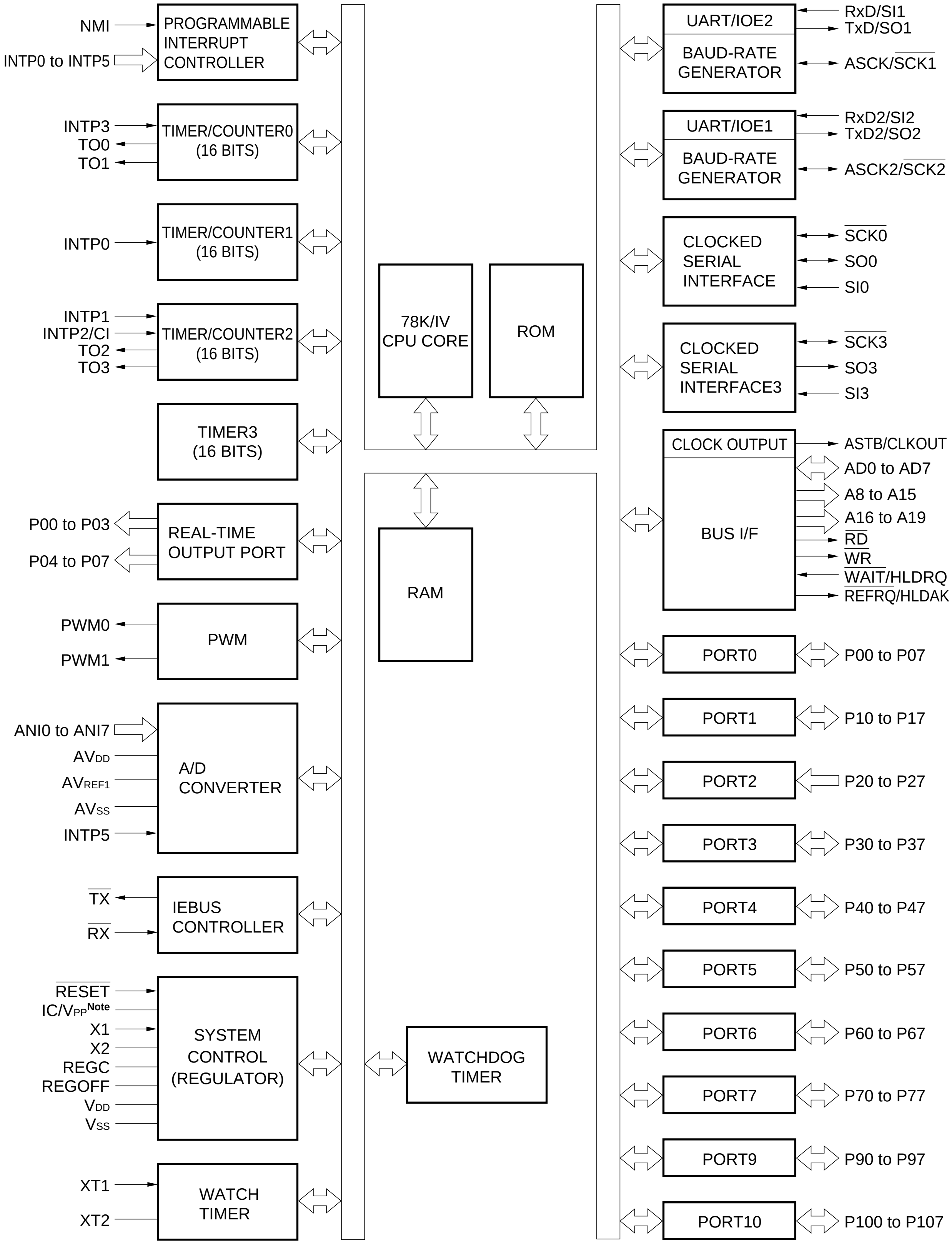
Item		Product Name: µPD784935	µPD784936	µPD784937	µPD78F4937
Number of basic instructions (mnemonics)		113			
General registers		8 bits × 16 registers × 8 banks or 16 bits × 8 registers × 8 banks (memory mapping)			
Minimum instruction execution time		160 ns/320 ns/636 ns/1.27 µs (at 12.58-kHz operation)			
Internal memory capacity	ROM	92 Kbytes (Mask ROM)	128 Kbytes (Mask ROM)	192 Kbytes (Mask ROM)	192 Kbytes (Flash memory)
	RAM	5,120 bytes	6,656 bytes	8,192 bytes	
Memory space		1 Mbyte in total of program and data			
I/O ports	Total	80			
	Inputs	8			
	I/O	72			
Pins with added functions[Note]	LED direct drive outputs	24			
	Transistor direct drive	8			
	N-ch open-drain	4			
Real-time output ports		4 bits × 2, or 8 bits × 1			
IEBus controller		Internal (simplify)			
Timer/counters		Timer/counter 0: (16 bits)	Timer register × 1 Capture register × 1 Compare register × 2	Pulse output capability • Toggle output • PWM/PPG output • One-shot pulse output	
		Timer/counter 1: (16 bits)	Timer register × 1 Capture register × 1 Capture/compare register × 1 Compare register × 1	Real-time output port	
		Timer/counter 2: (16 bits)	Timer register × 1 Capture register × 1 Capture/compare register × 1 Compare register × 1	Pulse output capability • Toggle output • PWM/PPG output	
		Timer 3: (16 bits)	Timer register × 1 Compare register × 1	Pulse output capability • Toggle output • PWM/PPG output	
Watch timer		Interrupt occurs at an interval of 0.5 sec. (Has an internal clock oscillator.) The input clock can be selected from among the main clock (12.58 MHz) or clock (32.7 kHz).			
PWM output		12-bit resolution × 2 channels			
Serial interfaces		UART/IOE (3-wire serial I/O) : 2 channels (on-chip baud rate generator) CSI (3-wire serial I/O): 2 channels			
A/D converter		8-bit resolution × 8 channels			
Clock output function		Selectable from f_{CLK}, $f_{CLK}/2$, $f_{CLK}/4$, $f_{CLK}/8$, $f_{CLK}/16$ (can be used as 1-bit output port)			
Watchdog timer		1 channel			

Note The pins with additional functions are included in the I/O pins.

(2/2)

Item \ Product Name		μPD784935	μPD784936	μPD784937	μPD78F4937
ROM correction function		Internal (can be set for 4 points of correction address)			
External expansion function		Available (can be set up to 1 Mbyte)			
Standby function		HALT/STOP/IDLE mode			
Interrupts	Hardware sources	27 (internal: 20, external: 7 (sampling clock variable input: 1))			
	Software sources	BRK instruction, BRKCS instruction, operand error			
	Non-maskable	Internal: 1, external: 1			
	Maskable	Internal: 19, external: 6			
		4-level programmable priority Three processing formats: Macro service/vectored interrupt/context switching			
Power supply voltage		V_{DD} = 2.7 to 5.5 V			
Package		• 100-pin plastic QFP (14 × 20 mm) • 100-pin plastic LQFP (14 × 14 mm)			

1.16.5 Block diagram



Note In the flash memory programming mode of the μPD78F4937.

Remark Internal ROM and RAM capacities vary depending on the products.

★ 1.17 Product Outline of μPD784955 Subseries (μPD784953, 784955, 78F4956)

1.17.1 Features

- Minimum instruction execution time: 160 ns (at f_{CLK} = 12.5-MHz operation)
- On-chip memory
  - ROM
    Mask ROM : 24 Kbytes (μPD784953)
    48 Kbytes (μPD784955)
    Flash memory : 64 Kbytes (μPD78F4956)
  - RAM : 768 bytes (μPD784953)
    : 2,048 bytes (μPD784955, 78F4956)
- I/O port : 67
- Timer/counter: 16-bit timer/counter × 6 units
  8-bit timer/counter × 2 units
- Serial interface: 2 channels
  UART: 1 channel (on-chip baud rate generator)
  CSI (3-wire serial I/O): 1 channel
- A/D converter: 8-bit resolution × 8 channels
- Real-time output function: 6-bit resolution × 2 channels
- Watchdog timer: 1 channel
- Standby function
  HALT/STOP/IDLE mode
  Low power consumption mode: HALT/IDLE mode (subsystem clock operation)
- Interrupt controller (4-level priority)
  Vector interrupt/macro service/context switching
- Power supply voltage: V_{DD} = 4.5 to 5.5 V

1.17.2 Applications

Motor control for inverter air conditioners, etc.

1.17.3 Ordering information and quality grade

(1) Ordering information

Part Number	Package	Internal ROM
μPD784935GC-××× -8BT	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD784955GC-××× -8BT	80-pin plastic QFP (14 × 14 mm)	Mask ROM
μPD78F4956GC-8BT	80-pin plastic QFP (14 × 14 mm)	Flash Memory

Remark ××× indicates ROM code suffix.

(2) Quality grades

Part Number	Package	Quality Grade
μPD784935GC-××× -8BT	80-pin plastic QFP (14 × 14 mm)	Standard
μPD784955GC-××× -8BT	80-pin plastic QFP (14 × 14 mm)	Standard
μPD78F4956GC-8BT	80-pin plastic QFP (14 × 14 mm)	Standard

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

Remark ××× indicates ROM code suffix.

Caution The μPD784955 Subseries is under development.

1.17.4 Outline of functions

(1/2)

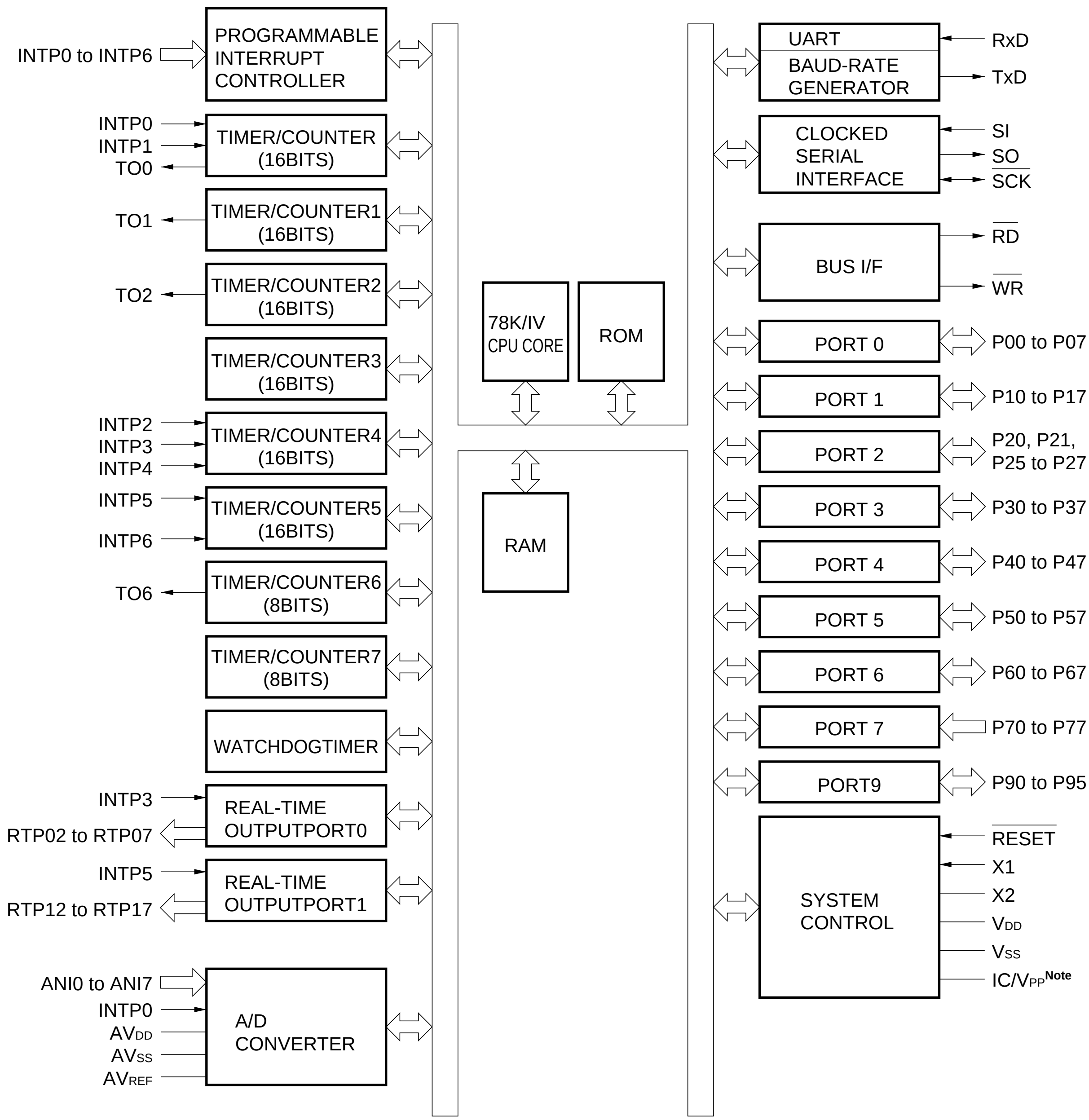
Item			μPD784953	μPD784955	μPD78F4956
Number of basic instructions (mnemonics)			113		
General registers			8 bits × 16 registers × 8 banks or 16 bits × 8 registers × 8 banks (memory mapped)		
Minimum instruction execution time			160 ns (at f_{CLK} = 12.5-MHz operation)		
Internal memory capacity	ROM		24 Kbytes (Mask ROM)	48 Kbytes (Mask ROM)	64 Kbytes (Flash memory)
	RAM		768 bytes	2,048 bytes	
I/O port	Total		67		
	CMOS input		8		
	CMOS I/O		59		
	Additional function pin Note	Pin with pull-up resistor	59		
		LED direct drive output	32		
Real-time output port			6 bits × 2		
Timer/counter			16-bit timer/counter: Timer register × 1 Capture/compare register × 2 Pulse output capability • PWM output		
			16-bit timer/counter 1: Timer register × 1 Compare register × 2 Pulse output capability • PWM output		
			16-bit timer/counter 2: Timer register × 1 Compare register × 2 Pulse output capability • PWM output		
			16-bit timer/counter 3: Timer register × 1 Compare register × 2		
			16-bit timer/counter 4: Timer register × 1 Capture/compare register × 3		
			16-bit timer/counter 5: Timer register × 1 Compare register × 1 Capture/compare register × 2		
			8-bit timer/counter 6: Timer register × 1 Compare register × 1 Pulse output capability • PWM output		
			8-bit timer/counter 7: Timer register × 1 Compare register × 1		
Serial interface			• UART: 1 channel (on-chip baud rate generator) • CSI (3-wire serial I/O): 1 channel		
A/D converter			8-bit resolution × 8 channels		
Watchdog timer			1 channel		
Standby function			HALT/STOP/IDLE mode		

Note The pins with additional functions are included in the I/O pins.

(2/2)

Item \ Product Name		μPD784953	μPD784955	μPD78F4956
Interrupt	Hardware sources	28 (internal: 22, external: 8 (shared with internal: 2))		
	Software sources	BRK instruction, BRKCS instruction, operand error		
	Non-maskable	Internal: 1, external: 1		
	Maskable	Internal: 20, external: 7		
		• 4-level programmable priority • 3 processing modes: vectored interrupt, macro service, context switching		
Power supply voltage		V_{DD} = 4.5 to 5.5 V		
Package		80-pin plastic QFP (14 × 14 mm)		

1.17.5 Block diagram



Note In the flash memory programming mode of the μPD78F4956.

Remark Internal ROM and RAM capacities vary depending on the products.

2.1 Memory Space

The 78K/IV Series can access a maximum memory space of 16 Mbytes. However, memory mapping varies from product to product according to the on-chip memory capacity and pin status. Therefore, the **User's Manual — Hardware** for the individual products should be consulted for details of the memory map address areas.

The 78K/IV Series can access a 16-Mbyte memory space. The mapping of the internal data area (special function registers and internal RAM) depends on the LOCATION instruction. A LOCATION instruction must be executed after reset release, and can only be used once.

The program after reset release must be as follows.

```
RSTVCT   CSEG  AT   0
         DW    RSTSTRT
             ≀
INITSEG  CSEG  BASE
RSTSTRT:LOCATION 0H; or LOCATION 0FH
         MOVG SP, #STKBGN
```

(1) When LOCATION 0 instruction is executed

The internal data area is mapped with the maximum address as FFFFH.

An area in the internal ROM that overlaps an internal data area cannot be used as internal ROM when the LOCATION 0 instruction is executed.

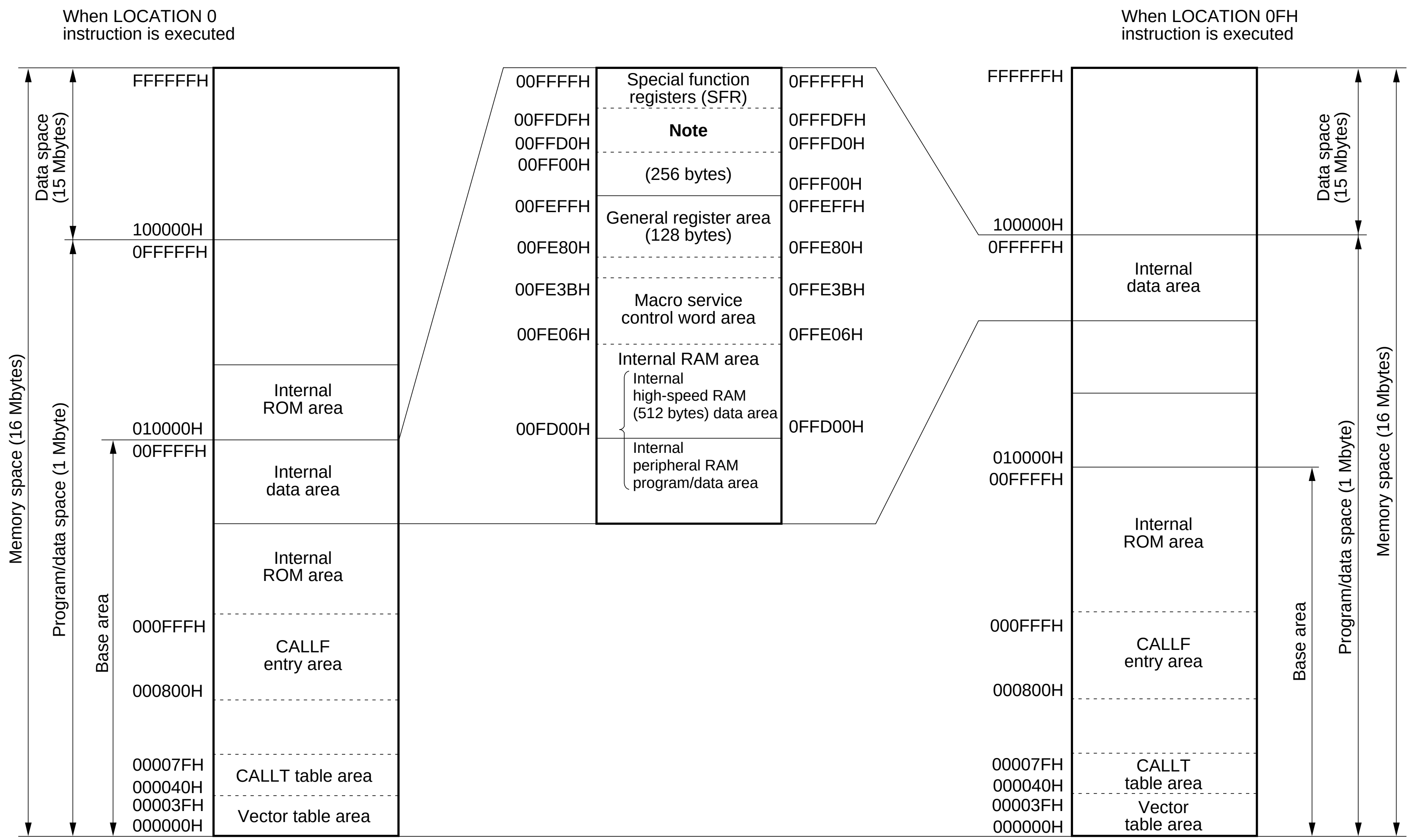
External memory is accessed in external memory extension mode.

(2) When LOCATION 0FH instruction is executed

The internal data area is mapped with the maximum address as FFFFFH.

The μPD784915 Subseries is fixed to the LOCATION 0 instruction.

Figure 2-1. Memory Map



Note External SFR area

Caution The μPD784915 Subseries is fixed to the LOCATION 0 instruction.

2.2 Internal ROM Area

The 78K/IV Series products shown below incorporate ROM which is used to store programs, table data, etc.

If the internal ROM area and internal data area overlap when the LOCATION 0 instruction is executed, the internal data area is accessed, and the overlapping part of the internal ROM area cannot be accessed.

The μPD784915 Subseries is fixed to the LOCATION 0 instruction.

Table 2-1. List of Internal ROM Space for 78K/IV Series Products (1/2)

Subseries Name	Product	Address Space	Internal ROM
μPD784026 Subseries	μPD784020 μPD784021	None	
	μPD784025	00000H to 0BFFFH	48 K × 8 bits
	μPD784026 μPD78P4026	00000H to 0FFFFH	64 K × 8 bits
μPD784038 Subseries μPD784038Y Subseries	μPD784031 μPD784031Y	None	
	μPD784035 μPD784035Y	00000H to 0BFFFH	48 K × 8 bits
	μPD784036 μPD784036Y	00000H to 0FFFFH	64 K × 8 bits
	μPD784037 μPD784037Y	00000H to 17FFFH	96 K × 8 bits
	μPD784038 μPD78P4038 μPD784038Y μPD78P4038Y	00000H to 1FFFFH	128 K × 8 bits
μPD784046 Subseries	μPD784044 μPD784054	00000H to 07FFFH	32 K × 8 bits
	μPD784046 μPD78F4046	00000H to 0FFFFH	64 K × 8 bits
μPD784216 Subseries μPD784216Y Subseries	μPD784214 μPD784214Y	00000H to 17FFFH	96 K × 8 bits
	μPD784215 μPD784215Y μPD784216 μPD784216Y μPD78F4216 μPD78F4216Y	00000H to 1FFFFH	128 K × 8 bits
μPD784218 Subseries μPD784218Y Subseries	μPD784217 μPD784217Y	00000H to 2FFFFH	192 K × 8 bits
	μPD784218 μPD784218Y μPD78F4218 μPD78F4218Y	00000H to 3FFFFH	256 K × 8 bits

Remark In case of a ROM-less product, this address space is an external memory.

Table 2-1. List of Internal ROM Space for 78K/IV Series Products (2/2)

	Subseries Name	Product	Address Space	Internal ROM
	μPD784225 Subseries μPD784225Y Subseries	μPD784224 μPD784224Y	00000H to 17FFFH	96 K × 8 bits
		μPD784225 μPD784225Y μPD78F4225 μPD78F4225Y	00000H to 1FFFFH	128 K × 8 bits
	μPD784908 Subseries	μPD784907	00000H to 17FFFH	96 K × 8 bits
		μPD784908 μPD78P4908	00000H to 1FFFFH	128 K × 8 bits
	μPD784915 Subseries	μPD784915 μPD784915A	00000H to 0BFFFH	48 K × 8 bits
		μPD784916A μPD78P4916	00000H to 0F6FFH	62 K × 8 bits
	μPD784928 Subseries μPD784928Y Subseries	μPD784927 μPD784927Y	00000H to 17FFFH	96 K × 8 bits
★		μPD78F4928 μPD78F4928Y	00000H to 1FFFFH	128 K × 8 bits
★	μPD784937 Subseries	μPD784935	00000H to 17FFFH	96 K × 8 bits
		μPD784936	00000H to 1FFFFH	128 K × 8 bits
		μPD784937 μPD78F4937	00000H to 2FFFFH	192 K × 8 bits
★	μPD784955 Subseries	μPD784953	00000H to 05FFFH	24 K × 8 bits
		μPD784955	00000H to 0BFFFH	48 K × 8 bits
		μPD78F4956	00000H to 0F6FFH	64 K × 8 bits

2.3 Base Area

The space from 00000H to FFFFFH comprises the base area. The base area is the object for the following uses.

- Reset entry address
- Interrupt entry address
- CALLT instruction entry address
- 16-bit immediate addressing mode (with instruction address addressing)
- 16-bit direct addressing mode
- 16-bit register addressing mode (with instruction address addressing)
- 16-bit register indirect addressing mode
- Short direct 16-bit memory indirect addressing mode

The vector table area, CALLT instruction table area and CALLF instruction entry area are allocated to the base area.

When the LOCATION 0 instruction is executed, the internal data area is located in the base area. Note that, in the internal data area, program fetches cannot be performed from the internal high-speed RAM area and special function register (SFR) area. Also, internal RAM area data should only be used after initialization has been performed.

2.3.1 Vector table area

The 64-byte area from 00000H to 0003FH is reserved as the vector table area. The vector table area holds the program start addresses used when a jump is performed as the result of $\overline{\text{RESET}}$ input or generation of an interrupt request. When context switching is used by an interrupt, the number of the register bank to be switched to is stored here.

Any portion not used by the vector table can be used as program memory or data memory.

16-bit values can be written to the vector table. Therefore, branches can only be made within the base area.

Table 2-2. Vector Table

Vector Table Address	Interrupts
00000H	Reset ($\overline{\text{RESET}}$ input)
00002H	NMI **Note**
00004H	WDT **Note**
00006H to 0003AH	Differs for each product
0003CH	Operand error interrupt
0003EH	BRK

Note Not used by some products.

2.3.2 CALLT instruction table area

The 1-byte call instruction (CALLT) subroutine entry addresses can be stored in the 64-byte area from 00040H to 0007FH.

The CALLT instruction references this table, and branches to a base area address written in the table as a subroutine. As the CALLT instruction is one byte in length, use of the CALLT instruction for subroutine calls written frequently throughout the program enables the program object size to be reduced. The table can contain up to 32 subroutine entry addresses, and therefore it is recommended that they be recorded in order of frequency.

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

Values that can be written to the CALLT instruction table are 16-bit values. Therefore, a branch can only be made within the base area.

2.3.3 CALLF instruction entry area

A subroutine call can be made directly to the area from 00800H to 00FFFH with the 2-byte call instruction (CALLF).

As the CALLF instruction is a two-byte call instruction, it enables the object size to be reduced compared with use of the direct subroutine call CALL instruction (3 bytes).

Writing subroutines directly in this area is an effective means of exploiting the high-speed capability of the device.

If you wish to reduce the object size, writing an unconditional branch (BR) instruction in this area and locating the subroutine itself outside this area will result in a reduced object size for subroutines that are called from five or more points. In this case, only the 4 bytes of the BR instruction are occupied in the CALLF entry area, enabling the object size to be reduced with a large number of subroutines.

2.4 Internal Data Area

The internal data area comprises the internal RAM area and special function register area. In some products, memories dependent on other hardware are also allocated to this areas (see the **User's Manual — Hardware** of each product).

The final address of the internal data area can be specified by means of the LOCATION instruction as either FFFFH (when a LOCATION 0 instruction is executed) or FFFFFH (when a LOCATION 0FH instruction is executed). Selection of the addresses of the internal data area by means of the LOCATION instruction must be executed once immediately after reset release, and once the selection is made, it cannot be changed. The program after reset release must be as shown in the example below. If the internal data area and another area are allocated to the same addresses, the internal data area is accessed and the other area cannot be accessed.

Example

```
RSTVCT   CSEG  AT 0
         DW    RSTSTRT
          ≀
INITSEG  CSEG  BASE
RSTSTRT: LOCATION 0H; or LOCATION 0FH
         MOVG SP, #STKBGN
```

Cautions 1. When the LOCATION 0 instruction is executed, it is necessary to ensure that the program after reset release does not overlap the internal data area. It is also necessary to make sure that the entry addresses of the service routines for non-maskable interrupts such as NMI do not overlap the internal data area. Also, initialization must be performed for maskable interrupt entry areas, etc., before the internal data area is referenced.

2. The μPD784915 Subseries is fixed to the LOCATION 0 instruction.

2.4.1 Internal RAM area

78K/IV Series products incorporate general-purpose static RAM.

This area is configured as follows:

- Internal RAM area
  - Peripheral RAM (PRAM)
  - Internal high-speed RAM (IRAM)

Table 2-3. Internal RAM Area in 78K/IV Series Products (1/2)

Subseries Name	Product	Internal RAM area		
			Peripheral RAM: PRAM	Internal high-speed RAM: IRAM
μPD784026 Subseries	μPD784020	512 Bytes (0FD00H to 0FEFFH)	0 Byte	512 Bytes (0FD00H to 0FEFFH)
	μPD784021 μPD784025 μPD784026 μPD78P4026	2,048 Bytes (0F700H to 0FEFFH)	1,536 Bytes (0F700H to 0FCFFH)	
μPD784038 Subseries μPD784038Y Subseries	μPD784031 μPD784031Y μPD784035 μPD784036 μPD784035Y μPD784036Y	2,048 Bytes (0F700H to 0FEFFH)	1,536 Bytes (0F700H to 0FCFFH)	
	μPD784037 μPD784037Y	3,584 Bytes (0F100H to 0FEFFH)	3,072 Bytes (0F100H to 0FCFFH)	
	μPD784038 μPD78P4038 μPD784038Y μPD78P4038Y	4,352 Bytes (0EE00H to 0FEFFH)	3,840 Bytes (0FE00H to 0FCFFH)	
μPD784046 Subseries	μPD784044 μPD784045	1,024 Bytes (0FB00H to 0FEFFH)	512 Bytes (0FB00H to 0FCFFH)	
	μPD784046 μPD78F4046	2,048 Bytes (0F700H to 0FEFFH)	1,536 Bytes (0F700H to 0FCFFH)	
μPD784216 Subseries μPD784216Y Subseries	μPD784214 μPD784214Y	3,584 Bytes (0F100H to 0FEFFH)	3,072 Bytes (0F100H to 0FCFFH)	
	μPD784215 μPD784215Y	5,120 Bytes (0EB00H to 0FEFFH)	4,608 Bytes (0FB00H to 0FCFFH)	
	μPD784216 μPD784216Y μPD78F4216 μPD78F4216Y	8,192 Bytes (0DF00H to 0FEFFH)	7,680 Bytes (0DF00H to 0FCFFH)	
μPD784218 Subseries μPD784218Y Subseries	μPD784217 μPD784217Y	12,800 Bytes (0CD00H to 0FEFFH)	12,288 Bytes (0CD00H to 0FCFFH)	
	μPD784218 μPD784218Y μPD78F4218 μPD78F4218Y			
μPD784225 Subseries μPD784225Y Subseries	μPD784224 μPD784224Y	3,584 Bytes (0F100H to 0FEFFH)	3,072 Bytes (0CF10H to 0FCFFH)	
	μPD784225 μPD784225Y μPD78F4225 μPD78F4225Y	4,352 Bytes (0EE00H to 0FEFFH)	3,840 Bytes (0EE00H to 0FCFFH)	

Remark The addresses in the table are the values that apply when the LOCATION 0 instruction is executed. When the LOCATION 0FH instruction is executed, 0F0000H should be added to the values shown above.

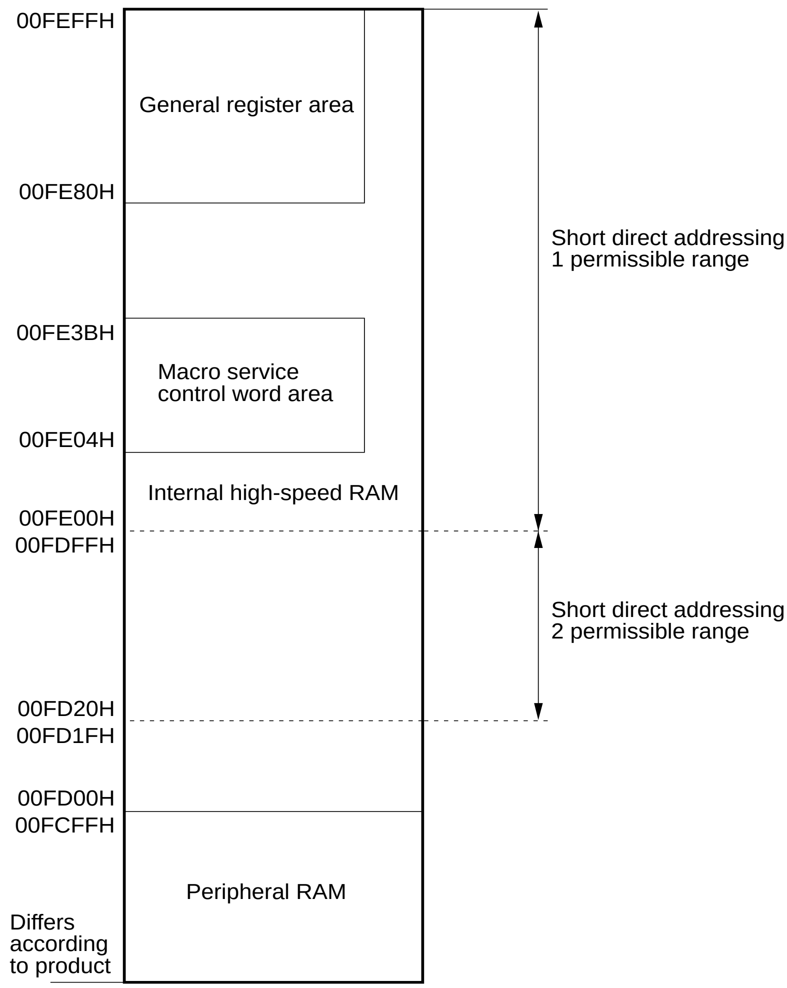
Table 2-3. Internal RAM Area in 78K/IV Series Products (2/2)

Subseries Name	Product	Internal RAM area		
			Peripheral RAM: PRAM	Internal high-speed RAM: IRAM
μPD784908 Subseries	μPD784907	3,584 Bytes (0F100H to 0FEFFH)	3,072 Byte (0F100H to 0FEFFH)	512 Bytes (0FD00H to 0FEFFH)
	μPD784908 μPD78P4908	4,352 Bytes (0EE00H to 0FEFFH)	3,840 Bytes (0EE00H to 0FCFFH)	
μPD784915 Subseries	μPD784915 μPD784915A μPD784916A	1,280 Bytes (0FA00H to 0FEFFH)	768 Bytes (0FA00H to 0FCFFH)	
	μPD78P4916	2,048 Bytes (0F700H to 0FEFFH)	1,536 Bytes (0F700H to 0FCFFH)	
μPD784928 Subseries μPD784928Y Subseries	μPD784927 μPD784927Y	2,048 Bytes (0F700H to 0FEFFH)	1,536 Bytes (0F700H to 0FCFFH)	
★	μPD78F4928 μPD78F4928Y	3,584 Bytes (0F100H to 0FEFFH)		
★ μPD784937 Subseries	μPD784935	5,120 Bytes (0EB00H to 0FEFFH)	4,608 Bytes (0EB00H to 0FCFFH)	
	μPD784936	6,656 Bytes (0E500H to 0FEFFH)	6,144 Bytes (0E500H to 0FCFFH)	
	μPD784937 μPD78F4937	8,192 Bytes (0DF00H to 0FEFFH)	7,680 Bytes (0DF00H to 0FCFFH)	
★ μPD784955 Subseries	μPD784953	768 Bytes (0FC00H to 0FEFFH)	256 Bytes (0FC00H to 0FCFFH)	
	μPD784955 μPD78F4956	2,048 Bytes (0F700H to 0FEFFH)	1,536 Bytes (0F700H to 0FCFFH)	

Remark The addresses in the table are the values that apply when the LOCATION 0 instruction is executed. When the LOCATION 0FH instruction is executed, 0F0000H should be added to the values shown above. The μPD784915 Subseries is fixed to the LOCATION 0 instruction.

Internal RAM mapping is shown in Figure 2-2.

Figure 2-2. Internal RAM Memory Mapping



Remark The addresses in the figure are the values that apply when the LOCATION 0 instruction is executed. When the LOCATION 0FH instruction is executed, 0F0000H should be added to the values shown above. The μPD784915 Subseries is fixed to the LOCATION 0 instruction.

(1) Internal high-speed RAM (IRAM)

The internal high-speed RAM (IRAM) allows high-speed accesses to be made. The short direct addressing mode for high-speed accesses can be used on 0FD20H to 0FEFFH in this area. There are two kinds of short direct addressing mode, short direct addressing 1 and short direct addressing 2, according to the target address. The function is the same in both of these addressing modes. With some instructions, the word length is shorter with short direct addressing 2 than with short direct addressing 1. See **CHAPTER 6 INSTRUCTION SET** for details.

A program fetch cannot be performed from IRAM. If a program fetch is performed from an address onto which IRAM is mapped, CPU runaway will result.

The following areas are reserved in IRAM.

- General register area : 0FE80H to 0FEFFH
- Macro service control word area : 0FE06H to 0FE3BH (the addresses actually reserved differ from product to product)
- Macro service channel area : 0FE00H to 0FEFFH (the address is specified by the macro service control word)

If the reserved function is not used in these areas, they can be used as ordinary data memory.

Remark The addresses in this text are those that apply when the LOCATION 0 instruction is executed. When the LOCATION 0FH instruction is executed, 0F0000H should be added to the values shown. The μPD784915 Subseries is fixed to the LOCATION 0 instruction.

(2) Peripheral RAM (PRAM)

The peripheral RAM (PRAM) is used as ordinary program memory or data memory. When used as program memory, he program must be written to the peripheral RAM beforehand by a program.

2.4.2 Special function register (SFR) area

The on-chip peripheral hardware special function registers (SFRs) are mapped onto the area from 0FF00H to 0FFFFH (see the **User's Manual — Hardware** for the individual products).

In some products, the area from 0FFD0H to 0FFDFH is mapped as an external SFR area, and allows externally connected peripheral I/Os, etc., to be accessed in external memory extension mode (specified by the memory extension mode register (MM)) by ROM-less products or on-chip ROM products.

Caution Addresses onto which SFRs are not mapped should not be accessed in this area. If such an address is accessed by mistake, the CPU may become deadlocked. A deadlock can only be released by reset input.

Remark The addresses in this text are those that apply when the LOCATION 0 instruction is executed. When the LOCATION 0FH instruction is executed, 0F0000H should be added to the values shown.
The μPD784915 Subseries is fixed to the LOCATION 0 instruction.

2.4.3 External SFR area

In some 78K/IV Series products, the 16-byte area from 0FFD0H to 0FFDFH in the SFR area (when the LOCATION 0 instruction is executed; 0FFFD0H to 0FFFDFH when the LOCATION 0FH instruction is executed) is mapped as an external SFR area. When the external memory extension mode is set in a ROM-less product or on-chip ROM product, externally connected peripheral I/Os, etc., can be accessed using the address bus or address/data bus, etc.

As the external SFR area can be accessed by SFR addressing, peripheral I/O and similar operations can be performed easily, the object size can be reduced, and macro service can be used.

Bus operations for accesses to the external SFR area are performed in the same way as for ordinary memory accesses.

2.5 External Memory Space

The external memory space is a memory space that can be accessed in accordance with the setting of the memory extension mode register (MM). It can hold programs, table data, etc., and can have peripheral I/O devices allocated to it.

A program cannot be allocated to the area from 100000H to 0FFFFFFH in the external memory space.

Note also that some products do not have an external memory space.

3.1 Control Registers

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

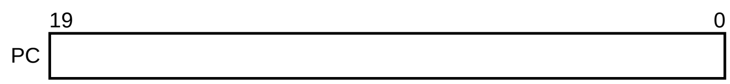
3.1.1 Program counter (PC)

This is a 20-bit binary counter that holds information on the next program address to be executed (see **Figure 3-1**).

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

Upon $\overline{\text{RESET}}$ input, the 16-bit data in address 0 and address 1 is set in the low-order 16 bits of the PC, and 0000 in the high-order 4 bits.

Figure 3-1. Program Counter (PC) Configuration



3.1.2 Program status word (PSW)

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

Read accesses and write accesses are performed in high-order 8-bit (PSWH) and low-order 8-bit (PSWL) units. Individual flags can be accessed by bit-manipulation instructions.

The contents of the PSW are automatically saved to the stack when a vectored interrupt request is acknowledged or a BRK instruction is executed, and automatically restored when an RETI or RETB instruction is executed. When context switching is used, the contents are automatically saved in RP3, and automatically restored when an RETCS or RETCSB instruction is executed.

$\overline{\text{RESET}}$ input resets (0) all bits.

“0” must always be written to the bits written as “0” in Figure 3-2. The contents of bits written as “–” are undefined when read.

Figure 3-2. Program Status Word (PSW) Configuration

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

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

The flags are described below.

(1) Carry flag (CY)

The carry flag stores a carry or borrow resulting from an operation.

This flag also stores the shifted-out value when a shift/rotate instruction is executed, and functions as a bit accumulator when a bit-manipulation instruction is executed.

The status of the CY flag can be tested with a conditional branch instruction.

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

The P/V flag performs the following two kinds of operation associated with execution of an operation instruction.

The status of the P/V flag can be tested with a conditional branch instruction.

- **Parity flag operation**

  Set (1) when the number of bits set (1) as the result of execution of a logical operation instruction, shift/ rotate instruction, or a CHKL or CHKLA instruction is even, and reset (0) if odd. When a 16-bit shift instruction is executed, however, only the low-order 8 bits of the operation result are valid for the parity flag.

- **Overflow flag operation**

  Set (1) when the numeric range expressed as a two's complement is exceeded as the result of execution of a logical operation instruction, and reset (0) otherwise. More specifically, the value of this flag is the exclusive OR of the carry into the MSB and the carry out of the MSB. For example, the two's complement range in an 8-bit arithmetic operation is 80H (–128) to 7FH (+127), and the flag is set (1) if the operation result is outside this range, and reset (0) if within this range.

Example The operation of the overflow flag when an 8-bit addition instruction is executed is shown below. When the addition of 78H (+120) and 69H (+105) is performed, the operation result is E1H (+225), and the two's complement limit is exceeded, with the result that the P/V flag is set (1). Expressed as a two's complement, E1H is –31.

```
     78H (+120)   =      0 1 1 1   1 0 0 0
+)   69H (+105)   =  +)  0 1 1 0   1 0 0 1
    ------------     ----------------------
                     0   1 1 1 0   0 0 0 1  =  –31    P/V = 1
                     ↑
                     CY
```

When the following two negative numbers are added together, the operation result is within the two's complement range, and therefore the P/V flag is reset.

```
     FBH (–5)     =      1 1 1 1   1 0 1 1
+)   F0H (–16)    =  +)  1 1 1 1   0 0 0 0
    ------------     ----------------------
                     1   1 1 1 0   1 0 1 1  =  –21    P/V = 0
                     ↑
                     CY
```

(3) Interrupt request enable flag (IE)

This flag controls CPU interrupt request acknowledgment operations.

When "0", interrupts are disabled, and only non-maskable interrupts and unmasked macro service requests can be acknowledged. All other interrupts are disabled.

When "1", the interrupt enabled state is set, and enabling of interrupt request acknowledgment is controlled by the interrupt mask flags corresponding to the individual interrupt requests and the priority of the individual interrupts.

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

(4) Auxiliary carry flag (AC)

The AC flag is set (1) when there is a carry out of bit 3 or a borrow into bit 3 as the result of an operation, and reset (0) otherwise.

This flag is used when the ADJBA or ADJBS instruction is executed.

(5) Register set selection flag (RSS)

The RSS flag specifies the general registers that function as X, A, C, and B, and the general register pairs (16-bit) that function as AX and BC.

This flag is provided to maintain compatibility with the 78K/III Series, and must be set to 0 except when using a 78K/III Series program.

(6) Zero flag (Z)

The Z flag records that the result of an operation is “0”.

It is set (1) when the result of an operation is “0”, and reset (0) otherwise. The status of the Z flag can be tested with a conditional branch instruction.

(7) Sign flag (S)

The S flag records that the MSB is “1” as the result of an operation.

It is set (1) when the MSB is “1” as the result of an operation, and reset (0) otherwise. The status of the S flag can be tested with a conditional branch instruction.

(8) Register bank selection flag (RBS0 to RBS2)

This is a 3-bit flag used to select one of the 8 register banks (register bank 0 to register bank 7) (see **Table 3-1**).

It holds 3-bit information which indicates the register bank selected by execution of a SEL RBn instruction, etc.

Table 3-1. Register Bank Selection

RBS2	RBS1	RBS0	Specified Register Bank
0	0	0	Register bank 0
0	0	1	Register bank 1
0	1	0	Register bank 2
0	1	1	Register bank 3
1	0	0	Register bank 4
1	0	1	Register bank 5
1	1	0	Register bank 6
1	1	1	Register bank 7

(9) User flag (UF)

This flag can be set and reset in the user program, and used for program control.

3.1.3 Use of RSS bit

Basically, the RSS bit should be fixed at 0 at all times.

The following explanation refers to the case where a 78K/III Series program is used, and the program used sets the RSS bit to 1. This explanation can be skipped if the RSS bit is fixed at 0.

The RSS bit is provided to allow the functions of A (R1), X (R0), B (R3), C (R2), AX (RP0), and BC (RP1) to be used by registers R4 to R7 (RP2, RP3) as well. Effective use of this bit enables efficient programs to be written in terms of program size and program execution.

However, careless use can result in unforeseen problems. Therefore, the RSS bit should always be set to 0. The RSS bit should only be set to 1 when a 78K/III Series program is used.

Use of the RSS bit set to 0 in all programs will improve programming and debugging efficiency.

Even when using a program in which the RSS bit is used set to 1, it is recommended that the program be amended if possible so that it does not set the RSS bit to 1.

(1) RSS bit functions

- Registers used by instructions for which the A, X, B, C, and AX registers are directly entered in the operand column of the instruction operation list (see **6.2**.)
- Registers specified as implied by instructions that use the A, AX, B, and C registers by means of implied addressing
- Registers used in addressing by instructions that use the A, B, and C registers in indexed addressing and based indexed addressing

The registers used in these cases are switched as follows according to the RSS bit.

– **When RSS = 0**

A → R1, X → R0, B → R3, C → R2, AX → RP0, BC → RP1

– **When RSS = 1**

A → R5, X → R4, B → R7, C → R6, AX → RP2, BC → RP3

Registers used other than those mentioned above are always the same irrespective of the value of the RSS bit. With the NEC assembler (RA78K4), the register operation code generated when the A, X, B, C, AX, and BC registers are described by those names is determined by the assembler RSS pseudo-instruction.
When the RSS bit is set or reset, an RSS pseudo-instruction must be written immediately before (or immediately after) the relevant instruction (see example below).

<Program example>

- **When RSS is set to 0**

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

- **When RSS is set to 1**

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

(2) Operation code generation method with RA78K4

- With RA78K4, if there is an instruction with the same function as an instruction for which A or AX is directly entered in the operand column of the instruction operation list, the operation code for which A or AX is directly entered in the operand column is generated first.

Example The function is the same when B is used for r in a MOV A, r instruction and when A is used as r and B is used as r’ in a MOV r, r’ instruction, and the same description (MOVA, B) is used in the assembler source program. In this case, RA78K4 generates code equivalent to the MOV A, r instruction.

Remark The register that is actually used with this instruction is determined when the program is run according to the contents of the RSS bit in the PSW. When RSS = 0, R1 or RP0 is used, and when RSS = 1, R5 or RP2 is used.

- If A, X, B, C, AX, or BC is written in an instruction for which r, r', rp and rp' are specified in the operand column, the A, X, B, C, AX, and BC instructions generate an operation code that specifies the following registers according to the operand of the RA78K4 pseudo-instruction.

Register	RSS 0	RSS 1
A	R1	R5
X	R0	R4
B	R3	R7
C	R2	R6
AX	RP0	RP2
BC	RP1	RP3

- If R0 to R7 or RP0 to RP4 is written as r, r', rp or rp' in the operand column, an operation code in accordance with that specification is output (an operation code for which A or AX is directly entered in the operand column is not output.)

- Descriptions R1, R3, R2 or R5, R7, R6 cannot be used for registers A, B, and C used in indexed addressing and based indexed addressing.

(3) Operating precautions

Switching the RSS bit has the same effect as having two register sets. However, the following point must be noted. If use with RSS = 1 is essential, these defects must be given full consideration when writing the program.

(a) When writing a program, care must be taken to ensure that the static program description and dynamic RSS bit changes at the time of program execution always coincide.
For example, when an MOV A, B instruction is assembled by RA78K4, MOV A, r code is generated. In this case, the registers actually used are as shown below according to the RSS pseudo-instruction written directly before the MOV A, B instruction in the source program and the RSS bit in the PSW when the program is run.

		RSS Pseudo-Instruction Operand	
		0	1
RSS bit in PSW	0	MOV R1, R3	MOV R1, R7
	1	MOV R5, R3	MOV R5, R7

(b) As a program that sets RSS to 1 cannot be used by a program that uses the context switching function, program applicability is poor.

(c) If interrupts are used by a program with more than one section in which the RSS bit in the PSW is set to “1”, it is necessary to set the RSS bit in the PSW to “0” or “1” at the beginning of the interrupt service program, and write an RSS pseudo-instruction corresponding to this in the source program. If this is not done, the execution results may sometimes be incorrect. For example, consider the following interrupt service program.

```
INT:
      PUSH  AX
      MOV   A, #byte
      ADD   !!addr24, A
      POP   AX
      RETI
```

In this program, the register determined at assembly time by the RSS pseudo-instruction written immediately before is used as the AX or A register in the “PUSH AX”, “MOV A, #byte”, and “POP AX” instructions. However, in the “ADD !!addr24, A” instruction, the register used as the A register is determined by the value to which the interrupted program set the RSS bit in the PSW. Therefore, either the expected value or an unexpected value may be stored in the memory specified by !!addr24.
In this example, only the interrupt service program execution result is in error, but if, for example, the ADD instruction operands are reversed (ADD A, !!addr24), the contents of the register used by the interrupt program might be corrupted.
Since the phenomenon occurs in an irregular fashion with this kind of bug, it is extremely difficult to find the cause during debugging.

(d) As different registers are used under the same name, program legibility is poor and debugging is difficult.

3.1.4 Stack pointer (SP)

The stack pointer is a 24-bit register that holds the start address of the stack area (LIFO type: 000000H to FFFFFFH) (see **Figure 3-3**). It is used to address the stack area when subroutine processing or interrupt servicing is performed.

The contents of the SP are decremented before a write to the stack area and incremented after a read from the stack area (see **Figures 3-4** and **3-5**).

The SP is accessed by special instructions.

The SP contents are undefined after $\overline{\text{RESET}}$ input, and therefore the SP must always be initialized by an initialization program directly after reset release (before a subroutine call or interrupt acknowledgment).

In some products a number of bits at the high-order end of the SP are fixed at 0. Please refer to the **User's Manual — Hardware** for the individual products for details.

Example SP initialization

MOVG SP, #0FEE0H;SP ← 0FEE0H (when used from FEDFH)

Figure 3-3. Stack Pointer (SP) Configuration

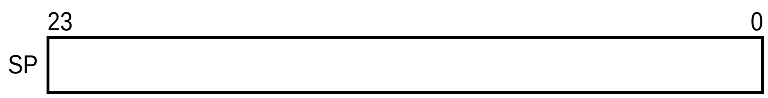


Figure 3-4. Data Saved to Stack Area

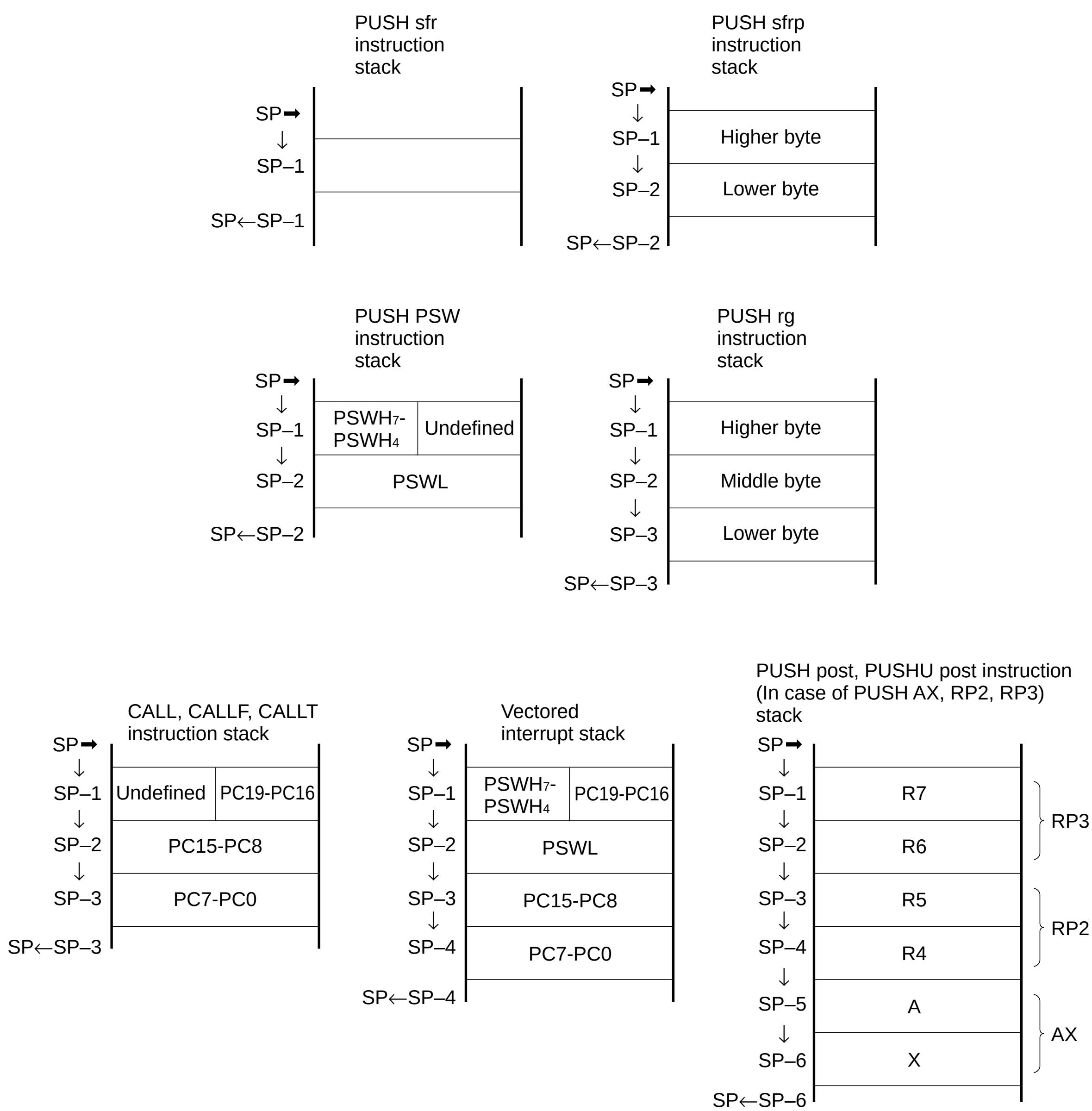
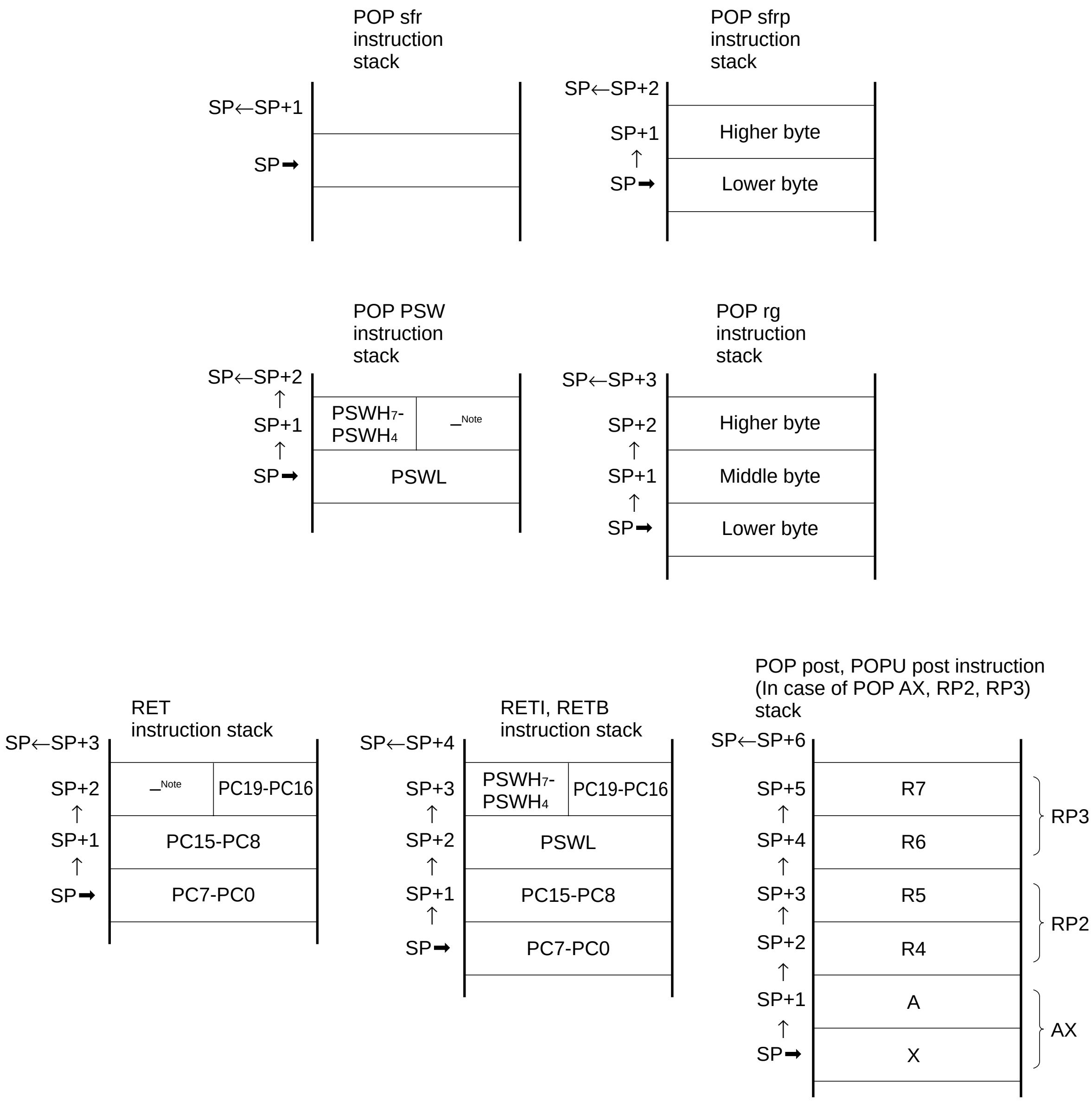


Figure 3-5. Data Restored from Stack Area



Note This 4-bit data is ignored.

Cautions 1. With stack addressing, the entire 16-Mbyte space can be accessed but a stack area cannot be reserved in the SFR area or internal ROM area.

2. The SP is undefined after $\overline{\text{RESET}}$ input. Moreover, non-maskable interrupts can still be acknowledged when the SP is in an undefined state. An unanticipated operation may therefore be performed if a non-maskable interrupt request is generated when the SP is in the undefined state directly after reset release. To avoid this risk, the program after reset release must be written as follows.
The μPD784915 Subseries is fixed to the LOCATION 0 instruction.

```
RSTVCT    CSEG  AT 0
          DW    RSTSTRT
           ~
INITSEG   CSEG  BASE
RSTSTRT:  LOCATION 0H; or LOCATION 0FH
          MOVG SP, #STKBGN
```

3.2 General Registers

3.2.1 Configuration

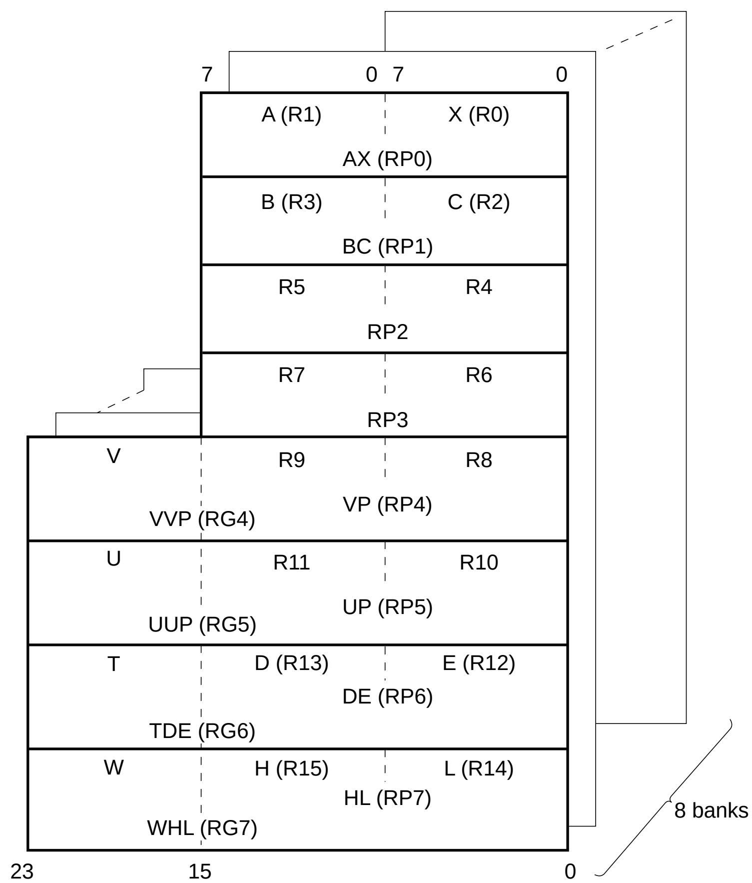
There are sixteen 8-bit general registers. Also, two general registers can be used together as a 16-bit general register. In addition, four of the 16-bit general registers can be combined with an 8-bit register for address extension and used as 24-bit address specification registers.

General registers other than the V, U, T, and W registers for address extension are mapped onto internal RAM.

These register sets are provided in 8 banks, and can be switched by means of software or the context switching function.

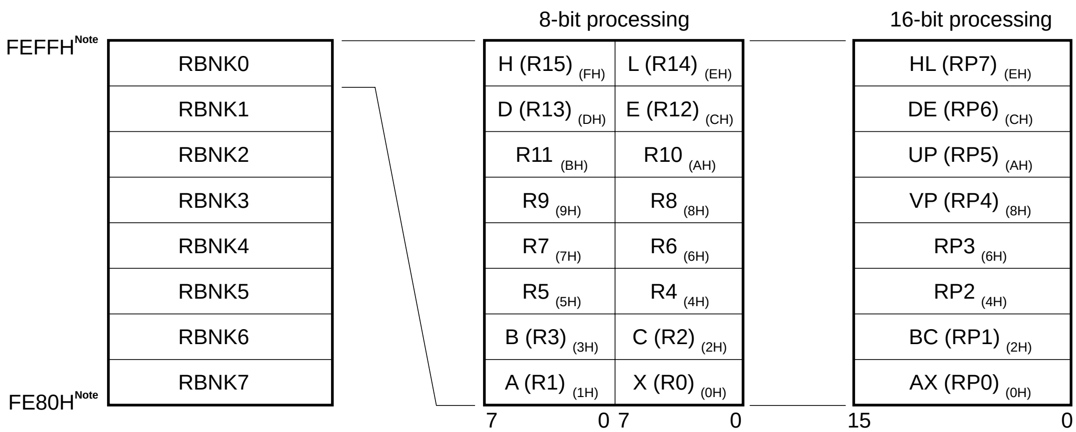
Upon $\overline{\text{RESET}}$ input, register bank 0 is selected. The register bank used during program execution can be checked by reading the register bank selection flag (RBS0, RBS1, RBS2) in the PSW.

Figure 3-6. General Register Configuration



Remark Absolute names are shown in parentheses.

Figure 3-7. General Register Addresses



Note When the LOCATION 0 instruction is executed. When the LOCATION 0FH instruction is executed, 0F0000H should be added to the address values shown.
The μPD784915 Subseries is fixed to the LOCATION 0 instruction.

Caution R4, R5, R6, R7, RP2, and RP3 can be used as the X, A, C, B, AX, and BC registers respectively by setting the RSS bit of the PSW to 1, but this function should only be used when using a 78K/III Series program.

Remark When the register bank is switched, and it is necessary to return to the original register bank, an SEL RBn instruction should be executed after first saving the PSW to the stack with a PUSH PSW instruction. When returning to the original register bank, if the stack location does not change the POP PSW instruction should be used.
When the register bank is changed by a vectored interrupt service program, etc., the PSW is automatically saved to the stack when an interrupt is acknowledged and restored by an RETI or RETB instruction, so that, if only one register bank is used in the interrupt service program, only an SEL RBn instruction need be executed, and execution of a PUSH PSW and POP W instruction is not necessary.

Example 1. When register bank 2 is specified

```
⋮
PUSH PSW
SEL RB2  }
⋮        } Operations in register bank 2
POP PSW  }
⋮          Operations in original register bank
```

2. When the register bank is specified by a vectored interrupt service program.

```
INT:
SEL RB5
⋮        } Operation in register bank 5
RETI       Automatic return to original register bank
```

3.2.2 Functions

In addition to being manipulated as 8-bit units, the general registers can also be manipulated as 16-bit units by pairing two 8-bit registers. Also, four of the 16-bit general registers can be combined with an 8-bit register for address extension and manipulated as 24-bit units.

Each register can be used in a general way for temporary storage of an operation result and as the operand of an inter-register operation instruction.

The area from 0FE80H to 0FEFFH (when the LOCATION 0 is executed; 0FFE80H to 0FFEFFH when the LOCATION 0FH instruction is executed) can be given an address specification and accessed as ordinary data memory irrespective of whether or not it is used as the general register area.

As 8 register banks are provided in the 78K/IV Series, efficient programs can be written by using different register banks for normal processing and processing in the event of an interrupt.

The registers have the following specific functions.

A (R1):

- Register mainly used for 8-bit data transfers and operation processing. Can be used in combination with all addressing modes for 8-bit data.
- Can also be used for bit data storage.
- Can be used as the register that holds the offset value in indexed addressing and based indexed addressing.

X (R0):

- Can be used for bit data storage.

AX (RP0):

- Register mainly used for 16-bit data transfers and operation processing. Can be used in combination with all addressing modes for 16-bit data.

AXDE:

- Used for 32-bit data storage when a DIVUX, MACW, or MACSW instruction is executed.

B (R3):

- Has a loop counter function, and can be used by the DBNZ instruction.
- Can be used as the register that holds the offset value in indexed addressing and based indexed addressing.
- Used as the MACW and MACSW instruction data pointer.

C (R2):

- Has a loop counter function, and can be used by the DBNZ instruction.
- Can be used as the register that holds the offset value in based indexed addressing.
- Used as the counter in a string instruction and the SACW instruction.
- Used as the MACW and MACSW instruction data pointer.

RP2:

- Used to save the low-order 16 bits of the program counter (PC) when context switching is used.

RP3:

- Used to save the high-order 4 bits of the program counter (PC) and the program status word (PSW) (excluding bits 0 to 3 of PSWH) when context switching is used.

VVP (RG4):

- Has a pointer function, and operates as the register that specifies the base address in register indirect addressing, based addressing and based indexed addressing.

UUP (RG5):

- Has a user stack pointer function, and enables a stack separate from the system stack to be implemented by means of the PUSHU and POPU instructions.
- Has a pointer function, and operates as the register that specifies the base address in register indirect addressing and based addressing.

DE (RP6), HL (RP7):

- Operate as the registers that specify the offset value in indexed addressing and based indexed addressing.

TDE (RG6):

- Has a pointer function, and operates as the register that specifies the base address in register indirect addressing and based addressing.
- Used as the pointer in a string instruction and the SACW instruction.

WHL (RG7):

- Register used mainly for 24-bit data transfers and operation processing.
- Has a pointer function, and operates as the register that specifies the base address in register indirect addressing and based addressing.
- Used as the pointer in a string instruction and the SACW instruction.

In addition to the function name that emphasizes the specific function of the register (X, A, C, B, E, D, L, H, AX, BC, VP, UP, DE, HL, VVP, UUP, TDE, WHL), each register can also be described by its absolute name (R0 to R15, RP0 to RP7, RG4 to RG7). The correspondence between these names is shown in Table 3-2.

Table 3-2. Function Names and Absolute Names

(a) 8-bit register

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

(b) 16-bit register

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

(c) 24-bit register

Absolute Name	Function Name
RG4	VVP
RG5	UUP
RG6	TDE
RG7	WHL

Note RSS should only be set to 1 when a 78K/III Series program is used.

Remark R8 to R11 have no function name.

3.3 Special Function Registers (SFR)

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

Note When the LOCATION 0 instruction is executed. When the LOCATION 0FH instruction is executed, the area is 0FFF00H to 0FFFFFH.
The μPD784915 Subseries is fixed to the LOCATION 0 instruction.

Caution Addresses onto which SFRs are not mapped should not be accessed in this area. If such an address is accessed by mistake, the CPU may become deadlocked. A deadlock can only be released by reset input.

CHAPTER 4 INTERRUPT FUNCTIONS

The three kinds of processing shown in Table 4-1 can be programmed as servicing for interrupt requests. Multiprocessing control using a 4-level priority system can easily be performed for maskable vectored interrupts.

Table 4-1. Interrupt Request Servicing

Service Mode	Service Performed by	Service	PC/PSW Contents
Vectored interrupts	Software	Executed by branching to service routine (any service contents)	Associated saving to & restoration from stack
Context switching		Executed by automatic switching of register bank and branching to service routine (any service contents)	Associated saving to & restoration from fixed area in register bank
Macro service	Firmware	Execution of memory-I/O data transfers, etc. (fixed service contents)	No change

Remark Please refer to the **User's Manual — Hardware** for the individual products for details.

4.1 Kinds of Interrupt Request

There are three kinds of interrupt request, as follows:

- Software interrupt requests
- Non-maskable interrupt requests
- Maskable interrupt requests

4.1.1 Software interrupt requests

An interrupt request by software is generated when a BRK instruction or BRKCS RBn instruction is executed, or if here is an error in an operand of an MOV WDM, #byte instruction or MOV STBC, #byte instruction, LOCATION instruction (operand error interrupt). Interrupt requests by software are acknowledged even in the interrupt disabled (DI) state, and are not subject to interrupt priority control. Therefore, when an interrupt request is generated by software, a branch is made to the interrupt service routine unconditionally.

To return from a BRK instruction, an RETB instruction is executed.

To return from a BRKCS RBn instruction service routine, a RETCSB !addr16 instruction is executed.

As an operand error interrupt is an interrupt generated if there is an error in an operand, processing is required for branching to the initialization program by a reset release after the necessary processing has been performed, etc.

4.1.2 Non-maskable interrupt requests

A non-maskable interrupt request is generated when a valid edge is input to the NMI pin or when the watchdog timer overflows. The provision of the NMI pin and watchdog timer functions varies from product to product. Please refer to the **User's Manual — Hardware** for the individual products for details.

Non-maskable interrupt requests are acknowledged unconditionally, even in the interrupt disabled (DI) state. Also, they are not subject to interrupt priority control, and are of higher priority that any other interrupt.

4.1.3 Maskable interrupt requests

A maskable interrupt request is one subject to masking control according to the setting of the interrupt control register. In addition, acknowledgment enabling/disabling can be set for all maskable interrupts by means of the IE flag in the PSW.

The priority order for maskable interrupt requests when interrupt requests of the same priority are generated simultaneously is predetermined (default priority). Also, multiprocessing can be performed with interrupt priorities divided into 4 levels in accordance with the specification of the interrupt control register. However, macro service requests are acknowledged without regard to priority control or the IE flag.

4.2 Interrupt Service Modes

4.2.1 Vectored interrupts

A branch is made to the service routine using the memory contents of the vector table address corresponding to the interrupt source as the branch destination address.

The following operations are executed to enable the CPU to perform interrupt servicing.

- When branching : The CPU state (PC & PSW contents) is saved to the stack.
- When returning : CPU statuses (PC & PSW contents) are restored from the stack.

The return from the service routine to the main routine is performed by an RETI instruction (or an RETB instruction in the case of a BRK instruction or operand error interrupt).

The branch destination address is restricted to the base area from 0000H to FFFFH.

Please refer to the **User's Manual — Hardware** for the individual products for details of the vector table.

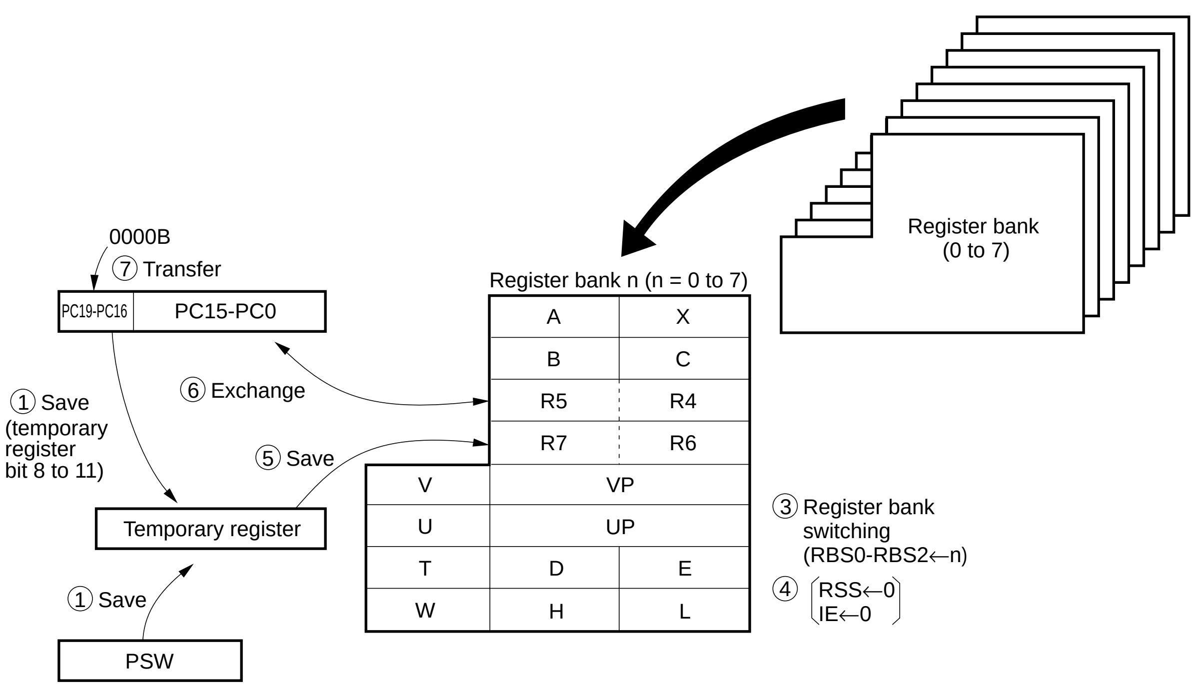
4.2.2 Context switching

The prescribed register bank is selected by hardware by generation of an interrupt request or execution of a BRKCS RBn instruction. With this function, a branch is made to the vector address stored beforehand in the register bank, and at the same time the contents of the program counter (PC) and program status word (PSW) are stacked in the register bank.

The return from the service routine is performed by a RETCS !addr16 instruction (or an RETCSB !addr16 instruction in the case of a BRKCS RBn instruction).

The branch destination address is restricted to the base area from 0000H to FFFFH.

Figure 4-1. Context Switching Operation by Interrupt Request Generation



4.2.3 Macro service function

In macro service, CPU execution is temporarily suspended when an interrupt is acknowledged, and the service set by firmware is executed. Since macro service is performed without the intermediation of the CPU, it is not necessary to save CPU statuses such as the PC and PSW contents. This is therefore very effective in improving the CPU service time.

Please refer to the **User's Manual — Hardware** for the individual products for details of macro service.

CHAPTER 5 ADDRESSING

5.1 Instruction Address Addressing

The instruction address is determined by the contents of the program counter (PC), and is normally incremented (by 1 for one byte) automatically in accordance with the number of bytes in the fetched instruction each time an instruction is executed. However, when an instruction associated with a branch is executed, branch address information is set in the PC and a branch performed by means of the addressing modes shown below.

The following kinds of instruction address addressing are provided:

- (8-bit/16-bit) relative addressing
- (11-bit/16-bit/20-bit) immediate addressing
- Table indirect addressing
- 16-bit register addressing
- 20-bit register addressing
- 16-bit register indirect addressing
- 20-bit register indirect addressing

Details of each kind of addressing are given in the following sections.

5.1.1 Relative addressing

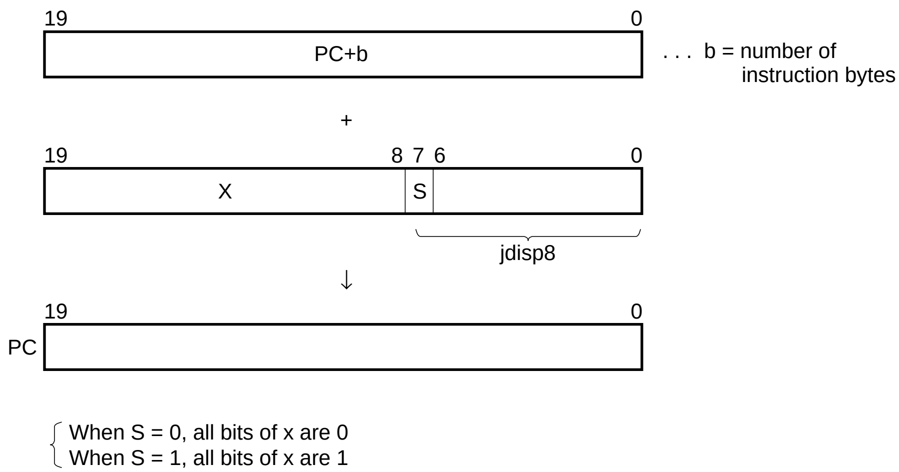
[Function]

The value obtained by adding the 8-bit or 16-bit immediate data in the operation code (displacement value: jdisp8, jdisp16) to the start address of the next instruction is transferred to the program counter (PC), and a branch is made. The displacement value is treated as signed two's complement data (–128 to +127, –32,768 to +32,767), with the MSB as the sign bit.

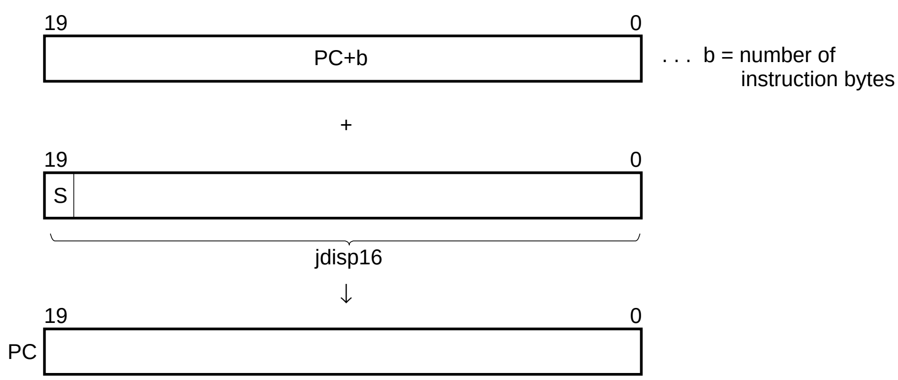
This is performed when a CALL $!addr20, BR $!addr20, BR $addr20, or conditional branch instruction is executed (only 8-bit immediate data can be used in a conditional branch instruction).

[Explanatory Diagrams]

8-bit relative addressing



16-bit relative addressing



5.1.2 Immediate addressing

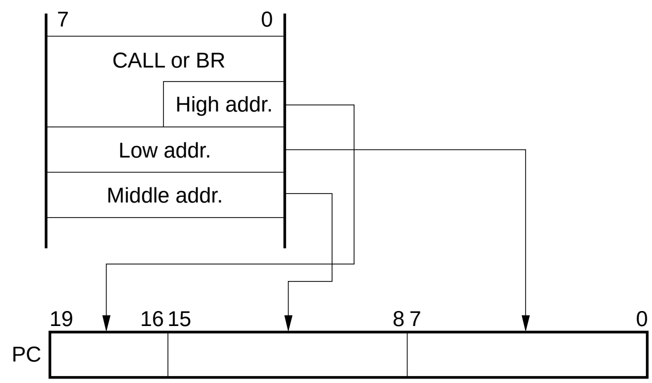
[Function]

The immediate data in the instruction word is transferred to the program counter (PC), and a branch is made. This is performed when a CALL !!addr20, BR !!addr20, CALL !addr16, BR !addr16, or CALLF !addr11 instruction is executed.

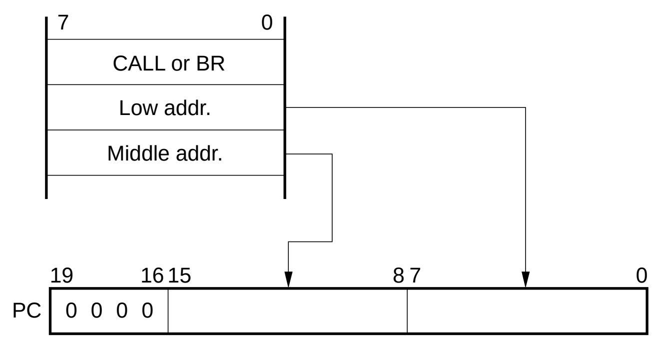
In the case of a CALL !addr16 or BR !addr16 instruction (16-bit immediate addressing), the high-order 4-bit address is fixed at 0, and a branch is made to the base area. In the case of the CALLF !addr11 instruction, the high-order 9-bit address is fixed at 000000001.

[Explanatory Diagrams]

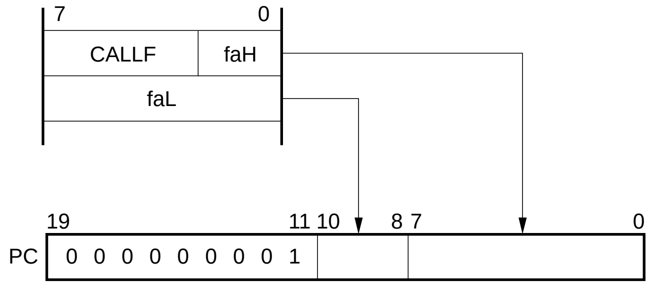
20-bit immediate addressing



16-bit immediate addressing



11-bit immediate addressing



[Caution]

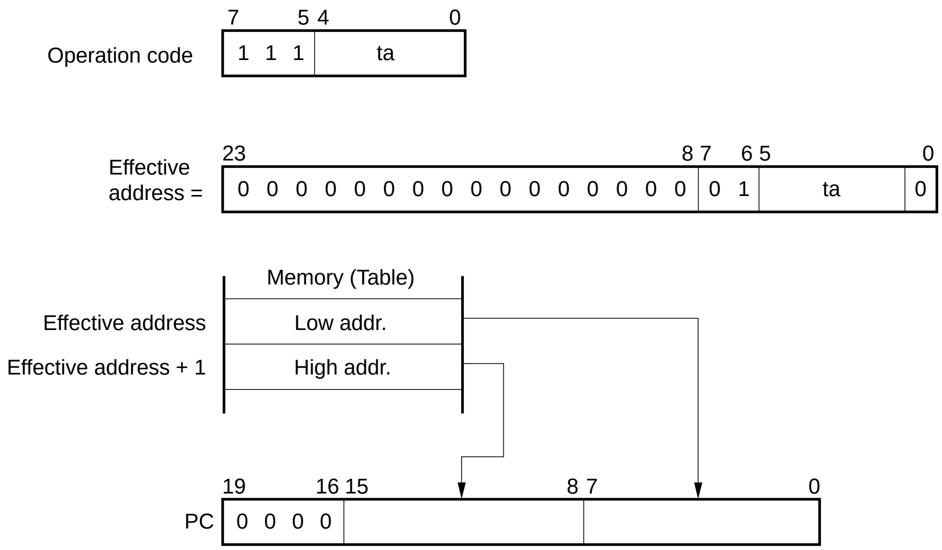
As the branch destination of the BR !addr16 instruction is restricted, it should only be used when using a 78K/0, 78K/I, 78K/II, or 78K/III Series program.

5.1.3 Table indirect addressing

[Function]

The specific location table contents (branch destination address) addressed by the immediate data in the low-order 5 bits of the operation code are transferred to the low-order 16 bits of the program counter (PC), 0000 is transferred to the high-order 4 bits, and a branch is made (the branch destination address is restricted to the base area). This is performed when a CALLT [addr5] instruction is executed.

[Explanatory Diagram]



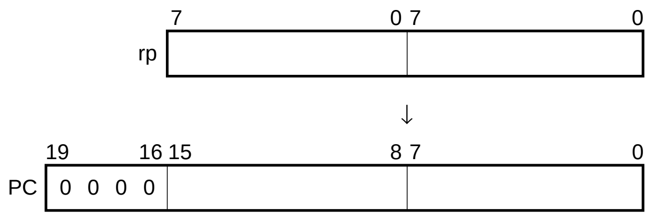
5.1.4 16-bit register addressing

[Function]

The contents of register rp (RP0 to RP7) specified by the instruction word are transferred to the low-order 16 bits of the program counter (PC), 0000 is transferred to the high-order 4 bits, and a branch is made (the branch destination address is restricted to the base area).

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

[Explanatory Diagrams]



[Caution]

As the branch destination of the BR rp instruction is restricted, it should only be used when using a 78K/0, 78K/I, 78K/II, or 78K/III Series program.

If AX or BC is written for rp, with the NEC RA78K4 assembler the object code generated depends on the RSS pseudo-instruction written immediately before. “1” should be specified by the RSS pseudo-instruction only when a 78K/III Series program is used (see **3.1.3 Use of RSS bit**).

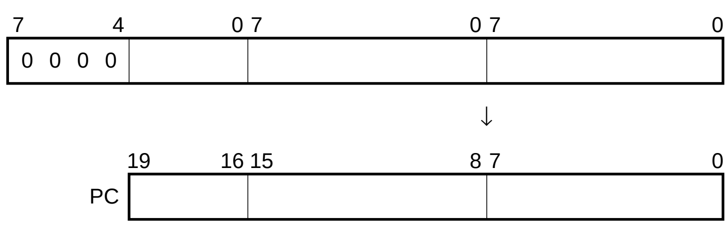
5.1.5 20-bit register addressing

[Function]

The contents of register rg (RG4 to RG7) specified by the instruction word are transferred to the program counter (PC), and a branch is made. The high-order 4 bits of rg should be set to 0000.

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

[Explanatory Diagram]

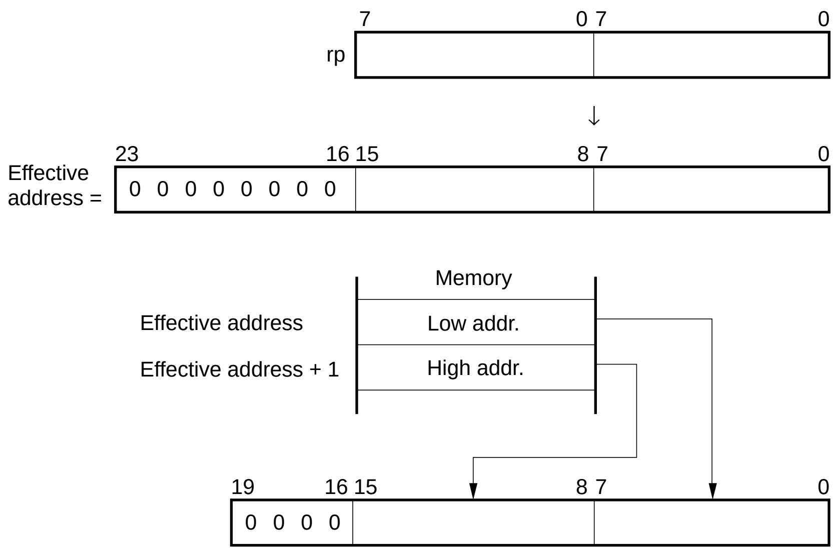


5.1.6 16-bit register indirect addressing

[Function]

The 2 consecutive bytes of data in the memory addressed by the contents of register rp (RP0 to RP7) specified by the instruction word are transferred to the low-order 16 bits of the program counter (PC), 0000 is transferred to the high-order 4 bits, and a branch is made (the branch destination address is restricted to the base area). This is performed when a BR [rp] or CALL [rp] instruction is executed.

[Explanatory Diagram]



[Caution]

As the address that holds the branch destination address and the branch destination of the BR [rp] instruction are restricted, it should only be used when using a 78K/III Series program.
If AX or BC is written for rp, with the NEC RA78K4 assembler the object code generated depends on the RSS pseudo-instruction written immediately before. “1” should be specified by the RSS pseudo-instruction only when a 78K/III Series program is used (see **3.1.3 Use of RSS bit**).

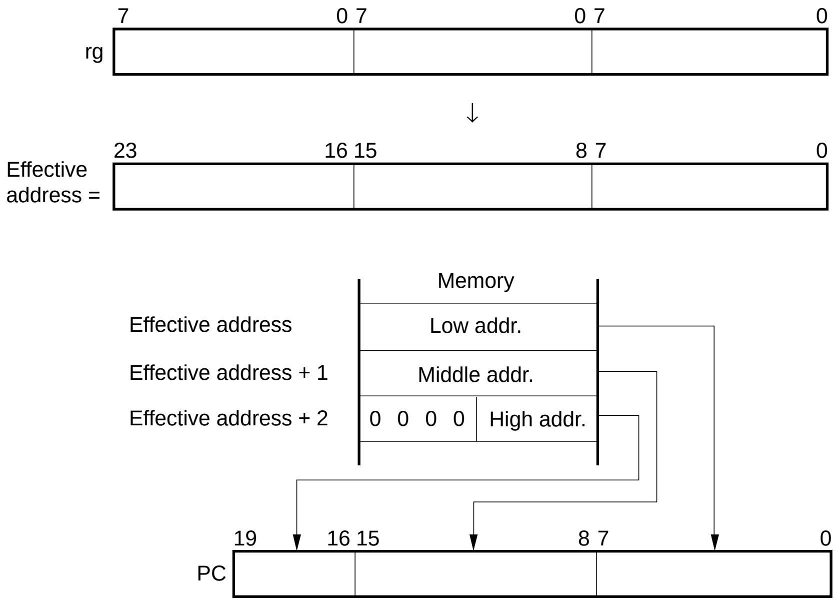
5.1.7 20-bit register indirect addressing

[Function]

The 3 consecutive bytes of data in the memory addressed by the contents of register rg (RP0 to RP7) specified by the instruction word are transferred to the program counter (PC), and a branch is made. The high-order 4 bits of the 3-byte data stored in the memory should be set to 0000.

This is performed when a BR [rg] or CALL [rg] instruction is executed.

[Explanatory Diagram]



5.2 Operand Address Addressing

The following methods are available for specifying the register, memory, etc., to be manipulated when an instruction is executed.

- Implied addressing
- Register addressing
- Immediate addressing
- 8-bit direct addressing
- 16-bit direct addressing
- 24-bit direct addressing
- Short direct addressing
- Special function register (SFR) addressing
- Short direct 16-bit memory indirect addressing
- Short direct 24-bit memory indirect addressing
- Stack addressing
- 24-bit register indirect addressing (including 24-bit register indirect addressing with auto-increment/auto-decrement)
- 16-bit register indirect addressing
- Based addressing
- Indexed addressing
- Based indexed addressing

Details of each kind of addressing are given in the following sections.

5.2.1 Implied addressing

[Function]

This type of addressing automatically addresses registers in the register bank specified by the register bank selection flags (RBS2, RBS1, and RBS0).

Instructions that use implied addressing in the 78K/IV Series instruction word are shown below.

The A, AX, C, and B registers used by these instructions are affected by the RSS bit in the PSW. When RSS = 0, R1, RP0, R2, and R2, respectively are accessed for the A, AX, C, and B registers, and when RSS = 1, R5, RP2, R6, and R7 are accessed. RSS should only be set to 1 when a 78K/III Series program is used (see **3.1.3 Use of RSS bit**).

Instruction	Registers Specified by Implied Addressing
MULU	A register as multiplicand, AX register as that holds product
MULUW, MULW	AX register as multiplicand and register that holds high-order 16 bits of product
DIVUW	AX register as register that holds dividend and quotient
DIVUX	AXDE register as register that holds dividend and quotient
MACW, MACSW	AXDE register as register that holds result of sum of products operation, B and C registers as pointer registers that specify data
ADJBA, ADJBS	A register as register that holds numeric value subject to decimal adjustment
CVTBW	A register as register that holds data before sign extension is performed, and AX register as register that holds result of sign extension
CHKLA	A register as register that holds result of comparison between pin level and port output latch
ROR4, ROL4	A register as register that holds digit data subject to digit rotation (only low-order 4 bits are used)
SACW, string instruction	C register as data counter string instruction

[Operand Format]

As this is used automatically according to the instruction, there is no specific operand format.

[Description Example]

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

5.2.2 Register addressing

[Function]

This type of addressing accesses as an operand the general register specified by the register specification code in the instruction word in the register bank specified by the register bank selection flag (RBS2, RBS1, RBS0). Register addressing is performed when an instruction with one of the operand formats shown below is executed.

[Operand Format]

Performed when an instruction with one of the operand formats shown below is executed.

Identifier	Description Format
A	A
C	C
X	X
B	B
r	X(R0), A(R1), C(R2), B(R3), R4, R5, R6, R7, R8, R9, R10, R11, E(R12), D(R13), L(R14), H(R15)
r1	X(R0), A(R1), C(R2), B(R3), R4, R5, R6, R7
r2	R8, R9, R10, R11, E(R12), D(R13), L(R14), H(R15)
r3	V, U, T, W
AX	AX
rp	AX(RP0), BC(RP1), RP2, RP3, VP(RP4), UP(RP5), DE(RP6), HL(RP7)
rp1	AX(RP0), BC(RP1), RP2, RP3
rp2	VP(RP4), UP(RP5), DE(RP6), HL(RP7)
WHL	WHL
rg	VVP(RG4), UUP(RG5), TDE(RG6), WHL(RP7)

Remarks 1. Absolute names are shown in parentheses.

2. With an instruction (such as ADDW AX, #word) in which A, X, AX, B, or C is specified directly as the register addressing operand, the register used as A, X, AX, B, or C is determined by the RSS bit in the PSW when the instruction is executed. The RSS bit in the PSW should be set to “1” only when a 78K/III Series program is used (see **3.1.3 Use of RSS bit**).

3. If A, X, B, C, AX, or BC is written as an operand in an instruction in which r, r1, rp, or rp1 is specified as the register addressing operand, with the NEC RA78K4 assembler the object code generated depends on the RSS pseudo-instruction written immediately before. “1” should be specified in the RSS pseudo-instruction operand only when a 78K/III Series program is used (see **3.1.3 Use of RSS bit**).

[Description Example 1]

- General example
  MOV A, r
- Specific example
  MOV A, C ; When the C register is selected as r

[Description Example 2]

- General example
  INCW rp
- Specific example
  INCW DE ; When the DE register pair is selected as rp

5.2.3 Immediate addressing

[Function]

This type of addressing has 8-bit data, 16-bit data and 24-bit data subject to manipulation in the operation code.

[Operand Format]

Performed when an instruction with one of the operand formats shown below is executed.

Identifier	Description Format
byte	Label or 8-bit immediate data
word	Label or 16-bit immediate data
imm24	Label or 24-bit immediate data

[Description Example]

- General example
  ADD A, #byte
- Specific example
  ADD A, #77H ; When 77H is used as byte

5.2.4 8-bit direct addressing

[Function]

With this kind of addressing, the immediate data in the instruction word is the operand address and the memory to be manipulated is addressed. It is used with the MOVTBLW instruction. Memory from 0FE00H to 0FEFFH is addressed when a LOCATION 0 instruction is executed, and memory from 0FFE00H to 0FFEFFH when a LOCATION 0FH instruction is executed.

[Operand Format]

Performed when an instruction with the operands shown below is executed.

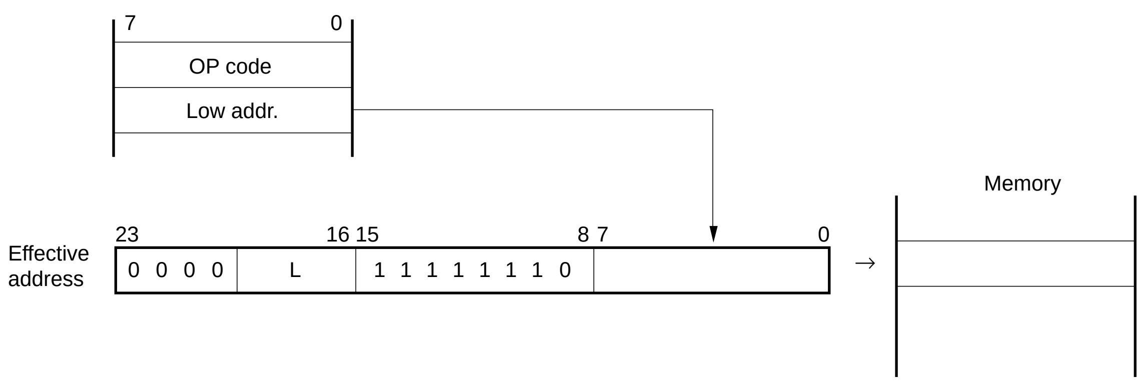
Identifier	Description Format
!addr8	Label, or immediate data 0FE00H to 0FEFFH **Note**

Note When the LOCATION 0 instruction is executed. When the LOCATION 0FH instruction is executed, the range is 0FFE00H to 0FFEFFH.
The μPD784915 Subseries is fixed to the LOCATION 0 instruction.

[Description Examples]

- General example
  MOVTBLW !addr8, n
- Specific example
  MOVTBLW !0FE24H, n; When FE24H is used as addr8

[Explanatory Diagram]



Remark L depends on the LOCATION instruction.
- When LOCATION 0 instruction is executed : 0000
- When LOCATION 0FH instruction is executed : 1111

5.2.5 16-bit direct addressing

[Function]

This type of addressing addresses memory subject to manipulation with the immediate data in the instruction word as the operand address. The base area can be addressed.

[Operand Format]

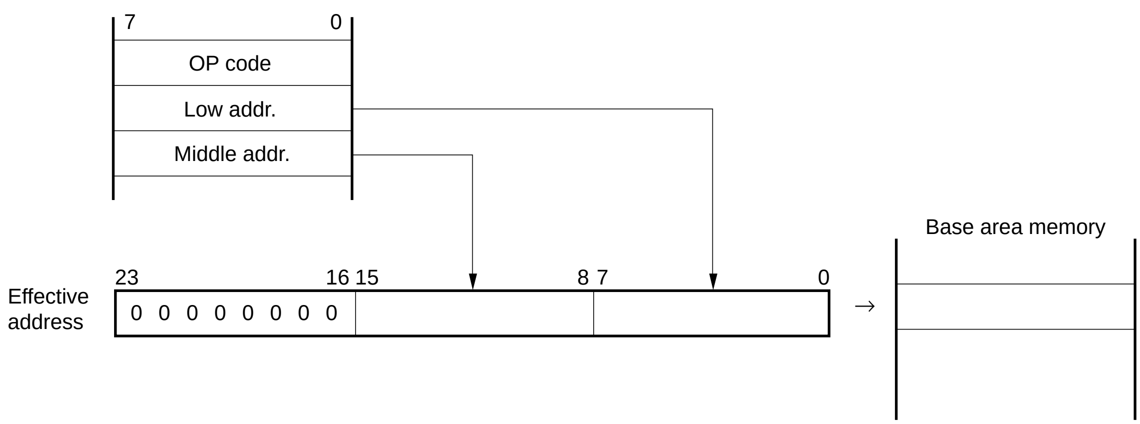
Performed when an instruction with the operand format shown below is executed.

Identifier	Description Format
addr16	Label or 16-bit immediate data

[Description Example]

- General example
  MOV A, !addr16
- Specific example
  MOV A, !0FE00H ; When FE00H is used as addr16

[Explanatory Diagram]



[Remarks]

This kind of addressing should only be used when it is absolutely essential to reduce the execution time or object size, or when 78K/0, 78K/I, 78K/II, or 78K/III Series software is used and program amendment is difficult. Amendments may be necessary in order to make further use of a program that uses this kind of addressing.

5.2.6 24-bit direct addressing

[Function]

This type of addressing addresses memory subject to manipulation with the immediate data in the instruction word as the operand address. The entire memory space can be addressed.

[Operand Format]

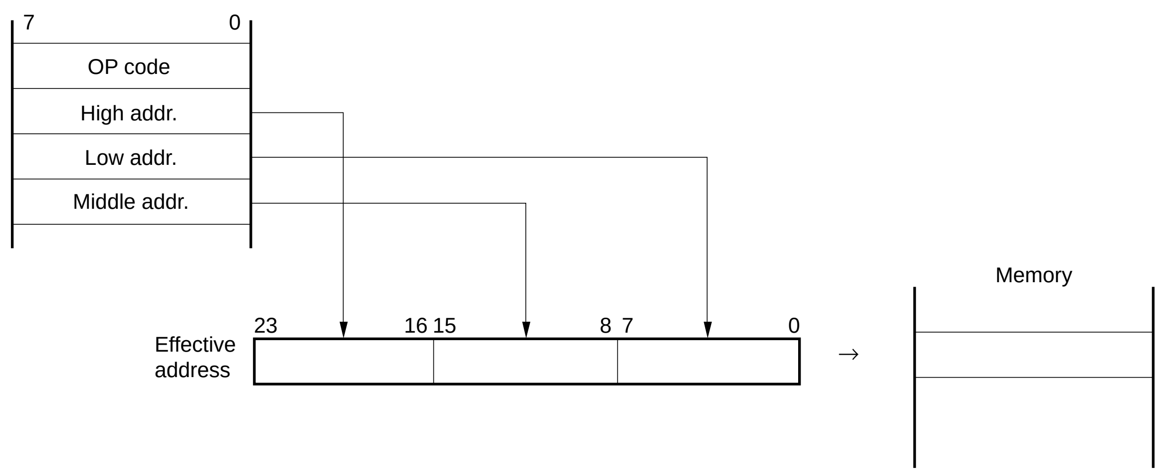
Performed when an instruction with the operand format shown below is executed.

Identifier	Description Format
addr24	Label or 24-bit immediate data

[Description Example]

- General example
  MOV A, !!addr24
- Specific example
  MOV A, !!54FE00H ; When 54FE00H is used as addr24

[Explanatory Diagram]



5.2.7 Short direct addressing

[Function]

This type of addressing directly addresses memory subject to manipulation in a fixed space with the 8-bit immediate data in the instruction word. 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 small number of clocks.

With short direct addressing, the applicable address range varies according to the LOCATION instruction in the same way as the internal data area location addresses. When a LOCATION 0 instruction is executed, internal RAM from 0FD20H to 0FEFFH and special function registers (SFRs) from 0FF00H to 0FF1FH can be accessed. When a LOCATION 0FH instruction is executed, internal RAM from 0FFD20H to 0FFEFFH and SFRs from 0FFF00H to 0FFF1FH can be accessed.

Ports frequently accessed in the program, timer/counter unit compare registers and capture registers are mapped onto the SFR area on which short direct addressing is used. These special function registers can be manipulated using a small number of bytes and small number of clocks.

[Operand Format]

Performed when an instruction with one of the operand formats shown below is executed.

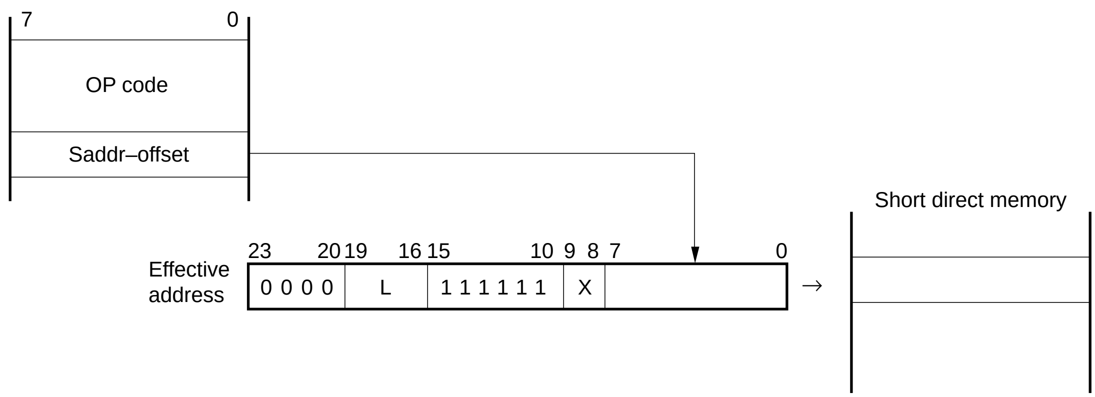
Identifier	Description Format
saddr	Label or immediate data 0FD20H to 0FF1FH
saddr1	Label or immediate data 0FE00H to 0FEFFH
saddr2	Label or immediate data 0FD20H to 0FDFFH and 0FF00H to 0FF1FH
saddrp	Label or immediate data 0FD20H to 0FF1EH
saddrp1	Label or immediate data 0FE00H to 0FEFEH
saddrp2	Label or immediate data 0FD20H to 0FDFFH and 0FF00H to 0FF1EH (If 0FDFFH is specified, the high-order byte is 0FE00H)
saddrg saddrg1 saddrg2	Label or immediate data 0FD20H to 0FEFDH Label or immediate data 0FE00H to 0FEFDH (during 24-bit manipulation) Label or immediate data 0FD20H to 0FDFFH (during 24-bit manipulation)

Remark The addresses in this table are those that apply when the LOCATION 0 instruction is executed. When the LOCATION 0FH instruction is executed, F0000H should be added to the values shown. The μPD784915 Subseries is fixed to the LOCATION 0 instruction.

[Description Example]

- General example
  MOV saddr, saddr
- Specific example
  MOV 0FE30H, 0FE50H

[Explanatory Diagram]



Remark L depends on the LOCATION instruction.

- When LOCATION 0 instruction is executed : 0000
- When LOCATION 0FH instruction is executed : 1111

X is determined by the op code information and the value of Saddr-offset.

- When saddr1 is specified by op code: 10
- When saddr2 is specified by op code and Saddr-offset is 20H to FFH: 01
- When saddr2 is specified by op code and Saddr-offset is 00H to 1FH: 11

5.2.8 Special function register (SFR) addressing function

[Function]

This type of addressing addresses memory-mapped special function registers (SFRs) with the 8-bit immediate data in the instruction word.

The space used by this kind of addressing varies according to the LOCATION instruction in the same way as the internal data area location addresses. When a LOCATION 0 instruction is executed, it is the 256-byte space from 0FF00H to 0FFFFH, and when a LOCATION 0FH instruction is executed, it is the 256-byte space from 0FFF00H to 0FFFFFH. However, SFRs mapped onto 0FF00H to 0FF1FH (when the LOCATION 0 instruction is executed; 0FFF00H to 0FFF1FH accessed by short direct addressing.

Remarks 1. With the NEC assembler package (RA78K4), short direct addressing is automatically (forcibly) used for instructions on SFRs in addresses that can be accessed by short direct addressing.

2. The μPD784915 Subseries is fixed to the LOCATION 0 instruction.

[Operand Format]

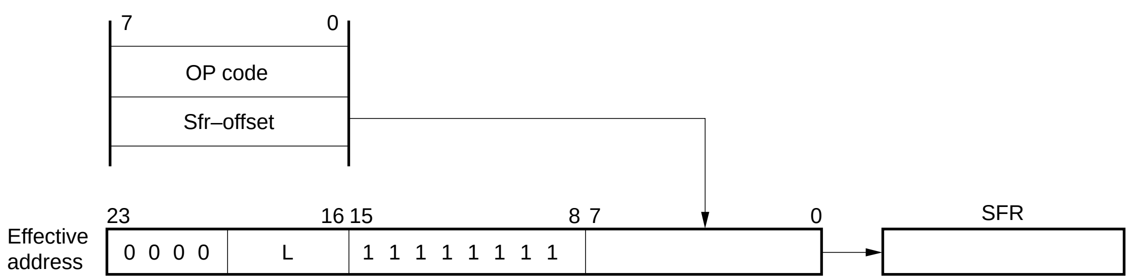
Performed when an instruction with one of the operand formats shown below is executed.

Identifier	Description Format
sfr	Special function register name
sfrp	Name of special function register for which 16-bit operation is possible

[Description Example]

- General example
  MOV sfr, A
- Specific example
  MOV PM0, A ; When PM0 is specified as sfr

[Explanatory Diagram]



Remark L depends on the LOCATION instruction.

- When LOCATION 0 instruction is executed : 0000
- When LOCATION 0FH instruction is executed : 1111

5.2.9 Short direct 16-bit memory indirect addressing

[Function]

This type of addressing addresses base area memory subject to manipulation with the contents of the two consecutive bytes of short direct memory addressed by the 8-bit bits of the operand address and the high-order 8 bits of the operand address set to 00000000.

This addressing is used when an instruction with [saddrp] in an operand is executed.

[Operand Format]

Performed when an instruction with the operand format shown below is executed.

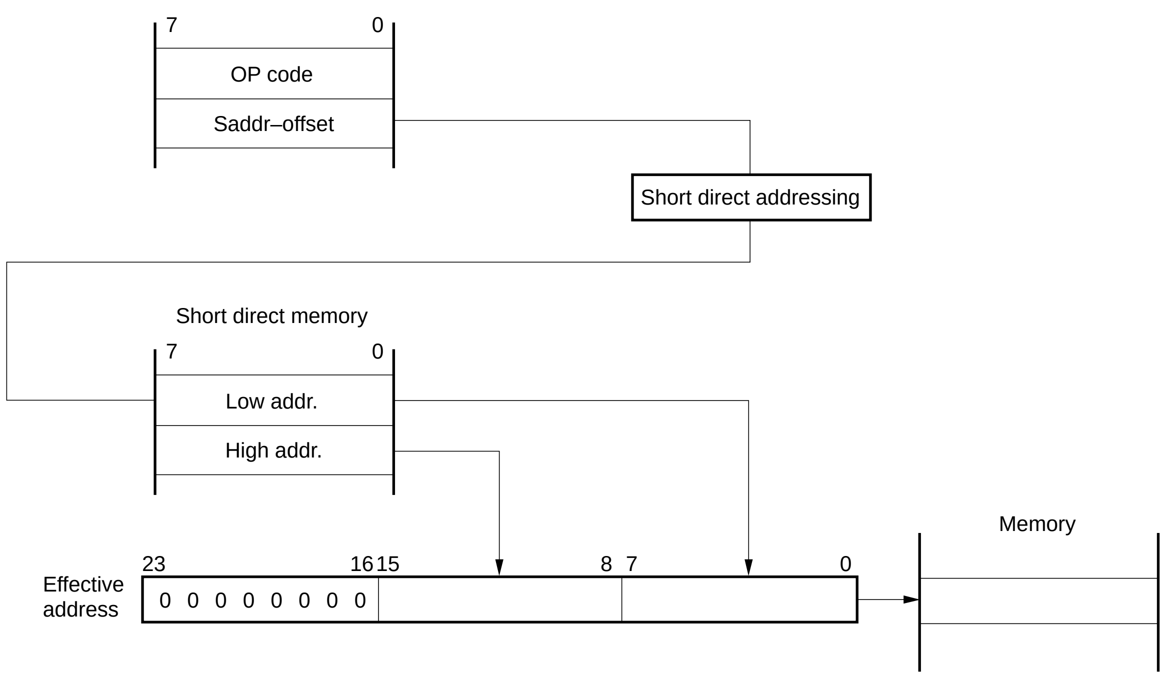
Identifier	Description Format
[saddrp]	[Label, immediate data FD20H to FEFEH Note]

Note When the LOCATION 0 instruction is executed. When the LOCATION 0FH instruction is executed, the range is FFD20H to FFEFEH.
The μPD784915 Subseries is fixed to the LOCATION 0 instruction.

[Description Example]

- General example
  XCH A, [saddrp]
- Specific example
  XCH A, [0FEA0H] ; When memory indicated by 2-byte data in addresses 0FEA0H and 0FEA1H is specified

[Explanatory Diagram]



[Remarks]

This kind of addressing should only be used when it is absolutely essential to reduce the execution time or object size, or when 78K/0, 78K/I, 78K/II, or 78K/III Series software is used and program amendment is difficult.

Amendments may be necessary in order to make further use of a program that uses this kind of addressing.

5.2.10 Short direct 24-bit memory indirect addressing

[Function]

This type of addressing addresses memory subject to manipulation with the contents of the 3 consecutive bytes of short direct memory addressed by the 8-bit immediate data in the instruction word as the operand address. This addressing is used when an instruction with [%saddrg] in an operand is executed.

[Operand Format]

Performed when an instruction with the operand format shown below is executed.

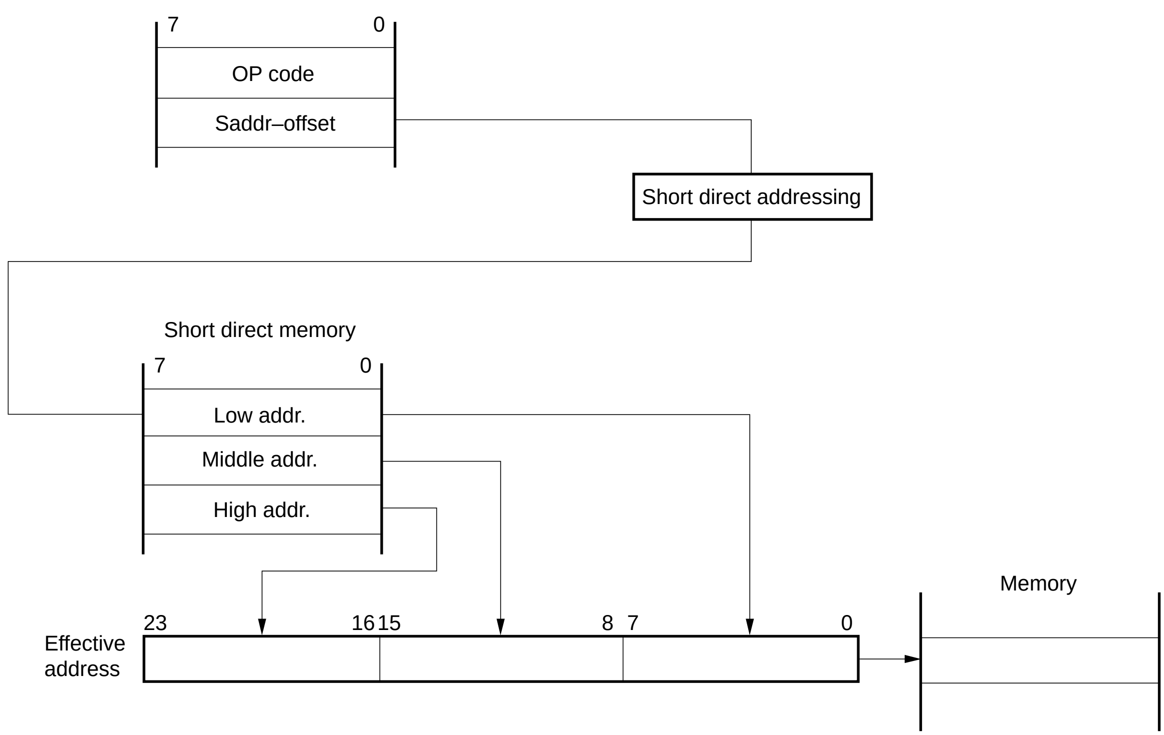
Identifier	Description Format
[%saddrg]	[%label, immediate data FD20H to FEFDH Note]

Note When the LOCATION 0 instruction is executed. When the LOCATION 0FH instruction is executed, the range is 0FFD20H to 0FFEFDH.
The μPD784915 Subseries is fixed to the LOCATION 0 instruction.

[Description Example]

- General example
  XCH A, [%saddrg]
- Specific example
  XCH A, [%0FEA0H] ; When memory indicated by 3-byte data in addresses 0FEA0H, 0FEA1H and 0FEA2H is specified

[Explanatory Diagram]



5.2.11 Stack addressing

[Function]

This type of addressing indirectly addresses the stack area in accordance with the contents of the stack pointer (SP) and user stack pointer (UUP).

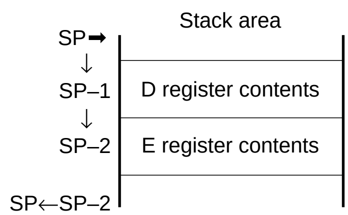
The SP is used automatically when a PUSH or POP instruction is executed, when register saving/restoration is performed as the result of interrupt request generation, and when a subroutine call or return instruction is executed.

The UUP is used automatically when a PUSHU or POPU instruction is executed.

[Description Example]

PUSH DE ; When the contents of the DE register are saved to the stack using a PUSH instruction When this instruction is executed, the SP is automatically decremented (by 2) and the contents of the DE register are saved to the stack.

[Explanatory Diagram]



Caution With stack addressing, the entire 16-Mbyte space can be accessed but a stack area cannot be reserved in the SFR area or internal ROM area.

5.2.12 24-bit register indirect addressing

[Function]

This type of addressing addresses the memory to be manipulated with the contents of register rg (RG4 to RG7) specified by the register pair specification code in the instruction word in the register bank specified by the register bank selection flag (RBS2, RBS1, RBS0) as the operand address. The entire memory space can be addressed. In addition, register indirect addressing with auto-increment that increments (+1/+2/+3) the register for which an address specification was made after instruction execution and register indirect addressing with auto-decrement that decrements (–1/–2/–3) the register after instruction execution are provided. The increment and decrement values are determined by the size of data manipulated.

This type of addressing is ideal for consecutive processing of multiple items of data.

[Operand Format]

Performed when an instruction with one of the operand formats shown below is executed.

Identifier	Description Format
mem	[TDE], [WHL], [TDE+], [WHL+], [TDE–], [WHL–], [VVP], [UUP]
mem1	[TDE], [WHL], [TDE+], [TDE–]
mem2	[TDE], [WHL]
mem3	[TDE], [WHL], [VVP], [UUP]

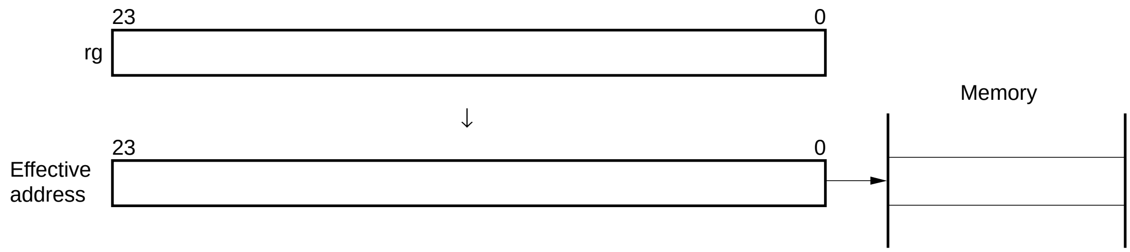
Remark “+” after register name: With auto-increment
“–” after register name: With auto-decrement

[Description Example]

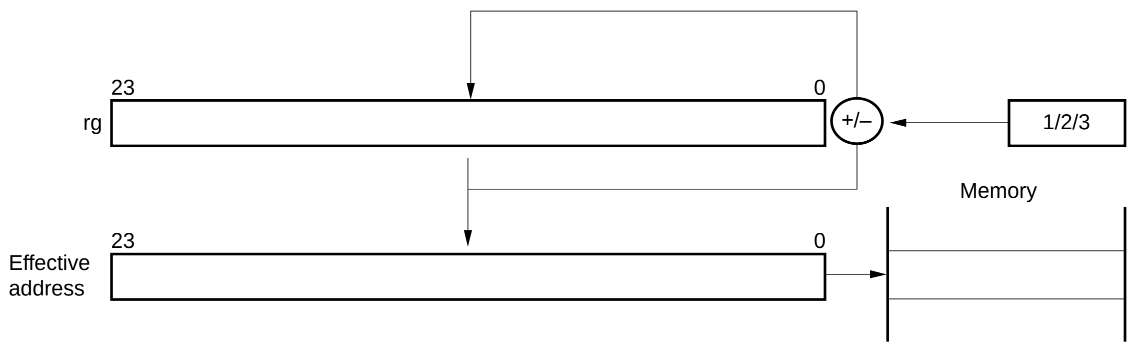
- General example
  MOV A, mem
- Specific example
  ADD A, [TDE] ; When [TDE] is specified as mem

[Explanatory Diagram]

24-bit register indirect addressing



Register indirect addressing with auto-increment/decrement



Remark +/–

+ : With auto-increment

– : With auto-decrement

1/2/3

1 : When data size is 1 byte

2 : When data size is 2 bytes (1 word)

3 : When data size is 3 bytes

5.2.13 16-bit register indirect addressing

[Function]

This type of addressing addresses the memory to be manipulated with the contents of register rp (RP0 to RP3) specified by the register specification code in the instruction word in the register bank specified by the register bank selection flag (RBS2, RBS1, RBS0) as the operand address. The base area memory space can be addressed. This type of addressing is only used with the ROR4 and ROL4 instructions, and is used when processing multiple consecutive bytes of BCD data.

This addressing is provided to maintain compatibility with the 78K/III Series, and should only be used when using a 78K/III Series program.

[Operand Format]

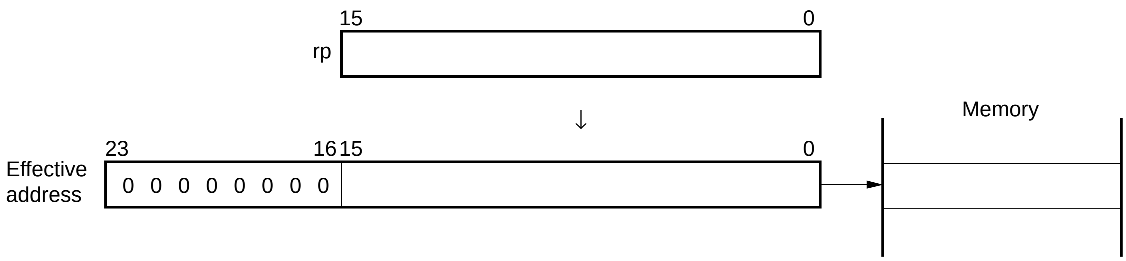
Performed when an instruction with the operand format shown below is executed.

Identifier	Description Format
mem3	[AX], [BC], [RP2], [RP3]

[Description Example]

- General example
  ROR4 mem3
- Specific example
  ROR4 [BC] ; When [BC] is written as mem3

[Explanatory Diagram]



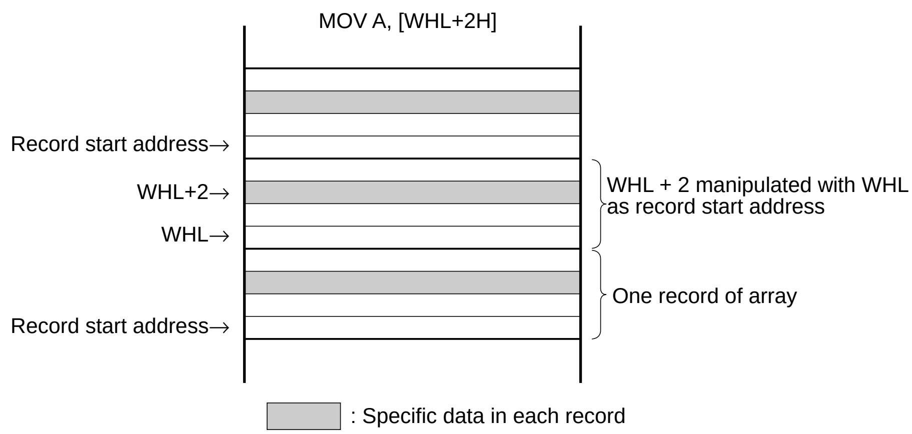
5.2.14 Based addressing

[Function]

With this type of addressing, register rg (RG4 to RG7) specified by the register specification code in the instruction word or the stack pointer (SP) in the register bank specified by the register bank selection flag (RBS2, RBS1, RBS0) addressed with the result of adding 8-bit immediate data to addition is performed with the offset data extended to 24 bits as a positive number. A carry from the 24th bit is ignored.

The entire memory space can be addressed.

This type of addressing is used when specific data is specified in an array in which one record consists of a number of bytes of data.



[Operand Format]

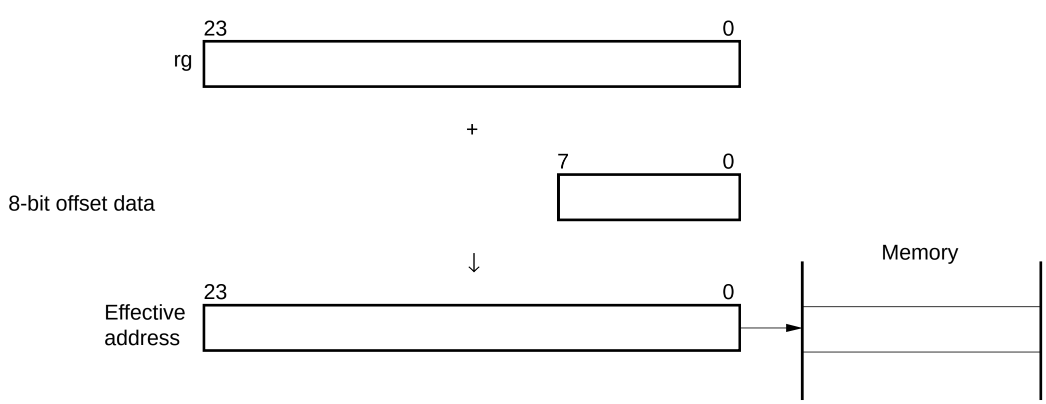
Performed when an instruction with one of the operand formats shown below is executed.

Identifier	Description Format
mem	[TDE + byte], [WHL + byte], [SP + byte], [VVP + byte], [UUP + byte]
mem1	[TDE + byte], [WHL + byte], [SP + byte], [VVP + byte], [UUP + byte]

[Description Example]

- General example
  AND A, mem
- Specific example
  AND A, [TDE+10H] ; When based addressing using the sum of register TDE as mem and 10H is selected

[Explanatory Diagram]



5.2.15 Indexed addressing

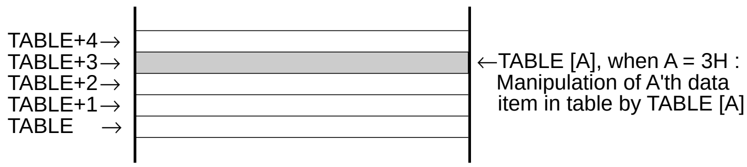
[Function]

With this type of addressing, the 24-bit address data written as the operand in the instruction word is used as the index, and memory is addressed with the result of adding the contents of the register specified in the instruction word in the register bank specified by the register bank selection flag (RBS2, RBS1, RBS0) to this value. The addition is performed with the register carry from the 24th bit is ignored.

The entire memory space can be addressed.

This type of addressing is used for table data reads, etc.

The A and B registers used in this addressing vary according to the value of the RSS bit in the PSW. When RSS = 0, these registers are R1 and R3 respectively, and when RSS = 1 they are R5 and R7. RSS should only be set to 1 when using a 78K/III Series program.



[Operand Format]

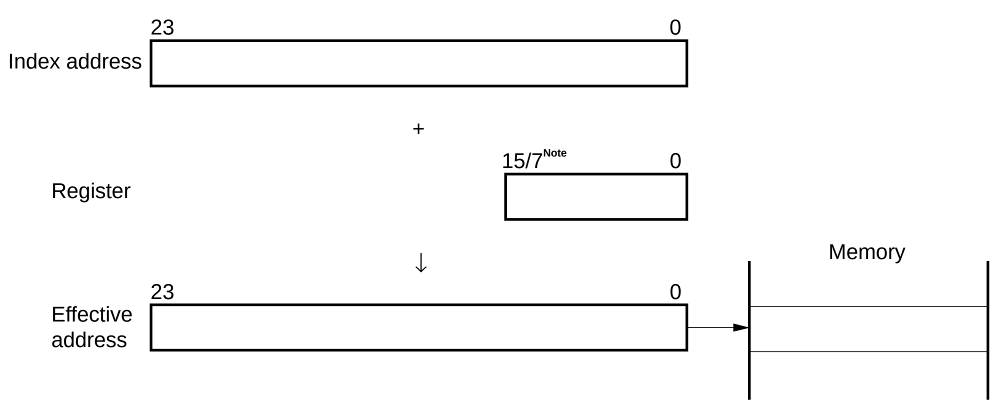
Performed when an instruction with one of the operand formats shown below is executed.

Identifier	Description Format
mem	imm24[A], imm24[B], imm24[DE], imm24[HL]
mem1	imm24[A], imm24[B], imm24[DE], imm24[HL]

[Description Example]

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

[Explanatory Diagram]



Note 15 : When register is DE or HL
7 : When register is A or B

5.2.16 Based indexed addressing

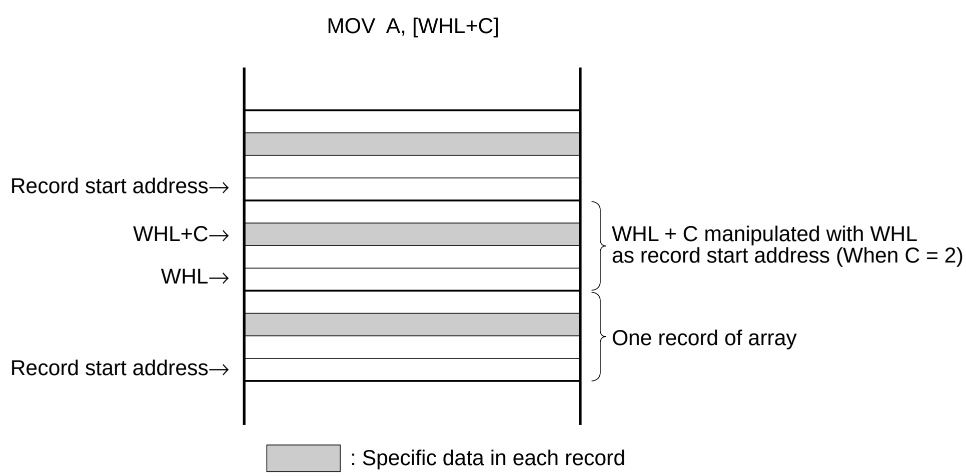
[Function]

With this type of addressing, the register specified by the register specification code in the instruction word in the register bank specified by the register bank selection flag (RBS2, RBS1, RBS0) is used as the base register, and memory is addressed with the result of adding the value of a register specified in the same way to the contents of this base register as offset data. The addition is performed with the offset data extended to 24 bits as a positive number. A carry from the 24th bit is ignored.

The entire memory space can be addressed.

This type of addressing is used to specify in order data in an array in which one record consists of a number of bytes of data.

The A, B, and C registers used in this addressing vary according to the value of the RSS bit in the PSW. When RSS = 0, these registers are R1, R3, and R2 respectively, and when RSS = 1 they are R5, R7, and R6. RSS should only be set to 1 when using a 78K/III Series program.



[Operand Format]

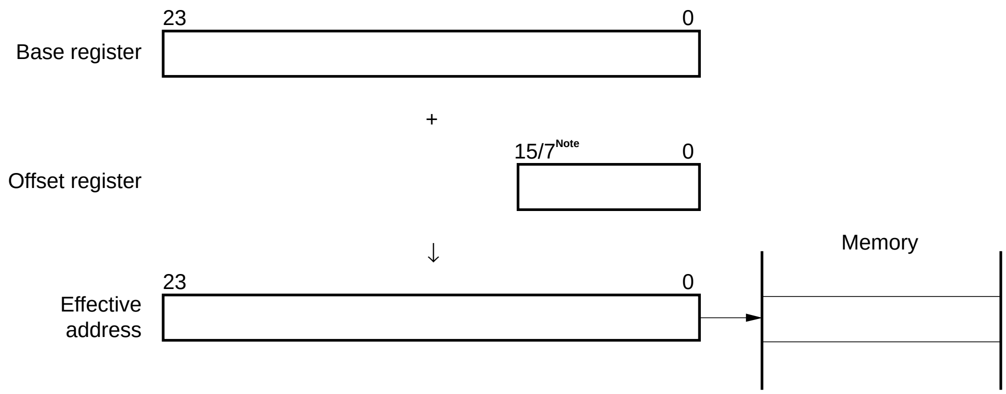
Performed when an instruction with one of the operand formats shown below is executed.

Identifier	Description Format
mem	[TDE + A], [TDE + B], [TDE + C], [WHL + A], [WHL + B], [WHL + C], [VVP + DE], [VVP + HL]
mem1	[TDE + A], [TDE + B], [TDE + C], [WHL + A], [WHL + B], [WHL + C], [VVP + DE], [VVP + HL]

[Description Example]

- General example
  AND A, mem
- Specific example
  AND A, [TDE+B] ; When based addressing using the sum of register TDE as mem and register B is selected

[Explanatory Diagram]



Note 15 : When register is DE or HL
7 : When register is A, B or C

This chapter shows the 78K/IV Series instruction set.

6.1 Legend

(1) Operand identifiers and descriptions (1/2)

Identifier	Description Format
r, r' **Note 1** r1 **Note 1** r2 r3 rp, rp' **Note 2** rp1 **Note 2** rp2 rg, rg' sfr sfrp	X(R0), A(R1), C(R2), B(R3), R4, R5, R6, R7, R8, R9, R10, R11, E(R12), D(R13), L(R14), H(R15) X(R0), A(R1), C(R2), B(R3), R4, R5, R6, R7 R8, R9, R10, R11, E(R12), D(R13), L(R14), H(R15) V, U, T, W AX(RP0), BC(RP1), RP2, RP3, VP(RP4), UP(RP5), DE(RP6), HL(RP7) AX(RP0), BC(RP1), RP2, RP3 VP(RP4), UP(RP5), DE(RP6), HL(RP7) VVP(RG4), UUP(RG5), TDE(RG6), WHL(RG7) Special function register symbol (see **Special Function Register Application Table**) Special function register symbol (register for which 16-bit operation is possible: see **Special Function Register Application Table**)
post **Note 2**	Multiple descriptions of AX(RP0), BC(RP1), RP2, RP3, VP(RP4), UP(RP5)/PSW, DE(RP6) and HL(RP7) are permissible. However, UP is only used with PUSH/POP instructions, and PSW with PUSHU/POPU instructions.
mem	[TDE], [WHL], [TDE +], [WHL +], [TDE –], [WHL –], [VVP], [UUP]: Register indirect addressing [TDE + byte], [WHL + byte], [SP + byte], [UUP + byte], [VVP + byte]: Based addressing imm24[A], imm24[B], imm24[DE], imm24[HL]: Indexed addressing [TDE + A], [TDE + B], [TDE + C], [WHL + A], [WHL + B], [WHL + C], [VVP + DE], [VVP + HL]: Based indexed addressing
mem1	All with [WHL +], [WHL –] excluded from mem
mem2	[TDE], [WHL]
mem3	[AX], [BC], [RP2], [RP3], [VVP], [UUP], [TDE], [WHL]

Notes 1. Setting the RSS bit to 1 enables R4 to R7 to be used as X, A, C, and B, but this function should only be used when using a 78K/III Series program.

2. Setting the RSS bit to 1 enables RP2 and RP3 to be used as AX and BC, but this function should only be used when using a 78K/III Series program.

(1) Operand identifiers and descriptions (2/2)

Identifier	Description Format
Note	
saddr, saddr'	FD20H to FF1FH immediate data or label
saddr1, saddr1'	FE00H to FEFFH immediate data or label
saddr2, saddr2'	FD20H to FDFFH, FF00H to FF1FH immediate data or label
saddrp	FD20H to FF1EH immediate data or label (16-bit operation)
saddrp1	FE00H to FEFEH immediate data or label (16-bit operation)
saddrp2	FD20H to FDFFH, FF00H to FF1EH immediate data or label (16-bit operation)
saddrg	FD20H to FEFDH immediate data or label (24-bit operation)
saddrg1	FE00H to FEFDH immediate data or label (24-bit operation)
saddrg2	FD20H to FDFFH immediate data or label (24-bit operation)
addr24	0H to FFFFFFH immediate data or label
addr20	0H to FFFFFH immediate data or label
addr16	0H to FFFFH immediate data or label
addr11	800H to FFFH immediate data or label
addr8	0FE00H to 0FEFFH **Note** immediate data or label
addr5	40H to 7EH immediate data or label
imm24	24-bit immediate data or label
word	16-bit immediate data or label
byte	8-bit immediate data or label
bit	3-bit immediate data or label
n	3-bit immediate data
locaddr	00H or 0FH

Note The addresses shown here apply when 00H is specified by the LOCATION instruction.
When 0FH is specified by the LOCATION instruction, F0000H should be added to the address values shown.
The μPD784915 Subseries is fixed to the LOCATION instruction.

(2) Operand column symbols

Symbol	Description
+	Auto-increment
–	Auto-decrement
#	Immediate data
!	16-bit absolute address
!!	24-bit/20-bit absolute address
$	8-bit relative address
$!	16-bit relative address
/	Bit inversion
[ ]	Indirect addressing
[%]	24-bit indirect addressing

(3) Flag column symbols

Symbol	Description
(Blank)	No change
0	Cleared to 0
1	Set to 1
x	Set or cleared depending on result
P	P/V flag operates as parity flag
V	P/V flag operates as overflow flag
R	Previously saved value is restored

(4) Operation field symbols

Symbol	Description
jdisp8	Signed two’s complement data (8 bits) indicating relative address distance between start address of next instruction and branch address
jdisp16	Signed two’s complement data (16 bits) indicating relative address distance between start address of next instruction and branch address
PC_{HW}	PC bits 16 to 19
PC_{LW}	PC bits 0 to 15

(5) Number of bytes of instruction that includes mem in operands

mem Mode	Register Indirect Addressing		Based Addressing	Indexed Addressing	Based Indexed Addressing
Number of bytes	1	2 Note	3	5	2

Note One-byte instruction only when [TDE], [WHL], [TDE +], [TDE –], [WHL +], or [WHL –] is written as mem in a MOV instruction .

(6) Number of bytes of instruction that includes saddr, saddrp, r or rp in operands

In some instructions which include saddr, saddrp, r, rp as operands, the number of bytes is written divided into two with “/”. Which number of bytes is to be used depends on the table below.

Identifier	Number of Bytes: Left Side	Number of Bytes: Right Side
saddr	saddr2	saddr1
saddrp	saddrp2	saddrp1
r	r1	r2
rp	rp1	rp2

(7) Description of instructions that include mem in operands and string instructions

Operands TDE, WHL, VVP, and UUP (24-bit registers) can also be written as DE, HL, VP, and UP respectively. However, they are still treated as TDE, WHL, VVP, and UUP (24-bit registers) when written as DE, HL, VP, and UP.

6.2 List of Instruction Operations

(1) 8-bit data transfer instruction: MOV

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
MOV	r, #byte	2/3	r ← byte					
	saddr, #byte	3/4	(saddr) ← byte					
	sfr, #byte	3	sfr ← byte					
	!addr16, #byte	5	(addr16) ← byte					
	!!addr24, #byte	6	(addr24) ← byte					
	r, r'	2/3	r ← r'					
	A, r	1/2	A ← r					
	A, saddr2	2	A ← (saddr2)					
	r, saddr	3	r ← (saddr)					
	saddr2, A	2	(saddr2) ← A					
	saddr, r	3	(saddr) ← r					
	A, sfr	2	A ← sfr					
	r, sfr	3	r ← sfr					
	sfr, A	2	sfr ← A					
	sfr, r	3	sfr ← r					
	saddr, saddr'	4	(saddr) ← (saddr')					
	r, !addr16	4	r ← (addr16)					
	!addr16, r	4	(addr16) ← r					
	r, !!addr24	5	r ← (addr24)					
	!!addr24, r	5	(addr24) ← r					
	A, [saddrp]	2/3	A ← ((saddrp))					
	A, [%saddrg]	3/4	A ← ((saddrg))					
	A, mem	1-5	A ← (mem)					
	[saddrp], A	2/3	((saddrp)) ← A					
	[%saddrg], A	3/4	((saddrg)) ← A					
	mem, A	1-5	(mem) ← A					
	PSWL, #byte	3	PSW_L ← byte	×	×	×	×	×
	PSWH, #byte	3	PSW_H ← byte					
	PSWL, A	2	PSW_L ← A	×	×	×	×	×
	PSWH, A	2	PSW_H ← A					
	A, PSWL	2	A ← PSW_L					
	A, PSWH	2	A ← PSW_H					
	r3, #byte	3	r3 ← byte					
	A, r3	2	A ← r3					
	r3, A	2	r3 ← A					

(2) 16-bit data transfer instruction: MOVW

Mnemonic	Operands	Bytes	Operation	Flags
				S Z AC P/V CY
MOVW	rp, #word	3	rp ← word	
	saddrp, #word	4/5	(saddrp) ← word	
	sfrp, #word	4	sfrp ← word	
	!addr16, #word	6	(addr16) ← word	
	!!addr24, #word	7	(addr24) ← word	
	rp, rp’	2	rp ← rp’	
	AX, saddrp2	2	AX ← (saddrp2)	
	rp, saddrp	3	rp ← (saddrp)	
	saddrp2, AX	2	(saddrp2) ← AX	
	saddrp, rp	3	(saddrp) ← rp	
	AX, sfrp	2	AX ← sfrp	
	rp, sfrp	3	rp ← sfrp	
	sfrp, AX	2	sfrp ← AX	
	sfrp, rp	3	sfrp ← rp	
	saddrp, saddrp’	4	(saddrp) ← (saddrp’)	
	rp, !addr16	4	rp ← (addr16)	
	!addr16, rp	4	(addr16) ← rp	
	rp, !!addr24	5	rp ← (addr24)	
	!!addr24, rp	5	(addr24) ← rp	
	AX, [saddrp]	3/4	AX ← ((saddrp))	
	AX, [%saddrg]	3/4	AX ← ((saddrg))	
	AX, mem	2-5	AX ← (mem)	
	[saddrp], AX	3/4	((saddrp)) ← AX	
	[%saddrg], AX	3/4	((saddrg)) ← AX	
	mem, AX	2-5	(mem) ← AX	

(3) 24-bit data transfer instruction: MOVG

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
MOVG	rg, #imm24	5	rg ← imm24					
	rg, rg'	2	rg ← rg'					
	rg, !!addr24	5	rg ← (addr24)					
	!!addr24, rg	5	(addr24) ← rg					
	rg, saddrg	3	rg ← (saddrg)					
	saddrg, rg	3	(saddrg) ← rg					
	WHL, [%saddrg]	3/4	WHL ← ((saddrg))					
	[%saddrg], WHL	3/4	((saddrg)) ← WHL					
	WHL, mem1	2-5	WHL ← (mem1)					
	mem1, WHL	2-5	(mem1) ← WHL					

(4) 8-bit data exchange instruction: XCH

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
XCH	r, r'	2/3	r ↔ r'					
	A, r	1/2	A ↔ r					
	A, saddr2	2	A ↔ (saddr2)					
	r, saddr	3	r ↔ (saddr)					
	r, sfr	3	r ↔ sfr					
	saddr, saddr'	4	(saddr) ↔ (saddr')					
	r, !addr16	4	r ↔ (addr16)					
	r, !!addr24	5	r ↔ (addr24)					
	A, [saddrp]	2/3	A ↔ ((saddrp))					
	A, [%saddrg]	3/4	A ↔ ((saddrg))					
	A, mem	2-5	A ↔ (mem)					

(5) 16-bit data exchange instruction: XCHW

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
XCHW	rp, rp'	2	rp ↔ rp'					
	AX, saddrp2	2	AX ↔ (saddrp2)					
	rp, saddrp	3	rp ↔ (saddrp)					
	rp, sfrp	3	rp ↔ sfrp					
	AX, [saddrp]	3/4	AX ↔ ((saddrp))					
	AX, [%saddrg]	3/4	AX ↔ ((saddrg))					
	AX, !addr16	4	AX ↔ (addr16)					
	AX, !!addr24	5	AX ↔ (addr24)					
	saddrp, saddrp'	4	(saddrp) ↔ (saddrp')					
	AX, mem	2-5	AX ↔ (mem)					

(6) 8-bit operation instructions: ADD, ADDC, SUB, SUBC, CMP, AND, OR, XOR

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
ADD	A, #byte	2	A, CY ← A + byte	×	×	×	V	×
	r, #byte	3	r, CY ← r + byte	×	×	×	V	×
	saddr, #byte	3/4	(saddr), CY ← (saddr) + byte	×	×	×	V	×
	sfr, #byte	4	sfr, CY ← sfr + byte	×	×	×	V	×
	r, r'	2/3	r, CY ← r + r'	×	×	×	V	×
	A, saddr2	2	A, CY ← A + (saddr2)	×	×	×	V	×
	r, saddr	3	r, CY ← r + (saddr)	×	×	×	V	×
	saddr, r	3	(saddr), CY ← (saddr) + r	×	×	×	V	×
	r, sfr	3	r, CY ← r + sfr	×	×	×	V	×
	sfr, r	3	sfr, CY ← sfr + r	×	×	×	V	×
	saddr, saddr'	4	(saddr), CY ← (saddr) + (saddr')	×	×	×	V	×
	A, [saddrp]	3/4	A, CY ← A + ((saddrp))	×	×	×	V	×
	A, [%saddrg]	3/4	A, CY ← A + ((saddrg))	×	×	×	V	×
	[saddrp], A	3/4	((saddrp)), CY ← ((saddrp)) + A	×	×	×	V	×
	[%saddrg], A	3/4	((saddrg)), CY ← ((saddrg)) + A	×	×	×	V	×
	A, !addr16	4	A, CY ← A + (addr16)	×	×	×	V	×
	A, !!addr24	5	A, CY ← A + (addr24)	×	×	×	V	×
	!addr16, A	4	(addr16), CY ← (addr16) + A	×	×	×	V	×
	!!addr24, A	5	(addr24), CY ← (addr24) + A	×	×	×	V	×
	A, mem	2-5	A, CY ← A + (mem)	×	×	×	V	×
	mem, A	2-5	(mem), CY ← (mem) + A	×	×	×	V	×

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
ADDC	A, #byte	2	A, CY ← A + byte + CY	×	×	×	V	×
	r, #byte	3	r, CY ← r + byte + CY	×	×	×	V	×
	saddr, #byte	3/4	(saddr), CY ← (saddr) + byte + CY	×	×	×	V	×
	sfr, #byte	4	sfr, CY ← sfr + byte + CY	×	×	×	V	×
	r, r’	2/3	r, CY ← r + r’ + CY	×	×	×	V	×
	A, saddr2	2	A, CY ← A + (saddr2) + CY	×	×	×	V	×
	r, saddr	3	r, CY ← r + (saddr) + CY	×	×	×	V	×
	saddr, r	3	(saddr), CY ← (saddr) + r + CY	×	×	×	V	×
	r, sfr	3	r, CY ← r + sfr + CY	×	×	×	V	×
	sfr, r	3	sfr, CY ← sfr + r + CY	×	×	×	V	×
	saddr, saddr’	4	(saddr), CY ← (saddr) + (saddr’) + CY	×	×	×	V	×
	A, [saddrp]	3/4	A, CY ← A + ((saddrp)) + CY	×	×	×	V	×
	A, [%saddrg]	3/4	A, CY ← A + ((saddrg)) + CY	×	×	×	V	×
	[saddrp], A	3/4	((saddrp)), CY ← ((saddrp)) + A + CY	×	×	×	V	×
	[%saddrg], A	3/4	((saddrg)), CY ← ((saddrg)) + A + CY	×	×	×	V	×
	A, !addr16	4	A, CY ← A + (addr16) + CY	×	×	×	V	×
	A, !!addr24	5	A, CY ← A + (addr24) + CY	×	×	×	V	×
	!addr16, A	4	(addr16), CY ← (addr16) + A + CY	×	×	×	V	×
	!!addr24, A	5	(addr24), CY ← (addr24) + A + CY	×	×	×	V	×
	A, mem	2-5	A, CY ← A + (mem) + CY	×	×	×	V	×
	mem, A	2-5	(mem), CY ← (mem) + A + CY	×	×	×	V	×

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
SUB	A, #byte	2	A, CY ← A – byte	×	×	×	V	×
	r, #byte	3	r, CY ← r – byte	×	×	×	V	×
	saddr, #byte	3/4	(saddr), CY ← (saddr) – byte	×	×	×	V	×
	sfr, #byte	4	sfr, CY ← sfr – byte	×	×	×	V	×
	r, r’	2/3	r, CY ← r – r’	×	×	×	V	×
	A, saddr2	2	A, CY ← A – (saddr2)	×	×	×	V	×
	r, saddr	3	r, CY ← r – (saddr)	×	×	×	V	×
	saddr, r	3	(saddr), CY ← (saddr) – r	×	×	×	V	×
	r, sfr	3	r, CY ← r – sfr	×	×	×	V	×
	sfr, r	3	sfr, CY ← sfr – r	×	×	×	V	×
	saddr, saddr’	4	(saddr), CY ← (saddr) – (saddr’)	×	×	×	V	×
	A, [saddrp]	3/4	A, CY ← A – ((saddrp))	×	×	×	V	×
	A, [%saddrg]	3/4	A, CY ← A – ((saddrg))	×	×	×	V	×
	[saddrp], A	3/4	((saddrp)), CY ← ((saddrp)) – A	×	×	×	V	×
	[%saddrg], A	3/4	((saddrg)), CY ← ((saddrg)) – A	×	×	×	V	×
	A, !addr16	4	A, CY ← A – (addr16)	×	×	×	V	×
	A, !!addr24	5	A, CY ← A – (addr24)	×	×	×	V	×
	!addr16, A	4	(addr16), CY ← (addr16) – A	×	×	×	V	×
	!!addr24, A	5	(addr24), CY ← (addr24) – A	×	×	×	V	×
	A, mem	2-5	A, CY ← A – (mem)	×	×	×	V	×
	mem, A	2-5	(mem), CY ← (mem) – A	×	×	×	V	×

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
SUBC	A, #byte	2	A, CY ← A – byte – CY	×	×	×	V	×
	r, #byte	3	r, CY ← r – byte – CY	×	×	×	V	×
	saddr, #byte	3/4	(saddr), CY ← (saddr) – byte – CY	×	×	×	V	×
	sfr, #byte	4	sfr, CY ← sfr – byte – CY	×	×	×	V	×
	r, r'	2/3	r, CY ← r – r' – CY	×	×	×	V	×
	A, saddr2	2	A, CY ← A – (saddr2) – CY	×	×	×	V	×
	r, saddr	3	r, CY ← r – (saddr) – CY	×	×	×	V	×
	saddr, r	3	(saddr), CY ← (saddr) – r – CY	×	×	×	V	×
	r, sfr	3	r, CY ← r – sfr – CY	×	×	×	V	×
	sfr, r	3	sfr, CY ← sfr – r – CY	×	×	×	V	×
	saddr, saddr'	4	(saddr), CY ← (saddr) – (saddr') – CY	×	×	×	V	×
	A, [saddrp]	3/4	A, CY ← A – ((saddrp)) – CY	×	×	×	V	×
	A, [%saddrg]	3/4	A, CY ← A – ((saddrg)) – CY	×	×	×	V	×
	[saddrp], A	3/4	((saddrp)), CY ← ((saddrp)) – A – CY	×	×	×	V	×
	[%saddrg], A	3/4	((saddrg)), CY ← ((saddrg)) – A – CY	×	×	×	V	×
	A, !addr16	4	A, CY ← A – (addr16) – CY	×	×	×	V	×
	A, !!addr24	5	A, CY ← A – (addr24) – CY	×	×	×	V	×
	!addr16, A	4	(addr16), CY ← (addr16) – A – CY	×	×	×	V	×
	!!addr24, A	5	(addr24), CY ← (addr24) – A – CY	×	×	×	V	×
	A, mem	2-5	A, CY ← A – (mem) – CY	×	×	×	V	×
	mem, A	2-5	(mem), CY ← (mem) – A – CY	×	×	×	V	×

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
CMP	A, #byte	2	A – byte	×	×	×	V	×
	r, #byte	3	r – byte	×	×	×	V	×
	saddr, #byte	3/4	(saddr) – byte	×	×	×	V	×
	sfr, #byte	4	sfr – byte	×	×	×	V	×
	r, r’	2/3	r – r’	×	×	×	V	×
	A, saddr2	2	A – (saddr2)	×	×	×	V	×
	r, saddr	3	r – (saddr)	×	×	×	V	×
	saddr, r	3	(saddr) – r	×	×	×	V	×
	r, sfr	3	r – sfr	×	×	×	V	×
	sfr, r	3	sfr – r	×	×	×	V	×
	saddr, saddr’	4	(saddr) – (saddr’)	×	×	×	V	×
	A, [saddrp]	3/4	A – ((saddrp))	×	×	×	V	×
	A, [%saddrg]	3/4	A – ((saddrg))	×	×	×	V	×
	[saddrp], A	3/4	((saddrp)) – A	×	×	×	V	×
	[%saddrg], A	3/4	((saddrg)) – A	×	×	×	V	×
	A, !addr16	4	A – (addr16)	×	×	×	V	×
	A, !!addr24	5	A – (addr24)	×	×	×	V	×
	!addr16, A	4	(addr16) – A	×	×	×	V	×
	!!addr24, A	5	(addr24) – A	×	×	×	V	×
	A, mem	2-5	A – (mem)	×	×	×	V	×
	mem, A	2-5	(mem) – A	×	×	×	V	×

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
AND	A, #byte	2	A ← A ∧ byte	×	×		P	
	r, #byte	3	r ← r ∧ byte	×	×		P	
	saddr, #byte	3/4	(saddr) ← (saddr) ∧ byte	×	×		P	
	sfr, #byte	4	sfr ← sfr ∧ byte	×	×		P	
	r, r'	2/3	r ← r ∧ r'	×	×		P	
	A, saddr2	2	A ← A ∧ (saddr2)	×	×		P	
	r, saddr	3	r ← r ∧ (saddr)	×	×		P	
	saddr, r	3	(saddr) ← (saddr) ∧ r	×	×		P	
	r, sfr	3	r ← r ∧ sfr	×	×		P	
	sfr, r	3	sfr ← sfr ∧ r	×	×		P	
	saddr, saddr'	4	(saddr) ← (saddr) ∧ (saddr')	×	×		P	
	A, [saddrp]	3/4	A ← A ∧ ((saddrp))	×	×		P	
	A, [%saddrg]	3/4	A ← A ∧ ((saddrg))	×	×		P	
	[saddrp], A	3/4	((saddrp)) ← ((saddrp)) ∧ A	×	×		P	
	[%saddrg], A	3/4	((saddrg)) ← ((saddrg)) ∧ A	×	×		P	
	A, !addr16	4	A ← A ∧ (addr16)	×	×		P	
	A, !!addr24	5	A ← A ∧ (addr24)	×	×		P	
	!addr16, A	4	(addr16) ← (addr16) ∧ A	×	×		P	
	!!addr24, A	5	(addr24) ← (addr24) ∧ A	×	×		P	
	A, mem	2-5	A ← A ∧ (mem)	×	×		P	
	mem, A	2-5	(mem) ← (mem) ∧ A	×	×		P	

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
OR	A, #byte	2	A ← A ∨ byte	×	×		P	
	r, #byte	3	r ← r ∨ byte	×	×		P	
	saddr, #byte	3/4	(saddr) ← (saddr) ∨ byte	×	×		P	
	sfr, #byte	4	sfr ← sfr ∨ byte	×	×		P	
	r, r’	2/3	r ← r ∨ r’	×	×		P	
	A, saddr2	2	A ← A ∨ (saddr2)	×	×		P	
	r, saddr	3	r ← r ∨ (saddr)	×	×		P	
	saddr, r	3	(saddr) ← (saddr) ∨ r	×	×		P	
	r, sfr	3	r ← r ∨ sfr	×	×		P	
	sfr, r	3	sfr ← sfr ∨ r	×	×		P	
	saddr, saddr’	4	(saddr) ← (saddr) ∨ (saddr’)	×	×		P	
	A, [saddrp]	3/4	A ← A ∨ ((saddrp))	×	×		P	
	A, [%saddrg]	3/4	A ← A ∨ ((saddrg))	×	×		P	
	[saddrp], A	3/4	((saddrp)) ← ((saddrp)) ∨ A	×	×		P	
	[%saddrg], A	3/4	((saddrg)) ← ((saddrg)) ∨ A	×	×		P	
	A, !addr16	4	A ← A ∨ (saddr16)	×	×		P	
	A, !!addr24	5	A ← A ∨ (saddr24)	×	×		P	
	!addr16, A	4	(addr16) ← (addr16) ∨ A	×	×		P	
	!!addr24, A	5	(addr24) ← (addr24) ∨ A	×	×		P	
	A, mem	2-5	A ← A ∨ (mem)	×	×		P	
	mem, A	2-5	(mem) ← (mem) ∨ A	×	×		P	

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
XOR	A, #byte	2	A ← A ∀ byte	×	×		P	
	r, #byte	3	r ← r ∀ byte	×	×		P	
	saddr, #byte	3/4	(saddr) ← (saddr) ∀ byte	×	×		P	
	sfr, #byte	4	sfr ← sfr ∀ byte	×	×		P	
	r, r’	2/3	r ← r ∀ r’	×	×		P	
	A, saddr2	2	A ← A ∀ (saddr2)	×	×		P	
	r, saddr	3	r ← r ∀ (saddr)	×	×		P	
	saddr, r	3	(saddr) ← (saddr) ∀ r	×	×		P	
	r, sfr	3	r ← r ∀ sfr	×	×		P	
	sfr, r	3	sfr ← sfr ∀ r	×	×		P	
	saddr, saddr’	4	(saddr) ← (saddr) ∀ (saddr’)	×	×		P	
	A, [saddrp]	3/4	A ← A ∀ ((saddrp))	×	×		P	
	A, [%saddrg]	3/4	A ← A ∀ ((saddrg))	×	×		P	
	[saddrp], A	3/4	((saddrp)) ← ((saddrp)) ∀ A	×	×		P	
	[%saddrg], A	3/4	((saddrg)) ← ((saddrg)) ∀ A	×	×		P	
	A, !addr16	4	A ← A ∀ (addr16)	×	×		P	
	A, !!addr24	5	A ← A ∀ (addr24)	×	×		P	
	!addr16, A	4	(addr16) ← (addr16) ∀ A	×	×		P	
	!!addr24, A	5	(addr24) ← (addr24) ∀ A	×	×		P	
	A, mem	2-5	A ← A ∀ (mem)	×	×		P	
	mem, A	2-5	(mem) ← (mem) ∀ A	×	×		P	

(7) 16-bit operation instructions: ADDW, SUBW, CMPW

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
ADDW	AX, #word	3	AX, CY ← AX + word	×	×	×	V	×
	rp, #word	4	rp, CY ← rp + word	×	×	×	V	×
	rp, rp'	2	rp, CY ← rp + rp'	×	×	×	V	×
	AX, saddrp2	2	AX, CY ← AX + (saddrp2)	×	×	×	V	×
	rp, saddrp	3	rp, CY ← rp + (saddrp)	×	×	×	V	×
	saddrp, rp	3	(saddrp), CY ← (saddrp) + rp	×	×	×	V	×
	rp, sfrp	3	rp, CY ← rp + sfrp	×	×	×	V	×
	sfrp, rp	3	sfrp, CY ← sfrp + rp	×	×	×	V	×
	saddrp, #word	4/5	(saddrp), CY ← (saddrp) + word	×	×	×	V	×
	sfrp, #word	5	sfrp, CY ← sfrp + word	×	×	×	V	×
	saddrp, saddrp'	4	(saddrp), CY ← (saddrp) + (saddrp')	×	×	×	V	×
SUBW	AX, #word	3	AX, CY ← AX – word	×	×	×	V	×
	rp, #word	4	rp, CY ← rp – word	×	×	×	V	×
	rp, rp'	2	rp, CY ← rp – rp'	×	×	×	V	×
	AX, saddrp2	2	AX, CY ← AX – (saddrp2)	×	×	×	V	×
	rp, saddrp	3	rp, CY ← rp – (saddrp)	×	×	×	V	×
	saddrp, rp	3	(saddrp), CY ← (saddrp) – rp	×	×	×	V	×
	rp, sfrp	3	rp, CY ← rp – sfrp	×	×	×	V	×
	sfrp, rp	3	sfrp, CY ← sfrp – rp	×	×	×	V	×
	saddrp, #word	4/5	(saddrp), CY ← (saddrp) – word	×	×	×	V	×
	sfrp, #word	5	sfrp, CY ← sfrp – word	×	×	×	V	×
	saddrp, saddrp'	4	(saddrp), CY ← (saddrp) – (saddrp')	×	×	×	V	×
CMPW	AX, #word	3	AX – word	×	×	×	V	×
	rp, #word	4	rp – word	×	×	×	V	×
	rp, rp'	2	rp – rp'	×	×	×	V	×
	AX, saddrp2	2	AX – (saddrp2)	×	×	×	V	×
	rp, saddrp	3	rp – (saddrp)	×	×	×	V	×
	saddrp, rp	3	(saddrp) – rp	×	×	×	V	×
	rp, sfrp	3	rp – sfrp	×	×	×	V	×
	sfrp, rp	3	sfrp – rp	×	×	×	V	×
	saddrp, #word	4/5	(saddrp) – word	×	×	×	V	×
	sfrp, #word	5	sfrp – word	×	×	×	V	×
	saddrp, saddrp'	4	(saddrp) – (saddrp')	×	×	×	V	×

(8) 24-bit operation instructions: ADDG, SUBG

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
ADDG	rg, rg'	2	rg, CY ← rg + rg'	×	×	×	V	×
	rg, #imm24	5	rg, CY ← rg + imm24	×	×	×	V	×
	WHL, saddrg	3	WHL, CY ← WHL + (saddrg)	×	×	×	V	×
SUBG	rg, rg'	2	rg, CY ← rg – rg'	×	×	×	V	×
	rg, #imm24	5	rg, CY ← rg – imm24	×	×	×	V	×
	WHL, saddrg	3	WHL, CY ← WHL – (saddrg)	×	×	×	V	×

(9) Multiplication instructions: MULU, MULUW, MULW, DIVUW, DIVUX

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
MULU	r	2/3	AX ← A × r					
MULUW	rp	2	AX (higher half), rp (lower half) ← AX × rp					
MULW	rp	2	AX (higher half), rp (lower half) ← AX × rp					
DIVUW	r	2/3	AX (quotient), r (remainder) ← AX ÷ r **Note 1**					
DIVUX	rp	2	AXDE (quotient), rp (remainder) ← AXDE ÷ rp **Note 2**					

Notes 1. When r = 0, r ← X, AX ← FFFFH

2. When rp = 0, rp ← DE, AXDE ← FFFFFFFFH

(10) Special operation instructions: MACW, MACSW, SACW

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
MACW	byte	3	AXDE ← (B) × (C) + AXDE, B ← B + 2, C ← C + 2, byte ← byte – 1 End if (byte = 0 or P/V = 1)	×	×	×	V	×
MACSW	byte	3	AXDE ← (B) × (C) + AXDE, B ← B + 2, C ← C + 2, byte ← byte – 1 if byte = 0 then End if P/V = 1 then if overflow AXDE ← 7FFFFFFFH, End if underflow AXDE ← 80000000H, End	×	×	×	V	×
SACW	[TDE +], [WHL +]	4	AX ← | (TDE) – (WHL) | + AX, TDE ← TDE + 2, WHL ← WHL + 2 C ← C – 1 End if (C = 0 or CY = 1)	×	×	×	V	×

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

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
INC	r	1/2	r ← r + 1	×	×	×	V	
	saddr	2/3	(saddr) ← (saddr) + 1	×	×	×	V	
DEC	r	1/2	r ← r – 1	×	×	×	V	
	saddr	2/3	(saddr) ← (saddr) – 1	×	×	×	V	
INCW	rp	2/1	rp ← rp + 1					
	saddrp	3/4	(saddrp) ← (saddrp) + 1					
DECW	rp	2/1	rp ← rp – 1					
	saddrp	3/4	(saddrp) ← (saddrp) – 1					
INCG	rg	2	rg ← rg + 1					
DECG	rg	2	rg ← rg – 1					

(12) Adjustment instructions: ADJBA, ADJBS, CVTBW

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
ADJBA		2	Decimal Adjust Accumulator after Addition	×	×	×	P	×
ADJBS		2	Decimal Adjust Accumulator after Subtract	×	×	×	P	×
CVTBW		1	X ← A, A ← 00H if A_7 = 0 X ← A, A ← FFH if A_7 = 1					

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

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
ROR	r, n	2/3	$(CY, r_7 \leftarrow r_0, r_{m-1} \leftarrow r_m) \times n$ $n = 0 - 7$				P	×
ROL	r, n	2/3	$(CY, r_0 \leftarrow r_7, r_{m+1} \leftarrow r_m) \times n$ $n = 0 - 7$				P	×
RORC	r, n	2/3	$(CY \leftarrow r_0, r_7 \leftarrow CY, r_{m-1} \leftarrow r_m) \times n$ $n = 0 - 7$				P	×
ROLC	r, n	2/3	$(CY \leftarrow r_7, r_0 \leftarrow CY, r_{m+1} \leftarrow r_m) \times n$ $n = 0 - 7$				P	×
SHR	r, n	2/3	$(CY \leftarrow r_0, r_7 \leftarrow 0, r_{m-1} \leftarrow r_m) \times n$ $n = 0 - 7$	×	×	0	P	×
SHL	r, n	2/3	$(CY \leftarrow r_7, r_0 \leftarrow 0, r_{m+1} \leftarrow r_m) \times n$ $n = 0 - 7$	×	×	0	P	×
SHRW	rp, n	2	$(CY \leftarrow rp_0, rp_{15} \leftarrow 0, rp_{m-1} \leftarrow rp_m) \times n$ $n = 0 - 7$	×	×	0	P	×
SHLW	rp, n	2	$(CY \leftarrow rp_{15}, rp_0 \leftarrow 0, rp_{m+1} \leftarrow rp_m) \times n$ $n = 0 - 7$	×	×	0	P	×
ROR4	mem3	2	$A_{3-0} \leftarrow (mem3)_{3-0}, (mem3)_{7-4} \leftarrow A_{3-0},$ $(mem3)_{3-0} \leftarrow (mem3)_{7-4}$					
ROL4	mem3	2	$A_{3-0} \leftarrow (mem3)_{7-4}, (mem3)_{3-0} \leftarrow A_{3-0},$ $(mem3)_{7-4} \leftarrow (mem3)_{3-0}$					

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

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
MOV1	CY, saddr.bit	3/4	CY ← (saddr.bit)					×
	CY, sfr.bit	3	CY ← sfr.bit					×
	CY, X.bit	2	CY ← X.bit					×
	CY, A.bit	2	CY ← A.bit					×
	CY, PSWL.bit	2	CY ← PSW$_L$.bit					×
	CY, PSWH.bit	2	CY ← PSW$_H$.bit					×
	CY, !addr16.bit	5	CY ← !addr16.bit					×
	CY, !!addr24.bit	6	CY ← !!addr24.bit					×
	CY, mem2.bit	2	CY ← mem2.bit					×
	saddr.bit, CY	3/4	(saddr.bit) ← CY					
	sfr.bit, CY	3	sfr.bit ← CY					
	X.bit, CY	2	X.bit ← CY					
	A.bit, CY	2	A.bit, ← CY					
	PSWL.bit, CY	2	PSW$_L$.bit ← CY	×	×	×	×	×
	PSWH.bit, CY	2	PSW$_H$.bit ← CY					
	!addr16.bit, CY	5	!addr16.bit ← CY					
	!!addr24.bit, CY	6	!!addr24.bit ← CY					
	mem2.bit, CY	2	mem2.bit ← CY					
AND1	CY, saddr.bit	3/4	CY ← CY ∧ (saddr.bit)					×
	CY, /saddr.bit	3/4	CY ← CY ∧ $\overline{\text{(saddr.bit)}}$					×
	CY, sfr.bit	3	CY ← CY ∧ sfr.bit					×
	CY, /sfr.bit	3	CY ← CY ∧ $\overline{\text{sfr.bit}}$					×
	CY, X.bit	2	CY ← CY ∧ X.bit					×
	CY, /X.bit	2	CY ← CY ∧ $\overline{\text{X.bit}}$					×
	CY, A.bit	2	CY ← CY ∧ A.bit					×
	CY, /A.bit	2	CY ← CY ∧ $\overline{\text{A.bit}}$					×
	CY, PSWL.bit	2	CY ← CY ∧ PSW$_L$.bit					×
	CY, /PSWL.bit	2	CY ← CY ∧ $\overline{\text{PSW}_L\text{.bit}}$					×
	CY, PSWH.bit	2	CY ← CY ∧ PSW$_H$.bit					×
	CY, /PSWH.bit	2	CY ← CY ∧ $\overline{\text{PSW}_H\text{.bit}}$					×
	CY, !addr16.bit	5	CY ← CY ∧ !addr16.bit					×
	CY, /!addr16.bit	5	CY ← CY ∧ $\overline{\text{!addr16.bit}}$					×
	CY, !!addr24.bit	6	CY ← CY ∧ !!addr24.bit					×
	CY, /!!addr24.bit	6	CY ← CY ∧ $\overline{\text{!!addr24.bit}}$					×
	CY, mem2.bit	2	CY ← CY ∧ mem2.bit					×
	CY, /mem2.bit	2	CY ← CY ∧ $\overline{\text{mem2.bit}}$					×

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
OR1	CY, saddr.bit	3/4	CY ← CY ∨ (saddr.bit)					×
	CY, /saddr.bit	3/4	CY ← CY ∨ $\overline{\text{(saddr.bit)}}$					×
	CY, sfr.bit	3	CY ← CY ∨ sfr.bit					×
	CY, /sfr.bit	3	CY ← CY ∨ $\overline{\text{sfr.bit}}$					×
	CY, X.bit	2	CY ← CY ∨ X.bit					×
	CY, /X.bit	2	CY ← CY ∨ $\overline{\text{X.bit}}$					×
	CY, A.bit	2	CY ← CY ∨ A.bit					×
	CY, /A.bit	2	CY ← CY ∨ $\overline{\text{A.bit}}$					×
	CY, PSWL.bit	2	CY ← CY ∨ PSW_L.bit					×
	CY, /PSWL.bit	2	CY ← CY ∨ $\overline{PSW_L\text{.bit}}$					×
	CY, PSWH.bit	2	CY ← CY ∨ PSW_H.bit					×
	CY, /PSWH.bit	2	CY ← CY ∨ $\overline{PSW_H\text{.bit}}$					×
	CY, !addr16.bit	5	CY ← CY ∨ !addr16.bit					×
	CY, /!addr16.bit	5	CY ← CY ∨ $\overline{\text{!addr16.bit}}$					×
	CY, !!addr24.bit	6	CY ← CY ∨ !!addr24.bit					×
	CY, /!!addr24.bit	6	CY ← CY ∨ $\overline{\text{!!addr24.bit}}$					×
	CY, mem2.bit	2	CY ← CY ∨ mem2.bit					×
	CY, /mem2.bit	2	CY ← CY ∨ $\overline{\text{mem2.bit}}$					×
XOR1	CY, saddr.bit	3/4	CY ← CY ⊻ (saddr.bit)					×
	CY, sfr.bit	3	CY ← CY ⊻ sfr.bit					×
	CY, X.bit	2	CY ← CY ⊻ X.bit					×
	CY, A.bit	2	CY ← CY ⊻ A.bit					×
	CY, PSWL.bit	2	CY ← CY ⊻ PSW_L.bit					×
	CY, PSWH.bit	2	CY ← CY ⊻ PSW_H.bit					×
	CY, !addr16.bit	5	CY ← CY ⊻ !addr16.bit					×
	CY, !!addr24.bit	6	CY ← CY ⊻ !!addr24.bit					×
	CY, mem2.bit	2	CY ← CY ⊻ mem2.bit					×
NOT1	saddr.bit	3/4	(saddr.bit) ← $\overline{\text{(saddr.bit)}}$					
	sfr.bit	3	sfr.bit ← $\overline{\text{sfr.bit}}$					
	X.bit	2	X.bit ← $\overline{\text{X.bit}}$					
	A.bit	2	A.bit ← $\overline{\text{A.bit}}$					
	PSWL.bit	2	PSWL.bit ← $\overline{PSW_L\text{.bit}}$	×	×	×	×	×
	PSWH.bit	2	PSWH.bit ← $\overline{PSW_H\text{.bit}}$					
	!addr16.bit	5	!addr16.bit ← $\overline{\text{!addr16.bit}}$					
	!!addr24.bit	6	!!addr24.bit ← $\overline{\text{!!addr24.bit}}$					
	mem2.bit	2	mem2.bit ← $\overline{\text{mem2.bit}}$					
	CY	1	CY ← $\overline{\text{CY}}$					×

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
SET1	saddr.bit	2/3	(saddr.bit) ← 1					
	sfr.bit	3	sfr.bit ← 1					
	X.bit	2	X.bit ← 1					
	A.bit	2	A.bit ← 1					
	PSWL.bit	2	PSW_L.bit ← 1	×	×	×	×	×
	PSWH.bit	2	PSW_H.bit ← 1					
	!addr16.bit	5	!addr16.bit ← 1					
	!!addr24.bit	6	!!addr24.bit ← 1					
	mem2.bit	2	mem2.bit ← 1					
	CY	1	CY ← 1					1
CLR1	saddr.bit	2/3	(saddr.bit) ← 0					
	sfr.bit	3	sfr.bit ← 0					
	X.bit	2	X.bit ← 0					
	A.bit	2	A.bit ← 0					
	PSWL.bit	2	PSW_L.bit ← 0	×	×	×	×	×
	PSWH.bit	2	PSW_H.bit ← 0					
	!addr16.bit	5	!addr16.bit ← 0					
	!!addr24.bit	6	!!addr24.bit ← 0					
	mem2.bit	2	mem2.bit ← 0					
	CY	1	CY ← 0					0

(15) Stack manipulation instructions: PUSH, PUSHU, POP, POPU, MOVG, ADDWG, SUBWG, INCG, DECG

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
PUSH Note 1	PSW	1	(SP – 2) ← PSW, SP ← SP – 2					
	sfrp	3	(SP – 2) ← sfrp, SP ← SP – 2					
	sfr	3	(SP – 1) ← sfr, SP ← SP – 1					
	post	2	{(SP – 2) ← post, SP ← SP – 2} × m Note 2					
	rg	2	(SP – 3) ← rg, SP ← SP – 3					
PUSHU Note 1	post	2	{(UUP – 2) ← post, UUP ← UUP – 2} × m Note 2					
POP Note 1	PSW	1	PSW ← (SP), SP ← SP + 2	R	R	R	R	R
	sfrp	3	sfrp ← (SP), SP ← SP + 2					
	sfr	3	sfr ← (SP), SP ← SP + 1					
	post	2	{post ← (SP), SP ← SP + 2} × m Note 2					
	rg	2	rg ← (SP), SP ← SP + 3					
POPU Note 1	post	2	{post ← (UUP), UUP ← UUP + 2} × m Note 2					
MOVG	SP, #imm24	5	SP ← imm24					
	SP, WHL	2	SP ← WHL					
	WHL, SP	2	WHL ← SP					
ADDWG	SP, #word	4	SP ← SP + word					
SUBWG	SP, #word	4	SP ← SP – word					
INCG	SP	2	SP ← SP + 1					
DECG	SP	2	SP ← SP – 1					

Notes 1. For details about operation, refer to **CHAPTER 3, Figure 3-4 Data Saved to Stack Area**, and **Figure 3-5 Data Restored from Stack Area.**

2. m = number of registers specified by post

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

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
CALL Note	!addr16	3	(SP – 3) ← (PC + 3), SP ← SP – 3, PC_{HW} ← 0, PC_{LW} ← addr16					
	!!addr20	4	(SP – 3) ← (PC + 4), SP ← SP – 3, PC ← addr20					
	rp	2	(SP – 3) ← (PC + 2), SP ← SP – 3, PC_{HW} ← 0, PC_{LW} ← rp					
	rg	2	(SP – 3) ← (PC + 2), SP ← SP – 3, PC ← rg					
	[rp]	2	(SP – 3) ← (PC + 2), SP ← SP – 3, PC_{HW} ← 0, PC_{LW} ← (rp)					
	[rg]	2	(SP – 3) ← (PC + 2), SP ← SP – 3, PC ← (rg)					
	$!addr20	3	(SP – 3) ← (PC + 3), SP ← SP – 3, PC ← PC + 3 + jdisp16					
CALLF Note	!addr11	2	(SP – 3) ← (PC + 2), SP ← SP – 3 PC_{19-12} ← 0, PC_{11} ← 1, PC_{10-0} ← addr11					
CALLT Note	[addr5]	1	(SP – 3) ← (PC + 1), SP ← SP – 3, PC_{HW} ← 0, PC_{LW} ← (addr5)					
BRK		1	(SP – 2) ← PSW, $(SP – 1)_{0-3}$ ← $(PC + 1)_{HW}$, (SP – 4) ← PC + 1, SP ← SP – 4 PC_{HW} ← 0, PC_{LW} ← (003EH)					
BRKCS	RBn	2	PC_{LW} ↔ RP2, RP3 ← PSW, RBS2 – 0 ← n, RSS ← 0, IE ← 0, $RP3_{8-11}$ ← PC_{HW}, PC_{HW} ← 0					
RET Note		1	PC ← (SP), SP ← SP + 3					
RETI Note		1	PC ← (SP), PSW ← (SP + 2), SP ← SP + 4	R	R	R	R	R
RETB Note		1	PC ← (SP), PSW ← (SP + 2), SP ← SP + 4	R	R	R	R	R
RETCS	!addr16	3	PSW ← RP3, PC_{LW} ← RP2, RP2 ← addr16, PC_{HW} ← $RP3_{8-11}$	R	R	R	R	R
RETCSB	!addr16	4	PSW ← RP3, PC_{LW} ← RP2, RP2 ← addr16, PC_{HW} ← $RP3_{8-11}$	R	R	R	R	R

Note For details about operation, refer to **CHAPTER 3, Figure 3-4 Data Saved to Stack Area**, and **Figure 3-5 Data Restored from Stack Area.**

(17) Unconditional branch instruction: BR

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
BR	!addr16	3	$PC_{HW} \leftarrow 0$, $PC_{LW} \leftarrow$ addr16					
	!!addr20	4	PC ← addr20					
	rp	2	$PC_{HW} \leftarrow 0$, $PC_{LW} \leftarrow$ rp					
	rg	2	PC ← rg					
	[rp]	2	$PC_{HW} \leftarrow 0$, $PC_{LW} \leftarrow$ (rp)					
	[rg]	2	PC ← (rg)					
	$addr20	2	PC ← PC + 2 + jdisp8					
	$!addr20	3	PC ← PC + 3 + jdisp16					

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

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
BNZ	$addr20	2	PC ← PC + 2 + jdisp8 if Z = 0					
BNE								
BZ	$addr20	2	PC ← PC + 2 + jdisp8 if Z = 1					
BE								
BNC	$addr20	2	PC ← PC + 2 + jdisp8 if CY = 0					
BNL								
BC	$addr20	2	PC ← PC + 2 + jdisp8 if CY = 1					
BL								
BNV	$addr20	2	PC ← PC + 2 + jdisp8 if P/V = 0					
BPO								
BV	$addr20	2	PC ← PC + 2 + jdisp8 if P/V = 1					
BPE								
BP	$addr20	2	PC ← PC + 2 + jdisp8 if S = 0					
BN	$addr20	2	PC ← PC + 2 + jdisp8 if S = 1					
BLT	$addr20	3	PC ← PC + 3 + jdisp8 if P/V ∀ S = 1					
BGE	$addr20	3	PC ← PC + 3 + jdisp8 if P/V ∀ S = 0					
BLE	$addr20	3	PC ← PC + 3 + jdisp8 if (P/V ∀ S) ∨ Z = 1					
BGT	$addr20	3	PC ← PC + 3 + jdisp8 if (P/V ∀ S) ∨ Z = 0					
BNH	$addr20	3	PC ← PC + 3 + jdisp8 if Z ∨ CY = 1					
BH	$addr20	3	PC ← PC + 3 + jdisp8 if Z ∨ CY = 0					
BF	saddr.bit, $addr20	4/5	PC ← PC + 4 [Note] + jdisp8 if(saddr.bit) = 0					
	sfr.bit, $addr20	4	PC ← PC + 4 + jdisp8 if sfr.bit = 0					
	X.bit, $addr20	3	PC ← PC + 3 + jdisp8 if X.bit = 0					
	A.bit, $addr20	3	PC ← PC + 3 + jdisp8 if A.bit = 0					
	PSWL.bit, $addr20	3	PC ← PC + 3 + jdisp8 if PSW_L.bit = 0					
	PSWH.bit, $addr20	3	PC ← PC + 3 + jdisp8 if PSW_H.bit = 0					
	!addr16.bit, $addr20	6	PC ← PC + 3 + jdisp8 if !addr16.bit = 0					
	!!addr24.bit, $addr20	7	PC ← PC + 3 + jdisp8 if !!addr24.bit = 0					
	mem2.bit, $addr20	3	PC ← PC + 3 + jdisp8 if mem2.bit = 0					

Note When the number of bytes is 4; when 5, the operation is: PC ← PC + 5 + jdisp8.

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
BT	saddr.bit, $addr20	3/4	PC ← PC + 3 **Note 1** + jdisp8 if(saddr.bit) = 1					
	sfr.bit, $addr20	4	PC ← PC + 4 + jdisp8 if sfr.bit = 1					
	X.bit, $addr20	3	PC ← PC + 3 + jdisp8 if X.bit = 1					
	A.bit, $addr20	3	PC ← PC + 3 + jdisp8 if A.bit = 1					
	PSWL.bit, $addr20	3	PC ← PC + 3 + jdisp8 if PSW_L.bit = 1					
	PSWH.bit, $addr20	3	PC ← PC + 3 + jdisp8 if PSW_H.bit = 1					
	!addr16.bit, $addr20	6	PC ← PC + 3 + jdisp8 if !addr16.bit = 1					
	!!addr24.bit, $addr20	7	PC ← PC + 3 + jdisp8 if !!addr24.bit = 1					
	mem2.bit, $addr20	3	PC ← PC + 3 + jdisp8 if mem2.bit = 1					
BTCLR	saddr.bit, $addr20	4/5	{PC ← PC + 4 **Note 2** + jdisp8, (saddr.bit) ← 0} if(saddr.bit) = 1					
	sfr.bit, $addr20	4	{PC ← PC + 4 + jdisp8, sfr.bit ← 0} if sfr.bit = 1					
	X.bit, $addr20	3	{PC ← PC + 3 + jdisp8, X.bit ← 0} if X.bit = 1					
	A.bit, $addr20	3	{PC ← PC + 3 + jdisp8, A.bit ← 0} if A.bit = 1					
	PSWL.bit, $addr20	3	{PC ← PC + 3 + jdisp8, PSW_L.bit ← 0} if PSW_L.bit = 1	×	×	×	×	×
	PSWH.bit, $addr20	3	{PC ← PC + 3 + jdisp8, PSW_H.bit ← 0} if PSW_H.bit = 1					
	!addr16.bit, $addr20	6	{PC ← PC + 3 + jdisp8, !addr16.bit ← 0} if !addr16 = 1					
	!!addr24.bit, $addr20	7	{PC ← PC + 3 + jdisp8, !!addr24.bit ← 0} if !!addr24 = 1					
	mem2.bit, $addr20	3	{PC ← PC + 3 + jdisp8, mem2.bit ← 0} if mem2. bit = 1					

Notes 1. When the number of bytes is 3; when 4, the operation is: PC ← PC + 4 + jdisp8.

2. When the number of bytes is 4; when 5, the operation is: PC ← PC + 5 + jdisp8.

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
BFSET	saddr.bit, $addr20	4/5	{PC ← PC + 4 [Note 1] + jdisp8, (saddr.bit) ← 1} if(saddr.bit) = 0					
	sfr.bit, $addr20	4	{PC ← PC + 4 + jdisp8, sfr.bit ← 1} if sfr.bit = 0					
	X.bit, $addr20	3	{PC ← PC + 3 + jdisp8, X.bit ← 1} if X.bit = 0					
	A.bit, $addr20	3	{PC ← PC + 3 + jdisp8, A.bit ← 1} if A.bit = 0					
	PSWL.bit, $addr20	3	{PC ← PC + 3 + jdisp8, PSW_L.bit ← 1} if PSW_L.bit = 0	×	×	×	×	×
	PSWH.bit, $addr20	3	{PC ← PC + 3 + jdisp8, PSW_H.bit ← 1} if PSW_H.bit = 0					
	!addr16.bit, $addr20	6	{PC ← PC + 3 + jdisp8, !addr16.bit ← 1} if !addr16 = 0					
	!!addr24.bit, $addr20	7	{PC ← PC + 3 + jdisp8, !!addr24.bit ← 1} if !!addr24 = 0					
	mem2.bit, $addr20	3	{PC ← PC + 3 + jdisp8, mem2.bit ← 1} if mem2.bit = 0					
DBNZ	B, $addr20	2	B ← B – 1, PC ← PC + 2 + jdisp8 if B ≠ 0					
	C, $addr20	2	C ← C – 1, PC ← PC + 2 + jdisp8 if C ≠ 0					
	saddr, $addr20	3/4	(saddr) ← (saddr) – 1, PC ← PC + 3 [Note 2] + jdisp8 if (saddr) ≠ 0					

Notes 1. When the number of bytes is 4; when 5, the operation is: PC ← PC + 5 + jdisp8.

2. When the number of bytes is 3; when 4, the operation is: PC ← PC + 4 + jdisp8.

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

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
MOV	STBC, #byte	4	STBC ← byte					
	WDM, #byte	4	WDM ← byte					
LOCATION	locaddr	4	SFR, internal data area location address high-order word specification					
SEL	RBn	2	RSS ← 0, RBS2 – 0 ← n					
	RBn, ALT	2	RSS ← 1, RBS2 – 0 ← n					
SWRS		2	RSS ← $\overline{\text{RSS}}$					
NOP		1	No Operation					
EI		1	IE ← 1(Enable interrupt)					
DI		1	IE ← 0(Disable interrupt)					

(20) Special instructions: CHKL, CHKLA

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
CHKL	sfr	3	(pin level) $\forall$ (output latch)	×	×		P	
CHKLA	sfr	3	A ← (pin level) $\forall$ (output latch)	×	×		P	

★ **Caution The CHKL and CHKLA instructions are not available in the μPD784216, 784216Y, 784218, 784218Y, 784225, 784225Y, 784937 Subseries. Do not execute these instructions. If these instructions are executed, the following operations will result.**

- **CHKL instruction .... After the pin levels of the output pins are read two times, they are exclusive-ORed. As a result, if the pins checked with this instruction are used in the port output mode, the exclusive-OR result is always 0 for all bits, and the Z flag is set to (1).**
- **CHKLA instruction .... After the pin levels of output pins are read two times, they are exclusive-ORed. As a result, if the pins checked with this instruction are used in the port output mode, the exclusive-OR result is always 0 for all bits, and the Z flag is set to (1) along with that the result is stored in the A register.**

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

Mnemonic	Operands	Bytes	Operation	Flags				
				S	Z	AC	P/V	CY
MOVTBLW	!addr8, byte	4	(addr8 + 2) ← (addr8), byte ← byte – 1, addr8 ← addr8 – 2 End if byte = 0					
MOVM	[TDE +], A	2	(TDE) ← A, TDE ← TDE + 1, C ← C – 1 End if C = 0					
	[TDE –], A	2	(TDE) ← A, TDE ← TDE – 1, C ← C – 1 End if C = 0					
XCHM	[TDE +], A	2	(TDE) ↔ A, TDE ← TDE + 1, C ← C – 1 End if C = 0					
	[TDE –], A	2	(TDE) ↔ A, TDE ← TDE – 1, C ← C – 1 End if C = 0					
MOVBK	[TDE +], [WHL +]	2	(TDE) ← (WHL), TDE ← TDE + 1, WHL ← WHL + 1, C ← C – 1 End if C = 0					
	[TDE –], [WHL –]	2	(TDE) ← (WHL), TDE ← TDE – 1, WHL ← WHL – 1, C ← C – 1 End if C = 0					
XCHBK	[TDE +], [WHL +]	2	(TDE) ↔ (WHL), TDE ← TDE + 1, WHL ← WHL + 1, C ← C – 1 End if C = 0					
	[TDE –], [WHL –]	2	(TDE) ↔ (WHL), TDE ← TDE – 1, WHL ← WHL – 1, C ← C – 1 End if C = 0					
CMPME	[TDE +], A	2	(TDE) – A, TDE ← TDE + 1, C ← C – 1 End if C = 0 or Z = 0	×	×	×	V	×
	[TDE –], A	2	(TDE) – A, TDE ← TDE – 1, C ← C – 1 End if C = 0 or Z = 0	×	×	×	V	×
CMPMNE	[TDE +], A	2	(TDE) – A, TDE ← TDE + 1, C ← C – 1 End if C = 0 or Z = 1	×	×	×	V	×
	[TDE –], A	2	(TDE) – A, TDE ← TDE – 1, C ← C – 1 End if C = 0 or Z = 1	×	×	×	V	×
CMPMC	[TDE +], A	2	(TDE) – A, TDE ← TDE + 1, C ← C – 1 End if C = 0 or CY = 0	×	×	×	V	×
	[TDE –], A	2	(TDE) – A, TDE ← TDE – 1, C ← C – 1 End if C = 0 or CY = 0	×	×	×	V	×
CMPMNC	[TDE +], A	2	(TDE) – A, TDE ← TDE + 1, C ← C – 1 End if C = 0 or CY = 1	×	×	×	V	×
	[TDE –], A	2	(TDE) – A, TDE ← TDE – 1, C ← C – 1 End if C = 0 or CY = 1	×	×	×	V	×
CMPBKE	[TDE +], [WHL +]	2	(TDE) – (WHL), TDE ← TDE + 1, WHL ← WHL + 1, C ← C – 1 End if C = 0 or Z = 0	×	×	×	V	×
	[TDE –], [WHL –]	2	(TDE) – (WHL), TDE ← TDE – 1, WHL ← WHL – 1, C ← C – 1 End if C = 0 or Z = 0	×	×	×	V	×
CMPBKNE	[TDE +], [WHL +]	2	(TDE) – (WHL), TDE ← TDE + 1, WHL ← WHL + 1, C ← C – 1 End if C = 0 or Z = 1	×	×	×	V	×
	[TDE –], [WHL –]	2	(TDE) – (WHL), TDE ← TDE – 1, WHL ← WHL – 1, C ← C – 1 End if C = 0 or Z = 1	×	×	×	V	×
CMPBKC	[TDE +], [WHL +]	2	(TDE) – (WHL), TDE ← TDE + 1, WHL ← WHL + 1, C ← C – 1 End if C = 0 or CY = 0	×	×	×	V	×
	[TDE –], [WHL –]	2	(TDE) – (WHL), TDE ← TDE – 1, WHL ← WHL – 1, C ← C – 1 End if C = 0 or CY = 0	×	×	×	V	×
CMPBKNC	[TDE +], [WHL +]	2	(TDE) – (WHL), TDE ← TDE + 1, WHL ← WHL + 1, C ← C – 1 End if C = 0 or CY = 1	×	×	×	V	×
	[TDE –], [WHL –]	2	(TDE) – (WHL), TDE ← TDE – 1, WHL ← WHL – 1, C ← C – 1 End if C = 0 or CY = 1	×	×	×	V	×

6.3 Instructions Listed by Type of Addressing

(1) 8-bit instructions (combinations expressed by writing A for r are shown in parentheses)

MOV, XCH, ADD, ADDC, SUB, SUBC, AND, OR, XOR, CMP, MULU, DIVUW, INC, DEC, ROR, ROL, RORC, ROLC, SHR, SHL, ROR4, ROL4, DBNZ, PUSH, POP, MOVM, XCHM, CMPME, CMPMNE, CMPMNC, CMPMC, MOVBK, XCHBK, CMPBKE, CMPBKNE, CMPBKNC, CMPBKC, CHKL, CHKLA

Table 6-1. List of Instructions by 8-Bit Addressing

1st Operand \ 2nd Operand	#byte	A	r r'	saddr saddr'	sfr	!addr16 !!addr24	mem [saddrp] [%saddrg]	r3 PSWL PSWH	[WHL +] [WHL –]	n	No Note 2
A	(MOV) ADD Note 1	(MOV) (XCH) (ADD) Note 1	MOV XCH (ADD) Note 1	(MOV) Note 6 (XCH) Note 6 (ADD) Notes 1, 6	MOV (XCH) (ADD) Note 1	(MOV) (XCH) ADD Note 1	MOV XCH ADD Note 1	MOV	(MOV) (XCH) (ADD) Note 1		
r	MOV ADD Note 1	(MOV) (XCH) (ADD) Note 1	MOV XCH ADD Note 1	MOV XCH ADD Note 1	MOV XCH ADD Note 1	MOV XCH				ROR Note 3	MULU DIVUW INC DEC
saddr	MOV ADD Note 1	(MOV) Note 6 (ADD) Note 1	MOV ADD Note 1	MOV XCH ADD Note 1							INC DEC DBNZ
sfr	MOV ADD Note 1	MOV (ADD) Note 1	MOV ADD Note 1								PUSH POP CHKL CHKLA
!addr16 !!addr24	MOV	(MOV) ADD Note 1	MOV								
mem [saddrp] [%saddrg]		MOV ADD Note 1									
mem3											ROR4 ROL4
r3 PSWL PSWH	MOV	MOV									
B, C											DBNZ
STBC, WDM	MOV										
[TDE +] [TDE –]		(MOV) (ADD) Note 1 MOVM Note 4							MOVBK Note 5		

Notes 1. ADDC, SUB, SUBC, AND, OR, XOR, and CMP are equivalent to ADD.
2. There is no 2nd operand, or the 2nd operand is not an operand address.
3. ROL, RORC, ROLC, SHR, and SHL are equivalent to ROR.
4. XCHM, CMPME, CMPMNE, CMPMNC, and CMPMC are equivalent to MOVM.
5. XCHBK, CMPBKE, CMPBKNE, CMPBKNC, and CMPBKC are equvalent to MOVBK.
6. When saddr is saddr2 in this combination, a short code length instruction can be used.

(2) 16-bit instructions (combinations expressed by writing AX for rp are shown in parentheses)
MOVW, XCHW, ADDW, SUBW, CMPW, MULUW, MULW, DIVUX, INCW, DECW, SHRW, SHLW, PUSH, POP, ADDWG, SUBWG, PUSHU, POPU, MOVTBLW, MACW, MACSW, SACW

Table 6-2. List of Instructions by 16-Bit Addressing

2nd Operand / 1st Operand	#word	AX	rp rp'	saddrp saddrp'	sfrp	!addr16 !!addr24	mem [saddrp] [%saddrg]	[WHL +]	byte	n	No Note 2
AX	(MOVW) ADDW Note 1	(MOVW) (XCHW) (ADD) Note 1	(MOVW) (XCHW) (ADDW) Note 1	(MOVW) Note 3 (XCHW) Note 3 (ADDW) Notes 1, 3	MOVW (XCHW) (ADDW) Note 1	(MOVW) XCHW	MOVW XCHW	(MOVW) (XCHW)			
rp	MOVW ADDW Note 1	(MOVW) (XCHW) (ADDW) Note 1	MOVW XCHW ADDW Note 1	MOVW XCHW ADDW Note 1	MOVW XCHW ADDW Note 1	MOVW				SHRW SHLW	MULW Note 4 INCW DECW
saddrp	MOVW ADDW Note 1	(MOVW) Note 3 (ADDW) Note 1	MOVW ADDW Note 1	MOVW XCHW ADDW Note 1							INCW DECW
sfrp	MOVW ADDW Note 1	MOVW (ADDW) Note 1	MOVW ADDW Note 1								PUSH POP
!addr16 !!addr24	MOVW	(MOVW)	MOVW						MOVTBLW		
mem [saddrp] [%saddrg]		MOVW									
PSW											PUSH POP
SP	ADDWG SUBWG										
post											PUSH POP PUSHU POPU
[TDE +]		(MOVW)						SACW			
byte											MACW MACSW

Notes 1. SUBW and CMPW are equivalent to ADDW.
2. There is no 2nd operand, or the 2nd operand is not an operand address.
3. When saddrp is saddrp2 in this combination, a short code length instruction can be used.
4. MULUW and DIVUX are equivalent to MULW.

(3) 24-bit instructions (combinations expressed by writing WHL for rg are shown in parentheses)
MOVG, ADDG, SUBG, INCG, DECG, PUSH, POP

Table 6-3. List of Instructions by 24-Bit Addressing

2nd Operand / 1st Operand	#imm24	WHL	rg rg'	saddrg	!!addr24	mem1	[%saddrg]	SP	No Note
WHL	(MOVG) (ADDG) (SUBG)	(MOVG) (ADDG) (SUBG)	(MOVG) (ADDG) (SUBG)	(MOVG) ADDG SUBG	(MOVG)	MOVG	MOVG	MOVG	
rg	MOVG ADDG SUBG	(MOVG) (ADDG) (SUBG)	MOVG ADDG SUBG	MOVG	MOVG				INCG DECG PUSH POP
saddrg		(MOVG)	MOVG						
!!addr24		(MOVG)	MOVG						
mem1		MOVG							
[%saddrg]		MOVG							
SP	MOVG	MOVG							INCG DECG

Note There is no 2nd operand, or the 2nd operand is not an operand address.

(4) Bit manipulation instructions

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

Table 6-4. List of Instructions by Bit Manipulation Instruction Addressing

2nd Operand / 1st Operand	CY	saddr.bit sfr.bit A.bit X.bit PSWL.bit PSWH.bit mem2.bit !addr16.bit !!addr24.bit	/saddr.bit /sfr.bit /A.bit /X.bit /PSWL.bit /PSWH.bit /mem2.bit /!addr16.bit /!!addr24.bit	No Note
CY		MOV1 AND1 OR1 XOR1	AND1 OR1	NOT1 SET1 CLR1
saddr.bit sfr.bit A.bit X.bit PSWL.bit PSWH. bit mem2.bit !addr16.bit !!addr24.bit	MOV1			NOT1 SET1 CLR1 BF BT BTCLR BFSET

Note There is no 2nd operand, or the 2nd operand is not an operand address.

(5) Call/return instructions/branch instructions

CALL, CALLF, CALLT, BRK, RET, RETI, RETB, RETCS, RETCSB, BRKCS, BR, BNZ, BNE, BZ, BE, BNC, BNL, BC, BL, BNV, BPO, BV, BPE, BP, BN, BLT, BGE, BLE, BGT, BNH, BH, BF, BT, BTCLR, BFSET, DBNZ

Table 6-5. List of Instructions by Call/Return Instruction/Branch Instruction Addressing

Instruction Address Operand	$addr20	$!addr20	!addr16	!!addr20	rp	rg	[rp]	[rg]	!addr11	[addr5]	RBn	No
Basic instructions	BC Note BR	CALL BR	CALL BR RETCS RETCSB	CALL BR	CALL BR	CALL BR	CALL BR	CALL BR	CALLF	CALLT	BRKCS	BRK RET RETI RETB
Compound instructions	BF BT BTCLR BFSET DBNZ											

Note BNZ, BNE, BZ, BE, BNC, BNL, BL, BNV, BPO, BV, BPE, BP, BN, BLT, BGE, BLE, BGT, BNH, and BH are equivalent to BC.

(6) Other instructions

ADJBA, ADJBS, CVTBW, LOCATION, SEL, NOT, EI, DI, SWRS

6.4 Operation Codes

6.4.1 Operation code symbols

(1) r1

R_2	R_1	R_0	r1
0	0	0	R0
0	0	1	R1
0	1	0	R2
0	1	1	R3
1	0	0	R4
1	0	1	R5
1	1	0	R6
1	1	1	R7

(2) r2

R_2	R_1	R_0	r2
0	0	0	R8
0	0	1	R9
0	1	0	R10
0	1	1	R11
1	0	0	R12
1	0	1	R13
1	1	0	R14
1	1	1	R15

(3) r, r'

R_3	R_2	R_1	R_0	r
R_7	R_6	R_5	R_4	r'
0	0	0	0	R0
0	0	0	1	R1
0	0	1	0	R2
0	0	1	1	R3
0	1	0	0	R4
0	1	0	1	R5
0	1	1	0	R6
0	1	1	1	R7
1	0	0	0	R8
1	0	0	1	R9
1	0	1	0	R10
1	0	1	1	R11
1	1	0	0	R12
1	1	0	1	R13
1	1	1	0	R14
1	1	1	1	R15

(4) rp

P_7	P_6	P_5	rp
0	0	0	RP0
0	0	1	RP1
0	1	0	RP2
0	1	1	RP3
1	0	0	RP4
1	0	1	RP5
1	1	0	RP6
1	1	1	RP7

(5) rp, rp'

P_2 P_1 P_0	rp rp'
0 0 0	RP0
0 0 1	RP4
0 1 0	RP1
0 1 1	RP5
1 0 0	RP2
1 0 1	RP6
1 1 0	RP3
1 1 1	RP7

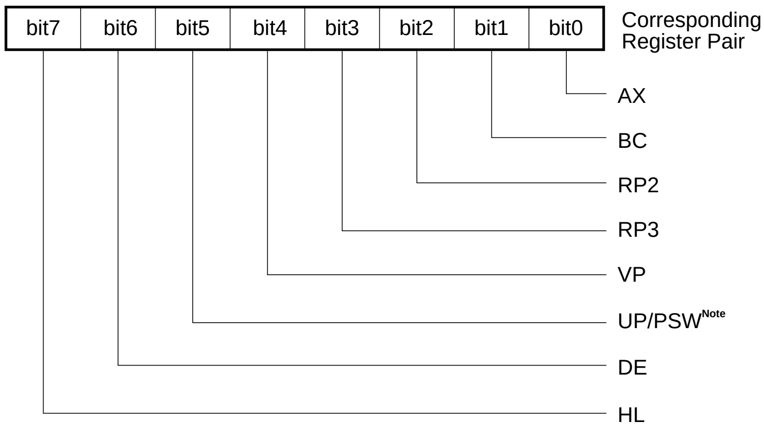
(6) rg, rg'

G_6 G_5	rg
G_2 G_1	rg'
0 0	RG4
0 1	RG5
1 0	RG6
1 1	RG7

(7) mem3

P_2 P_1 P_0	mem3
0 0 0	[RP0]
0 0 1	[RG4]
0 1 0	[RP1]
0 1 1	[RG5]
1 0 0	[RP2]
1 0 1	[RG6]
1 1 0	[RP3]
1 1 1	[RG7]

(8) post byte



0	Save/restore operation not performed on stack memory
1	Save/restore operation performed on stack memory

Note UP in the case of a PUSH/POP instruction, PSW in the case of a PUSHU/POPU instruction.

(9) locaddr

locaddr	locaddrl	locaddrh
0	FEH	01H
0FH	FFH	00H

6.4.2 List of operation codes

(1) 8-bit data transfer instruction: MOV

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
MOV	r1, #byte	1 0 1 1 1 $R_2R_1R_0$	← #byte →	
	r2, #byte	0 0 1 1 1 1 0 0	1 0 1 1 1 $R_2R_1R_0$	← #byte →
	saddr2, #byte	0 0 1 1 1 0 1 0	← Saddr2-offset →	← #byte →
	saddr1, #byte	0 0 1 1 1 1 0 0	0 0 1 1 1 0 1 0	← Saddr1-offset →
		← #byte →		
	sfr, #byte	0 0 1 0 1 0 1 1	← Sfr-offset →	← #byte →
	!addr16, #byte	0 0 0 0 1 0 0 1	0 1 0 0 0 0 0 0	← Low Address →
		← High Address →	← #byte →	
	!!addr24, #byte	0 0 0 0 1 0 0 1	0 1 0 1 0 0 0 0	← High-w Address →
		← Low Address →	← High Address →	← #byte →
	r, r1	0 0 1 0 0 1 0 0	$R_7R_6R_5R_4$ 0 $R_2R_1R_0$	
	r, r2	0 0 1 1 1 1 0 0	0 0 1 0 0 1 0 0	$R_7R_6R_5R_4$ 0 $R_2R_1R_0$
	A, r1	1 1 0 1 0 $R_2R_1R_0$		
	A, r2	0 0 1 1 1 1 0 0	1 1 0 1 0 $R_2R_1R_0$	
	A, saddr2	0 0 1 0 0 0 0 0	← Saddr2-offset →	
	r, saddr2	0 0 1 1 1 0 0 0	$R_3R_2R_1R_0$ 0 0 0 0	← Saddr2-offset →
	r, saddr1	0 0 1 1 1 0 0 0	$R_3R_2R_1R_0$ 0 0 0 1	← Saddr1-offset →
	saddr2, A	0 0 1 0 0 0 1 0	← Saddr2-offset →	
	saddr2, r	0 0 1 1 1 0 0 0	$R_3R_2R_1R_0$ 0 1 0 0	← Saddr2-offset →
	saddr1, r	0 0 1 1 1 0 0 0	$R_3R_2R_1R_0$ 0 1 0 1	← Saddr1-offset →
	A, sfr	0 0 0 1 0 0 0 0	← Sfr-offset →	
	r, sfr	0 0 1 1 1 0 0 0	$R_3R_2R_1R_0$ 0 0 1 0	← Sfr-offset →
	sfr, A	0 0 0 1 0 0 1 0	← Sfr-offset →	
	sfr, r	0 0 1 1 1 0 0 0	$R_3R_2R_1R_0$ 0 1 1 0	← Sfr-offset →
	saddr2, saddr2'	0 0 1 0 1 0 1 0	0 0 0 0 0 0 0 0	← Saddr2'-offset →
		← Saddr2-offset →		
	saddr2, saddr1	0 0 1 0 1 0 1 0	0 0 0 1 0 0 0 0	← Saddr1-offset →
		← Saddr2-offset →		
	saddr1, saddr2	0 0 1 0 1 0 1 0	0 0 1 0 0 0 0 0	← Saddr2-offset →
		← Saddr1-offset →		

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
MOV	saddr1, saddr1’	0 0 1 0 1 0 1 0	0 0 1 1 0 0 0 0	← Saddr1’-offset →
		← Saddr1-offset →		
	r, !addr16	0 0 1 1 1 1 1 0	$R_3 R_2 R_1 R_0$ 0 0 0 0	← Low Address →
		← High Address →		
	!addr16, r	0 0 1 1 1 1 1 0	$R_3 R_2 R_1 R_0$ 0 0 0 1	← Low Address →
		← High Address →		
	r, !!addr24	0 0 1 1 1 1 1 0	$R_3 R_2 R_1 R_0$ 0 0 1 0	← High-w Address →
		← Low Address →	← High Address →	
	!!addr24, r	0 0 1 1 1 1 1 0	$R_3 R_2 R_1 R_0$ 0 0 1 1	← High-w Address →
		← Low Address →	← High Address →	
	A, [saddrp2]	0 0 0 1 1 0 0 0	← Saddr2-offset →	
	A, [saddrp1]	0 0 1 1 1 1 0 0	0 0 0 1 1 0 0 0	← Saddr1-offset →
	A, [%saddrg2]	0 0 0 0 0 1 1 1	0 0 1 1 0 0 0 0	← Saddr2-offset →
	A, [%saddrg1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 1 0 0 0 0
		← Saddr1-offset →		
	A, [TDE +]	0 1 0 1 1 0 0 0		
	A, [WHL +]	0 1 0 1 1 0 0 1		
	A, [TDE –]	0 1 0 1 1 0 1 0		
	A, [WHL –]	0 1 0 1 1 0 1 1		
	A, [TDE]	0 1 0 1 1 1 0 0		
	A, [WHL]	0 1 0 1 1 1 0 1		
	A, [VVP]	0 0 0 1 0 1 1 0	0 1 1 0 0 0 0 0	
	A, [UUP]	0 0 0 1 0 1 1 0	0 1 1 1 0 0 0 0	
	A, [TDE + byte]	0 0 0 0 0 1 1 0	0 0 0 0 0 0 0 0	← Low Offset →
	A, [SP + byte]	0 0 0 0 0 1 1 0	0 0 0 1 0 0 0 0	← Low Offset →
	A, [WHL + byte]	0 0 0 0 0 1 1 0	0 0 1 0 0 0 0 0	← Low Offset →
	A, [UUP + byte]	0 0 0 0 0 1 1 0	0 0 1 1 0 0 0 0	← Low Offset →
	A, [VVP + byte]	0 0 0 0 0 1 1 0	0 1 0 0 0 0 0 0	← Low Offset →
	A, imm24 [DE]	0 0 0 0 1 0 1 0	0 0 0 0 0 0 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [A]	0 0 0 0 1 0 1 0	0 0 0 1 0 0 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [HL]	0 0 0 0 1 0 1 0	0 0 1 0 0 0 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
MOV	A, imm24 [B]	0 0 0 0 1 0 1 0	0 0 1 1 0 0 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, [TDE + A]	0 0 0 1 0 1 1 1	0 0 0 0 0 0 0 0	
	A, [WHL + A]	0 0 0 1 0 1 1 1	0 0 0 1 0 0 0 0	
	A, [TDE + B]	0 0 0 1 0 1 1 1	0 0 1 0 0 0 0 0	
	A, [WHL + B]	0 0 0 1 0 1 1 1	0 0 1 1 0 0 0 0	
	A, [VVP + DE]	0 0 0 1 0 1 1 1	0 1 0 0 0 0 0 0	
	A, [VVP + HL]	0 0 0 1 0 1 1 1	0 1 0 1 0 0 0 0	
	A, [TDE + C]	0 0 0 1 0 1 1 1	0 1 1 0 0 0 0 0	
	A, [WHL + C]	0 0 0 1 0 1 1 1	0 1 1 1 0 0 0 0	
	[saddrp2], A	0 0 0 1 1 0 0 1	← Saddr2-offset →	
	[saddrp1], A	0 0 1 1 1 1 0 0	0 0 0 1 1 0 0 1	← Saddr1-offset →
	[%saddrg2], A	0 0 0 0 0 1 1 1	1 0 1 1 0 0 0 0	← Saddr2-offset →
	[%saddrg1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 1 0 0 0 0
		← Saddr1-offset →		
	[TDE +], A	0 1 0 1 0 0 0 0		
	[WHL +], A	0 1 0 1 0 0 0 1		
	[TDE –], A	0 1 0 1 0 0 1 0		
	[WHL –], A	0 1 0 1 0 0 1 1		
	[TDE], A	0 1 0 1 0 1 0 0		
	[WHL], A	0 1 0 1 0 1 0 1		
	[VVP], A	0 0 0 1 0 1 1 0	1 1 1 0 0 0 0 0	
	[UUP], A	0 0 0 1 0 1 1 0	1 1 1 1 0 0 0 0	
	[TDE + byte], A	0 0 0 0 0 1 1 0	1 0 0 0 0 0 0 0	← Low Offset →
	[SP + byte], A	0 0 0 0 0 1 1 0	1 0 0 1 0 0 0 0	← Low Offset →
	[WHL + byte], A	0 0 0 0 0 1 1 0	1 0 1 0 0 0 0 0	← Low Offset →
	[UUP + byte], A	0 0 0 0 0 1 1 0	1 0 1 1 0 0 0 0	← Low Offset →
	[VVP + byte], A	0 0 0 0 0 1 1 0	1 1 0 0 0 0 0 0	← Low Offset →
	imm24 [DE], A	0 0 0 0 1 0 1 0	1 0 0 0 0 0 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [A], A	0 0 0 0 1 0 1 0	1 0 0 1 0 0 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [HL], A	0 0 0 0 1 0 1 0	1 0 1 0 0 0 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
MOV	imm24 [B], A	0 0 0 0 1 0 1 0	1 0 1 1 0 0 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	[TDE + A], A	0 0 0 1 0 1 1 1	1 0 0 0 0 0 0 0	
	[WHL + A], A	0 0 0 1 0 1 1 1	1 0 0 1 0 0 0 0	
	[TDE + B], A	0 0 0 1 0 1 1 1	1 0 1 0 0 0 0 0	
	[WHL + B], A	0 0 0 1 0 1 1 1	1 0 1 1 0 0 0 0	
	[VVP + DE], A	0 0 0 1 0 1 1 1	1 1 0 0 0 0 0 0	
	[VVP + HL], A	0 0 0 1 0 1 1 1	1 1 0 1 0 0 0 0	
	[TDE + C], A	0 0 0 1 0 1 1 1	1 1 1 0 0 0 0 0	
	[WHL + C], A	0 0 0 1 0 1 1 1	1 1 1 1 0 0 0 0	
	PSWL, #byte	0 0 1 0 1 0 1 1	1 1 1 1 1 1 1 0	← #byte →
	PSWH, #byte	0 0 1 0 1 0 1 1	1 1 1 1 1 1 1 1	← #byte →
	PSWL, A	0 0 0 1 0 0 1 0	1 1 1 1 1 1 1 0	
	PSWH, A	0 0 0 1 0 0 1 0	1 1 1 1 1 1 1 1	
	A, PSWL	0 0 0 1 0 0 0 0	1 1 1 1 1 1 1 0	
	A, PSWH	0 0 0 1 0 0 0 0	1 1 1 1 1 1 1 1	
	V, #byte	0 0 0 0 0 1 1 1	0 1 1 0 0 0 0 1	← #byte →
	U, #byte	0 0 0 0 0 1 1 1	0 1 1 0 0 0 1 1	← #byte →
	T, #byte	0 0 0 0 0 1 1 1	0 1 1 0 0 1 0 1	← #byte →
	W, #byte	0 0 0 0 0 1 1 1	0 1 1 0 0 1 1 1	← #byte →
	A, V	0 0 0 0 0 1 0 1	1 1 0 0 0 0 0 1	
	A, U	0 0 0 0 0 1 0 1	1 1 0 0 0 0 1 1	
	A, T	0 0 0 0 0 1 0 1	1 1 0 0 0 1 0 1	
	A, W	0 0 0 0 0 1 0 1	1 1 0 0 0 1 1 1	
	V, A	0 0 0 0 0 1 0 1	1 1 0 0 1 0 0 1	
	U, A	0 0 0 0 0 1 0 1	1 1 0 0 1 0 1 1	
	T, A	0 0 0 0 0 1 0 1	1 1 0 0 1 1 0 1	
	W, A	0 0 0 0 0 1 0 1	1 1 0 0 1 1 1 1	

(2) 16-bit data transfer instruction: MOVW

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
MOVW	rp, #word	0 1 1 0 0 P_2 P_1 P_0	← Low Byte →	← High Byte →
	saddrp2, #word	0 0 0 0 1 1 0 0	← Saddr2-offset →	← Low Byte →
		← High Byte →		
	saddrp1, #word	0 0 1 1 1 1 0 0	0 0 0 0 1 1 0 0	← Saddr1-offset →
		← Low Byte →	← High Byte →	
	sfrp, #word	0 0 0 0 1 0 1 1	← Sfr-offset →	← Low Byte →
		← High Byte →		
	!addr16, #word	0 0 0 0 1 0 0 1	0 1 0 0 0 0 0 1	← Low Address →
		← High Address →	← Low Byte →	← High Byte →
	!!addr24, #word	0 0 0 0 1 0 0 1	0 1 0 1 0 0 0 1	← High-w Address →
		← Low Address →	← High Address →	← Low Byte →
		← High Byte →		
	rp, rp'	0 0 1 0 0 1 0 0	P_7 P_6 P_5 0 1 P_2 P_1 P_0	
	AX, saddrp2	0 0 0 1 1 1 0 0	← Saddr2-offset →	
	rp, saddrp2	0 0 1 1 1 0 0 0	P_7 P_6 P_5 0 1 0 0 0	← Saddr2-offset →
	rp, saddrp1	0 0 1 1 1 0 0 0	P_7 P_6 P_5 0 1 0 0 1	← Saddr1-offset →
	saddrp2, AX	0 0 0 1 1 0 1 0	← Saddr2-offset →	
	saddrp2, rp	0 0 1 1 1 0 0 0	P_7 P_6 P_5 0 1 1 0 0	← Saddr2-offset →
	saddrp1, rp	0 0 1 1 1 0 0 0	P_7 P_6 P_5 0 1 1 0 1	← Saddr1-offset →
	AX, sfrp	0 0 0 1 0 0 0 1	← Sfr-offset →	
	rp, sfrp	0 0 1 1 1 0 0 0	P_7 P_6 P_5 0 1 0 1 0	← Sfr-offset →
	sfrp, AX	0 0 0 1 0 0 1 1	← Sfr-offset →	
	sfrp, rp	0 0 1 1 1 0 0 0	P_7 P_6 P_5 0 1 1 1 0	← Sfr-offset →
	saddrp2, saddrp2'	0 0 1 0 1 0 1 0	1 0 0 0 0 0 0 0	← Saddr2'-offset →
		← Saddr2-offset →		
	saddrp2, saddrp1	0 0 1 0 1 0 1 0	1 0 0 1 0 0 0 0	← Saddr1-offset →
		← Saddr2-offset →		
	saddrp1, saddrp2	0 0 1 0 1 0 1 0	1 0 1 0 0 0 0 0	← Saddr2-offset →
		← Saddr1-offset →		
	saddrp1, saddrp1'	0 0 1 0 1 0 1 0	1 0 1 1 0 0 0 0	← Saddr1'-offset →
		← Saddr1-offset →		
	rp, !addr16	0 0 1 1 1 1 1 0	P_7 P_6 P_5 0 1 0 0 0	← Low Address →
		← High Address →		

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
MOVW	!addr16, rp	0 0 1 1 1 1 1 0	$P_7 P_6 P_5$ 0 1 0 0 1	← Low Address →
		← High Address →		
	rp, !!addr24	0 0 1 1 1 1 1 0	$P_7 P_6 P_5$ 0 1 0 1 0	← High-w Address →
		← Low Address →	← High Address →	
	!!addr24, rp	0 0 1 1 1 1 1 0	$P_7 P_6 P_5$ 0 1 0 1 1	← High-w Address →
		← Low Address →	← High Address →	
	AX, [saddrp2]	0 0 0 0 0 1 1 1	0 0 1 0 0 0 0 1	← Saddr2-offset →
	AX, [saddrp1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 0 0 0 0 1
		← Saddr1-offset →		
	AX, [%saddrg2]	0 0 0 0 0 1 1 1	0 0 1 1 0 0 0 1	← Saddr2-offset →
	AX, [%saddrg1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 1 0 0 0 1
		← Saddr1-offset →		
	AX, [TDE +]	0 0 0 1 0 1 1 0	0 0 0 0 0 0 0 1	
	AX, [WHL +]	0 0 0 1 0 1 1 0	0 0 0 1 0 0 0 1	
	AX, [TDE –]	0 0 0 1 0 1 1 0	0 0 1 0 0 0 0 1	
	AX, [WHL –]	0 0 0 1 0 1 1 0	0 0 1 1 0 0 0 1	
	AX, [TDE]	0 0 0 1 0 1 1 0	0 1 0 0 0 0 0 1	
	AX, [WHL]	0 0 0 1 0 1 1 0	0 1 0 1 0 0 0 1	
	AX, [VVP]	0 0 0 1 0 1 1 0	0 1 1 0 0 0 0 1	
	AX, [UUP]	0 0 0 1 0 1 1 0	0 1 1 1 0 0 0 1	
	AX, [TDE + byte]	0 0 0 0 0 1 1 0	0 0 0 0 0 0 0 1	← Low Offset →
	AX, [SP + byte]	0 0 0 0 0 1 1 0	0 0 0 1 0 0 0 1	← Low Offset →
	AX, [WHL + byte]	0 0 0 0 0 1 1 0	0 0 1 0 0 0 0 1	← Low Offset →
	AX, [UUP + byte]	0 0 0 0 0 1 1 0	0 0 1 1 0 0 0 1	← Low Offset →
	AX, [VVP + byte]	0 0 0 0 0 1 1 0	0 1 0 0 0 0 0 1	← Low Offset →
	AX, imm24 [DE]	0 0 0 0 1 0 1 0	0 0 0 0 0 0 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	AX, imm24 [A]	0 0 0 0 1 0 1 0	0 0 0 1 0 0 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	AX, imm24 [HL]	0 0 0 0 1 0 1 0	0 0 1 0 0 0 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	AX, imm24 [B]	0 0 0 0 1 0 1 0	0 0 1 1 0 0 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	AX, [TDE + A]	0 0 0 1 0 1 1 1	0 0 0 0 0 0 0 1	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
MOVW	AX, [WHL + A]	0 0 0 1 0 1 1 1	0 0 0 1 0 0 0 1	
	AX, [TDE + B]	0 0 0 1 0 1 1 1	0 0 1 0 0 0 0 1	
	AX, [WHL + B]	0 0 0 1 0 1 1 1	0 0 1 1 0 0 0 1	
	AX, [VVP + DE]	0 0 0 1 0 1 1 1	0 1 0 0 0 0 0 1	
	AX, [VVP + HL]	0 0 0 1 0 1 1 1	0 1 0 1 0 0 0 1	
	AX, [TDE + C]	0 0 0 1 0 1 1 1	0 1 1 0 0 0 0 1	
	AX, [WHL + C]	0 0 0 1 0 1 1 1	0 1 1 1 0 0 0 1	
	[saddrp2], AX	0 0 0 0 0 1 1 1	1 0 1 0 0 0 0 1	← Saddr2-offset →
	[saddrp1], AX	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 0 0 0 0 1
		← Saddr1-offset →		
	[%saddrg2], AX	0 0 0 0 0 1 1 1	1 0 1 1 0 0 0 1	← Saddr2-offset →
	[%saddrg1], AX	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 1 0 0 0 1
		← Saddr1-offset →		
	[TDE +], AX	0 0 0 1 0 1 1 0	1 0 0 0 0 0 0 1	
	[WHL +], AX	0 0 0 1 0 1 1 0	1 0 0 1 0 0 0 1	
	[TDE –], AX	0 0 0 1 0 1 1 0	1 0 1 0 0 0 0 1	
	[WHL–], AX	0 0 0 1 0 1 1 0	1 0 1 1 0 0 0 1	
	[TDE], AX	0 0 0 1 0 1 1 0	1 1 0 0 0 0 0 1	
	[WHL], AX	0 0 0 1 0 1 1 0	1 1 0 1 0 0 0 1	
	[VVP], AX	0 0 0 1 0 1 1 0	1 1 1 0 0 0 0 1	
	[UUP], AX	0 0 0 1 0 1 1 0	1 1 1 1 0 0 0 1	
	[TDE + byte], AX	0 0 0 0 0 1 1 0	1 0 0 0 0 0 0 1	← Low Offset →
	[SP + byte], AX	0 0 0 0 0 1 1 0	1 0 0 1 0 0 0 1	← Low Offset →
	[WHL + byte], AX	0 0 0 0 0 1 1 0	1 0 1 0 0 0 0 1	← Low Offset →
	[UUP + byte], AX	0 0 0 0 0 1 1 0	1 0 1 1 0 0 0 1	← Low Offset →
	[VVP + byte], AX	0 0 0 0 0 1 1 0	1 1 0 0 0 0 0 1	← Low Offset →
	imm24 [DE], AX	0 0 0 0 1 0 1 0	1 0 0 0 0 0 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [A], AX	0 0 0 0 1 0 1 0	1 0 0 1 0 0 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [HL], AX	0 0 0 0 1 0 1 0	1 0 1 0 0 0 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [B], AX	0 0 0 0 1 0 1 0	1 0 1 1 0 0 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	

(Continued on next page)

Mnemonic	Operands	Operation Code				
		B1		B2		B3
		B4		B5		B6
		B7				
MOVW	[TDE + A], AX	0 0 0 1	0 1 1 1	1 0 0 0	0 0 0 1	
	[WHL + A], AX	0 0 0 1	0 1 1 1	1 0 0 1	0 0 0 1	
	[TDE + B], AX	0 0 0 1	0 1 1 1	1 0 1 0	0 0 0 1	
	[WHL + B], AX	0 0 0 1	0 1 1 1	1 0 1 1	0 0 0 1	
	[VVP + DE], AX	0 0 0 1	0 1 1 1	1 1 0 0	0 0 0 1	
	[VVP + HL], AX	0 0 0 1	0 1 1 1	1 1 0 1	0 0 0 1	
	[TDE + C], AX	0 0 0 1	0 1 1 1	1 1 1 0	0 0 0 1	
	[WHL + C], AX	0 0 0 1	0 1 1 1	1 1 1 1	0 0 0 1	

(3) 24-bit data transfer instruction: MOVG

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
MOVG	rg, #imm24	0 0 1 1 1 0 0 0	1 G_6G_5 1 1 0 1 1	← Low Byte →
		← High Byte →	← High-w Byte →	
	rg, rg'	0 0 1 0 0 1 0 0	1 G_6G_5 1 1 G_2G_1 1	
	rg, !!addr24	0 0 1 1 1 1 1 0	1 G_6G_5 1 1 0 1 0	← High-w Address →
		← Low Address →	← High Address →	
	!!addr24, rg	0 0 1 1 1 1 1 0	1 G_6G_5 1 1 0 1 1	← High-w Address →
		← Low Address →	← High Address →	
	rg, saddrg2	0 0 1 1 1 0 0 0	1 G_6G_5 1 1 0 0 0	← Saddr2-offset →
	rg, saddrg1	0 0 1 1 1 0 0 0	1 G_6G_5 1 1 0 0 1	← Saddr1-offset →
	saddrg2, rg	0 0 1 1 1 0 0 0	1 G_6G_5 1 1 1 0 0	← Saddr2-offset →
	saddrg1, rg	0 0 1 1 1 0 0 0	1 G_6G_5 1 1 1 0 1	← Saddr1-offset →
	WHL, [%saddrg2]	0 0 0 0 0 1 1 1	0 0 1 1 0 0 1 0	← Saddr2-offset →
	WHL, [%saddrg1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 1 0 0 1 0
		← Saddr1-offset →		
	[%saddrg2], WHL	0 0 0 0 0 1 1 1	1 0 1 1 0 0 1 0	← Saddr2-offset →
	[%saddrg1], WHL	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 1 0 0 1 0
		← Saddr1-offset →		
	WHL, [TDE +]	0 0 0 1 0 1 1 0	0 0 0 0 0 0 1 0	
	WHL, [TDE –]	0 0 0 1 0 1 1 0	0 0 1 0 0 0 1 0	
	WHL, [TDE]	0 0 0 1 0 1 1 0	0 1 0 0 0 0 1 0	
	WHL, [WHL]	0 0 0 1 0 1 1 0	0 1 0 1 0 0 1 0	
	WHL, [VVP]	0 0 0 1 0 1 1 0	0 1 1 0 0 0 1 0	
	WHL, [UUP]	0 0 0 1 0 1 1 0	0 1 1 1 0 0 1 0	
	WHL, [TDE + byte]	0 0 0 0 0 1 1 0	0 0 0 0 0 0 1 0	← Low Offset →
	WHL, [SP + byte]	0 0 0 0 0 1 1 0	0 0 0 1 0 0 1 0	← Low Offset →
	WHL, [WHL + byte]	0 0 0 0 0 1 1 0	0 0 1 0 0 0 1 0	← Low Offset →
	WHL, [UUP + byte]	0 0 0 0 0 1 1 0	0 0 1 1 0 0 1 0	← Low Offset →
	WHL, [VVP + byte]	0 0 0 0 0 1 1 0	0 1 0 0 0 0 1 0	← Low Offset →
	WHL, imm24 [DE]	0 0 0 0 1 0 1 0	0 0 0 0 0 0 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	WHL, imm24 [A]	0 0 0 0 1 0 1 0	0 0 0 1 0 0 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
MOVG	WHL, imm24 [HL]	0 0 0 0 1 0 1 0	0 0 1 0 0 0 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	WHL, imm24 [B]	0 0 0 0 1 0 1 0	0 0 1 1 0 0 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	WHL, [TDE + A]	0 0 0 1 0 1 1 1	0 0 0 0 0 0 1 0	
	WHL, [WHL + A]	0 0 0 1 0 1 1 1	0 0 0 1 0 0 1 0	
	WHL, [TDE + B]	0 0 0 1 0 1 1 1	0 0 1 0 0 0 1 0	
	WHl, [WHL + B]	0 0 0 1 0 1 1 1	0 0 1 1 0 0 1 0	
	WHL, [VVP + DE]	0 0 0 1 0 1 1 1	0 1 0 0 0 0 1 0	
	WHL, [VVP + HL]	0 0 0 1 0 1 1 1	0 1 0 1 0 0 1 0	
	WHL, [TDE + C]	0 0 0 1 0 1 1 1	0 1 1 0 0 0 1 0	
	WHL, [WHL + C]	0 0 0 1 0 1 1 1	0 1 1 1 0 0 1 0	
	[TDE +], WHL	0 0 0 1 0 1 1 0	1 0 0 0 0 0 1 0	
	[TDE –], WHL	0 0 0 1 0 1 1 0	1 0 1 0 0 0 1 0	
	[TDE], WHL	0 0 0 1 0 1 1 0	1 1 0 0 0 0 1 0	
	[WHL], WHL	0 0 0 1 0 1 1 0	1 1 0 1 0 0 1 0	
	[VVP], WHL	0 0 0 1 0 1 1 0	1 1 1 0 0 0 1 0	
	[UUP], WHL	0 0 0 1 0 1 1 0	1 1 1 1 0 0 1 0	
	[TDE + byte], WHL	0 0 0 0 0 1 1 0	1 0 0 0 0 0 1 0	← Low Offset →
	[SP + byte], WHL	0 0 0 0 0 1 1 0	1 0 0 1 0 0 1 0	← Low Offset →
	[WHL + byte], WHL	0 0 0 0 0 1 1 0	1 0 1 0 0 0 1 0	← Low Offset →
	[UUP + byte], WHL	0 0 0 0 0 1 1 0	1 0 1 1 0 0 1 0	← Low Offset →
	[VVP + byte], WHL	0 0 0 0 0 1 1 0	1 1 0 0 0 0 1 0	← Low Offset →
	imm24 [DE], WHL	0 0 0 0 1 0 1 0	1 0 0 0 0 0 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [A], WHL	0 0 0 0 1 0 1 0	1 0 0 1 0 0 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [HL], WHL	0 0 0 0 1 0 1 0	1 0 1 0 0 0 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [B], WHL	0 0 0 0 1 0 1 0	1 0 1 1 0 0 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	[TDE + A], WHL	0 0 0 1 0 1 1 1	1 0 0 0 0 0 1 0	
	[WHL + A], WHL	0 0 0 1 0 1 1 1	1 0 0 1 0 0 1 0	
	[TDE + B], WHL	0 0 0 1 0 1 1 1	1 0 1 0 0 0 1 0	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
MOVG	[WHL + B], WHL	0 0 0 1 0 1 1 1	1 0 1 1 0 0 1 0	
	[VVP + DE], WHL	0 0 0 1 0 1 1 1	1 1 0 0 0 0 1 0	
	[VVP + HL], WHL	0 0 0 1 0 1 1 1	1 1 0 1 0 0 1 0	
	[TDE + C], WHL	0 0 0 1 0 1 1 1	1 1 1 0 0 0 1 0	
	[WHL + C], WHL	0 0 0 1 0 1 1 1	1 1 1 1 0 0 1 0	

(4) 8-bit data exchange instruction: XCH

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
XCH	r, r1	0 0 1 0 0 1 0 1	$R_7 R_6 R_5 R_4$ 0 $R_2 R_1 R_0$	
	r, r2	0 0 1 1 1 1 0 0	0 0 1 0 0 1 0 1	$R_7 R_6 R_5 R_4$ 0 $R_2 R_1 R_0$
	A, r1	1 1 0 1 1 $R_2 R_1$ 0		
	A, r2	0 0 1 1 1 1 0 0	1 1 0 1 1 $R_2 R_1 R_0$	
	A, saddr2	0 0 1 0 0 0 0 1	← Saddr2-offset →	
	r, saddr2	0 0 1 1 1 0 0 1	$R_7 R_6 R_5 R_4$ 0 0 0 0	← Saddr2-offset →
	r, saddr1	0 0 1 1 1 0 0 1	$R_7 R_6 R_5 R_4$ 0 0 0 1	← Saddr1-offset →
	r, sfr	0 0 1 1 1 0 0 1	$R_7 R_6 R_5 R_4$ 0 0 1 0	← Sfr-offset →
	saddr2, saddr2’	0 0 1 0 1 0 1 0	0 0 0 0 0 1 0 0	← Saddr2’-offset →
		← Saddr2-offset →		
	saddr2, saddr1	0 0 1 0 1 0 1 0	0 0 0 1 0 1 0 0	← Saddr1-offset →
		← Saddr2-offset →		
	saddr1, saddr2	0 0 1 0 1 0 1 0	0 0 1 0 0 1 0 0	← Saddr2-offset →
		← Saddr1-offset →		
	saddr1, saddr1’	0 0 1 0 1 0 1 0	0 0 1 1 0 1 0 0	← Saddr1’-offset →
		← Saddr1-offset →		
	r, !addr16	0 0 1 1 1 1 1 0	$R_7 R_6 R_5 R_4$ 0 1 0 0	← Low Address →
		← High Address →		
	r, !!addr24	0 0 1 1 1 1 1 0	$R_7 R_6 R_5 R_4$ 0 1 1 0	← High-w Address →
		← Low Address →	← High Address →	
	A, [saddrp2]	0 0 1 0 0 0 1 1	← Saddr2-offset →	
	A, [saddrp1]	0 0 1 1 1 1 0 0	0 0 1 0 0 0 1 1	← Saddr1-offset →
	A, [%saddrg2]	0 0 0 0 0 1 1 1	0 0 1 1 0 1 0 0	← Saddr2-offset →
	A, [%saddrg1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 1 0 1 0 0
		← Saddr1-offset →		
	A, [TDE +]	0 0 0 1 0 1 1 0	0 0 0 0 0 1 0 0	
	A, [WHL +]	0 0 0 1 0 1 1 0	0 0 0 1 0 1 0 0	
	A, [TDE –]	0 0 0 1 0 1 1 0	0 0 1 0 0 1 0 0	
	A, [WHL –]	0 0 0 1 0 1 1 0	0 0 1 1 0 1 0 0	
	A, [TDE]	0 0 0 1 0 1 1 0	0 1 0 0 0 1 0 0	
	A, [WHL]	0 0 0 1 0 1 1 0	0 1 0 1 0 1 0 0	
	A, [VVP]	0 0 0 1 0 1 1 0	0 1 1 0 0 1 0 0	
	A, [UUP]	0 0 0 1 0 1 1 0	0 1 1 1 0 1 0 0	

(Continued on next page)

<table>
<tr><th rowspan="3">Mnemonic</th><th rowspan="3">Operands</th><th colspan="3">Operation Code</th></tr>
<tr><th>B1</th><th>B2</th><th>B3</th></tr>
<tr><th>B4
B7</th><th>B5</th><th>B6</th></tr>
<tr><td>XCH</td><td>A, [TDE + byte]</td><td>0 0 0 0 0 1 1 0</td><td>0 0 0 0 0 1 0 0</td><td>← Low Offset →</td></tr>
<tr><td></td><td>A, [SP + byte]</td><td>0 0 0 0 0 1 1 0</td><td>0 0 0 1 0 1 0 0</td><td>← Low Offset →</td></tr>
<tr><td></td><td>A, [WHL + byte]</td><td>0 0 0 0 0 1 1 0</td><td>0 0 1 0 0 1 0 0</td><td>← Low Offset →</td></tr>
<tr><td></td><td>A, [UUP + byte]</td><td>0 0 0 0 0 1 1 0</td><td>0 0 1 1 0 1 0 0</td><td>← Low Offset →</td></tr>
<tr><td></td><td>A, [VVP + byte]</td><td>0 0 0 0 0 1 1 0</td><td>0 1 0 0 0 1 0 0</td><td>← Low Offset →</td></tr>
<tr><td></td><td rowspan="2">A, imm24 [DE]</td><td>0 0 0 0 1 0 1 0</td><td>0 0 0 0 0 1 0 0</td><td>← Low Offset →</td></tr>
<tr><td></td><td>← High Offset →</td><td>← High-w Offset →</td><td></td></tr>
<tr><td></td><td rowspan="2">A, imm24 [A]</td><td>0 0 0 0 1 0 1 0</td><td>0 0 0 1 0 1 0 0</td><td>← Low Offset →</td></tr>
<tr><td></td><td>← High Offset →</td><td>← High-w Offset →</td><td></td></tr>
<tr><td></td><td rowspan="2">A, imm24 [HL]</td><td>0 0 0 0 1 0 1 0</td><td>0 0 1 0 0 1 0 0</td><td>← Low Offset →</td></tr>
<tr><td></td><td>← High Offset →</td><td>← High-w Offset →</td><td></td></tr>
<tr><td></td><td rowspan="2">A, imm24 [B]</td><td>0 0 0 0 1 0 1 0</td><td>0 0 1 1 0 1 0 0</td><td>← Low Offset →</td></tr>
<tr><td></td><td>← High Offset →</td><td>← High-w Offset →</td><td></td></tr>
<tr><td></td><td>A, [TDE + A]</td><td>0 0 0 1 0 1 1 1</td><td>0 0 0 0 0 1 0 0</td><td></td></tr>
<tr><td></td><td>A, [WHL + A]</td><td>0 0 0 1 0 1 1 1</td><td>0 0 0 1 0 1 0 0</td><td></td></tr>
<tr><td></td><td>A, [TDE + B]</td><td>0 0 0 1 0 1 1 1</td><td>0 0 1 0 0 1 0 0</td><td></td></tr>
<tr><td></td><td>A, [WHL + B]</td><td>0 0 0 1 0 1 1 1</td><td>0 0 1 1 0 1 0 0</td><td></td></tr>
<tr><td></td><td>A, [VVP + DE]</td><td>0 0 0 1 0 1 1 1</td><td>0 1 0 0 0 1 0 0</td><td></td></tr>
<tr><td></td><td>A, [VVP + HL]</td><td>0 0 0 1 0 1 1 1</td><td>0 1 0 1 0 1 0 0</td><td></td></tr>
<tr><td></td><td>A, [TDE + C]</td><td>0 0 0 1 0 1 1 1</td><td>0 1 1 0 0 1 0 0</td><td></td></tr>
<tr><td></td><td>A, [WHL + C]</td><td>0 0 0 1 0 1 1 1</td><td>0 1 1 1 0 1 0 0</td><td></td></tr>
</table>

(5) 16-bit data exchange instruction: XCHW

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
XCHW	rp, rp'	0 0 1 0 0 1 0 1	$P_7 P_6 P_5$ 0 1 $P_2 P_1 P_0$	
	AX, saddrp2	0 0 0 1 1 0 1 1	← Saddr2-offset →	
	rp, saddrp2	0 0 1 1 1 0 0 1	$P_7 P_6 P_5$ 0 1 0 0 0	← Saddr2-offset →
	rp, saddrp1	0 0 1 1 1 0 0 1	$P_7 P_6 P_5$ 0 1 0 0 1	← Saddr1-offset →
	rp, sfrp	0 0 1 1 1 0 0 1	$P_7 P_6 P_5$ 0 1 0 1 0	← Sfr-offset →
	AX, [saddrp2]	0 0 0 0 0 1 1 1	0 0 1 0 0 1 0 1	← Saddr2-offset →
	AX, [saddrp1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 0 0 1 0 1
		← Saddr1-offset →		
	AX, [%saddrg2]	0 0 0 0 0 1 1 1	0 0 1 1 0 1 0 1	← Saddr2-offset →
	AX, [%saddrg1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 1 0 1 0 1
		← Saddr1-offset →		
	AX, !addr16	0 0 0 0 1 0 1 0	0 1 0 0 0 1 0 1	← Low Address →
		← High Address →		
	AX, !!addr24	0 0 0 0 1 0 1 0	0 1 0 1 0 1 0 1	← High-w Address →
		← Low Address →	← High Address →	
	saddrp2, saddrp2'	0 0 1 0 1 0 1 0	1 0 0 0 0 1 0 0	← Saddr2'-offset →
		← Saddr2-offset →		
	saddrp2, saddrp1	0 0 1 0 1 0 1 0	1 0 0 1 0 1 0 0	← Saddr1-offset →
		← Saddr2-offset →		
	saddrp1, saddrp2	0 0 1 0 1 0 1 0	1 0 1 0 0 1 0 0	← Saddr2-offset →
		← Saddr1-offset →		
	saddrp1, saddrp1'	0 0 1 0 1 0 1 0	1 0 1 1 0 1 0 0	← Saddr1'-offset →
		← Saddr1-offset →		
	AX, [TDE + byte]	0 0 0 0 0 1 1 0	0 0 0 0 0 1 0 1	← Low Offset →
	AX, [SP + byte]	0 0 0 0 0 1 1 0	0 0 0 1 0 1 0 1	← Low Offset →
	AX, [WHL + byte]	0 0 0 0 0 1 1 0	0 0 1 0 0 1 0 1	← Low Offset →
	AX, [UUP + byte]	0 0 0 0 0 1 1 0	0 0 1 1 0 1 0 1	← Low Offset →
	AX, [VVP + byte]	0 0 0 0 0 1 1 0	0 1 0 0 0 1 0 1	← Low Offset →
	AX, imm24 [DE]	0 0 0 0 1 0 1 0	0 0 0 0 0 1 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	AX, imm24 [A]	0 0 0 0 1 0 1 0	0 0 0 1 0 1 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
XCHW	AX, imm24 [HL]	0 0 0 0 1 0 1 0	0 0 1 0 0 1 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	AX, imm24 [B]	0 0 0 0 1 0 1 0	0 0 1 1 0 1 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	AX, [TDE +]	0 0 0 1 0 1 1 0	0 0 0 0 0 1 0 1	
	AX, [WHL +]	0 0 0 1 0 1 1 0	0 0 0 1 0 1 0 1	
	AX, [TDE –]	0 0 0 1 0 1 1 0	0 0 1 0 0 1 0 1	
	AX, [WHL –]	0 0 0 1 0 1 1 0	0 0 1 1 0 1 0 1	
	AX, [TDE]	0 0 0 1 0 1 1 0	0 1 0 0 0 1 0 1	
	AX, [WHL]	0 0 0 1 0 1 1 0	0 1 0 1 0 1 0 1	
	AX, [VVP]	0 0 0 1 0 1 1 0	0 1 1 0 0 1 0 1	
	AX, [UUP]	0 0 0 1 0 1 1 0	0 1 1 1 0 1 0 1	
	AX, [TDE + A]	0 0 0 1 0 1 1 1	0 0 0 0 0 1 0 1	
	AX, [WHL + A]	0 0 0 1 0 1 1 1	0 0 0 1 0 1 0 1	
	AX, [TDE + B]	0 0 0 1 0 1 1 1	0 0 1 0 0 1 0 1	
	AX, [WHL + B]	0 0 0 1 0 1 1 1	0 0 1 1 0 1 0 1	
	AX, [VVP + DE]	0 0 0 1 0 1 1 1	0 1 0 0 0 1 0 1	
	AX, [VVP + HL]	0 0 0 1 0 1 1 1	0 1 0 1 0 1 0 1	
	AX, [TDE + C]	0 0 0 1 0 1 1 1	0 1 1 0 0 1 0 1	
	AX, [WHL + C]	0 0 0 1 0 1 1 1	0 1 1 1 0 1 0 1	

(6) 8-bit operation instructions: ADD, ADDC, SUB, SUBC, CMP, AND, OR, XOR

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
ADD	A, #byte	1 0 1 0 1 0 0 0	← #byte →	
	r, #byte	0 1 1 1 1 0 0 0	$R_7R_6R_5R_4$ 0 0 1 1	← #byte →
	saddr2, #byte	0 1 1 0 1 0 0 0	← Saddr2-offset →	← #byte →
	saddr1, #byte	0 0 1 1 1 1 0 0	0 1 1 0 1 0 0 0	← Saddr1-offset →
		← #byte →		
	sfr, #byte	0 0 0 0 0 0 0 1	0 1 1 0 1 0 0 0	← Sfr-offset →
		← #byte →		
	r, r1	1 0 0 0 1 0 0 0	$R_7R_6R_5R_4$ 0 $R_2R_1R_0$	
	r, r2	0 0 1 1 1 1 0 0	1 0 0 0 1 0 0 0	$R_7R_6R_5R_4$ 0 $R_2R_1R_0$
	A, saddr2	1 0 0 1 1 0 0 0	← Saddr2-offset →	
	r, saddr2	0 1 1 1 1 0 0 0	$R_7R_6R_5R_4$ 0 0 0 0	← Saddr2-offset →
	r, saddr1	0 1 1 1 1 0 0 0	$R_7R_6R_5R_4$ 0 0 0 1	← Saddr1-offset →
	saddr2, r	0 1 1 1 1 0 0 0	$R_7R_6R_5R_4$ 0 1 0 0	← Saddr2-offset →
	saddr1, r	0 1 1 1 1 0 0 0	$R_7R_6R_5R_4$ 0 1 0 1	← Saddr1-offset →
	r, sfr	0 1 1 1 1 0 0 0	$R_7R_6R_5R_4$ 0 0 1 0	← Sfr-offset →
	sfr, r	0 1 1 1 1 0 0 0	$R_7R_6R_5R_4$ 0 1 1 0	← Sfr-offset →
	saddr2, saddr2'	0 0 1 0 1 0 1 0	0 0 0 0 1 0 0 0	← Saddr2'-offset →
		← Saddr2-offset →		
	saddr2, saddr1	0 0 1 0 1 0 1 0	0 0 0 1 1 0 0 0	← Saddr1-offset →
		← Saddr2-offset →		
	saddr1, saddr2	0 0 1 0 1 0 1 0	0 0 1 0 1 0 0 0	← Saddr2-offset →
		← Saddr1-offset →		
	saddr1, saddr1'	0 0 1 0 1 0 1 0	0 0 1 1 1 0 0 0	← Saddr1'-offset →
		← Saddr1-offset →		
	A, [saddrp2]	0 0 0 0 0 1 1 1	0 0 1 0 1 0 0 0	← Saddr2-offset →
	A, [saddrp1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 0 1 0 0 0
		← Saddr1-offset →		
	A, [%saddrg2]	0 0 0 0 0 1 1 1	0 0 1 1 1 0 0 0	← Saddr2-offset →
	A, [%saddrg1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 1 1 0 0 0
		← Saddr1-offset →		
	[saddrp2], A	0 0 0 0 0 1 1 1	1 0 1 0 1 0 0 0	← Saddr2-offset →
	[saddrp1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 0 1 0 0 0
		← Saddr1-offset →		

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
ADD	[%saddrg2], A	0 0 0 0 0 1 1 1	1 0 1 1 1 0 0 0	← Saddr2-offset →
	[%saddrg1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 1 1 0 0 0
		← Saddr1 Offset →		
	A, !addr16	0 0 0 0 1 0 1 0	0 1 0 0 1 0 0 0	← Low Address →
		← High Address →		
	A, !!addr24	0 0 0 0 1 0 1 0	0 1 0 1 1 0 0 0	← High-w Address →
		← Low Address →	← High Address →	
	!addr16, A	0 0 0 0 1 0 1 0	1 1 0 0 1 0 0 0	← Low Address →
		← High Address →		
	!!addr24, A	0 0 0 0 1 0 1 0	1 1 0 1 1 0 0 0	← High-w Address →
		← Low Address →	← High Address →	
	A, [TDE +]	0 0 0 1 0 1 1 0	0 0 0 0 1 0 0 0	
	A, [WHL +]	0 0 0 1 0 1 1 0	0 0 0 1 1 0 0 0	
	A, [TDE –]	0 0 0 1 0 1 1 0	0 0 1 0 1 0 0 0	
	A, [WHL –]	0 0 0 1 0 1 1 0	0 0 1 1 1 0 0 0	
	A, [TDE]	0 0 0 1 0 1 1 0	0 1 0 0 1 0 0 0	
	A, [WHL]	0 0 0 1 0 1 1 0	0 1 0 1 1 0 0 0	
	A, [VVP]	0 0 0 1 0 1 1 0	0 1 1 0 1 0 0 0	
	A, [UUP]	0 0 0 1 0 1 1 0	0 1 1 1 1 0 0 0	
	A, [TDE + byte]	0 0 0 0 0 1 1 0	0 0 0 0 1 0 0 0	← Low Offset →
	A, [SP + byte]	0 0 0 0 0 1 1 0	0 0 0 1 1 0 0 0	← Low Offset →
	A, [WHL + byte]	0 0 0 0 0 1 1 0	0 0 1 0 1 0 0 0	← Low Offset →
	A, [UUP + byte]	0 0 0 0 0 1 1 0	0 0 1 1 1 0 0 0	← Low Offset →
	A, [VVP + byte]	0 0 0 0 0 1 1 0	0 1 0 0 1 0 0 0	← Low Offset →
	A, imm24 [DE]	0 0 0 0 1 0 1 0	0 0 0 0 1 0 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [A]	0 0 0 0 1 0 1 0	0 0 0 1 1 0 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [HL]	0 0 0 0 1 0 1 0	0 0 1 0 1 0 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [B]	0 0 0 0 1 0 1 0	0 0 1 1 1 0 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, [TDE + A]	0 0 0 1 0 1 1 1	0 0 0 0 1 0 0 0	
	A, [WHL + A]	0 0 0 1 0 1 1 1	0 0 0 1 1 0 0 0	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
ADD	A, [TDE + B]	0 0 0 1 0 1 1 1	0 0 1 0 1 0 0 0	
	A, [WHL + B]	0 0 0 1 0 1 1 1	0 0 1 1 1 0 0 0	
	A, [VVP + DE]	0 0 0 1 0 1 1 1	0 1 0 0 1 0 0 0	
	A, [VVP + HL]	0 0 0 1 0 1 1 1	0 1 0 1 1 0 0 0	
	A, [TDE + C]	0 0 0 1 0 1 1 1	0 1 1 0 1 0 0 0	
	A, [WHL + C]	0 0 0 1 0 1 1 1	0 1 1 1 1 0 0 0	
	[TDE +], A	0 0 0 1 0 1 1 0	1 0 0 0 1 0 0 0	
	[WHL +], A	0 0 0 1 0 1 1 0	1 0 0 1 1 0 0 0	
	[TDE –], A	0 0 0 1 0 1 1 0	1 0 1 0 1 0 0 0	
	[WHL –], A	0 0 0 1 0 1 1 0	1 0 1 1 1 0 0 0	
	[TDE], A	0 0 0 1 0 1 1 0	1 1 0 0 1 0 0 0	
	[WHL], A	0 0 0 1 0 1 1 0	1 1 0 1 1 0 0 0	
	[VVP], A	0 0 0 1 0 1 1 0	1 1 1 0 1 0 0 0	
	[UUP], A	0 0 0 1 0 1 1 0	1 1 1 1 1 0 0 0	
	[TDE + byte], A	0 0 0 0 0 1 1 0	1 0 0 0 1 0 0 0	← Low Offset →
	[SP + byte], A	0 0 0 0 0 1 1 0	1 0 0 1 1 0 0 0	← Low Offset →
	[WHL + byte], A	0 0 0 0 0 1 1 0	1 0 1 0 1 0 0 0	← Low Offset →
	[UUP + byte], A	0 0 0 0 0 1 1 0	1 0 1 1 1 0 0 0	← Low Offset →
	[VVP + byte], A	0 0 0 0 0 1 1 0	1 1 0 0 1 0 0 0	← Low Offset →
	imm24 [DE], A	0 0 0 0 1 0 1 0	1 0 0 0 1 0 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [A], A	0 0 0 0 1 0 1 0	1 0 0 1 1 0 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [HL], A	0 0 0 0 1 0 1 0	1 0 1 0 1 0 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [B], A	0 0 0 0 1 0 1 0	1 0 1 1 1 0 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	[TDE + A], A	0 0 0 1 0 1 1 1	1 0 0 0 1 0 0 0	
	[WHL + A], A	0 0 0 1 0 1 1 1	1 0 0 1 1 0 0 0	
	[TDE + B], A	0 0 0 1 0 1 1 1	1 0 1 0 1 0 0 0	
	[WHL + B], A	0 0 0 1 0 1 1 1	1 0 1 1 1 0 0 0	
	[VVP + DE], A	0 0 0 1 0 1 1 1	1 1 0 0 1 0 0 0	
	[VVP + HL], A	0 0 0 1 0 1 1 1	1 1 0 1 1 0 0 0	
	[TDE + C], A	0 0 0 1 0 1 1 1	1 1 1 0 1 0 0 0	
	[WHL + C], A	0 0 0 1 0 1 1 1	1 1 1 1 1 0 0 0	

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
ADDC	A, #byte	1 0 1 0 1 0 0 1	← #byte →	
	r, #byte	0 1 1 1 1 0 0 1	$R_7 R_6 R_5 R_4$ 0 0 1 1	← #byte →
	saddr2, #byte	0 1 1 0 1 0 0 1	← Saddr2-offset →	← #byte →
	saddr1, #byte	0 0 1 1 1 1 0 0	0 1 1 0 1 0 0 1	← Saddr1-offset →
		← #byte →		
	sfr, #byte	0 0 0 0 0 0 0 1	0 1 1 0 1 0 0 1	← Sfr-offset →
		← #byte →		
	r, r1	1 0 0 0 1 0 0 1	$R_7 R_6 R_5 R_4$ 0 $R_2 R_1 R_0$	
	r, r2	0 0 1 1 1 1 0 0	1 0 0 0 1 0 0 1	$R_7 R_6 R_5 R_4$ 0 $R_2 R_1 R_0$
	A, saddr2	1 0 0 1 1 0 0 1	← Saddr2-offset →	
	r, saddr2	0 1 1 1 1 0 0 1	$R_7 R_6 R_5 R_4$ 0 0 0 0	← Saddr2-offset →
	r, saddr1	0 1 1 1 1 0 0 1	$R_7 R_6 R_5 R_4$ 0 0 0 1	← Saddr1-offset →
	saddr2, r	0 1 1 1 1 0 0 1	$R_7 R_6 R_5 R_4$ 0 1 0 0	← Saddr2-offset →
	saddr1, r	0 1 1 1 1 0 0 1	$R_7 R_6 R_5 R_4$ 0 1 0 1	← Saddr1-offset →
	r, sfr	0 1 1 1 1 0 0 1	$R_7 R_6 R_5 R_4$ 0 0 1 0	← Sfr-offset →
	sfr, r	0 1 1 1 1 0 0 1	$R_7 R_6 R_5 R_4$ 0 1 1 0	← Sfr-offset →
	saddr2, saddr2’	0 0 1 0 1 0 1 0	0 0 0 0 1 0 0 1	← Saddr2’-offset →
		← Saddr2-offset →		
	saddr2, saddr1	0 0 1 0 1 0 1 0	0 0 0 1 1 0 0 1	← Saddr1-offset →
		← Saddr2-offset →		
	saddr1, saddr2	0 0 1 0 1 0 1 0	0 0 1 0 1 0 0 1	← Saddr2-offset →
		← Saddr1-offset →		
	saddr1, saddr1’	0 0 1 0 1 0 1 0	0 0 1 1 1 0 0 1	← Saddr1’-offset →
		← Saddr1-offset →		
	A, [saddrp2]	0 0 0 0 0 1 1 1	0 0 1 0 1 0 0 1	← Saddr2-offset →
	A, [saddrp1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 0 1 0 0 1
		← Saddr1-offset →		
	A, [%saddrg2]	0 0 0 0 0 1 1 1	0 0 1 1 1 0 0 1	← Saddr2-offset →
	A, [%saddrg1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 1 1 0 0 1
		← Saddr1-offset →		
	[saddrp2], A	0 0 0 0 0 1 1 1	1 0 1 0 1 0 0 1	← Saddr2-offset →
	[saddrp1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 0 1 0 0 1
		← Saddr1-offset →		
	[%saddrg2], A	0 0 0 0 0 1 1 1	1 0 1 1 1 0 0 1	← Saddr2-offset →

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
ADDC	[%saddrg1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 1 1 0 0 1
		← Saddr1-offset →		
	A, !addr16	0 0 0 0 1 0 1 0	0 1 0 0 1 0 0 1	← Low Address →
		← High Address →		
	A, !!addr24	0 0 0 0 1 0 1 0	0 1 0 1 1 0 0 1	← High-w Address →
		← Low Address →	← High Address →	
	!addr16, A	0 0 0 0 1 0 1 0	1 1 0 0 1 0 0 1	← Low Address →
		← High Address →		
	!!addr24, A	0 0 0 0 1 0 1 0	1 1 0 1 1 0 0 1	← High-w Address →
		← Low Address →	← High Address →	
	A, [TDE +]	0 0 0 1 0 1 1 0	0 0 0 0 1 0 0 1	
	A, [WHL +]	0 0 0 1 0 1 1 0	0 0 0 1 1 0 0 1	
	A, [TDE –]	0 0 0 1 0 1 1 0	0 0 1 0 1 0 0 1	
	A, [WHL –]	0 0 0 1 0 1 1 0	0 0 1 1 1 0 0 1	
	A, [TDE]	0 0 0 1 0 1 1 0	0 1 0 0 1 0 0 1	
	A, [WHL]	0 0 0 1 0 1 1 0	0 1 0 1 1 0 0 1	
	A, [VVP]	0 0 0 1 0 1 1 0	0 1 1 0 1 0 0 1	
	A, [UUP]	0 0 0 1 0 1 1 0	0 1 1 1 1 0 0 1	
	A, [TDE + byte]	0 0 0 0 0 1 1 0	0 0 0 0 1 0 0 1	← Low Offset →
	A, [SP + byte]	0 0 0 0 0 1 1 0	0 0 0 1 1 0 0 1	← Low Offset →
	A, [WHL + byte]	0 0 0 0 0 1 1 0	0 0 1 0 1 0 0 1	← Low Offset →
	A, [UUP + byte]	0 0 0 0 0 1 1 0	0 0 1 1 1 0 0 1	← Low Offset →
	A, [VVP + byte]	0 0 0 0 0 1 1 0	0 1 0 0 1 0 0 1	← Low Offset →
	A, imm24 [DE]	0 0 0 0 1 0 1 0	0 0 0 0 1 0 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [A]	0 0 0 0 1 0 1 0	0 0 0 1 1 0 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [HL]	0 0 0 0 1 0 1 0	0 0 1 0 1 0 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [B]	0 0 0 0 1 0 1 0	0 0 1 1 1 0 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, [TDE + A]	0 0 0 1 0 1 1 1	0 0 0 0 1 0 0 1	
	A, [WHL + A]	0 0 0 1 0 1 1 1	0 0 0 1 1 0 0 1	
	A, [TDE + B]	0 0 0 1 0 1 1 1	0 0 1 0 1 0 0 1	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
ADDC	A, [WHL + B]	0 0 0 1 0 1 1 1	0 0 1 1 1 0 0 1	
	A, [VVP + DE]	0 0 0 1 0 1 1 1	0 1 0 0 1 0 0 1	
	A, [VVP + HL]	0 0 0 1 0 1 1 1	0 1 0 1 1 0 0 1	
	A, [TDE + C]	0 0 0 1 0 1 1 1	0 1 1 0 1 0 0 1	
	A, [WHL + C]	0 0 0 1 0 1 1 1	0 1 1 1 1 0 0 1	
	[TDE +], A	0 0 0 1 0 1 1 0	1 0 0 0 1 0 0 1	
	[WHL +], A	0 0 0 1 0 1 1 0	1 0 0 1 1 0 0 1	
	[TDE –], A	0 0 0 1 0 1 1 0	1 0 1 0 1 0 0 1	
	[WHL –], A	0 0 0 1 0 1 1 0	1 0 1 1 1 0 0 1	
	[TDE], A	0 0 0 1 0 1 1 0	1 1 0 0 1 0 0 1	
	[WHL], A	0 0 0 1 0 1 1 0	1 1 0 1 1 0 0 1	
	[VVP], A	0 0 0 1 0 1 1 0	1 1 1 0 1 0 0 1	
	[UUP], A	0 0 0 1 0 1 1 0	1 1 1 1 1 0 0 1	
	[TDE + byte], A	0 0 0 0 0 1 1 0	1 0 0 0 1 0 0 1	← Low Offset →
	[SP + byte], A	0 0 0 0 0 1 1 0	1 0 0 1 1 0 0 1	← Low Offset →
	[WHL + byte], A	0 0 0 0 0 1 1 0	1 0 1 0 1 0 0 1	← Low Offset →
	[UUP + byte], A	0 0 0 0 0 1 1 0	1 0 1 1 1 0 0 1	← Low Offset →
	[VVP + byte], A	0 0 0 0 0 1 1 0	1 1 0 0 1 0 0 1	← Low Offset →
	imm24 [DE], A	0 0 0 0 1 0 1 0	1 0 0 0 1 0 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [A], A	0 0 0 0 1 0 1 0	1 0 0 1 1 0 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [HL], A	0 0 0 0 1 0 1 0	1 0 1 0 1 0 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [B], A	0 0 0 0 1 0 1 0	1 0 1 1 1 0 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	[TDE + A], A	0 0 0 1 0 1 1 1	1 0 0 0 1 0 0 1	
	[WHL + A], A	0 0 0 1 0 1 1 1	1 0 0 1 1 0 0 1	
	[TDE + B], A	0 0 0 1 0 1 1 1	1 0 1 0 1 0 0 1	
	[WHL + B], A	0 0 0 1 0 1 1 1	1 0 1 1 1 0 0 1	
	[VVP + DE], A	0 0 0 1 0 1 1 1	1 1 0 0 1 0 0 1	
	[VVP + HL], A	0 0 0 1 0 1 1 1	1 1 0 1 1 0 0 1	
	[TDE + C], A	0 0 0 1 0 1 1 1	1 1 1 0 1 0 0 1	
	[WHL + C], A	0 0 0 1 0 1 1 1	1 1 1 1 1 0 0 1	

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
SUB	A, #byte	1 0 1 0 1 0 1 0	← #byte →	
	r, #byte	0 1 1 1 1 0 1 0	$R_7R_6R_5R_4$ 0 0 1 1	← #byte →
	saddr2, #byte	0 1 1 0 1 0 1 0	← Saddr2 Offset →	← #byte →
	saddr1, #byte	0 0 1 1 1 1 0 0	0 1 1 0 1 0 1 0	← Saddr1-Offset →
		← #byte →		
	sfr, #byte	0 0 0 0 0 0 0 1	0 1 1 0 1 0 1 0	← Sfr-offset →
		← #byte →		
	r, r1	1 0 0 0 1 0 1 0	$R_7R_6R_5R_4$ 0 $R_2R_1R_0$	
	r, r2	0 0 1 1 1 1 0 0	1 0 0 0 1 0 1 0	$R_7R_6R_5R_4$ 0 $R_2R_1R_0$
	A, saddr2	1 0 0 1 1 0 1 0	← Saddr2-offset →	
	r, saddr2	0 1 1 1 1 0 1 0	$R_7R_6R_5R_4$ 0 0 0 0	← Saddr2-offset →
	r, saddr1	0 1 1 1 1 0 1 0	$R_7R_6R_5R_4$ 0 0 0 1	← Saddr1-offset →
	saddr2, r	0 1 1 1 1 0 1 0	$R_7R_6R_5R_4$ 0 1 0 0	← Saddr2-offset →
	saddr1, r	0 1 1 1 1 0 1 0	$R_7R_6R_5R_4$ 0 1 0 1	← Saddr1-offset →
	r, sfr	0 1 1 1 1 0 1 0	$R_7R_6R_5R_4$ 0 0 1 0	← Sfr-offset →
	sfr, r	0 1 1 1 1 0 1 0	$R_7R_6R_5R_4$ 0 1 1 0	← Sfr-offset →
	saddr2, saddr2’	0 0 1 0 1 0 1 0	0 0 0 0 1 0 1 0	← Saddr2’-offset →
		← Saddr2-offset →		
	saddr2, saddr1	0 0 1 0 1 0 1 0	0 0 0 1 1 0 1 0	← Saddr1-offset →
		← Saddr2-offset →		
	saddr1, saddr2	0 0 1 0 1 0 1 0	0 0 1 0 1 0 1 0	← Saddr2-offset →
		← Saddr1-offset →		
	saddr1, saddr1’	0 0 1 0 1 0 1 0	0 0 1 1 1 0 1 0	← Saddr1’-offset →
		← Saddr1-offset →		
	A, [saddrp2]	0 0 0 0 0 1 1 1	0 0 1 0 1 0 1 0	← Saddr2-offset →
	A, [saddrp1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 0 1 0 1 0
		← Saddr1-offset →		
	A, [%saddrg2]	0 0 0 0 0 1 1 1	0 0 1 1 1 0 1 0	← Saddr2-offset →
	A, [%saddrg1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 1 1 0 1 0
		← Saddr1-offset →		
	[saddrp2], A	0 0 0 0 0 1 1 1	1 0 1 0 1 0 1 0	← Saddr2-offset →
	[saddrp1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 0 1 0 1 0
		← Saddr1-offset →		
	[%saddrg2], A	0 0 0 0 0 1 1 1	1 0 1 1 1 0 1 0	← Saddr2-offset →

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
SUB	[%saddrg1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 1 1 0 1 0
		← Saddr1-offset →		
	A, !addr16	0 0 0 0 1 0 1 0	0 1 0 0 1 0 1 0	← Low Address →
		← High Address →		
	A, !!addr24	0 0 0 0 1 0 1 0	0 1 0 1 1 0 1 0	← High-w Address →
		← Low Address →	← High Address →	
	!addr16, A	0 0 0 0 1 0 1 0	1 1 0 0 1 0 1 0	← Low Address →
		← High Address →		
	!!addr24, A	0 0 0 0 1 0 1 0	1 1 0 1 1 0 1 0	← High-w Address →
		← Low Address →	← High Address →	
	A, [TDE +]	0 0 0 1 0 1 1 0	0 0 0 0 1 0 1 0	
	A, [WHL +]	0 0 0 1 0 1 1 0	0 0 0 1 1 0 1 0	
	A, [TDE –]	0 0 0 1 0 1 1 0	0 0 1 0 1 0 1 0	
	A, [WHL –]	0 0 0 1 0 1 1 0	0 0 1 1 1 0 1 0	
	A, [TDE]	0 0 0 1 0 1 1 0	0 1 0 0 1 0 1 0	
	A, [WHL]	0 0 0 1 0 1 1 0	0 1 0 1 1 0 1 0	
	A, [VVP]	0 0 0 1 0 1 1 0	0 1 1 0 1 0 1 0	
	A, [UUP]	0 0 0 1 0 1 1 0	0 1 1 1 1 0 1 0	
	A, [TDE + byte]	0 0 0 0 0 1 1 0	0 0 0 0 1 0 1 0	← Low Offset →
	A, [SP + byte]	0 0 0 0 0 1 1 0	0 0 0 1 1 0 1 0	← Low Offset →
	A, [WHL + byte]	0 0 0 0 0 1 1 0	0 0 1 0 1 0 1 0	← Low Offset →
	A, [UUP + byte]	0 0 0 0 0 1 1 0	0 0 1 1 1 0 1 0	← Low Offset →
	A, [VVP + byte]	0 0 0 0 0 1 1 0	0 1 0 0 1 0 1 0	← Low Offset →
	A, imm24 [DE]	0 0 0 0 1 0 1 0	0 0 0 0 1 0 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [A]	0 0 0 0 1 0 1 0	0 0 0 1 1 0 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [HL]	0 0 0 0 1 0 1 0	0 0 1 0 1 0 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [B]	0 0 0 0 1 0 1 0	0 0 1 1 1 0 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, [TDE + A]	0 0 0 1 0 1 1 1	0 0 0 0 1 0 1 0	
	A, [WHL + A]	0 0 0 1 0 1 1 1	0 0 0 1 1 0 1 0	
	A, [TDE + B]	0 0 0 1 0 1 1 1	0 0 1 0 1 0 1 0	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
SUB	A, [WHL + B]	0 0 0 1 0 1 1 1	0 0 1 1 1 0 1 0	
	A, [VVP + DE]	0 0 0 1 0 1 1 1	0 1 0 0 1 0 1 0	
	A, [VVP + HL]	0 0 0 1 0 1 1 1	0 1 0 1 1 0 1 0	
	A, [TDE + C]	0 0 0 1 0 1 1 1	0 1 1 0 1 0 1 0	
	A, [WHL + C]	0 0 0 1 0 1 1 1	0 1 1 1 1 0 1 0	
	[TDE +], A	0 0 0 1 0 1 1 0	1 0 0 0 1 0 1 0	
	[WHL +], A	0 0 0 1 0 1 1 0	1 0 0 1 1 0 1 0	
	[TDE –], A	0 0 0 1 0 1 1 0	1 0 1 0 1 0 1 0	
	[WHL –], A	0 0 0 1 0 1 1 0	1 0 1 1 1 0 1 0	
	[TDE], A	0 0 0 1 0 1 1 0	1 1 0 0 1 0 1 0	
	[WHL], A	0 0 0 1 0 1 1 0	1 1 0 1 1 0 1 0	
	[VVP], A	0 0 0 1 0 1 1 0	1 1 1 0 1 0 1 0	
	[UUP], A	0 0 0 1 0 1 1 0	1 1 1 1 1 0 1 0	
	[TDE + byte], A	0 0 0 0 0 1 1 0	1 0 0 0 1 0 1 0	← Low Offset →
	[SP + byte], A	0 0 0 0 0 1 1 0	1 0 0 1 1 0 1 0	← Low Offset →
	[WHL + byte], A	0 0 0 0 0 1 1 0	1 0 1 0 1 0 1 0	← Low Offset →
	[UUP + byte], A	0 0 0 0 0 1 1 0	1 0 1 1 1 0 1 0	← Low Offset →
	[VVP + byte], A	0 0 0 0 0 1 1 0	1 1 0 0 1 0 1 0	← Low Offset →
	imm24 [DE], A	0 0 0 0 1 0 1 0	1 0 0 0 1 0 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [A], A	0 0 0 0 1 0 1 0	1 0 0 1 1 0 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [HL], A	0 0 0 0 1 0 1 0	1 0 1 0 1 0 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [B], A	0 0 0 0 1 0 1 0	1 0 1 1 1 0 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	[TDE + A], A	0 0 0 1 0 1 1 1	1 0 0 0 1 0 1 0	
	[WHL + A], A	0 0 0 1 0 1 1 1	1 0 0 1 1 0 1 0	
	[TDE + B], A	0 0 0 1 0 1 1 1	1 0 1 0 1 0 1 0	
	[WHL + B], A	0 0 0 1 0 1 1 1	1 0 1 1 1 0 1 0	
	[VVP + DE], A	0 0 0 1 0 1 1 1	1 1 0 0 1 0 1 0	
	[VVP + HL], A	0 0 0 1 0 1 1 1	1 1 0 1 1 0 1 0	
	[TDE + C], A	0 0 0 1 0 1 1 1	1 1 1 0 1 0 1 0	
	[WHL + C], A	0 0 0 1 0 1 1 1	1 1 1 1 1 0 1 0	

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
SUBC	A, #byte	1 0 1 0 1 0 1 1	← #byte →	
	r, #byte	0 1 1 1 1 0 1 1	$R_7 R_6 R_5 R_4$ 0 0 1 1	← #byte →
	saddr2, #byte	0 1 1 0 1 0 1 1	← Saddr2-offset →	← #byte →
	saddr1, #byte	0 0 1 1 1 1 0 0	0 1 1 0 1 0 1 1	← Saddr1-offset →
		← #byte →		
	sfr, #byte	0 0 0 0 0 0 0 1	0 1 1 0 1 0 1 1	← Sfr-offset →
		← #byte →		
	r, r1	1 0 0 0 1 0 1 1	$R_7 R_6 R_5 R_4$ 0 $R_2 R_1 R_0$	
	r, r2	0 0 1 1 1 1 0 0	1 0 0 0 1 0 1 1	$R_7 R_6 R_5 R_4$ 0 $R_2 R_1 R_0$
	A, saddr2	1 0 0 1 1 0 1 1	← Saddr2-offset →	
	r, saddr2	0 1 1 1 1 0 1 1	$R_7 R_6 R_5 R_4$ 0 0 0 0	← Saddr2-offset →
	r, saddr1	0 1 1 1 1 0 1 1	$R_7 R_6 R_5 R_4$ 0 0 0 1	← Saddr1-offset →
	saddr2, r	0 1 1 1 1 0 1 1	$R_7 R_6 R_5 R_4$ 0 1 0 0	← Saddr2-offset →
	saddr1, r	0 1 1 1 1 0 1 1	$R_7 R_6 R_5 R_4$ 0 1 0 1	← Saddr1-offset →
	r, sfr	0 1 1 1 1 0 1 1	$R_7 R_6 R_5 R_4$ 0 0 1 0	← Sfr-offset →
	sfr, r	0 1 1 1 1 0 1 1	$R_7 R_6 R_5 R_4$ 0 1 1 0	← Sfr-offset →
	saddr2, saddr2'	0 0 1 0 1 0 1 0	0 0 0 0 1 0 1 1	← Saddr2'-offset →
		← Saddr2-offset →		
	saddr2, saddr1	0 0 1 0 1 0 1 0	0 0 0 1 1 0 1 1	← Saddr1-offset →
		← Saddr2-offset →		
	saddr1, saddr2	0 0 1 0 1 0 1 0	0 0 1 0 1 0 1 1	← Saddr2-offset →
		← Saddr1-offset →		
	saddr1, saddr1'	0 0 1 0 1 0 1 0	0 0 1 1 1 0 1 1	← Saddr1'-offset →
		← Saddr1-offset →		
	A, [saddrp2]	0 0 0 0 0 1 1 1	0 0 1 0 1 0 1 1	← Saddr2-offset →
	A, [saddrp1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 0 1 0 1 1
		← Saddr1-offset →		
	A, [%saddrg2]	0 0 0 0 0 1 1 1	0 0 1 1 1 0 1 1	← Saddr2-offset →
	A, [%saddrg1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 1 1 0 1 1
		← Saddr1-offset →		
	[saddrp2], A	0 0 0 0 0 1 1 1	1 0 1 0 1 0 1 1	← Saddr2-offset →
	[saddrp1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 0 1 0 1 1
		← Saddr1-offset →		
	[%saddrg2], A	0 0 0 0 0 1 1 1	1 0 1 1 1 0 1 1	← Saddr2-offset →

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
SUBC	[%saddrg1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 1 1 0 1 1
		← Saddr1-offset →		
	A, !addr16	0 0 0 0 1 0 1 0	0 1 0 0 1 0 1 1	← Low Address →
		← High Address →		
	A, !!addr24	0 0 0 0 1 0 1 0	0 1 0 1 1 0 1 1	← High-w Address →
		← Low Address →	← High Address →	
	!addr16, A	0 0 0 0 1 0 1 0	1 1 0 0 1 0 1 1	← Low Address →
		← High Address →		
	!!addr24, A	0 0 0 0 1 0 1 0	1 1 0 1 1 0 1 1	← High-w Address →
		← Low Address →	← High Address →	
	A, [TDE +]	0 0 0 1 0 1 1 0	0 0 0 0 1 0 1 1	
	A, [WHL +]	0 0 0 1 0 1 1 0	0 0 0 1 1 0 1 1	
	A, [TDE –]	0 0 0 1 0 1 1 0	0 0 1 0 1 0 1 1	
	A, [WHL –]	0 0 0 1 0 1 1 0	0 0 1 1 1 0 1 1	
	A, [TDE]	0 0 0 1 0 1 1 0	0 1 0 0 1 0 1 1	
	A, [WHL]	0 0 0 1 0 1 1 0	0 1 0 1 1 0 1 1	
	A, [VVP]	0 0 0 1 0 1 1 0	0 1 1 0 1 0 1 1	
	A, [UUP]	0 0 0 1 0 1 1 0	0 1 1 1 1 0 1 1	
	A, [TDE + byte]	0 0 0 0 0 1 1 0	0 0 0 0 1 0 1 1	← Low Offset →
	A, [SP + byte]	0 0 0 0 0 1 1 0	0 0 0 1 1 0 1 1	← Low Offset →
	A, [WHL + byte]	0 0 0 0 0 1 1 0	0 0 1 0 1 0 1 1	← Low Offset →
	A, [UUP + byte]	0 0 0 0 0 1 1 0	0 0 1 1 1 0 1 1	← Low Offset →
	A, [VVP + byte]	0 0 0 0 0 1 1 0	0 1 0 0 1 0 1 1	← Low Offset →
	A, imm24 [DE]	0 0 0 0 1 0 1 0	0 0 0 0 1 0 1 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [A]	0 0 0 0 1 0 1 0	0 0 0 1 1 0 1 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [HL]	0 0 0 0 1 0 1 0	0 0 1 0 1 0 1 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [B]	0 0 0 0 1 0 1 0	0 0 1 1 1 0 1 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, [TDE + A]	0 0 0 1 0 1 1 1	0 0 0 0 1 0 1 1	
	A, [WHL + A]	0 0 0 1 0 1 1 1	0 0 0 1 1 0 1 1	
	A, [TDE + B]	0 0 0 1 0 1 1 1	0 0 1 0 1 0 1 1	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
SUBC	A, [WHL + B]	0 0 0 1 0 1 1 1	0 0 1 1 1 0 1 1	
	A, [VVP + DE]	0 0 0 1 0 1 1 1	0 1 0 0 1 0 1 1	
	A, [VVP + HL]	0 0 0 1 0 1 1 1	0 1 0 1 1 0 1 1	
	A, [TDE + C]	0 0 0 1 0 1 1 1	0 1 1 0 1 0 1 1	
	A, [WHL + C]	0 0 0 1 0 1 1 1	0 1 1 1 1 0 1 1	
	[TDE +], A	0 0 0 1 0 1 1 0	1 0 0 0 1 0 1 1	
	[WHL +], A	0 0 0 1 0 1 1 0	1 0 0 1 1 0 1 1	
	[TDE –], A	0 0 0 1 0 1 1 0	1 0 1 0 1 0 1 1	
	[WHL –], A	0 0 0 1 0 1 1 0	1 0 1 1 1 0 1 1	
	[TDE], A	0 0 0 1 0 1 1 0	1 1 0 0 1 0 1 1	
	[WHL], A	0 0 0 1 0 1 1 0	1 1 0 1 1 0 1 1	
	[VVP], A	0 0 0 1 0 1 1 0	1 1 1 0 1 0 1 1	
	[UUP], A	0 0 0 1 0 1 1 0	1 1 1 1 1 0 1 1	
	[TDE + byte], A	0 0 0 0 0 1 1 0	1 0 0 0 1 0 1 1	← Low Offset →
	[SP + byte], A	0 0 0 0 0 1 1 0	1 0 0 1 1 0 1 1	← Low Offset →
	[WHL + byte], A	0 0 0 0 0 1 1 0	1 0 1 0 1 0 1 1	← Low Offset →
	[UUP + byte], A	0 0 0 0 0 1 1 0	1 0 1 1 1 0 1 1	← Low Offset →
	[VVP + byte], A	0 0 0 0 0 1 1 0	1 1 0 0 1 0 1 1	← Low Offset →
	imm24 [DE], A	0 0 0 0 1 0 1 0	1 0 0 0 1 0 1 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [A], A	0 0 0 0 1 0 1 0	1 0 0 1 1 0 1 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [HL], A	0 0 0 0 1 0 1 0	1 0 1 0 1 0 1 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [B], A	0 0 0 0 1 0 1 0	1 0 1 1 1 0 1 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	[TDE + A], A	0 0 0 1 0 1 1 1	1 0 0 0 1 0 1 1	
	[WHL + A], A	0 0 0 1 0 1 1 1	1 0 0 1 1 0 1 1	
	[TDE + B], A	0 0 0 1 0 1 1 1	1 0 1 0 1 0 1 1	
	[WHL + B], A	0 0 0 1 0 1 1 1	1 0 1 1 1 0 1 1	
	[VVP + DE], A	0 0 0 1 0 1 1 1	1 1 0 0 1 0 1 1	
	[VVP + HL], A	0 0 0 1 0 1 1 1	1 1 0 1 1 0 1 1	
	[TDE + C], A	0 0 0 1 0 1 1 1	1 1 1 0 1 0 1 1	
	[WHL + C], A	0 0 0 1 0 1 1 1	1 1 1 1 1 0 1 1	

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
CMP	A, #byte	1 0 1 0 1 1 1 1	← #byte →	
	r, #byte	0 1 1 1 1 1 1 1	$R_7R_6R_5R_4$ 0 0 1 1	← #byte →
	saddr2, #byte	0 1 1 0 1 1 1 1	← Saddr2-offset →	← #byte →
	saddr1, #byte	0 0 1 1 1 1 0 0	0 1 1 0 1 1 1 1	← Saddr1-offset →
		← #byte →		
	sfr, #byte	0 0 0 0 0 0 0 1	0 1 1 0 1 1 1 1	← Sfr-offset →
		← #byte →		
	r, r1	1 0 0 0 1 1 1 1	$R_7R_6R_5R_4$ 0 $R_2R_1R_0$	
	r, r2	0 0 1 1 1 1 0 0	1 0 0 0 1 1 1 1	$R_7R_6R_5R_4$ 0 $R_2R_1R_0$
	A, saddr2	1 0 0 1 1 1 1 1	← Saddr2-offset →	
	r, saddr2	0 1 1 1 1 1 1 1	$R_7R_6R_5R_4$ 0 0 0 0	← Saddr2-offset →
	r, saddr1	0 1 1 1 1 1 1 1	$R_7R_6R_5R_4$ 0 0 0 1	← Saddr1-offset →
	saddr2, r	0 1 1 1 1 1 1 1	$R_7R_6R_5R_4$ 0 1 0 0	← Saddr2-offset →
	saddr1, r	0 1 1 1 1 1 1 1	$R_7R_6R_5R_4$ 0 1 0 1	← Saddr1-offset →
	r, sfr	0 1 1 1 1 1 1 1	$R_7R_6R_5R_4$ 0 0 1 0	← Sfr-offset →
	sfr, r	0 1 1 1 1 1 1 1	$R_7R_6R_5R_4$ 0 1 1 0	← Sfr-offset →
	saddr2, saddr2’	0 0 1 0 1 0 1 0	0 0 0 0 1 1 1 1	← Saddr2’-offset →
		← Saddr2-offset →		
	saddr2, saddr1	0 0 1 0 1 0 1 0	0 0 0 1 1 1 1 1	← Saddr1-offset →
		← Saddr2-offset →		
	saddr1, saddr2	0 0 1 0 1 0 1 0	0 0 1 0 1 1 1 1	← Saddr2-offset →
		← Saddr1-offset →		
	saddr1, saddr1’	0 0 1 0 1 0 1 0	0 0 1 1 1 1 1 1	← Saddr1’-offset →
		← Saddr1-offset →		
	A, [saddrp2]	0 0 0 0 0 1 1 1	0 0 1 0 1 1 1 1	← Saddr2-offset →
	A, [saddrp1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 0 1 1 1 1
		← Saddr1-offset →		
	A, [%saddrg2]	0 0 0 0 0 1 1 1	0 0 1 1 1 1 1 1	← Saddr2-offset →
	A, [%saddrg1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 1 1 1 1 1
		← Saddr1-offset →		
	[saddrp2], A	0 0 0 0 0 1 1 1	1 0 1 0 1 1 1 1	← Saddr2-offset →
	[saddrp1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 0 1 1 1 1
		← Saddr1-offset →		
	[%saddrg2], A	0 0 0 0 0 1 1 1	1 0 1 1 1 1 1 1	← Saddr2-offset →

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
CMP	[%saddrg1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 1 1 1 1 1
		← Saddr1-offset →		
	A, !addr16	0 0 0 0 1 0 1 0	0 1 0 0 1 1 1 1	← Low Address →
		← High Address →		
	A, !!addr24	0 0 0 0 1 0 1 0	0 1 0 1 1 1 1 1	← High-w Address →
		← Low Address →	← High Address →	
	!addr16, A	0 0 0 0 1 0 1 0	1 1 0 0 1 1 1 1	← Low Address →
		← High Address →		
	!!addr24, A	0 0 0 0 1 0 1 0	1 1 0 1 1 1 1 1	← High-w Address →
		← Low Address →	← High Address →	
	A, [TDE +]	0 0 0 1 0 1 1 0	0 0 0 0 1 1 1 1	
	A, [WHL +]	0 0 0 1 0 1 1 0	0 0 0 1 1 1 1 1	
	A, [TDE –]	0 0 0 1 0 1 1 0	0 0 1 0 1 1 1 1	
	A, [WHL –]	0 0 0 1 0 1 1 0	0 0 1 1 1 1 1 1	
	A, [TDE]	0 0 0 1 0 1 1 0	0 1 0 0 1 1 1 1	
	A, [WHL]	0 0 0 1 0 1 1 0	0 1 0 1 1 1 1 1	
	A, [VVP]	0 0 0 1 0 1 1 0	0 1 1 0 1 1 1 1	
	A, [UUP]	0 0 0 1 0 1 1 0	0 1 1 1 1 1 1 1	
	A, [TDE + byte]	0 0 0 0 0 1 1 0	0 0 0 0 1 1 1 1	← Low Offset →
	A, [SP + byte]	0 0 0 0 0 1 1 0	0 0 0 1 1 1 1 1	← Low Offset →
	A, [WHL + byte]	0 0 0 0 0 1 1 0	0 0 1 0 1 1 1 1	← Low Offset →
	A, [UUP + byte]	0 0 0 0 0 1 1 0	0 0 1 1 1 1 1 1	← Low Offset →
	A, [VVP + byte]	0 0 0 0 0 1 1 0	0 1 0 0 1 1 1 1	← Low Offset →
	A, imm24 [DE]	0 0 0 0 1 0 1 0	0 0 0 0 1 1 1 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [A]	0 0 0 0 1 0 1 0	0 0 0 1 1 1 1 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [HL]	0 0 0 0 1 0 1 0	0 0 1 0 1 1 1 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [B]	0 0 0 0 1 0 1 0	0 0 1 1 1 1 1 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, [TDE + A]	0 0 0 1 0 1 1 1	0 0 0 0 1 1 1 1	
	A, [WHL + A]	0 0 0 1 0 1 1 1	0 0 0 1 1 1 1 1	
	A, [TDE + B]	0 0 0 1 0 1 1 1	0 0 1 0 1 1 1 1	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
CMP	A, [WHL + B]	0 0 0 1 0 1 1 1	0 0 1 1 1 1 1 1	
	A, [VVP + DE]	0 0 0 1 0 1 1 1	0 1 0 0 1 1 1 1	
	A, [VVP + HL]	0 0 0 1 0 1 1 1	0 1 0 1 1 1 1 1	
	A, [TDE + C]	0 0 0 1 0 1 1 1	0 1 1 0 1 1 1 1	
	A, [WHL + C]	0 0 0 1 0 1 1 1	0 1 1 1 1 1 1 1	
	[TDE +], A	0 0 0 1 0 1 1 0	1 0 0 0 1 1 1 1	
	[WHL +], A	0 0 0 1 0 1 1 0	1 0 0 1 1 1 1 1	
	[TDE –], A	0 0 0 1 0 1 1 0	1 0 1 0 1 1 1 1	
	[WHL –], A	0 0 0 1 0 1 1 0	1 0 1 1 1 1 1 1	
	[TDE], A	0 0 0 1 0 1 1 0	1 1 0 0 1 1 1 1	
	[WHL], A	0 0 0 1 0 1 1 0	1 1 0 1 1 1 1 1	
	[VVP], A	0 0 0 1 0 1 1 0	1 1 1 0 1 1 1 1	
	[UUP], A	0 0 0 1 0 1 1 0	1 1 1 1 1 1 1 1	
	[TDE + byte], A	0 0 0 0 0 1 1 0	1 0 0 0 1 1 1 1	← Low Offset →
	[SP + byte], A	0 0 0 0 0 1 1 0	1 0 0 1 1 1 1 1	← Low Offset →
	[WHL + byte], A	0 0 0 0 0 1 1 0	1 0 1 0 1 1 1 1	← Low Offset →
	[UUP + byte], A	0 0 0 0 0 1 1 0	1 0 1 1 1 1 1 1	← Low Offset →
	[VVP + byte], A	0 0 0 0 0 1 1 0	1 1 0 0 1 1 1 1	← Low Offset →
	imm24 [DE], A	0 0 0 0 1 0 1 0	1 0 0 0 1 1 1 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [A], A	0 0 0 0 1 0 1 0	1 0 0 1 1 1 1 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [HL], A	0 0 0 0 1 0 1 0	1 0 1 0 1 1 1 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [B], A	0 0 0 0 1 0 1 0	1 0 1 1 1 1 1 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	[TDE + A], A	0 0 0 1 0 1 1 1	1 0 0 0 1 1 1 1	
	[WHL + A], A	0 0 0 1 0 1 1 1	1 0 0 1 1 1 1 1	
	[TDE + B], A	0 0 0 1 0 1 1 1	1 0 1 0 1 1 1 1	
	[WHL + B], A	0 0 0 1 0 1 1 1	1 0 1 1 1 1 1 1	
	[VVP + DE], A	0 0 0 1 0 1 1 1	1 1 0 0 1 1 1 1	
	[VVP + HL], A	0 0 0 1 0 1 1 1	1 1 0 1 1 1 1 1	
	[TDE + C], A	0 0 0 1 0 1 1 1	1 1 1 0 1 1 1 1	
	[WHL + C], A	0 0 0 1 0 1 1 1	1 1 1 1 1 1 1 1	

Mnemonic	Operands	Operation Code B1 / B4 / B7	B2 / B5	B3 / B6
AND	A, #byte	1 0 1 0 1 1 0 0	← #byte →	
	r, #byte	0 1 1 1 1 1 0 0	$R_7R_6R_5R_4$ 0 0 1 1	← #byte →
	saddr2, #byte	0 1 1 0 1 1 0 0	← Saddr2-offset →	← #byte →
	saddr1, #byte	0 0 1 1 1 1 0 0	0 1 1 0 1 1 0 0	← Saddr1-offset →
		← #byte →		
	sfr, #byte	0 0 0 0 0 0 0 1	0 1 1 0 1 1 0 0	← Sfr-offset →
		← #byte →		
	r, r1	1 0 0 0 1 1 0 0	$R_7R_6R_5R_4$ 0 $R_2R_1R_0$	
	r, r2	0 0 1 1 1 1 0 0	1 0 0 0 1 1 0 0	$R_7R_6R_5R_4$ 0 $R_2R_1R_0$
	A, saddr2	1 0 0 1 1 1 0 0	← Saddr2-offset →	
	r, saddr2	0 1 1 1 1 1 0 0	$R_7R_6R_5R_4$ 0 0 0 0	← Saddr2-offset →
	r, saddr1	0 1 1 1 1 1 0 0	$R_7R_6R_5R_4$ 0 0 0 1	← Saddr1-offset →
	saddr2, r	0 1 1 1 1 1 0 0	$R_7R_6R_5R_4$ 0 1 0 0	← Saddr2-offset →
	saddr1, r	0 1 1 1 1 1 0 0	$R_7R_6R_5R_4$ 0 1 0 1	← Saddr1-offset →
	r, sfr	0 1 1 1 1 1 0 0	$R_7R_6R_5R_4$ 0 0 1 0	← Sfr-offset →
	sfr, r	0 1 1 1 1 1 0 0	$R_7R_6R_5R_4$ 0 1 1 0	← Sfr-offset →
	saddr2, saddr2'	0 0 1 0 1 0 1 0	0 0 0 0 1 1 0 0	← Saddr2'-offset →
		← Saddr2-offset →		
	saddr2, saddr1	0 0 1 0 1 0 1 0	0 0 0 1 1 1 0 0	← Saddr1-offset →
		← Saddr2-offset →		
	saddr1, saddr2	0 0 1 0 1 0 1 0	0 0 1 0 1 1 0 0	← Saddr2-offset →
		← Saddr1-offset →		
	saddr1, saddr1'	0 0 1 0 1 0 1 0	0 0 1 1 1 1 0 0	← Saddr1'-offset →
		← Saddr1-offset →		
	A, [saddrp2]	0 0 0 0 0 1 1 1	0 0 1 0 1 1 0 0	← Saddr2-offset →
	A, [saddrp1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 0 1 1 0 0
		← Saddr1-offset →		
	A, [%saddrg2]	0 0 0 0 0 1 1 1	0 0 1 1 1 1 0 0	← Saddr2-offset →
	A, [%saddrg1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 1 1 1 0 0
		← Saddr1-offset →		
	[saddrp2], A	0 0 0 0 0 1 1 1	1 0 1 0 1 1 0 0	← Saddr2-offset →
	[saddrp1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 0 1 1 0 0
		← Saddr1-offset →		
	[%saddrg2], A	0 0 0 0 0 1 1 1	1 0 1 1 1 1 0 0	← Saddr2-offset →

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
AND	[%saddrg1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 1 1 1 0 0
		← Saddr1-offset →		
	A, !addr16	0 0 0 0 1 0 1 0	0 1 0 0 1 1 0 0	← Low Address →
		← High Address →		
	A, !!addr24	0 0 0 0 1 0 1 0	0 1 0 1 1 1 0 0	← High-w Address →
		← Low Address →	← High Address →	
	!addr16, A	0 0 0 0 1 0 1 0	1 1 0 0 1 1 0 0	← Low Address →
		← High Address →		
	!!addr24, A	0 0 0 0 1 0 1 0	1 1 0 1 1 1 0 0	← High-w Address →
		← Low Address →	← High Address →	
	A, [TDE +]	0 0 0 1 0 1 1 0	0 0 0 0 1 1 0 0	
	A, [WHL +]	0 0 0 1 0 1 1 0	0 0 0 1 1 1 0 0	
	A, [TDE –]	0 0 0 1 0 1 1 0	0 0 1 0 1 1 0 0	
	A, [WHL –]	0 0 0 1 0 1 1 0	0 0 1 1 1 1 0 0	
	A, [TDE]	0 0 0 1 0 1 1 0	0 1 0 0 1 1 0 0	
	A, [WHL]	0 0 0 1 0 1 1 0	0 1 0 1 1 1 0 0	
	A, [VVP]	0 0 0 1 0 1 1 0	0 1 1 0 1 1 0 0	
	A, [UUP]	0 0 0 1 0 1 1 0	0 1 1 1 1 1 0 0	
	A, [TDE + byte]	0 0 0 0 0 1 1 0	0 0 0 0 1 1 0 0	← Low Offset →
	A, [SP + byte]	0 0 0 0 0 1 1 0	0 0 0 1 1 1 0 0	← Low Offset →
	A, [WHL + byte]	0 0 0 0 0 1 1 0	0 0 1 0 1 1 0 0	← Low Offset →
	A, [UUP + byte]	0 0 0 0 0 1 1 0	0 0 1 1 1 1 0 0	← Low Offset →
	A, [VVP + byte]	0 0 0 0 0 1 1 0	0 1 0 0 1 1 0 0	← Low Offset →
	A, imm24 [DE]	0 0 0 0 1 0 1 0	0 0 0 0 1 1 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [A]	0 0 0 0 1 0 1 0	0 0 0 1 1 1 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [HL]	0 0 0 0 1 0 1 0	0 0 1 0 1 1 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [B]	0 0 0 0 1 0 1 0	0 0 1 1 1 1 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, [TDE + A]	0 0 0 1 0 1 1 1	0 0 0 0 1 1 0 0	
	A, [WHL + A]	0 0 0 1 0 1 1 1	0 0 0 1 1 1 0 0	
	A, [TDE + B]	0 0 0 1 0 1 1 1	0 0 1 0 1 1 0 0	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
AND	A, [WHL + B]	0 0 0 1 0 1 1 1	0 0 1 1 1 1 0 0	
	A, [VVP + DE]	0 0 0 1 0 1 1 1	0 1 0 0 1 1 0 0	
	A, [VVP + HL]	0 0 0 1 0 1 1 1	0 1 0 1 1 1 0 0	
	A, [TDE + C]	0 0 0 1 0 1 1 1	0 1 1 0 1 1 0 0	
	A, [WHL + C]	0 0 0 1 0 1 1 1	0 1 1 1 1 1 0 0	
	[TDE +], A	0 0 0 1 0 1 1 0	1 0 0 0 1 1 0 0	
	[WHL +], A	0 0 0 1 0 1 1 0	1 0 0 1 1 1 0 0	
	[TDE –], A	0 0 0 1 0 1 1 0	1 0 1 0 1 1 0 0	
	[WHL –], A	0 0 0 1 0 1 1 0	1 0 1 1 1 1 0 0	
	[TDE], A	0 0 0 1 0 1 1 0	1 1 0 0 1 1 0 0	
	[WHL], A	0 0 0 1 0 1 1 0	1 1 0 1 1 1 0 0	
	[VVP], A	0 0 0 1 0 1 1 0	1 1 1 0 1 1 0 0	
	[UUP], A	0 0 0 1 0 1 1 0	1 1 1 1 1 1 0 0	
	[TDE + byte], A	0 0 0 0 0 1 1 0	1 0 0 0 1 1 0 0	← Low Offset →
	[SP + byte], A	0 0 0 0 0 1 1 0	1 0 0 1 1 1 0 0	← Low Offset →
	[WHL + byte], A	0 0 0 0 0 1 1 0	1 0 1 0 1 1 0 0	← Low Offset →
	[UUP + byte], A	0 0 0 0 0 1 1 0	1 0 1 1 1 1 0 0	← Low Offset →
	[VVP + byte], A	0 0 0 0 0 1 1 0	1 1 0 0 1 1 0 0	← Low Offset →
	imm24 [DE], A	0 0 0 0 1 0 1 0	1 0 0 0 1 1 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [A], A	0 0 0 0 1 0 1 0	1 0 0 1 1 1 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [HL], A	0 0 0 0 1 0 1 0	1 0 1 0 1 1 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [B], A	0 0 0 0 1 0 1 0	1 0 1 1 1 1 0 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	[TDE + A], A	0 0 0 1 0 1 1 1	1 0 0 0 1 1 0 0	
	[WHL + A], A	0 0 0 1 0 1 1 1	1 0 0 1 1 1 0 0	
	[TDE + B], A	0 0 0 1 0 1 1 1	1 0 1 0 1 1 0 0	
	[WHL + B], A	0 0 0 1 0 1 1 1	1 0 1 1 1 1 0 0	
	[VVP + DE], A	0 0 0 1 0 1 1 1	1 1 0 0 1 1 0 0	
	[VVP + HL], A	0 0 0 1 0 1 1 1	1 1 0 1 1 1 0 0	
	[TDE + C], A	0 0 0 1 0 1 1 1	1 1 1 0 1 1 0 0	
	[WHL + C], A	0 0 0 1 0 1 1 1	1 1 1 1 1 1 0 0	

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
OR	A, #byte	1 0 1 0 1 1 1 0	← #byte →	
	r, #byte	0 1 1 1 1 1 1 0	$R_7R_6R_5R_4$ 0 0 1 1	← #byte →
	saddr2, #byte	0 1 1 0 1 1 1 0	← Saddr2-offset →	← #byte →
	saddr1, #byte	0 0 1 1 1 1 0 0	0 1 1 0 1 1 1 0	← Saddr1-offset →
		← #byte →		
	sfr, #byte	0 0 0 0 0 0 0 1	0 1 1 0 1 1 1 0	← Sfr-offset →
		← #byte →		
	r, r1	1 0 0 0 1 1 1 0	$R_7R_6R_5R_4$ 0 $R_2R_1R_0$	
	r, r2	0 0 1 1 1 1 0 0	1 0 0 0 1 1 1 0	$R_7R_6R_5R_4$ 0 $R_2R_1R_0$
	A, saddr2	1 0 0 1 1 1 1 0	← Saddr2-offset →	
	r, saddr2	0 1 1 1 1 1 1 0	$R_7R_6R_5R_4$ 0 0 0 0	← Saddr2-offset →
	r, saddr1	0 1 1 1 1 1 1 0	$R_7R_6R_5R_4$ 0 0 0 1	← Saddr1-offset →
	saddr2, r	0 1 1 1 1 1 1 0	$R_7R_6R_5R_4$ 0 1 0 0	← Saddr2-offset →
	saddr1, r	0 1 1 1 1 1 1 0	$R_7R_6R_5R_4$ 0 1 0 1	← Saddr1-offset →
	r, sfr	0 1 1 1 1 1 1 0	$R_7R_6R_5R_4$ 0 0 1 0	← Sfr-offset →
	sfr, r	0 1 1 1 1 1 1 0	$R_7R_6R_5R_4$ 0 1 1 0	← Sfr-offset →
	saddr2, saddr2'	0 0 1 0 1 0 1 0	0 0 0 0 1 1 1 0	← Saddr2'-offset →
		← Saddr2-offset →		
	saddr2, saddr1	0 0 1 0 1 0 1 0	0 0 0 1 1 1 1 0	← Saddr1-offset →
		← Saddr2-offset →		
	saddr1, saddr2	0 0 1 0 1 0 1 0	0 0 1 0 1 1 1 0	← Saddr2-offset →
		← Saddr1-offset →		
	saddr1, saddr1'	0 0 1 0 1 0 1 0	0 0 1 1 1 1 1 0	← Saddr1'-offset →
		← Saddr1-offset →		
	A, [saddrp2]	0 0 0 0 0 1 1 1	0 0 1 0 1 1 1 0	← Saddr2-offset →
	A, [saddrp1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 0 1 1 1 0
		← Saddr1-offset →		
	A, [%saddrg2]	0 0 0 0 0 1 1 1	0 0 1 1 1 1 1 0	← Saddr2-offset →
	A, [%saddrg1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 1 1 1 1 0
		← Saddr1-offset →		
	[saddrp2], A	0 0 0 0 0 1 1 1	1 0 1 0 1 1 1 0	← Saddr2-offset →
	[saddrp1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 0 1 1 1 0
		← Saddr1-offset →		
	[%saddrg2], A	0 0 0 0 0 1 1 1	1 0 1 1 1 1 1 0	← Saddr2-offset →

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
OR	[%saddrg1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 1 1 1 1 0
		← Saddr1-offset →		
	A, !addr16	0 0 0 0 1 0 1 0	0 1 0 0 1 1 1 0	← Low Address →
		← High Address →		
	A, !!addr24	0 0 0 0 1 0 1 0	0 1 0 1 1 1 1 0	← High-w Address →
		← Low Address →	← High Address →	
	!addr16, A	0 0 0 0 1 0 1 0	1 1 0 0 1 1 1 0	← Low Address →
		← High Address →		
	!!addr24, A	0 0 0 0 1 0 1 0	1 1 0 1 1 1 1 0	← High-w Address →
		← Low Address →	← High Address →	
	A, [TDE +]	0 0 0 1 0 1 1 0	0 0 0 0 1 1 1 0	
	A, [WHL +]	0 0 0 1 0 1 1 0	0 0 0 1 1 1 1 0	
	A, [TDE –]	0 0 0 1 0 1 1 0	0 0 1 0 1 1 1 0	
	A, [WHL –]	0 0 0 1 0 1 1 0	0 0 1 1 1 1 1 0	
	A, [TDE]	0 0 0 1 0 1 1 0	0 1 0 0 1 1 1 0	
	A, [WHL]	0 0 0 1 0 1 1 0	0 1 0 1 1 1 1 0	
	A, [VVP]	0 0 0 1 0 1 1 0	0 1 1 0 1 1 1 0	
	A, [UUP]	0 0 0 1 0 1 1 0	0 1 1 1 1 1 1 0	
	A, [TDE + byte]	0 0 0 0 0 1 1 0	0 0 0 0 1 1 1 0	← Low Offset →
	A, [SP + byte]	0 0 0 0 0 1 1 0	0 0 0 1 1 1 1 0	← Low Offset →
	A, [WHL + byte]	0 0 0 0 0 1 1 0	0 0 1 0 1 1 1 0	← Low Offset →
	A, [UUP + byte]	0 0 0 0 0 1 1 0	0 0 1 1 1 1 1 0	← Low Offset →
	A, [VVP + byte]	0 0 0 0 0 1 1 0	0 1 0 0 1 1 1 0	← Low Offset →
	A, imm24 [DE]	0 0 0 0 1 0 1 0	0 0 0 0 1 1 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [A]	0 0 0 0 1 0 1 0	0 0 0 1 1 1 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [HL]	0 0 0 0 1 0 1 0	0 0 1 0 1 1 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [B]	0 0 0 0 1 0 1 0	0 0 1 1 1 1 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, [TDE + A]	0 0 0 1 0 1 1 1	0 0 0 0 1 1 1 0	
	A, [WHL + A]	0 0 0 1 0 1 1 1	0 0 0 1 1 1 1 0	
	A, [TDE + B]	0 0 0 1 0 1 1 1	0 0 1 0 1 1 1 0	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
OR	A, [WHL + B]	0 0 0 1 0 1 1 1	0 0 1 1 1 1 1 0	
	A, [VVP + DE]	0 0 0 1 0 1 1 1	0 1 0 0 1 1 1 0	
	A, [VVP + HL]	0 0 0 1 0 1 1 1	0 1 0 1 1 1 1 0	
	A, [TDE + C]	0 0 0 1 0 1 1 1	0 1 1 0 1 1 1 0	
	A, [WHL + C]	0 0 0 1 0 1 1 1	0 1 1 1 1 1 1 0	
	[TDE +], A	0 0 0 1 0 1 1 0	1 0 0 0 1 1 1 0	
	[WHL +], A	0 0 0 1 0 1 1 0	1 0 0 1 1 1 1 0	
	[TDE –], A	0 0 0 1 0 1 1 0	1 0 1 0 1 1 1 0	
	[WHL –], A	0 0 0 1 0 1 1 0	1 0 1 1 1 1 1 0	
	[TDE], A	0 0 0 1 0 1 1 0	1 1 0 0 1 1 1 0	
	[WHL], A	0 0 0 1 0 1 1 0	1 1 0 1 1 1 1 0	
	[VVP], A	0 0 0 1 0 1 1 0	1 1 1 0 1 1 1 0	
	[UUP], A	0 0 0 1 0 1 1 0	1 1 1 1 1 1 1 0	
	[TDE + byte], A	0 0 0 0 0 1 1 0	1 0 0 0 1 1 1 0	← Low Offset →
	[SP + byte], A	0 0 0 0 0 1 1 0	1 0 0 1 1 1 1 0	← Low Offset →
	[WHL + byte], A	0 0 0 0 0 1 1 0	1 0 1 0 1 1 1 0	← Low Offset →
	[UUP + byte], A	0 0 0 0 0 1 1 0	1 0 1 1 1 1 1 0	← Low Offset →
	[VVP + byte], A	0 0 0 0 0 1 1 0	1 1 0 0 1 1 1 0	← Low Offset →
	imm24 [DE], A	0 0 0 0 1 0 1 0	1 0 0 0 1 1 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [A], A	0 0 0 0 1 0 1 0	1 0 0 1 1 1 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [HL], A	0 0 0 0 1 0 1 0	1 0 1 0 1 1 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [B], A	0 0 0 0 1 0 1 0	1 0 1 1 1 1 1 0	← Low Offset →
		← High Offset →	← High-w Offset →	
	[TDE + A], A	0 0 0 1 0 1 1 1	1 0 0 0 1 1 1 0	
	[WHL + A], A	0 0 0 1 0 1 1 1	1 0 0 1 1 1 1 0	
	[TDE + B], A	0 0 0 1 0 1 1 1	1 0 1 0 1 1 1 0	
	[WHL + B], A	0 0 0 1 0 1 1 1	1 0 1 1 1 1 1 0	
	[VVP + DE], A	0 0 0 1 0 1 1 1	1 1 0 0 1 1 1 0	
	[VVP + HL], A	0 0 0 1 0 1 1 1	1 1 0 1 1 1 1 0	
	[TDE + C], A	0 0 0 1 0 1 1 1	1 1 1 0 1 1 1 0	
	[WHL + C], A	0 0 0 1 0 1 1 1	1 1 1 1 1 1 1 0	

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
XOR	A, #byte	1 0 1 0 1 1 0 1	← #byte →	
	r, #byte	0 1 1 1 1 1 0 1	$R_7R_6R_5R_4$ 0 0 1 1	← #byte →
	saddr2, #byte	0 1 1 0 1 1 0 1	← Saddr2-offset →	← #byte →
	saddr1, #byte	0 0 1 1 1 1 0 0	0 1 1 0 1 1 0 1	← Saddr1-offset →
		← #byte →		
	sfr, #byte	0 0 0 0 0 0 0 1	0 1 1 0 1 1 0 1	← Sfr-offset →
		← #byte →		
	r, r1	1 0 0 0 1 1 0 1	$R_7R_6R_5R_4$ 0 $R_2R_1R_0$	
	r, r2	0 0 1 1 1 1 0 0	1 0 0 0 1 1 0 1	$R_7R_6R_5R_4$ 0 $R_2R_1R_0$
	A, saddr2	1 0 0 1 1 1 0 1	← Saddr2-offset →	
	r, saddr2	0 1 1 1 1 1 0 1	$R_7R_6R_5R_4$ 0 0 0 0	← Saddr2-offset →
	r, saddr1	0 1 1 1 1 1 0 1	$R_7R_6R_5R_4$ 0 0 0 1	← Saddr1-offset →
	saddr2, r	0 1 1 1 1 1 0 1	$R_7R_6R_5R_4$ 0 1 0 0	← Saddr2-offset →
	saddr1, r	0 1 1 1 1 1 0 1	$R_7R_6R_5R_4$ 0 1 0 1	← Saddr1-offset →
	r, sfr	0 1 1 1 1 1 0 1	$R_7R_6R_5R_4$ 0 0 1 0	← Sfr-offset →
	sfr, r	0 1 1 1 1 1 0 1	$R_7R_6R_5R_4$ 0 1 1 0	← Sfr-offset →
	saddr2, saddr2'	0 0 1 0 1 0 1 0	0 0 0 0 1 1 0 1	← Saddr2'-offset →
		← Saddr2-offset →		
	saddr2, saddr1	0 0 1 0 1 0 1 0	0 0 0 1 1 1 0 1	← Saddr1-offset →
		← Saddr2-offset →		
	saddr1, saddr2	0 0 1 0 1 0 1 0	0 0 1 0 1 1 0 1	← Saddr2-offset →
		← Saddr1-offset →		
	saddr1, saddr1'	0 0 1 0 1 0 1 0	0 0 1 1 1 1 0 1	← Saddr1'-offset →
		← Saddr1-offset →		
	A, [saddrp2]	0 0 0 0 0 1 1 1	0 0 1 0 1 1 0 1	← Saddr2-offset →
	A, [saddrp1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 0 1 1 0 1
		← Saddr1-offset →		
	A, [%saddrg2]	0 0 0 0 0 1 1 1	0 0 1 1 1 1 0 1	← Saddr2-offset →
	A, [%saddrg1]	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	0 0 1 1 1 1 0 1
		← Saddr1-offset →		
	[saddrp2], A	0 0 0 0 0 1 1 1	1 0 1 0 1 1 0 1	← Saddr2-offset →
	[saddrp1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 0 1 1 0 1
		← Saddr1-offset →		
	[%saddrg2], A	0 0 0 0 0 1 1 1	1 0 1 1 1 1 0 1	← Saddr2-offset →

(Continued on next page)

Mnemonic	Operands	Operation Code B1 / B4 / B7	B2 / B5	B3 / B6
XOR	[%saddrg1], A	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 0 1 1 1 1 0 1
		← Saddr1-offset →		
	A, !addr16	0 0 0 0 1 0 1 0	0 1 0 0 1 1 0 1	← Low Address →
		← High Address →		
	A, !!addr24	0 0 0 0 1 0 1 0	0 1 0 1 1 1 0 1	← High-w Address →
		← Low Address →	← High Address →	
	!addr16, A	0 0 0 0 1 0 1 0	1 1 0 0 1 1 0 1	← Low Address →
		← High Address →		
	!!addr24, A	0 0 0 0 1 0 1 0	1 1 0 1 1 1 0 1	← High-w Address →
		← Low Address →	← High Address →	
	A, [TDE +]	0 0 0 1 0 1 1 0	0 0 0 0 1 1 0 1	
	A, [WHL +]	0 0 0 1 0 1 1 0	0 0 0 1 1 1 0 1	
	A, [TDE –]	0 0 0 1 0 1 1 0	0 0 1 0 1 1 0 1	
	A, [WHL –]	0 0 0 1 0 1 1 0	0 0 1 1 1 1 0 1	
	A, [TDE]	0 0 0 1 0 1 1 0	0 1 0 0 1 1 0 1	
	A, [WHL]	0 0 0 1 0 1 1 0	0 1 0 1 1 1 0 1	
	A, [VVP]	0 0 0 1 0 1 1 0	0 1 1 0 1 1 0 1	
	A, [UUP]	0 0 0 1 0 1 1 0	0 1 1 1 1 1 0 1	
	A, [TDE + byte]	0 0 0 0 0 1 1 0	0 0 0 0 1 1 0 1	← Low Offset →
	A, [SP + byte]	0 0 0 0 0 1 1 0	0 0 0 1 1 1 0 1	← Low Offset →
	A, [WHL + byte]	0 0 0 0 0 1 1 0	0 0 1 0 1 1 0 1	← Low Offset →
	A, [UUP + byte]	0 0 0 0 0 1 1 0	0 0 1 1 1 1 0 1	← Low Offset →
	A, [VVP + byte]	0 0 0 0 0 1 1 0	0 1 0 0 1 1 0 1	← Low Offset →
	A, imm24 [DE]	0 0 0 0 1 0 1 0	0 0 0 0 1 1 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [A]	0 0 0 0 1 0 1 0	0 0 0 1 1 1 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [HL]	0 0 0 0 1 0 1 0	0 0 1 0 1 1 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, imm24 [B]	0 0 0 0 1 0 1 0	0 0 1 1 1 1 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	A, [TDE + A]	0 0 0 1 0 1 1 1	0 0 0 0 1 1 0 1	
	A, [WHL + A]	0 0 0 1 0 1 1 1	0 0 0 1 1 1 0 1	
	A, [TDE + B]	0 0 0 1 0 1 1 1	0 0 1 0 1 1 0 1	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
XOR	A, [WHL + B]	0 0 0 1 0 1 1 1	0 0 1 1 1 1 0 1	
	A, [VVP + DE]	0 0 0 1 0 1 1 1	0 1 0 0 1 1 0 1	
	A, [VVP + HL]	0 0 0 1 0 1 1 1	0 1 0 1 1 1 0 1	
	A, [TDE + C]	0 0 0 1 0 1 1 1	0 1 1 0 1 1 0 1	
	A, [WHL + C]	0 0 0 1 0 1 1 1	0 1 1 1 1 1 0 1	
	[TDE +], A	0 0 0 1 0 1 1 0	1 0 0 0 1 1 0 1	
	[WHL +], A	0 0 0 1 0 1 1 0	1 0 0 1 1 1 0 1	
	[TDE –], A	0 0 0 1 0 1 1 0	1 0 1 0 1 1 0 1	
	[WHL –], A	0 0 0 1 0 1 1 0	1 0 1 1 1 1 0 1	
	[TDE], A	0 0 0 1 0 1 1 0	1 1 0 0 1 1 0 1	
	[WHL], A	0 0 0 1 0 1 1 0	1 1 0 1 1 1 0 1	
	[VVP], A	0 0 0 1 0 1 1 0	1 1 1 0 1 1 0 1	
	[UUP], A	0 0 0 1 0 1 1 0	1 1 1 1 1 1 0 1	
	[TDE + byte], A	0 0 0 0 0 1 1 0	1 0 0 0 1 1 0 1	← Low Offset →
	[SP + byte], A	0 0 0 0 0 1 1 0	1 0 0 1 1 1 0 1	← Low Offset →
	[WHL + byte], A	0 0 0 0 0 1 1 0	1 0 1 0 1 1 0 1	← Low Offset →
	[UUP + byte], A	0 0 0 0 0 1 1 0	1 0 1 1 1 1 0 1	← Low Offset →
	[VVP + byte], A	0 0 0 0 0 1 1 0	1 1 0 0 1 1 0 1	← Low Offset →
	imm24 [DE], A	0 0 0 0 1 0 1 0	1 0 0 0 1 1 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [A], A	0 0 0 0 1 0 1 0	1 0 0 1 1 1 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [HL], A	0 0 0 0 1 0 1 0	1 0 1 0 1 1 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	imm24 [B], A	0 0 0 0 1 0 1 0	1 0 1 1 1 1 0 1	← Low Offset →
		← High Offset →	← High-w Offset →	
	[TDE + A], A	0 0 0 1 0 1 1 1	1 0 0 0 1 1 0 1	
	[WHL + A], A	0 0 0 1 0 1 1 1	1 0 0 1 1 1 0 1	
	[TDE + B], A	0 0 0 1 0 1 1 1	1 0 1 0 1 1 0 1	
	[WHL + B], A	0 0 0 1 0 1 1 1	1 0 1 1 1 1 0 1	
	[VVP + DE], A	0 0 0 1 0 1 1 1	1 1 0 0 1 1 0 1	
	[VVP + HL], A	0 0 0 1 0 1 1 1	1 1 0 1 1 1 0 1	
	[TDE + C], A	0 0 0 1 0 1 1 1	1 1 1 0 1 1 0 1	
	[WHL + C], A	0 0 0 1 0 1 1 1	1 1 1 1 1 1 0 1	

(7) 16-bit operation instructions: ADDW, SUBW, CMPW

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
ADDW	AX, #word	0 0 1 0 1 1 0 1	← Low Byte →	← High Byte →
	rp, #word	0 1 1 1 1 0 0 0	$P_7 P_6 P_5$ 0 1 0 1 1	← Low Byte →
		← High Byte →		
	rp, rp'	1 0 0 0 1 0 0 0	$P_7 P_6 P_5$ 0 1 $P_2 P_1 P_0$	
	AX, saddrp2	0 0 0 1 1 1 0 1	← Saddr2-offset →	
	rp, saddrp2	0 1 1 1 1 0 0 0	$P_7 P_6 P_5$ 0 1 0 0 0	← Saddr2-offset →
	rp, saddrp1	0 1 1 1 1 0 0 0	$P_7 P_6 P_5$ 0 1 0 0 1	← Saddr1-offset →
	saddrp2, rp	0 1 1 1 1 0 0 0	$P_7 P_6 P_5$ 0 1 1 0 0	← Saddr2-offset →
	saddrp1, rp	0 1 1 1 1 0 0 0	$P_7 P_6 P_5$ 0 1 1 0 1	← Saddr1-offset →
	rp, sfrp	0 1 1 1 1 0 0 0	$P_7 P_6 P_5$ 0 1 0 1 0	← Sfr-offset →
	sfrp, rp	0 1 1 1 1 0 0 0	$P_7 P_6 P_5$ 0 1 1 1 0	← Sfr-offset →
	saddrp2, #word	0 0 0 0 1 1 0 1	← Saddr2-offset →	← Low Byte →
		← High Byte →		
	saddrp1, #word	0 0 1 1 1 1 0 0	0 0 0 0 1 1 0 1	← Saddr1-offset →
		← Low Byte →	← High Byte →	
	sfrp, #word	0 0 0 0 0 0 0 1	0 0 0 0 1 1 0 1	← Sfr-offset →
		← Low Byte →	← High Byte →	
	saddrp2, saddrp2'	0 0 1 0 1 0 1 0	1 0 0 0 1 1 0 1	← Saddr2'-offset →
		← Saddr2-offset →		
	saddrp2, saddrp1	0 0 1 0 1 0 1 0	1 0 0 1 1 1 0 1	← Saddr1-offset →
		← Saddr2-offset →		
	saddrp1, saddrp2	0 0 1 0 1 0 1 0	1 0 1 0 1 1 0 1	← Saddr2-offset →
		← Saddr1-offset →		
	saddrp1, saddrp1'	0 0 1 0 1 0 1 0	1 0 1 1 1 1 0 1	← Saddr1'-offset →
		← Saddr1-offset →		
SUBW	AX, #word	0 0 1 0 1 1 1 0	← Low Byte →	← High Byte →
	rp, #word	0 1 1 1 1 0 1 0	$P_7 P_6 P_5$ 0 1 0 1 1	← Low Byte →
		← High Byte →		
	rp, rp'	1 0 0 0 1 0 1 0	$P_7 P_6 P_5$ 0 1 $P_2 P_1 P_0$	
	AX, saddrp2	0 0 0 1 1 1 1 0	← Saddr2-offset →	
	rp, saddrp2	0 1 1 1 1 0 1 0	$P_7 P_6 P_5$ 0 1 0 0 0	← Saddr2-offset →
	rp, saddrp1	0 1 1 1 1 0 1 0	$P_7 P_6 P_5$ 0 1 0 0 1	← Saddr1-offset →
	saddrp2, rp	0 1 1 1 1 0 1 0	$P_7 P_6 P_5$ 0 1 1 0 0	← Saddr2-offset →

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
SUBW	saddrp1, rp	0 1 1 1 1 0 1 0	$P_7 P_6 P_5$ 0 1 1 0 1	← Saddr1-offset →
	rp, sfrp	0 1 1 1 1 0 1 0	$P_7 P_6 P_5$ 0 1 0 1 0	← Sfr-offset →
	sfrp, rp	0 1 1 1 1 0 1 0	$P_7 P_6 P_5$ 0 1 1 1 0	← Sfr-offset →
	saddrp2, #word	0 0 0 0 1 1 1 0	← Saddr2-offset →	← Low Byte →
		← High Byte →		
	saddrp1, #word	0 0 1 1 1 1 0 0	0 0 0 0 1 1 1 0	← Saddr1-offset →
		← Low Byte →	← High Byte →	
	sfrp, #word	0 0 0 0 0 0 0 1	0 0 0 0 1 1 1 0	← Sfr-offset →
		← Low Byte →	← High Byte →	
	saddrp2, saddrp2’	0 0 1 0 1 0 1 0	1 0 0 0 1 1 1 0	← Saddr2’-offset →
		← Saddr2-offset →		
	saddrp2, saddrp1	0 0 1 0 1 0 1 0	1 0 0 1 1 1 1 0	← Saddr1-offset →
		← Saddr2-offset →		
	saddrp1, saddrp2	0 0 1 0 1 0 1 0	1 0 1 0 1 1 1 0	← Saddr2-offset →
		← Saddr1-offset →		
	saddrp1, saddrp1’	0 0 1 0 1 0 1 0	1 0 1 1 1 1 1 0	← Saddr1’-offset →
		← Saddr1-offset →		
CMPW	AX, #word	0 0 1 0 1 1 1 1	← Low Byte →	← High Byte →
	rp, #word	0 1 1 1 1 1 1 1	$P_7 P_6 P_5$ 0 1 0 1 1	← Low Byte →
		← High Byte →		
	rp, rp’	1 0 0 0 1 1 1 1	$P_7 P_6 P_5$ 0 1 $P_2 P_1 P_0$	
	AX, saddrp2	0 0 0 1 1 1 1 1	← Saddr2-offset →	
	rp, saddrp2	0 1 1 1 1 1 1 1	$P_7 P_6 P_5$ 0 1 0 0 0	← Saddr2-offset →
	rp, saddrp1	0 1 1 1 1 1 1 1	$P_7 P_6 P_5$ 0 1 0 0 1	← Saddr1-offset →
	saddrp2, rp	0 1 1 1 1 1 1 1	$P_7 P_6 P_5$ 0 1 1 0 0	← Saddr2-offset →
	saddrp1, rp	0 1 1 1 1 1 1 1	$P_7 P_6 P_5$ 0 1 1 0 1	← Saddr1-offset →
	rp, sfrp	0 1 1 1 1 1 1 1	$P_7 P_6 P_5$ 0 1 0 1 0	← Sfr-offset →
	sfrp, rp	0 1 1 1 1 1 1 1	$P_7 P_6 P_5$ 0 1 1 1 0	← Sfr-offset →
	saddrp2, #word	0 0 0 0 1 1 1 1	← Saddr2-offset →	← Low Byte →
		← High Byte →		
	saddrp1, #word	0 0 1 1 1 1 0 0	0 0 0 0 1 1 1 1	← Saddr1-offset →
		← Low Byte →	← High Byte →	
	sfrp, #word	0 0 0 0 0 0 0 1	0 0 0 0 1 1 1 1	← Sfr-offset →
		← Low Byte →	← High Byte →	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
CMPW	saddrp2, saddrp2'	0 0 1 0 1 0 1 0	1 0 0 0 1 1 1 1	← Saddr2'-offset →
		← Saddr2-offset →		
	saddrp2, saddrp1	0 0 1 0 1 0 1 0	1 0 0 1 1 1 1 1	← Saddr1-offset →
		← Saddr2-offset →		
	saddrp1, saddrp2	0 0 1 0 1 0 1 0	1 0 1 0 1 1 1 1	← Saddr2-offset →
		← Saddr1-offset →		
	saddrp1, saddrp1'	0 0 1 0 1 0 1 0	1 0 1 1 1 1 1 1	← Saddr1'-offset →
		← Saddr1-offset →		

(8) 24-bit operation instructions: ADDG, SUBG

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
ADDG	rg, rg'	1 0 0 0 1 0 0 0	1 G_6G_5 1 1 G_2G_1 1	
	rg, #imm24	0 1 1 1 1 0 0 0	1 G_6G_5 1 1 0 1 1	← Low Byte →
		← High Byte →	← High-w Byte →	
	WHL, saddrg2	0 1 1 1 1 0 0 0	1 1 1 1 1 0 0 0	← Saddr2-offset →
	WHL, saddrg1	0 1 1 1 1 0 0 0	1 1 1 1 1 0 0 1	← Saddr1-offset →
SUBG	rg, rg'	1 0 0 0 1 0 1 0	1 G_6G_5 1 1 G_2G_1 1	
	rg, #imm24	0 1 1 1 1 0 1 0	1 G_6G_5 1 1 0 1 1	← Low Byte →
		← High Byte →	← High-w Byte →	
	WHL, saddrg2	0 1 1 1 1 0 1 0	1 1 1 1 1 0 0 0	← Saddr2-offset →
	WHL, saddrg1	0 1 1 1 1 0 1 0	1 1 1 1 1 0 0 1	← Saddr1-offset →

(9) Multiplication instructions: MULU, MULUW, MULW, DIVUW, DIVUX

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
MULU	r1	0 0 0 0 0 1 0 1	0 0 0 0 1 R_2 R_1 R_0	
	r2	0 0 1 1 1 1 0 0	0 0 0 0 0 1 0 1	0 0 0 0 1 R_2 R_1 R_0
MULUW	rp	0 0 0 0 0 1 0 1	0 0 1 0 1 P_2 P_1 P_0	
MULW	rp	0 0 0 0 0 1 0 1	0 0 1 1 1 P_2 P_1 P_0	
DIVUW	r1	0 0 0 0 0 1 0 1	0 0 0 1 1 R_2 R_1 R_0	
	r2	0 0 1 1 1 1 0 0	0 0 0 0 0 1 0 1	0 0 0 1 1 R_2 R_1 R_0
DIVUX	rp	0 0 0 0 0 1 0 1	1 1 1 0 1 P_2 P_1 P_0	

(10) Special operation instructions: MACW, MACSW, SACW

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
MACW	byte	0 0 0 0 0 1 1 1	1 0 0 0 0 1 0 1	← byte →
MACSW	byte	0 0 0 0 0 1 1 1	1 0 0 1 0 1 0 1	← byte →
SACW	[TDE +], [WHL +]	0 0 0 0 1 0 0 1	0 1 1 0 0 1 0 0	0 1 0 0 0 0 0 1
		0 1 0 0 0 1 1 0		

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

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
INC	r1	1 1 0 0 0 $R_2 R_1 R_0$		
	r2	0 0 1 1 1 1 0 0	1 1 0 0 0 $R_2 R_1 R_0$	
	saddr2	0 0 1 0 0 1 1 0	← Saddr2-offset →	
	saddr1	0 0 1 1 1 1 0 0	0 0 1 0 0 1 1 0	← Saddr1-offset →
DEC	r1	1 1 0 0 1 $R_2 R_1 R_0$		
	r2	0 0 1 1 1 1 0 0	1 1 0 0 1 $R_2 R_1 R_0$	
	saddr2	0 0 1 0 0 1 1 1	← Saddr2-offset →	
	saddr1	0 0 1 1 1 1 0 0	0 0 1 0 0 1 1 1	← Saddr1-offset →
INCW	RP0	0 0 1 1 1 1 1 0	0 0 0 0 1 1 0 1	
	RP1	0 0 1 1 1 1 1 0	0 0 1 0 1 1 0 1	
	RP2	0 0 1 1 1 1 1 0	0 1 0 0 1 1 0 1	
	RP3	0 0 1 1 1 1 1 0	0 1 1 0 1 1 0 1	
	VP (RP4)	0 1 0 0 0 1 0 0		
	UP (RP5)	0 1 0 0 0 1 0 1		
	DE (RP6)	0 1 0 0 0 1 1 0		
	HL (RP7)	0 1 0 0 0 1 1 1		
	saddrp2	0 0 0 0 0 1 1 1	1 1 1 0 1 0 0 0	← Saddr2-offset →
	saddrp1	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 1 1 0 1 0 0 0
		← Saddr1-offset →		
DECW	RP0	0 0 1 1 1 1 1 0	0 0 0 0 1 1 1 1	
	RP1	0 0 1 1 1 1 1 0	0 0 1 0 1 1 1 1	
	RP2	0 0 1 1 1 1 1 0	0 1 0 0 1 1 1 1	
	RP3	0 0 1 1 1 1 1 0	0 1 1 0 1 1 1 1	
	VP (RP4)	0 1 0 0 1 1 0 0		
	UP (RP5)	0 1 0 0 1 1 0 1		
	DE (RP6)	0 1 0 0 1 1 1 0		
	HL (RP7)	0 1 0 0 1 1 1 1		
	saddrp2	0 0 0 0 0 1 1 1	1 1 1 0 1 0 0 1	← Saddr2-offset →
	saddrp1	0 0 1 1 1 1 0 0	0 0 0 0 0 1 1 1	1 1 1 0 1 0 0 1
		← Saddr1-offset →		
INCG	rg	0 0 1 1 1 1 1 0	1 $G_6 G_5$ 1 1 1 0 1	
DECG	rg	0 0 1 1 1 1 1 0	1 $G_6 G_5$ 1 1 1 1 1	

(12) Adjustment instructions: ADJBA, ADJBS, CVTBW

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
ADJBA		0 0 0 0 0 1 0 1	1 1 1 1 1 1 1 0	
ADJBS		0 0 0 0 0 1 0 1	1 1 1 1 1 1 1 1	
CVTBW		0 0 0 0 0 1 0 0		

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

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
ROR	r1, n	0 0 1 1 0 0 0 0	0 1 N_2 N_1 N_0 R_2 R_1 R_0	
	r2, n	0 0 1 1 1 1 0 0	0 0 1 1 0 0 0 0	0 1 N_2 N_1 N_0 R_2 R_1 R_0
ROL	r1, n	0 0 1 1 0 0 0 1	0 1 N_2 N_1 N_0 R_2 R_1 R_0	
	r2, n	0 0 1 1 1 1 0 0	0 0 1 1 0 0 0 1	0 1 N_2 N_1 N_0 R_2 R_1 R_0
RORC	r1, n	0 0 1 1 0 0 0 0	0 0 N_2 N_1 N_0 R_2 R_1 R_0	
	r2, n	0 0 1 1 1 1 0 0	0 0 1 1 0 0 0 0	0 0 N_2 N_1 N_0 R_2 R_1 R_0
ROLC	r1, n	0 0 1 1 0 0 0 1	0 0 N_2 N_1 N_0 R_2 R_1 R_0	
	r2, n	0 0 1 1 1 1 0 0	0 0 1 1 0 0 0 1	0 0 N_2 N_1 N_0 R_2 R_1 R_0
SHR	r1, n	0 0 1 1 0 0 0 0	1 0 N_2 N_1 N_0 R_2 R_1 R_0	
	r2, n	0 0 1 1 1 1 0 0	0 0 1 1 0 0 0 0	1 0 N_2 N_1 N_0 R_2 R_1 R_0
SHL	r1, n	0 0 1 1 0 0 0 1	1 0 N_2 N_1 N_0 R_2 R_1 R_0	
	r2, n	0 0 1 1 1 1 0 0	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 P_0	
SHLW	rp, n	0 0 1 1 0 0 0 1	1 1 N_2 N_1 N_0 P_2 P_1 P_0	
ROR4	mem3	0 0 0 0 0 1 0 1	1 0 0 0 1 P_2 P_1 P_0	
ROL4	mem3	0 0 0 0 0 1 0 1	1 0 0 1 1 P_2 P_1 P_0	

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

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
MOV1	CY, saddr2. bit	0 0 0 0 1 0 0 0	0 0 0 0 0 $B_2 B_1 B_0$	← Saddr2-offset →
	CY, saddr1. bit	0 0 1 1 1 1 0 0	0 0 0 0 1 0 0 0	0 0 0 0 0 $B_2 B_1 B_0$
		← Saddr1-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, X. bit	0 0 0 0 0 0 1 1	0 0 0 0 0 $B_2 B_1 B_0$	
	CY, A. bit	0 0 0 0 0 0 1 1	0 0 0 0 1 $B_2 B_1 B_0$	
	CY, PSWL. bit	0 0 0 0 0 0 1 0	0 0 0 0 0 $B_2 B_1 B_0$	
	CY, PSWH. bit	0 0 0 0 0 0 1 0	0 0 0 0 1 $B_2 B_1 B_0$	
	CY, [TDE]. bit	0 0 1 1 1 1 0 1	0 0 0 0 0 $B_2 B_1 B_0$	
	CY, [WHL]. bit	0 0 1 1 1 1 0 1	0 0 0 0 1 $B_2 B_1 B_0$	
	CY, !addr16.bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	0 0 0 0 0 $B_2 B_1 B_0$
		← Low Address →	← High Address →	
	CY, !!addr24.bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	0 0 0 0 1 $B_2 B_1 B_0$
		← High-w Address →	← Low Address →	← High Address →
	saddr2. bit, CY	0 0 0 0 1 0 0 0	0 0 0 1 0 $B_2 B_1 B_0$	← Saddr2-offset →
	saddr1. bit, CY	0 0 1 1 1 1 0 0	0 0 0 0 1 0 0 0	0 0 0 1 0 $B_2 B_1 B_0$
		← Saddr1-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 →
	X. bit, CY	0 0 0 0 0 0 1 1	0 0 0 1 0 $B_2 B_1 B_0$	
	A. bit, CY	0 0 0 0 0 0 1 1	0 0 0 1 1 $B_2 B_1 B_0$	
	PSWL. bit, CY	0 0 0 0 0 0 1 0	0 0 0 1 0 $B_2 B_1 B_0$	
	PSWH. bit, CY	0 0 0 0 0 0 1 0	0 0 0 1 1 $B_2 B_1 B_0$	
	[TDE]. bit, CY	0 0 1 1 1 1 0 1	0 0 0 1 0 $B_2 B_1 B_0$	
	[WHL]. bit, CY	0 0 1 1 1 1 0 1	0 0 0 1 1 $B_2 B_1 B_0$	
	!addr16. bit, CY	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	0 0 0 1 0 $B_2 B_1 B_0$
		← Low Address →	← High Address →	
	!addr24. bit, CY	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	0 0 0 1 1 $B_2 B_1 B_0$
		← High-w Address →	← Low Address →	← High Address →
AND1	CY, saddr2. bit	0 0 0 0 1 0 0 0	0 0 1 0 0 $B_2 B_1 B_0$	← Saddr2-offset →
	CY, saddr1. bit	0 0 1 1 1 1 0 0	0 0 0 0 1 0 0 0	0 0 1 0 0 $B_2 B_1 B_0$
		← Saddr1-offset →		
	CY,/saddr2. bit	0 0 0 0 1 0 0 0	0 0 1 1 0 $B_2 B_1 B_0$	← Saddr2-offset →

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
AND1	CY,/saddr1. bit	0 0 1 1 1 1 0 0	0 0 0 0 1 0 0 0	0 0 1 1 0 $B_2 B_1 B_0$
		← Saddr1-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, 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, 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, PSWL. bit	0 0 0 0 0 0 1 0	0 0 1 0 0 $B_2 B_1 B_0$	
	CY,/PSWL. bit	0 0 0 0 0 0 1 0	0 0 1 1 0 $B_2 B_1 B_0$	
	CY, PSWH. bit	0 0 0 0 0 0 1 0	0 0 1 0 1 $B_2 B_1 B_0$	
	CY,/PSWH. bit	0 0 0 0 0 0 1 0	0 0 1 1 1 $B_2 B_1 B_0$	
	CY, [TDE]. bit	0 0 1 1 1 1 0 1	0 0 1 0 0 $B_2 B_1 B_0$	
	CY,/ [TDE]. bit	0 0 1 1 1 1 0 1	0 0 1 1 0 $B_2 B_1 B_0$	
	CY, [WHL]. bit	0 0 1 1 1 1 0 1	0 0 1 0 1 $B_2 B_1 B_0$	
	CY,/ [WHL]. bit	0 0 1 1 1 1 0 1	0 0 1 1 1 $B_2 B_1 B_0$	
	CY, !addr16.bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	0 0 1 0 0 $B_2 B_1 B_0$
		← Low Address →	← High Address →	
	CY, /!addr16.bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	0 0 1 1 0 $B_2 B_1 B_0$
		← Low Address →	← High Address →	
	CY, !!addr24.bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	0 0 1 0 1 $B_2 B_1 B_0$
		← High-w Address →	← Low Address →	← High Address →
	CY, /!!addr24.bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	0 0 1 1 1 $B_2 B_1 B_0$
		← High-w Address →	← Low Address →	← High Address →
OR1	CY, saddr2. bit	0 0 0 0 1 0 0 0	0 1 0 0 0 $B_2 B_1 B_0$	← Saddr2-offset →
	CY, saddr1. bit	0 0 1 1 1 1 0 0	0 0 0 0 1 0 0 0	0 1 0 0 0 $B_2 B_1 B_0$
		← Saddr1-offset →		
	CY, /saddr2. bit	0 0 0 0 1 0 0 0	0 1 0 1 0 $B_2 B_1 B_0$	← Saddr2-offset →
	CY, /saddr1. bit	0 0 1 1 1 1 0 0	0 0 0 0 1 0 0 0	0 1 0 1 0 $B_2 B_1 B_0$
		← Saddr1-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, 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 $B_2 B_1 B_0$	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
OR1	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, PSWL. bit	0 0 0 0 0 0 1 0	0 1 0 0 0 $B_2 B_1 B_0$	
	CY,/PSWL. bit	0 0 0 0 0 0 1 0	0 1 0 1 0 $B_2 B_1 B_0$	
	CY, PSWH. bit	0 0 0 0 0 0 1 0	0 1 0 0 1 $B_2 B_1 B_0$	
	CY,/PSWH. bit	0 0 0 0 0 0 1 0	0 1 0 1 1 $B_2 B_1 B_0$	
	CY, [TDE]. bit	0 0 1 1 1 1 0 1	0 1 0 0 0 $B_2 B_1 B_0$	
	CY,/ [TDE]. bit	0 0 1 1 1 1 0 1	0 1 0 1 0 $B_2 B_1 B_0$	
	CY, [WHL]. bit	0 0 1 1 1 1 0 1	0 1 0 0 1 $B_2 B_1 B_0$	
	CY,/ [WHL]. bit	0 0 1 1 1 1 0 1	0 1 0 1 1 $B_2 B_1 B_0$	
	CY, !addr16.bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	0 1 0 0 0 $B_2 B_1 B_0$
		← Low Address →	← High Address →	
	CY,/!addr16.bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	0 1 0 1 0 $B_2 B_1 B_0$
		← Low Address →	← High Address →	
	CY, !!addr24.bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	0 1 0 0 1 $B_2 B_1 B_0$
		← High-w Address →	← Low Address →	← High Address →
	CY,/!!addr24.bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	0 1 0 1 1 $B_2 B_1 B_0$
		← High-w Address →	← Low Address →	← High Address →
XOR1	CY, saddr2. bit	0 0 0 0 1 0 0 0	0 1 1 0 0 $B_2 B_1 B_0$	← Saddr2-offset →
	CY, saddr1. bit	0 0 1 1 1 1 0 0	0 0 0 0 1 0 0 0	0 1 1 0 0 $B_2 B_1 B_0$
		← Saddr1-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, X. bit	0 0 0 0 0 0 1 1	0 1 1 0 0 $B_2 B_1 B_0$	
	CY, A. bit	0 0 0 0 0 0 1 1	0 1 1 0 1 $B_2 B_1 B_0$	
	CY, PSWL. bit	0 0 0 0 0 0 1 0	0 1 1 0 0 $B_2 B_1 B_0$	
	CY, PSWH. bit	0 0 0 0 0 0 1 0	0 1 1 0 1 $B_2 B_1 B_0$	
	CY, [TDE]. bit	0 0 1 1 1 1 0 1	0 1 1 0 0 $B_2 B_1 B_0$	
	CY, [WHL]. bit	0 0 1 1 1 1 0 1	0 1 1 0 1 $B_2 B_1 B_0$	
	CY, !addr16.bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	0 1 1 0 0 $B_2 B_1 B_0$
		← Low Address →	← High Address →	
	CY, !!addr24.bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	0 1 1 0 1 $B_2 B_1 B_0$
		← High-w Address →	← Low Address →	← High Address →

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
NOT1	saddr2. bit	0 0 0 0 1 0 0 0	0 1 1 1 0 B_2 B_1 B_0	← Saddr2-offset →
	saddr1. bit	0 0 1 1 1 1 0 0	0 0 0 0 1 0 0 0	0 1 1 1 0 B_2 B_1 B_0
		← Saddr1-offset →		
	sfr. bit	0 0 0 0 1 0 0 0	0 1 1 1 1 B_2 B_1 B_0	← Sfr-offset →
	X. bit	0 0 0 0 0 0 1 1	0 1 1 1 0 B_2 B_1 B_0	
	A. bit	0 0 0 0 0 0 1 1	0 1 1 1 1 B_2 B_1 B_0	
	PSWL. bit	0 0 0 0 0 0 1 0	0 1 1 1 0 B_2 B_1 B_0	
	PSWH. bit	0 0 0 0 0 0 1 0	0 1 1 1 1 B_2 B_1 B_0	
	[TDE]. bit	0 0 1 1 1 1 0 1	0 1 1 1 0 B_2 B_1 B_0	
	[WHL]. bit	0 0 1 1 1 1 0 1	0 1 1 1 1 B_2 B_1 B_0	
	!addr16.bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	0 1 1 1 0 B_2 B_1 B_0
		← Low Address →	← High Address →	
	!!addr24.bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	0 1 1 1 1 B_2 B_1 B_0
		← High-w Address →	← Low Address →	← High Address →
	CY	0 1 0 0 0 0 1 0		
SET1	saddr2. bit	1 0 1 1 0 B_2 B_1 B_0	← Saddr2-offset →	
	saddr1. bit	0 0 1 1 1 1 0 0	1 0 1 1 0 B_2 B_1 B_0	← Saddr1-offset →
	sfr. bit	0 0 0 0 1 0 0 0	1 0 0 0 1 B_2 B_1 B_0	← Sfr-offset →
	X. bit	0 0 0 0 0 0 1 1	1 0 0 0 0 B_2 B_1 B_0	
	A. bit	0 0 0 0 0 0 1 1	1 0 0 0 1 B_2 B_1 B_0	
	PSWL. bit	0 0 0 0 0 0 1 0	1 0 0 0 0 B_2 B_1 B_0	
	PSWH. bit	0 0 0 0 0 0 1 0	1 0 0 0 1 B_2 B_1 B_0	
	[TDE]. bit	0 0 1 1 1 1 0 1	1 0 0 0 0 B_2 B_1 B_0	
	[WHL]. bit	0 0 1 1 1 1 0 1	1 0 0 0 1 B_2 B_1 B_0	
	!addr16. bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	1 0 0 0 0 B_2 B_1 B_0
		← Low Address →	← High Address →	
	!!addr24. bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	1 0 0 0 1 B_2 B_1 B_0
		← High-w Address →	← Low Address →	← High Address →
	CY	0 1 0 0 0 0 0 1		

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
CLR1	saddr2. bit	1 0 1 0 0 B_2 B_1 B_0	← Saddr2-offset →	
	saddr1. bit	0 0 1 1 1 1 0 0	1 0 1 0 0 B_2 B_1 B_0	← Saddr1-offset →
	sfr. bit	0 0 0 0 1 0 0 0	1 0 0 1 1 B_2 B_1 B_0	← Sfr-offset →
	X. bit	0 0 0 0 0 0 1 1	1 0 0 1 0 B_2 B_1 B_0	
	A. bit	0 0 0 0 0 0 1 1	1 0 0 1 1 B_2 B_1 B_0	
	PSWL. bit	0 0 0 0 0 0 1 0	1 0 0 1 0 B_2 B_1 B_0	
	PSWH. bit	0 0 0 0 0 0 1 0	1 0 0 1 1 B_2 B_1 B_0	
	[TDE]. bit	0 0 1 1 1 1 0 1	1 0 0 1 0 B_2 B_1 B_0	
	[WHL]. bit	0 0 1 1 1 1 0 1	1 0 0 1 1 B_2 B_1 B_0	
	!addr16.bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	1 0 0 1 0 B_2 B_1 B_0
		← Low Address →	← High Address →	
	!!addr24.bit	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	1 0 0 1 1 B_2 B_1 B_0
		← High-w address →	← Low Address →	← High Address →
	CY	0 1 0 0 0 0 0 0		

(15) Stack manipulation instructions: PUSH, PUSHU, POP, POPU, MOVG, ADDWG, SUBWG, INCG, DECG

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
PUSH	PSW	0 1 0 0 1 0 0 1		
	sfrp	0 0 0 0 0 1 1 1	1 1 0 1 1 0 0 1	← sfr-offset →
	sfr	0 0 0 0 0 1 1 1	1 1 0 1 1 0 1 1	← sfr-offset →
	post	0 0 1 1 0 1 0 1	← post →	
	rg	0 0 0 0 1 0 0 1	1 0 0 0 1 G_2G_1 1	
PUSHU	post	0 0 1 1 0 1 1 1	← post →	
POP	PSW	0 1 0 0 1 0 0 0		
	sfrp	0 0 0 0 0 1 1 1	1 1 0 1 1 0 0 0	← Sfr-offset →
	sfr	0 0 0 0 0 1 1 1	1 1 0 1 1 0 1 0	← Sfr-offset →
	post	0 0 1 1 0 1 0 0	← post →	
	rg	0 0 0 0 1 0 0 1	1 0 0 1 1 G_2G_1 1	
POPU	post	0 0 1 1 0 1 1 0	← post →	
MOVG	SP, #imm24	0 0 0 0 1 0 0 1	0 0 1 0 0 0 0 0	← Low Byte →
		← High Byte →	← High-w Byte →	
	SP, WHL	0 0 0 0 0 1 0 1	1 1 1 1 1 0 1 1	
	WHL, SP	0 0 0 0 0 1 0 1	1 1 1 1 1 0 1 0	
ADDWG	SP, #word	0 0 0 0 1 0 0 1	0 0 1 0 1 0 0 0	← Low Byte →
		← High Byte →		
SUBWG	SP, #word	0 0 0 0 1 0 0 1	0 0 1 0 1 0 1 0	← Low Byte →
		← High Byte →		
INCG	SP	0 0 0 0 0 1 0 1	1 1 1 1 1 0 0 0	
DECG	SP	0 0 0 0 0 1 0 1	1 1 1 1 1 0 0 1	

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

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
CALL	!addr16	0 0 1 0 1 0 0 0	← Low Address →	← High Address →
	!!addr20	0 0 0 0 1 0 0 1	1 1 1 1 Hi-w Add	← Low Address →
		← High Address →		
	rp	0 0 0 0 0 1 0 1	0 1 0 1 1 P_2 P_1 P_0	
	rg	0 0 0 0 0 1 0 1	0 1 0 1 0 G_2 G_1 1	
	[rp]	0 0 0 0 0 1 0 1	0 1 1 1 1 P_2 P_1 P_0	
	[rg]	0 0 0 0 0 1 0 1	0 1 1 1 0 G_2 G_1 1	
	$!addr20	0 0 1 1 1 1 1 1	← $addr Low →	← $addr High →
CALLF	!addr11	1 0 0 1 0	← fa →	
CALLT	[addr5]	1 1 1 T_4 T_3 T_2 T_1 T_0		
BRK		0 1 0 1 1 1 1 0		
BRKCS	RBn	0 0 0 0 0 1 0 1	1 1 0 1 1 E_2 E_1 E_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		
RETCS	!addr16	0 0 1 0 1 0 0 1	← Low Address →	← High Address →
RETCSB	!addr16	0 0 0 0 1 0 0 1	1 0 1 1 0 0 0 0	← Low Address →
		← High Address →		

(17) Unconditional branch instruction: BR

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
BR	!addr16	0 0 1 0 1 1 0 0	← Low Address →	← High Address →
	!!addr20	0 0 0 0 1 0 0 1	1 1 1 0 Hi-w Add	← Low Address →
		← High Address →		
	rp	0 0 0 0 0 1 0 1	0 1 0 0 1 P_2 P_1 P_0	
	rg	0 0 0 0 0 1 0 1	0 1 0 0 0 G_2 G_1 1	
	[rp]	0 0 0 0 0 1 0 1	0 1 1 0 1 P_2 P_1 P_0	
	[rg]	0 0 0 0 0 1 0 1	0 1 1 0 0 G_2 G_1 1	
	$addr20	0 0 0 1 0 1 0 0	← $addr20 →	
	$!addr20	0 1 0 0 0 0 1 1	← $addr Low →	← $addr High →

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

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
BNZ	$addr20	1 0 0 0 0 0 0 0	← $addr20 →	
BNE				
BZ	$addr20	1 0 0 0 0 0 0 1	← $addr20 →	
BE				
BNC	$addr20	1 0 0 0 0 0 1 0	← $addr20 →	
BNL				
BC	$addr20	1 0 0 0 0 0 1 1	← $addr20 →	
BL				
BNV	$addr20	1 0 0 0 0 1 0 0	← $addr20 →	
BPO				
BV	$addr20	1 0 0 0 0 1 0 1	← $addr20 →	
BPE				
BP	$addr20	1 0 0 0 0 1 1 0	← $addr20 →	
BN	$addr20	1 0 0 0 0 1 1 1	← $addr20 →	
BLT	$addr20	0 0 0 0 0 1 1 1	1 1 1 1 1 0 0 0	← $addr20 →

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
BGE	\$addr20	0 0 0 0 0 1 1 1	1 1 1 1 1 0 0 1	← \$addr20 →
BLE	\$addr20	0 0 0 0 0 1 1 1	1 1 1 1 1 0 1 0	← \$addr20 →
BGT	\$addr20	0 0 0 0 0 1 1 1	1 1 1 1 1 0 1 1	← \$addr20 →
BNH	\$addr20	0 0 0 0 0 1 1 1	1 1 1 1 1 1 0 0	← \$addr20 →
BH	\$addr20	0 0 0 0 0 1 1 1	1 1 1 1 1 1 0 1	← \$addr20 →
BF	saddr2. bit, \$addr20	0 0 0 0 1 0 0 0	1 0 1 0 0 B_2 B_1 B_0	← Saddr2-offset →
		← \$addr20 →		
	saddr1. bit, \$addr20	0 0 1 1 1 1 0 0	0 0 0 0 1 0 0 0	1 0 1 0 0 B_2 B_1 B_0
		← Saddr1-offset →	← \$addr20 →	
	sfr. bit, \$addr20	0 0 0 0 1 0 0 0	1 0 1 0 1 B_2 B_1 B_0	← Sfr-offset →
		← \$addr20 →		
	X. bit, \$addr20	0 0 0 0 0 0 1 1	1 0 1 0 0 B_2 B_1 B_0	← \$addr20 →
	A. bit, \$addr20	0 0 0 0 0 0 1 1	1 0 1 0 1 B_2 B_1 B_0	← \$addr20 →
	PSWL. bit, \$addr20	0 0 0 0 0 0 1 0	1 0 1 0 0 B_2 B_1 B_0	← \$addr20 →
	PSWH. bit, \$addr20	0 0 0 0 0 0 1 0	1 0 1 0 1 B_2 B_1 B_0	← \$addr20 →
	[TDE]. bit, \$addr20	0 0 1 1 1 1 0 1	1 0 1 0 0 B_2 B_1 B_0	← \$addr20 →
	[WHL]. bit, \$addr20	0 0 1 1 1 1 0 1	1 0 1 0 1 B_2 B_1 B_0	← \$addr20 →
	!addr16.bit, \$addr20	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	1 0 1 0 0 B_2 B_1 B_0
		← Low Address →	← High Address →	← \$addr20 →
	!!addr24.bit, \$addr20	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	1 0 1 0 1 B_2 B_1 B_0
		← High-w Address →	← Low Address →	← High Address →
		← \$addr20 →		
BT	saddr2. bit, \$addr20	0 1 1 1 0 B_2 B_1 B_0	← Saddr2-offset →	← \$addr20 →
	saddr1. bit, \$addr20	0 0 1 1 1 1 0 0	0 1 1 1 0 B_2 B_1 B_0	← Saddr1-offset →
		← \$addr20 →		
	sfr. bit, \$addr20	0 0 0 0 1 0 0 0	1 0 1 1 1 B_2 B_1 B_0	← Sfr-offset →
		← \$addr20 →		
	X. bit, \$addr20	0 0 0 0 0 0 1 1	1 0 1 1 0 B_2 B_1 B_0	← \$addr20 →
	A. bit, \$addr20	0 0 0 0 0 0 1 1	1 0 1 1 1 B_2 B_1 B_0	← \$addr20 →
	PSWL. bit, \$addr20	0 0 0 0 0 0 1 0	1 0 1 1 0 B_2 B_1 B_0	← \$addr20 →
	PSWH. bit, \$addr20	0 0 0 0 0 0 1 0	1 0 1 1 1 B_2 B_1 B_0	← \$addr20 →
	[TDE]. bit, \$addr20	0 0 1 1 1 1 0 1	1 0 1 1 0 B_2 B_1 B_0	← \$addr20 →
	[WHL]. bit, \$addr20	0 0 1 1 1 1 0 1	1 0 1 1 1 B_2 B_1 B_0	← \$addr20 →

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
BT	!addr16.bit, $addr20	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	1 0 1 1 0 B_2 B_1 B_0
		← Low Address →	← High Address →	← $addr20 →
	!!addr24.bit, $addr20	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	1 0 1 1 1 B_2 B_1 B_0
		← High-w Address →	← Low Address →	← High Address →
		← $addr20 →		
BTCLR	saddr2, bit, $addr20	0 0 0 0 1 0 0 0	1 1 0 1 0 B_2 B_1 B_0	← Saddr2-offset →
		← $addr20 →		
	saddr1. bit, $addr20	0 0 1 1 1 1 0 0	0 0 0 0 1 0 0 0	1 1 0 1 0 B_2 B_1 B_0
		← Saddr1-offset →	← $addr20 →	
	sfr. bit, $addr20	0 0 0 0 1 0 0 0	1 1 0 1 1 B_2 B_1 B_0	← Sfr-offset →
		← $addr20 →		
	X. bit, $addr20	0 0 0 0 0 0 1 1	1 1 0 1 0 B_2 B_1 B_0	← $addr20 →
	A. bit, $addr20	0 0 0 0 0 0 1 1	1 1 0 1 1 B_2 B_1 B_0	← $addr20 →
	PSWL. bit, $addr20	0 0 0 0 0 0 1 0	1 1 0 1 0 B_2 B_1 B_0	← $addr20 →
	PSWH. bit, $addr20	0 0 0 0 0 0 1 0	1 1 0 1 1 B_2 B_1 B_0	← $addr20 →
	[TDE]. bit, $addr20	0 0 1 1 1 1 0 1	1 1 0 1 0 B_2 B_1 B_0	← $addr20 →
	[WHL]. bit, $addr20	0 0 1 1 1 1 0 1	1 1 0 1 1 B_2 B_1 B_0	← $addr20 →
	!addr16.bit, $addr20	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	1 1 0 1 0 B_2 B_1 B_0
		← Low Address →	← High Address →	← $addr20 →
	!!addr24.bit, $addr20	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	1 1 0 1 1 B_2 B_1 B_0
		← High-w Address →	← Low Address →	← High Address →
		← $addr20 →		
BFSET	saddr2. bit, $addr20	0 0 0 0 1 0 0 0	1 1 0 0 0 B_2 B_1 B_0	← Saddr2-offset →
		← $addr20 →		
	saddr1. bit, $addr20	0 0 1 1 1 1 0 0	0 0 0 0 1 0 0 0	1 1 0 0 0 B_2 B_1 B_0
		← Saddr1-offset →	← $addr20 →	
	sfr. bit, $addr20	0 0 0 0 1 0 0 0	1 1 0 0 1 B_2 B_1 B_0	← Sfr-offset →
		← $addr20 →		
	X. bit, $addr20	0 0 0 0 0 0 1 1	1 1 0 0 0 B_2 B_1 B_0	← $addr20 →
	A. bit, $addr20	0 0 0 0 0 0 1 1	1 1 0 0 1 B_2 B_1 B_0	← $addr20 →
	PSWL. bit, $addr20	0 0 0 0 0 0 1 0	1 1 0 0 0 B_2 B_1 B_0	← $addr20 →
	PSWH. bit, $addr20	0 0 0 0 0 0 1 0	1 1 0 0 1 B_2 B_1 B_0	← $addr20 →
	[TDE]. bit, $addr20	0 0 1 1 1 1 0 1	1 1 0 0 0 B_2 B_1 B_0	← $addr20 →
	[WHL]. bit, $addr20	0 0 1 1 1 1 0 1	1 1 0 0 1 B_2 B_1 B_0	← $addr20 →

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
BFSET	!addr16.bit, $addr20	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	1 1 0 0 0 B_2 B_1 B_0
		← Low Address →	← High Address →	← $addr20 →
	!!addr24.bit, $addr20	0 0 0 0 1 0 0 1	1 1 0 1 0 0 0 0	1 1 0 0 1 B_2 B_1 B_0
		← High-w Address →	← Low Address →	← High Address →
		← $addr20 →		
DBNZ	B, $addr20	0 0 1 1 0 0 1 1	← $addr20 →	
	C, $addr20	0 0 1 1 0 0 1 0	← $addr20 →	
	saddr2, $addr20	0 0 1 1 1 0 1 1	← Saddr2-offset →	← $addr20 →
	saddr1, $addr20	0 0 1 1 1 1 0 0	0 0 1 1 1 0 1 1	← Saddr1-offset →
		← $addr20 →		

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

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
MOV	STBC, #byte	0 0 0 0 1 0 0 1	1 1 0 0 0 0 0 0	← $\overline{\text{\#byte}}$ →
		← #byte →		
	WDM, #byte	0 0 0 0 1 0 0 1	1 1 0 0 0 0 1 0	← $\overline{\text{\#byte}}$ →
		← #byte →		
LOCATION	locaddr	0 0 0 0 1 0 0 1	1 1 0 0 0 0 0 1	← locaddr1 →
		← locaddrh →		
SEL	RBn	0 0 0 0 0 1 0 1	1 0 1 0 1 E_2 E_1 E_0	
	RBn. ALT	0 0 0 0 0 1 0 1	1 0 1 1 1 E_2 E_1 E_0	
SWRS		0 0 0 0 0 1 0 1	1 1 1 1 1 1 0 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		

(20) Special instructions: CHKL, CHKLA

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
CHKL	sfr	0 0 0 0 0 1 1 1	1 1 0 0 1 0 0 0	← Sfr address →
CHKLA	sfr	0 0 0 0 0 1 1 1	1 1 0 0 1 0 0 1	← Sfr address →

★ **Caution The CHKL and CHKLA instructions are not available in the μPD784216, 784216Y, 784218, 784218Y, 784225, 784225Y, 784937 Subseries. Do not execute these instructions. If these instructions are executed, the following operations will result.**

- **CHKL instruction .... After the pin levels of the output pins are read two times, they are exclusive-ORed. As a result, if the pins checked with this instruction are used in the port output mode, the exclusive-OR result is always 0 for all bits, and the Z flag is set to (1).**
- **CHKLA instruction .... After the pin levels of output pins are read two times, they are exclusive-ORed. As a result, if the pins checked with this instruction are used in the port output mode, the exclusive-OR result is always 0 for all bits, and the Z flag is set to (1) along with that the result is stored in the A register.**

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

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
MOVTBLW	!addr8, byte	0 0 0 0 1 0 0 1	1 0 1 0 0 0 0 0	← Low Address →
		← byte →		
MOVM	[TDE +], A	0 0 0 1 0 1 0 1	0 0 0 0 0 0 0 0	
	[TDE –], A	0 0 0 1 0 1 0 1	0 0 0 1 0 0 0 0	
MOVBK	[TDE +], [WHL +]	0 0 0 1 0 1 0 1	0 0 1 0 0 0 0 0	
	[TDE –], [WHL –]	0 0 0 1 0 1 0 1	0 0 1 1 0 0 0 0	
XCHM	[TDE +], A	0 0 0 1 0 1 0 1	0 0 0 0 0 0 0 1	
	[TDE –], A	0 0 0 1 0 1 0 1	0 0 0 1 0 0 0 1	
XCHBK	[TDE +], [WHL +]	0 0 0 1 0 1 0 1	0 0 1 0 0 0 0 1	
	[TDE –], [WHL –]	0 0 0 1 0 1 0 1	0 0 1 1 0 0 0 1	
CMPME	[TDE +], A	0 0 0 1 0 1 0 1	0 0 0 0 0 1 0 0	
	[TDE –], A	0 0 0 1 0 1 0 1	0 0 0 1 0 1 0 0	
CMPBKE	[TDE +], [WHL +]	0 0 0 1 0 1 0 1	0 0 1 0 0 1 0 0	
	[TDE –], [WHL –]	0 0 0 1 0 1 0 1	0 0 1 1 0 1 0 0	
CMPMNE	[TDE +], A	0 0 0 1 0 1 0 1	0 0 0 0 0 1 0 1	
	[TDE –], A	0 0 0 1 0 1 0 1	0 0 0 1 0 1 0 1	

(Continued on next page)

Mnemonic	Operands	Operation Code		
		B1	B2	B3
		B4	B5	B6
		B7		
CMPBKNE	[TDE +], [WHL +]	0 0 0 1 0 1 0 1	0 0 1 0 0 1 0 1	
	[TDE –], [WHL –]	0 0 0 1 0 1 0 1	0 0 1 1 0 1 0 1	
CMPMC	[TDE +], A	0 0 0 1 0 1 0 1	0 0 0 0 0 1 1 1	
	[TDE –], A	0 0 0 1 0 1 0 1	0 0 0 1 0 1 1 1	
CMPBKC	[TDE +], [WHL +]	0 0 0 1 0 1 0 1	0 0 1 0 0 1 1 1	
	[TDE –], [WHL –]	0 0 0 1 0 1 0 1	0 0 1 1 0 1 1 1	
CMPMNC	[TDE +], A	0 0 0 1 0 1 0 1	0 0 0 0 0 1 1 0	
	[TDE –], A	0 0 0 1 0 1 0 1	0 0 0 1 0 1 1 0	
CMPBKNC	[TDE +], [WHL +]	0 0 0 1 0 1 0 1	0 0 1 0 0 1 1 0	
	[TDE –], [WHL –]	0 0 0 1 0 1 0 1	0 0 1 1 0 1 1 0	

6.5 Number of Instruction Clocks

6.5.1 Execution time of instruction

The execution time for instructions is shown as the number of clocks of f_{CLK}.

The CPU in the 78K/IV Series has an instruction queue, so that another instruction can be prefetched in parallel while one instruction is executed. Consequently, the actual execution time of an instruction is dependent on the preceding instruction.

The execution time of an instruction also changes with the number of wait states used for memory access. Therefore, the accurate execution time of the program cannot be calculated by merely adding the number of execution clocks of instructions.

The minimum number of execution clocks is shown for instructions except those used for branch operation, such as BR, CALL, and RET instructions. For the branch instructions, the number of clocks slightly more than the minimum value is shown.

6.5.2 Definitions for "Clocks" column

(1) Internal ROM

The number of clocks set to 1 if the data to be accessed by an instruction is stored in the internal ROM and if the IFCH bit, which is bit 7 of the memory mapping mode register (MM), is shown. If the IFCH bit is cleared to 0, refer to the column of PRAM, EMEM, or SFR.

(2) IRAM

The number of clocks if the data to be accessed by an instruction is stored in the internal high-speed RAM (the area of addresses FD00H through FEFFH when LOCATION 0 instruction is executed, and the area of FFD00H through FFEFFH when LOCATION 0FH instruction is executed) is shown.

The μPD784915 Subseries is fixed to the LOCATION instruction.

(3) PRAM/EMEM/SFR

The number of clocks if the data to be accessed by an instruction is stored in an area of the internal RAM which is not IRAM, in the external memory (including the external SFR), or in the SFR area is shown.

(4) Others

The number of clocks if no data is accessed by an instruction is shown.

6.5.3 Explanation of “Clocks” column

(1) Number of clocks for accessing word data

- The number of clocks shown in the PRAM, EMEM, and SFR columns is when the bus width is 16 bits and when data is located at an even address. If the bus width is 8 bits, or if data is located at an odd address even though the bus width is 16 bits, add 4 to the number of clocks shown in the table. Note that the width of the internal RAM is 16 bits. Also, if word data of the internal ROM is located at an odd address, add 4 to the number of clocks.
- If word data is saved to or restored from an odd address by a stack manipulation instruction marked “n”, add 4 to the coefficient of “n”.

(2) Number of clocks for accessing 3-byte data

The number of clocks shown in the PRAM, EMEM, or SFR column is used when the bus width is 16 bits. If the bus width is 8 bits, and if data is located at an odd address even though the bus width is 16 bits, add 4 to the number of clocks shown in the table. Note that the bus width of the internal RAM is 16 bits.

(3) If two types of numbers of clocks are shown with each delimited by “/” from the other

If two types of numbers of clocks are shown with each delimited by “/” from the other, two types of numbers of bytes are shown for that instruction with each delimited by “/” from the other. The execution time of this kind of instruction is the number of clocks shown at the same side as the number of bytes.

(4) When “n” is shown in “Clocks” column

- When the MACW, MACSW, and MOVTBLW instructions are used, the number specified by operand byte substitutes for “n”.
- In the case of the SACW, MOVM, XCHM, MOVBK, XCHBK, CMPME, CMPMNE, CMPMC, CMPMNC, CMPBKE, CMPBKNE, CMPBKC, and CMPBKNC instructions, the value set to the C register on starting execution of the instruction substitutes for “n”. This number of clocks is the value when the instruction execution is not stopped by an interrupt or macro service.
- When the shift or rotate instruction is used, the number of bits to be shifted or rotated substitutes for “n”.
- When the stack manipulation instruction is used, the number of registers to be saved to the stack or restored from the stack substitutes for “n”.

6.5.4 List of number of clocks

(1) 8-bit data transfer instruction: MOV

(1/3)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
MOV	r, #byte	2/3	–	2/3	–	–
	saddr, #byte	3/4		3/4	7	
	sfr, #byte	3		–	7	
	!addr16, #byte	5	–	7	9	
	!!addr24, #byte	6	–	8	10	
	r, r’	2/3	–	2/3	–	
	A, r	1/2				
	A, saddr2	2		3	7	
	r, saddr	3		4	8	
	saddr2, A	2		2	6	
	saddr, r	3		4	8	
	A, sfr	2		–	7	
	r, sfr	3			8	
	sfr, A	2			6	
	sfr, r	3			8	
	saddr, saddr’	4		6	14	
	r, !addr16	4	9	7	9	
	!addr16, r	4	–	6	8	
	r, !!addr24	5	10	8	10	
	!!addr24, r	5	–	7	9	
	A, [saddrp]	2/3	9/10	7/8	9/10	
	A, [%saddrg]	3/4	14/15	12/13	14/15	
	A, [TDE +]	1	9	7	9	
	A, [WHL +]	1				
	A, [TDE –]	1				
	A, [WHL –]	1				
	A, [TDE]	1	8	6	8	
	A, [WHL]	1				
	A, [VVP]	2	9	7	9	
	A, [UUP]	2				
	A, [TDE + byte]	3	10	8	10	
	A, [SP + byte]	3	11	9	11	

(2/3)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
MOV	A, [WHL + byte]	3	10	8	10	–
	A, [UUP + byte]	3				
	A, [VVP + byte]	3				
	A, imm24[DE]	5	12	10	12	
	A, imm24[A]	5				
	A, imm24[HL]	5				
	A, imm24[B]	5				
	A, [TDE + A]	2	10	8	10	
	A, [WHL + A]	2				
	A, [TDE + B]	2				
	A, [WHL + B]	2				
	A, [VVP + DE]	2				
	A, [VVP + HL]	2				
	A, [TDE + C]	2				
	A, [WHL + C]	2				
	[saddrp], A	2/3	–	6/7	8/9	
	[%saddrg], A	3/4		12/13	14/15	
	[TDE +], A	1		8	10	
	[WHL +], A	1				
	[TDE –], A	1				
	[WHL –], A	1				
	[TDE], A	1		5	7	
	[WHL], A	1				
	[VVP], A	2		7	9	
	[UUP], A	2				
	[TDE + byte], A	3		8	10	
	[SP + byte], A	3		9	11	
	[WHL + byte], A	3		8	10	
	[UUP + byte], A	3				
	[VVP + byte], A	3				
	imm24[DE], A	5		10	12	
	imm24[A], A	5				
	imm24[HL], A	5				
	imm24[B], A	5				

(3/3)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
MOV	[TDE + A], A	2	–	8	10	–
	[WHL + A], A	2				
	[TDE + B], A	2				
	[WHL + B], A	2				
	[VVP + DE], A	2				
	[VVP + HL], A	2				
	[TDE + C], A	2				
	[WHL + C], A	2				
	PSWL, #byte	3		–	–	7
	PSWH, #byte	3				
	PSWL, A	2				6
	PSWH, A	2				
	A, PSWL	2				7
	A, PSWH	2				
	r3, #byte	3				3
	A, r3	2				4
	r3, A	2				3

(2) 16-bit data transfer instruction: MOVW

(1/3)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
MOVW	rp, #word	3	–	3	–	–
	saddrp, #word	4/5		4	8	
	sfrp, #word	4		–		
	!addr16, #word	6		8	10	
	!!addr24, #word	7		9	11	
	rp, rp'	2		2	–	
	AX, saddrp2	2		3	7	
	rp, saddrp	3		4	8	
	saddrp2, AX	2		2	6	
	saddrp, rp	3		3	7	
	AX, sfrp	2		–	7	
	rp, sfrp	3			8	
	sfrp, AX	2			6	
	sfrp, rp	3			7	
	saddrp, saddrp'	4		6	14	
	rp, !addr16	4	9	7	9	
	!addr16, rp	4	–	6	8	
	rp, !!addr24	5	10	8	10	
	!!addr24, rp	5	–	7	9	
	AX, [saddrp]	3/4	10/11	8/9	10/11	
	AX, [%saddrg]	3/4	14/15	12/13	14/15	
	AX, [TDE +]	2	11	9	11	
	AX, [WHL +]	2				
	AX, [TDE –]	2				
	AX, [WHL –]	2				
	AX, [TDE]	2	9	7	9	
	AX, [WHL]	2				
	AX, [VVP]	2				
	AX, [UUP]	2				
	AX, [TDE + byte]	3	10	8	10	
	AX, [SP + byte]	3	11	9	11	
	AX, [WHL + byte]	3	10	8	10	
	AX, [UUP + byte]	3				
	AX, [VVP + byte]	3				

(2/3)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
MOVW	AX, imm24[DE]	5	12	10	12	–
	AX, imm24[A]	5				
	AX, imm24[HL]	5				
	AX, imm24[B]	5				
	AX, [TDE + A]	2	10	8	10	
	AX, [WHL + A]	2				
	AX, [TDE + B]	2				
	AX, [WHL + B]	2				
	AX, [VVP + DE]	2				
	AX, [VVP + HL]	2				
	AX, [TDE + C]	2				
	AX, [WHL + C]	2				
	[saddrp], AX	3/4	–	8/9	10/11	
	[%saddrg], AX	3/4		12/13	14/15	
	[TDE +], AX	2		9	11	
	[WHL +], AX	2				
	[TDE –], AX	2				
	[WHL –], AX	2				
	[TDE], AX	2		7	9	
	[WHL], AX	2				
	[VVP], AX	2				
	[UUP], AX	2				
	[TDE + byte], AX	3		8	10	
	[SP + byte], AX	3		9	11	
	[WHL + byte], AX	3		8	10	
	[UUP + byte], AX	3				
	[VVP + byte], AX	3				
	imm24[DE], AX	5		10	12	
	imm24[A], AX	5				
	imm24[HL], AX	5				
	imm24[B], AX	5				

(3/3)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
MOVW	[TDE + A], AX	2	–	8	10	–
	[WHL + A], AX	2				
	[TDE + B], AX	2				
	[WHL + B], AX	2				
	[VVP + DE], AX	2				
	[VVP + HL], AX	2				
	[TDE + C], AX	2				
	[WHL + C], AX	2				

(3) 24-bit data transfer instruction: MOVG

(1/2)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
MOVG	rg, #imm24	5	–	5	–	–
	rg, rg'	2		4		
	rg, !!addr24	5	17	13	17	
	!!addr24, rg	5	–	12	16	
	rg, saddrg	3		9	17	
	saddrg, rg	3		7	15	
	WHL, [%saddrg]	3/4	21/22	17/18	21/22	
	[%saddrg], WHL	3/4	–			
	WHL, [TDE +]	2	19	15	19	
	WHL, [TDE –]	2				
	WHL, [TDE]	2	16	12	16	
	WHL, [WHL]	2				
	WHL, [VVP]	2				
	WHL, [UUP]	2				
	WHL, [TDE + byte]	3	17	13	17	
	WHL, [SP + byte]	3	18	14	18	
	WHL, [WHL + byte]	3	17	13	17	
	WHL, [UUP + byte]	3				
	WHL, [VVP + byte]	3				
	WHL, imm24[DE]	5	19	15	19	
	WHL, imm24[A]	5				
	WHL, imm24[HL]	5				
	WHL, imm24[B]	5				
	WHL, [TDE + A]	2	17	13	17	
	WHL, [WHL + A]	2				
	WHL, [TDE + B]	2				
	WHL, [WHL + B]	2				
	WHL, [VVP + DE]	2				
	WHL, [VVP + HL]	2				
	WHL, [TDE + C]	2				
	WHL, [WHL + C]	2				

(2/2)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
MOVG	[TDE +], WHL	2	–	15	19	–
	[TDE –], WHL	2				
	[TDE], WHL	2		12	16	
	[WHL], WHL	2				
	[VVP], WHL	2				
	[UUP], WHL	2				
	[TDE + byte], WHL	3		13	17	
	[SP + byte], WHL	3		14	18	
	[WHL + byte], WHL	3		13	17	
	[UUP + byte], WHL	3				
	[VVP + byte], WHL	3				
	imm24[DE], WHL	5		15	19	
	imm24[A], WHL	5				
	imm24[HL], WHL	5				
	imm24[B], WHL	5				
	[TDE + A], WHL	2		13	17	
	[WHL + A], WHL	2				
	[TDE + B], WHL	2				
	[WHL + B], WHL	2				
	[VVP + DE], WHL	2				
	[VVP + HL], WHL	2				
	[TDE + C], WHL	2				
	[WHL + C], WHL	2				

(4) 8-bit data exchange instruction: XCH

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
XCH	r, r’	2/3	–	4	–	–
	A, r	1/2		4/5		
	A, saddr2	2		5	13	
	r, saddr	3		6	14	
	r, sfr	3		–	14	
	saddr, saddr’	4		8	24	
	r, !addr16	4		11	15	
	r, !!addr24	5				
	A, [saddrp]	2/3		8/9	10/11	
	A, [%saddrg]	3/4		17/18	21/22	
	A, [TDE +]	2		14	18	
	A, [WHL +]	2				
	A, [TDE –]	2				
	A, [WHL –]	2				
	A, [TDE]	2		12	16	
	A, [WHL]	2				
	A, [VVP]	2				
	A, [UUP]	2				
	A, [TDE + byte]	3		13	17	
	A, [SP + byte]	3		14	18	
	A, [WHL + byte]	3		13	17	
	A, [UUP + byte]	3				
	A, [VVP + byte]	3				
	A, imm24[DE]	5		15	19	
	A, imm24[A]	5				
	A, imm24[HL]	5				
	A, imm24[B]	5				
	A, [TDE + A]	2		13	17	
	A, [WHL + A]	2				
	A, [TDE + B]	2				
	A, [WHL + B]	2				
	A, [VVP + DE]	2				
	A, [VVP + HL]	2				
	A, [TDE + C]	2				
	A, [WHL + C]	2				

(5) 16-bit data exchange instruction: XCHW

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
XCHW	rp, rp'	2	–	4	–	–
	AX, saddrp2	2		5	13	
	rp, saddrp	3		6	14	
	rp, sfrp	3		–	14	
	AX, [saddrp]	3/4		13/14	17/18	
	AX, [%saddrg]	3/4		17/18	21/22	
	AX, !addr16	4		3	3	
	AX, !!addr24	5		4	4	
	saddrp, saddrp'	4		8	24	
	AX, [TDE +]	2		14	18	
	AX, [WHL +]	2				
	AX, [TDE –]	2				
	AX, [WHL –]	2				
	AX, [TDE]	2		12	16	
	AX, [WHL]	2				
	AX, [VVP]	2				
	AX, [UUP]	2				
	AX, [TDE + byte]	3		13	17	
	AX, [SP + byte]	3		14	18	
	AX, [WHL + byte]	3		13	17	
	AX, [UUP + byte]	3				
	AX, [VVP + byte]	3				
	AX, imm24[DE]	5		15	19	
	AX, imm24[A]	5				
	AX, imm24[HL]	5				
	AX, imm24[B]	5				
	AX, [TDE + A]	2		13	17	
	AX, [WHL + A]	2				
	AX, [TDE + B]	2				
	AX, [WHL + B]	2				
	AX, [VVP + DE]	2				
	AX, [VVP + HL]	2				
	AX, [TDE + C]	2				
	AX, [WHL + C]	2				

(6) 8-bit operation instructions: ADD, ADDC, SUB, SUBC, AND, OR, XOR, CMP

(1/5)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
ADD	A, #byte	2	–	2	–	–
ADDC	r, #byte	3		4		
SUB	saddr, #byte	3/4		6/7	12/13	
SUBC	sfr, #byte	4		–	13	
AND	r, r’	2/3		3/4	–	
OR	A, saddr2	4		3	7	
XOR	r, saddr	3		4	8	
	saddr, r	3		8	14	
	r, sfr	3		–	8	
	sfr, r	3		–	14	
	saddr, saddr’	4		8	18	
	A, [saddrp]	3/4	11/12	9/10	11/12	
	A, [%saddrg]	3/4	15/16	13/14	15/16	
	[saddrp], A	3/4	–	11/12	15/16	
	[%saddrg], A	3/4		15/16	19/20	
	A, !addr16	4	10	8	10	
	A, !!addr24	5	11	9	11	
	!addr16, A	4	–	10	14	
	!!addr24, A	5		11	15	
	A, [TDE +]	1	11	9	11	
	A, [WHL +]	1				
	A, [TDE –]	1				
	A, [WHL –]	1				
	A, [TDE]	1	10	8	10	
	A, [WHL]	1				
	A, [VVP]	2				
	A, [UUP]	2				
	A, [TDE + byte]	3	12	10	12	
	A, [SP + byte]	3				
	A, [WHL + byte]	3				
	A, [UUP + byte]	3				
	A, [VVP + byte]	3				

(2/5)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
ADD	A, imm24[DE]	5	13	11	13	–
ADDC	A, imm24[A]	5				
SUB	A, imm24[HL]	5				
SUBC	A, imm24[B]	5				
AND	A, [TDE + A]	2	11	9	11	
OR	A, [WHL + A]	2				
XOR	A, [TDE + B]	2				
	A, [WHL + B]	2				
	A, [VVP + DE]	2				
	A, [VVP + HL]	2				
	A, [TDE + C]	2				
	A, [WHL + C]	2				
	[TDE +], A	1	–	10	14	
	[WHL +], A	1				
	[TDE –], A	1				
	[WHL –], A	1				
	[TDE], A	1				
	[WHL], A	1				
	[VVP], A	2				
	[UUP], A	2				
	[TDE + byte], A	3		13	17	
	[SP + byte], A	3				
	[WHL + byte], A	3				
	[UUP + byte], A	3				
	[VVP + byte], A	3				
	imm24[DE], A	5		14	18	
	imm24[A], A	5				
	imm24[HL], A	5				
	imm24[B], A	5				

(3/5)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
ADD	[TDE + A], A	2	–	12	16	–
ADDC	[WHL + A], A	2				
SUB	[TDE + B], A	2				
SUBC	[WHL + B], A	2				
AND	[VVP + DE], A	2				
OR	[VVP + HL], A	2				
XOR	[TDE + C], A	2				
	[WHL + C], A	2				

(4/5)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
CMP	A, #byte	2	–	2	–	–
	r, #byte	3		4		
	saddr, #byte	3/4		4/5	8/9	
	sfr, #byte	4		–	9	
	r, r’	2/3		3/4	–	
	A, saddr2	4		3	7	
	r, saddr	3		4	8	
	saddr, r	3		6	10	
	r, sfr	3		–	9	
	sfr, r	3			10	
	saddr, saddr’	4		6	14	
	A, [saddrp]	3/4	11/12	9/10	11/12	
	A, [%saddrg]	3/4	15/16	13/14	15/16	
	[saddrp], A	3/4	11/12	9/10	11/12	
	[%saddrg], A	3/4	15/16	13/14	15/16	
	A, !addr16	4	10	8	10	
	A, !!addr24	5	11	9	11	
	!addr16, A	4	10	8	10	
	!!addr24, A	5	11	9	11	
	A, [TDE +]	1	11	9	11	
	A, [WHL +]	1				
	A, [TDE –]	1				
	A, [WHL –]	1				
	A, [TDE]	1	10	8	10	
	A, [WHL]	1				
	A, [VVP]	2				
	A, [UUP]	2				
	A, [TDE + byte]	3	12	10	12	
	A, [SP + byte]	3				
	A, [WHL + byte]	3				
	A, [UUP + byte]	3				
	A, [VVP + byte]	3				
	A, imm24[DE]	5	13	11	13	
	A, imm24[A]	5				
	A, imm24[HL]	5				
	A, imm24[B]	5				

(5/5)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
CMP	A, [TDE + A]	2	11	9	11	–
	A, [WHL + A]	2				
	A, [TDE + B]	2				
	A, [WHL + B]	2				
	A, [VVP + DE]	2				
	A, [VVP + HL]	2				
	A, [TDE + C]	2				
	A, [WHL + C]	2				
	[TDE +], A	1	10	8	10	
	[WHL +], A	1				
	[TDE –], A	1				
	[WHL –], A	1				
	[TDE], A	1				
	[WHL], A	1				
	[VVP], A	2				
	[UUP], A	2				
	[TDE + byte], A	3	13	11	13	
	[SP + byte], A	3				
	[WHL + byte], A	3				
	[UUP + byte], A	3				
	[VVP + byte], A	3				
	imm24[DE], A	5	14	12	14	
	imm24[A], A	5				
	imm24[HL], A	5				
	imm24[B], A	5				
	[TDE + A], A	2	12	10	12	
	[WHL + A], A	2				
	[TDE + B], A	2				
	[WHL + B], A	2				
	[VVP + DE], A	2				
	[VVP + HL], A	2				
	[TDE + C], A	2				
	[WHL + C], A	2				

(7) 16-bit operation instructions: ADDW, SUBW, CMPW

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
ADDW	AX, #word	3	–	3	–	–
SUBW	rp, #word	4		5		
	rp, rp'	2		3		
	AX, saddrp2	2			7	
	rp, saddrp	3		5	9	
	saddrp, rp	3		8	14	
	rp, sfrp	3		–	9	
	sfrp, rp	3		–	13	
	saddrp, #word	4/5		7/8		
	sfrp, #word	5		–	14	
	saddrp, saddrp'	4		8	20	
CMPW	AX, #word	3	–	3	–	–
	rp, #word	4		5		
	rp, rp'	2		3		
	AX, saddrp2	2			7	
	rp, saddrp	3		5	9	
	saddrp, rp	3				
	rp, sfrp	3		–		
	sfrp, rp	3				
	saddrp, #word	4/5		5/6	9	
	sfrp, #word	5		–	10	
	saddrp, saddrp'	4		6		

(8) 24-bit operation instructions: ADDG, SUBG

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
ADDG	rg, rg’	2	–	6	–	–
SUBG	rg, #imm24	5		8		
	WHL, saddrg	3		13	19	

(9) Multiplication instructions: MULU, MULUW, MULW, DIVUW, DIVUX

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
MULU	r	2/3	–	11/12	–	–
MULUW	rp	2	–	15	–	–
MULW	rp	2	–	14	–	–
DIVUW	r	2/3	–	23/24	–	–
DIVUX	rp	2	–	43	–	–

(10) Special operation instructions: MACW, MACSW, SACW

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
MACW	byte	3	–	5 + 21n	–	–
MACSW	byte	3	–	5 + 21n	–	–
SACW	[TDE +], [WHL +]	4	–	4 + 19n	4 + 23n	–

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

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
INC	r	1/2	–	2/3	–	–
DEC	saddr	2/3		5/6	11/12	
INCW	rp	2/1	–	3/2	–	–
DECW	saddrp	3/4		6/7	12/13	
INCG **DECG**	rg	2	–	4	–	–

(12) Adjustment instructions: ADJBA, ADJBS, CVTBW

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
ADJBA		2	–	5	–	–
ADJBS		2	–	5	–	–
CVTBW		1	–	3	–	–

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

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
ROR **ROL** **RORC** **ROLC** **SHR** **SHL**	r, n	2/3	–	5 + n/6 + n	–	–
SHRW **SHLW**	rp, n	2	–	5 + n	–	–
ROR4 **ROL4**	mem3	2	–	11	15	–

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

(1/3)

<table>
<tr><th rowspan="2">Mnemonic</th><th rowspan="2">Operands</th><th rowspan="2">Bytes</th><th colspan="4">Clocks</th></tr>
<tr><th>Internal ROM</th><th>IRAM</th><th>PRAM/EMEM/SFR</th><th>Others</th></tr>
<tr><td rowspan="20">MOV1</td><td>CY, saddr.bit</td><td>3/4</td><td rowspan="6">–</td><td>6/7</td><td>10/11</td><td rowspan="20">–</td></tr>
<tr><td>CY, sfr.bit</td><td>3</td><td>–</td><td>10</td></tr>
<tr><td>CY, X.bit</td><td>2</td><td rowspan="2">5</td><td rowspan="2">–</td></tr>
<tr><td>CY, A.bit</td><td>2</td></tr>
<tr><td>CY, PSWL.bit</td><td>2</td><td rowspan="2">–</td><td rowspan="2">5</td></tr>
<tr><td>CY, PSWH.bit</td><td>2</td></tr>
<tr><td>CY, [TDE].bit</td><td>2</td><td rowspan="2">11</td><td rowspan="2">9</td><td rowspan="2">11</td></tr>
<tr><td>CY, [WHL].bit</td><td>2</td></tr>
<tr><td>CY, !addr16.bit</td><td>5</td><td rowspan="2">16</td><td rowspan="2">14</td><td rowspan="2">16</td></tr>
<tr><td>CY, !!addr24.bit</td><td>6</td></tr>
<tr><td>saddr.bit, CY</td><td>3/4</td><td rowspan="10">–</td><td>5/6</td><td>13/14</td></tr>
<tr><td>sfr.bit, CY</td><td>3</td><td>–</td><td>13</td></tr>
<tr><td>X.bit, CY</td><td>2</td><td rowspan="2">6</td><td rowspan="2">–</td></tr>
<tr><td>A.bit, CY</td><td>2</td></tr>
<tr><td>PSWL.bit, CY</td><td>2</td><td rowspan="2">–</td><td>8</td></tr>
<tr><td>PSWH.bit, CY</td><td>2</td><td>7</td></tr>
<tr><td>[TDE].bit, CY</td><td>2</td><td rowspan="2">10</td><td rowspan="2">14</td></tr>
<tr><td>[WHL].bit, CY</td><td>2</td></tr>
<tr><td>!addr16.bit, CY</td><td>5</td><td rowspan="2">13</td><td rowspan="2">15</td></tr>
<tr><td>!!addr24.bit, CY</td><td>6</td></tr>
</table>

(2/3)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
AND1	CY, saddr.bit	3/4	–	6/7	10/11	–
OR1	CY, /saddr.bit	3/4				
	CY, sfr.bit	3		–	10	
	CY, /sfr.bit	3				
	CY, X.bit	2		5	–	
	CY, /X.bit	2				
	CY, A.bit	2				
	CY, /A.bit	2				
	CY, PSWL.bit	2				
	CY, /PSWL.bit	2				
	CY, PSWH.bit	2				
	CY, /PSWH.bit	2				
	CY, [TDE].bit	2	11	9	11	
	CY, /[TDE].bit	2				
	CY, [WHL].bit	2				
	CY, /[WHL].bit	2				
	CY, !addr16.bit	5	16	14	16	
	CY, /!addr16.bit	5				
	CY, !!addr24.bit	6				
	CY, /!!addr24.bit	6				
XOR1	CY, saddr.bit	3/4	–	6/7	10/11	
	CY, /sfr.bit	3		–	10	
	CY, X.bit	2		5	–	
	CY, A.bit	2				
	CY, PSWL.bit	2		–	5	
	CY, PSWH.bit	2				
	CY, [TDE].bit	2	11	9	11	
	CY, [WHL].bit	2				
	CY, !addr16.bit	5	16	14	16	
	CY, !!addr24.bit	6				

(3/3)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
NOT1	saddr.bit	3/4	–	5/6	13/14	–
	sfr.bit	3		–	13	
	X.bit	2		5	–	
	A.bit	2				
	PSWL.bit	2		–	7	
	PSWH.bit	2			6	
	[TDE].bit	2		10	14	
	[WHL].bit	2				
	!addr16.bit	5		13	15	
	!!addr24.bit	6				
	CY	1		–	2	
SET1	saddr.bit	2/3		4/5	12/13	
CLR1	sfr.bit	3		–	13	
	X.bit	2		5	–	
	A.bit	2				
	PSWL.bit	2		–	7	
	PSWH.bit	2			6	
	[TDE].bit	2		10	14	
	[WHL].bit	2				
	!addr16.bit	5		13	15	
	!!addr24.bit	6				
	CY	1		–	2	

(15) Stack manipulation instructions: PUSH, PUSHU, POP, POPU, MOVG, ADDWG, SUBWG, INCG, DECG

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
PUSH	PSW	1	–	5	7	–
	sfrp	3		10	14	
	sfr	3				
	post	2		4 + 5n	4 + 7n	
	rg	2		12	16	
PUSHU	post	2	–	6 + 5n	6 + 7n	–
POP	PSW	1	8	7	9	–
	sfrp	3	15	14	16	
	sfr	3				
	post	2	4 + 8n	4 + 6n	4 + 8n	
	rg	2	17	13	17	
POPU	post	2	7 + 8n	7 + 6n	7 + 8n	–
MOVG	SP, #imm24	5	–	–	–	5
	SP, WHL	2				
	WHL, SP	2				
ADDWG **SUBWG**	SP, #word	4	–	–	–	5
INCG **DECG**	SP	2	–	–	–	5

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

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
CALL	!addr16	3	–	19	23	–
	!!addr20	4	–	22	26	
	rp	2	–	20	24	
	rg	2	–	22	26	
	[rp]	2	30 **Note**	24	30	
	[rg]	2	37 **Note**	29	37	
	$!addr20	3	–	19	23	
CALLF	!addr11	2	–	19	23	–
CALLT	[addr5]	1	28 **Note**	22	28	–
BRK		1	–	23	29	–
BRKCS	RBn	2	–	–	–	13
RET		1	21	17	21	–
RETI		1	22	18	22	–
RETB		1	21	17	21	–
RETCS	!addr16	3	–	–	–	14
RETCSB	!addr16	4	–	–	–	14

Note When the stack is PRAM or EMEM

(17) Unconditional branch instruction: BR

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
BR	!addr16	3	–	–	–	11
	!!addr20	4	–	–	–	12
	rp	2	–	–	–	11
	rg	2	–	–	–	12
	[rp]	2	16	14	16	–
	[rg]	2	22	18	22	–
	$addr20	2	–	–	–	10
	$!addr20	3	–	–	–	11

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

(1/4)

Mnemonic	Operands	Bytes	Clocks				
			Not Branch	Branches			
				Internal ROM	IRAM	PRAM/EMEM/SFR	Others
BNZ	$addr20	2	3	–	–	–	10
BNE							
BZ	$addr20	2	3	–	–	–	10
BE							
BNC	$addr20	2	3	–	–	–	10
BNL							
BC	$addr20	2	3	–	–	–	10
BL							
BNV	$addr20	2	3	–	–	–	10
BPO							
BV	$addr20	2	3	–	–	–	10
BPE							
BP	$addr20	2	3	–	–	–	10
BN							
BLT	$addr20	3	4	–	–	–	11
BGE	$addr20	3	4	–	–	–	11
BLE	$addr20	3	4	–	–	–	11
BGT	$addr20	3	4	–	–	–	11
BNH	$addr20	3	4	–	–	–	11
BH	$addr20	3	4	–	–	–	11

(2/4)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
BF	saddr.bit, $addr20	4/5	–	14/15	18	–
			–	7/8	11	–
	sfr.bit, $addr20	4	–	–	18	–
			–	–	11	–
	X.bit, $addr20	3	–	13	–	–
			–	6	–	–
	A.bit, $addr20	3	–	13	–	–
			–	6	–	–
	PSWL.bit, $addr20	3	–	–	13	–
			–	–	6	–
	PSWH.bit, $addr20	3	–	–	13	–
			–	–	6	–
	mem2.bit, $addr20	3	19	17	19	–
			12	10	12	–
	!addr16.bit, $addr20	6	–	22	24	–
			–	15	17	–
	!!addr24.bit, $addr20	7	–	22	24	–
			–	15	17	–

Remark The number of clocks differs depending on the following cases. Therefore, two types of numbers of clocks are shown for each operand with one type shown at the top and the other at the bottom.

Top : Branches (internal ROM high-speed fetch, etc.)

Bottom : Does not branch

(3/4)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
BT	saddr.bit, $addr20	3/4	–	13/14	17	–
			–	6/7	10	–
	sfr.bit, $addr20	4	–	–	18	–
			–	–	11	–
	X.bit, $addr20	3	–	13	–	–
			–	6	–	–
	A.bit, $addr20	3	–	13	–	–
			–	6	–	–
	PSWL.bit, $addr20	3	–	–	13	–
			–	–	6	–
	PSWH.bit, $addr20	3	–	–	13	–
			–	–	6	–
	mem2.bit, $addr20	3	19	17	19	–
			12	10	12	–
	!addr16.bit, $addr20	6	–	22	24	–
			–	15	17	–
	!!addr24.bit, $addr20	7	–	22	24	–
			–	15	17	–

Remark The number of clocks differs depending on the following cases. Therefore, two types of numbers of clocks are shown for each operand with one type shown at the top and the other at the bottom.

Top : Branches (internal ROM high-speed fetch, etc.)

Bottom : Does not branch

(4/4)

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
BTCLR	saddr.bit, $addr20	4/5	–	16/17	24	–
BFSET			–	7/8	15	–
	sfr.bit, $addr20	4	–	–	24	–
			–	–	15	–
	X.bit, $addr20	3	–	15	–	–
			–	6	–	–
	A.bit, $addr20	3	–	15	–	–
			–	6	–	–
	PSWL.bit, $addr20	3	–	–	15	–
			–	–	6	–
	PSWH.bit, $addr20	3	–	–	16	–
			–	–	6	–
	mem2.bit, $addr20	3	–	21	25	–
			–	12	16	–
	!addr16.bit, $addr20	6	–	24	26	–
			–	15	17	–
	!!addr24.bit, $addr20	7	–	24	26	–
			–	15	17	–
DBNZ	B, $addr20	2	12	–	–	–
			4	–	–	–
	C, $addr20	2	12	–	–	–
			4	–	–	–
	saddr, $addr20	3	21	17	21	–
			5	5	5	–
		4	22	18	22	–
			6	6	6	–

Remark The number of clocks differs depending on the following cases. Therefore, two types of numbers of clocks are shown for each operand with one type shown at the top and the other at the bottom.

Top : Branches (internal ROM high-speed fetch, etc.)

Bottom : Does not branch

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

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
MOV	STBC, #byte	4	–	–	–	13
	WDM, #byte	4				
LOCATION	locaddr	4	–	–	–	13
SEL	RBn	2	–	–	–	3
	RBn, ALT	2				
SWRS		2	–	–	–	4
NOP		1	–	–	–	2
EI		1	–	–	–	2
DI		1	–	–	–	2

(20) Special instructions: CHKL, CHKLA

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
CHKL	sfr	3	–	–	14	–
CHKLA	sfr	3	–	–	14	–

★ **Caution The CHKL and CHKLA instructions are not available in the μPD784216, 784216Y, 784218, 784218Y, 784225, 784225Y, 784937 Subseries. Do not execute these instructions. If these instructions are executed, the following operations will result.**

- **CHKL instruction .... After the pin levels of the output pins are read two times, they are exclusive-ORed. As a result, if the pins checked with this instruction are used in the port output mode, the exclusive-OR result is always 0 for all bits, and the Z flag is set to (1).**
- **CHKLA instruction .... After the pin levels of output pins are read two times, they are exclusive-ORed. As a result, if the pins checked with this instruction are used in the port output mode, the exclusive-OR result is always 0 for all bits, and the Z flag is set to (1) along with that the result is stored in the A register.**

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

Mnemonic	Operands	Bytes	Clocks			
			Internal ROM	IRAM	PRAM/EMEM/SFR	Others
MOVTBLW	!addr16, byte	4	–	7 + 5n	–	–
MOVM	[TDE +], A	2	–	3 + 8n	3 + 10n	–
	[TDE –], A	2				
XCHM	[TDE +], A	2	–	3 + 14n	3 + 20n	–
	[TDE –], A	2				
MOVBK	[TDE +], [WHL +]	2	3 + 17n **Note 1**	3 + 13n **Note 2**	3 + 17n **Note 3**	–
	[TDE –], [WHL –]	2				
XCHBK	[TDE +], [WHL +]	2	–	3 + 21n **Note 2**	3 + 29n **Note 3**	–
	[TDE –], [WHL –]	2				
CMPME	[TDE +], A	2	3 + 12n	3 + 10n	3 + 12n	–
	[TDE –], A	2				
CMPMNE	[TDE +], A	2	3 + 12n	3 + 10n	3 + 12n	–
	[TDE –], A	2				
CMPMC	[TDE +], A	2	3 + 12n	3 + 10n	3 + 12n	–
	[TDE –], A	2				
CMPMNC	[TDE +], A	2	3 + 12n	3 + 10n	3 + 12n	–
	[TDE –], A	2				
CMPBKE	[TDE +], [WHL +]	2	3 + 19n **Note 1**	3 + 15n **Note 2**	3 + 19n **Note 3**	–
	[TDE –], [WHL –]	2				
CMPBKNE	[TDE +], [WHL +]	2	3 + 19n **Note 1**	3 + 15n **Note 2**	3 + 19n **Note 3**	–
	[TDE –], [WHL –]	2				
CMPBKC	[TDE +], [WHL +]	2	3 + 19n **Note 1**	3 + 15n **Note 2**	3 + 19n **Note 3**	–
	[TDE –], [WHL –]	2				
CMPBKNC	[TDE +], [WHL +]	2	3 + 19n **Note 1**	3 + 15n **Note 2**	3 + 19n **Note 3**	–
	[TDE –], [WHL –]	2				

Notes 1. When the memory specified by the WHL register is the internal ROM and the memory specified by the TDE register is PRAM or EMEM

2. If both the transfer source and destination memories are IRAM

3. If both the transfer source and destination memories are PRAM or EMEM

CHAPTER 7 DESCRIPTION OF INSTRUCTIONS

This chapter describes the instructions of 78K/IV Series products. Each instruction is described on a mnemonic basis, with a number of operands covered together.

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

Refer to **CHAPTER 6 INSTRUCTION SET** for the number of bytes in the instructions, and the operation codes.

Description Example

Mnemonic

Full name

MOV

Move

Byte Data Transfer

Meaning of Instruction

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

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

[Operands] : Shows the operands that can be specified in this instruction. See **CHAPTER 6 INSTRUCTION SET** for an explanation of the operand symbols.

Mnemonic	Operands
MOV	r, #byte
	saddr, #byte
	A, saddr2
	saddr2, A
	A, mem

Mnemonic	Operands
MOV	[saddrp], A
	[%saddrg], A
	mem, A
	A, r3
	r3, A

[Flags] : Shows the operation of flags changed by execution of the instruction.
The operation symbols for each flag are shown in the legend below.

S	Z	AC	P/V	CY

Legend

Symbol	Meaning
Blank	No change
0	Cleared to 0
1	Set to 1
×	Set or cleared depending on result
P	P/V flag operates as parity flag
V	P/V flag operates as overflow flag
R	Previously saved value is restored

[Description] : Explains the detailed operation of the instruction.

- 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 A register

7.1 8-bit Data Transfer Instruction

There is one 8-bit data transfer instruction, a 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
	!addr16, #byte
	!!addr24, #byte
	r, r'
	A, r
	A, saddr2
	r, saddr
	saddr2, A
	saddr, r
	A, sfr
	r, sfr
	sfr, A
	sfr, r
	saddr, saddr'
	r, !addr16
	!addr16, r

Mnemonic	Operands (dst, src)
MOV	r, !!addr24
	!!addr24, r
	A, [saddrp]
	A, [%saddrg]
	A, mem
	[saddrp], A
	[%saddrg], A
	mem, A
	PSWL, #byte
	PSWH, #byte
	PSWL, A
	PSWH, A
	A, PSWL
	A, PSWH
	r3, #byte
	A, r3
	r3, A

[Flags]

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

S	Z	AC	P/V	CY
×	×	×	×	×

In other cases

S	Z	AC	P/V	CY

[Description]

- The contents of the source operand (src) specified by the 2nd operand are transferred to the destination operand (dst) specified by the 1st operand.
- No interrupts or macro service requests are acknowledged between a MOV PSWL, #byte instruction or MOV PSWL, A instruction and the following instruction.
- Instructions with r3 (T, U, V, or W register) as an operand should only be used when the high-order 8-bit of the address are set when a 78K/0, 78K/I, 78K/II, or 78K/III Series program is used. Also, if possible, the program should be amended so that r3 need not be specified directly.

[Coding example]

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

7.2 16-bit Data Transfer Instruction

There is one 16-bit data transfer instruction, as follows:

MOVW

Move Word
Word Data Transfer

[Instruction format] MOVW dst, src

[Operation] dst ← src

[Operands]

Mnemonic	Operands (dst, src)
MOVW	rp, #word
	saddrp, #word
	sfrp, #word
	!addr16, #word
	!!addr24, #word
	rp, rp'
	AX, saddrp2
	rp, saddrp
	saddr2, AX
	saddrp, rp
	AX, sfrp
	rp, sfrp
	sfrp, AX

Mnemonic	Operands (dst, src)
MOVW	sfrp, rp
	saddrp, saddrp'
	rp, !addr16
	!addr16, rp
	rp, !!addr24
	!!addr24, rp
	AX, [saddrp]
	AX, [%saddrg]
	AX, mem
	[saddrp], AX
	[%saddrg], AX
	mem, AX

[Flags]

S	Z	AC	P/V	CY

[Description]
- The contents of the source operand (src) specified by the 2nd operand are transferred to the destination operand (dst) specified by the 1st operand.
- The following caution should be noted if all the following conditions apply when a 78K/0, 78K/I, 78K/II, or 78K/III Series program is used.

[Conditions]

- An instruction in which rp is specified as an operand is used .
- DE, HL, VP, or UP is actually written for rp .
- DE, HL, VP, or UP is used as an address pointer

[Caution]

Ensure that the contents of the T, W, V, or U register that indicates the high-order 8 bits of the address pointer are coordinated with DE, HL, VP, or UP that indicates the low-order 16 bits, and if program amendment is possible, change the program so that a 24-bit manipulation instruction is used.

[Coding example]

MOVW AX, [WHL] ; Transfers the contents of memory indicated by the WHL register to the AX register

7.3 24-bit Data Transfer Instruction

There is one 24-bit data transfer instruction, as follows:

MOVG

Move G Note
24-Bit Data Transfer

[Instruction format] MOVG dst, src

Note G is a character that indicates that 24-bit data is to be manipulated.

[Operation] dst ← src

[Operands]

Mnemonic	Operands (dst, src)
MOVG	rp, #imm24
	rg, rg'
	rg, !!addr24
	!!addr24, rg
	rg, saddrg
	saddrg, rg
	WHL, [%saddrg]
	[%saddrg], WHL
	WHL, mem1
	mem1, WHL

[Flags]

S	Z	AC	P/V	CY

[Description]

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

[Coding example]

MOVG VVP, SADG ; Transfers the 24-bit data in address SADG that can be accessed by short direct addressing to the VVP register.

7.4 8-bit Data Exchange Instruction

There is one 8-bit data exchange instruction, as follows:

XCH

Exchange
Byte Data Exchange

[Instruction format] XCH dst, src

[Operation] dst ↔ src

[Operands]

Mnemonic	Operands (dst, src)
XCH	r, r'
	A, r
	A, saddr2
	r, saddr
	r, sfr
	saddr, saddr'
	r, !addr16
	r, !!addr24
	A, [saddrp]
	A, [%saddrg]
	A, mem

[Flags]

S	Z	AC	P/V	CY

[Description]

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

[Coding example]

XCH B, D ; Exchanges the contents of the B register with the contents of the D register

7.5 16-bit Data Exchange Instruction

There is one 16-bit data exchange instruction, as follows:

XCHW

Exchange
Word Data Exchange

[Instruction format] XCHW dst, src

[Operation] dst ↔ src

[Operands]

Mnemonic	Operands (dst, src)
XCHW	rp, rp’
	AX, saddrp2
	rp, saddrp
	rp, sfrp
	AX, [saddrp]
	AX, [%saddrg]
	AX, !addr16
	AX, !!addr24
	saddrp, saddrp’
	AX, mem

[Flags]

S	Z	AC	P/V	CY

[Description]

- Exchanges the contents of the 1st operand and 2nd operand.
- The following caution should be noted if all the following conditions apply when a 78K/0, 78K/I, 78K/II, or 78K/III Series program is used.

[Conditions]

- An instruction in which rp is specified as an operand is used
- DE, HL, VP, or UP is actually written for rp
- DE, HL, VP, or UP is used as an address pointer

[Caution]

Ensure that the contents of the T, W, V, or U register that indicates the high-order 8 bits of the address pointer are coordinated with DE, HL, VP, or UP that indicates the low-order 16 bits, and if program amendment is possible, change the program so that a 24-bit manipulation instruction is used.

[Coding example]

XCHW AX, mem ; Exchanges the contents of the AX register with the memory contents addressed by memory addressing

7.6 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
	r, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, saddr2
	r, saddr
	saddr, r
	r, sfr
	sfr, r
	saddr, saddr'

Mnemonic	Operands (dst, src)
ADD	A, [saddrp]
	A, [%saddrg]
	[saddrp], A
	[%saddrg], A
	A, !addr16
	A, !!addr24
	!addr16, A
	!!addr24, A
	A, mem
	mem, A

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The source operand (src) specified by the 2nd operand is added to the destination operand (dst) specified by the 1st operand, and the result is stored in the CY flag and destination operand (dst).
- The S flag is set (1) if bit 7 of dst is set (1) as a result of the addition, and cleared (0) otherwise.
- The Z flag is set (1) if dst is 0 as a result of the addition, and cleared (0) otherwise.
- The AC flag is set (1) if a carry is generated out of bit 3 into bit 4 as a result of the addition, and cleared (0) otherwise.
- The P/V flag is set (1) if a carry is generated out of bit 6 into bit 7 and a carry is not generated out of bit 7 as a result of the addition (when overflow is generated by a two's complement type operation), or if a carry is not generated out of bit 6 into bit 7 and a carry is generated out of bit 7 (when underflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a carry is generated out of bit 7 as a result of the addition, and cleared (0) otherwise.

[Coding example]

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

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
	r, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, saddr2
	r, saddr
	saddr, r
	r, sfr
	sfr, r
	saddr, saddr'

Mnemonic	Operands (dst, src)
ADDC	A, [saddrp]
	A, [%saddrg]
	[saddrp], A
	[%saddrg], A
	A, !addr16
	A, !!addr24
	!addr16, A
	!!addr24, A
	A, mem
	mem, A

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The source operand (src) specified by the 2nd operand and the CY flag are added to the destination operand (dst) specified by the 1st operand, and the result is stored in the destination operand (dst) and the CY flag. This instruction is mainly used when performing multiple byte addition.
- The S flag is set (1) if bit 7 of dst is set (1) as a result of the addition, and cleared (0) otherwise.
- The Z flag is set (1) if dst is 0 as a result of the addition, and cleared (0) otherwise.
- The AC flag is set (1) if a carry is generated out of bit 3 into bit 4 as a result of the addition, and cleared (0) otherwise.
- The P/V flag is set (1) if a carry is generated out of bit 6 into bit 7 and a carry is not generated out of bit 7 as a result of the addition (when overflow is generated by a two's complement type operation), or if a carry is not generated out of bit 6 into bit 7 and a carry is generated out of bit 7 (when underflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a carry is generated out of bit 7 as a result of the addition, and cleared (0) otherwise.

[Coding example]

ADDC A, 12345H [B] ; Adds the contents of address (12345H + (B register)) and the CY flag to the A register, 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
	r, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, saddr2
	r, saddr
	saddr, r
	r, sfr
	sfr, r
	saddr, saddr'

Mnemonic	Operands (dst, src)
SUB	A, [saddrp]
	A, [%saddrg]
	[saddrp], A
	[%saddrg], A
	A, !addr16
	A, !!addr24
	!addr16, A
	!!addr24, A
	A, mem
	mem, A

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Description]
- The source operand (src) specified by the 2nd operand is subtracted from the destination operand (dst) specified by the 1st operand, and the result is stored in the destination operand (dst) and the CY flag.
- The destination operand (dst) can be cleared to 0 by making the source operand (src) and destination operand (dst) the same.
- The S flag is set (1) if bit 7 of dst is set (1) as a result of the subtraction, and cleared (0) otherwise.
- The Z flag is set (1) if dst is 0 as a result of the subtraction, and cleared (0) otherwise.
- The AC flag is set (1) if a borrow is generated out of bit 4 into bit 3 as a result of the subtraction, and cleared (0) otherwise.
- The P/V flag is set (1) if a borrow is generated out of bit 6 into bit 7 and a borrow is not generated in bit 7 as a result of the subtraction (when underflow is generated by a two's complement type operation), or if a borrow is not generated out of bit 6 into bit 7 and a borrow is generated in bit 7 (when overflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a borrow is generated in bit 7 as a result of the subtraction, and cleared (0) otherwise.

[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
	r, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, saddr2
	r, saddr
	saddr, r
	r, sfr
	sfr, r
	saddr, saddr'

Mnemonic	Operands (dst, src)
SUBC	A, [saddrp]
	A, [%saddrg]
	[saddrp], A
	[%saddrg], A
	A, !addr16
	A, !!addr24
	!addr16, A
	!!addr24, A
	A, mem
	mem, A

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The source operand (src) specified by the 2nd operand and the CY flag are subtracted from the destination operand (dst) specified by the 1st operand, and the result is stored in the destination operand (dst) and the CY flag. The CY flag is subtracted from the LSB. This instruction is mainly used when performing multiple byte subtraction.
- The S flag is set (1) if bit 7 of dst is set (1) as a result of the subtraction, and cleared (0) otherwise.
- The Z flag is set (1) if dst is 0 as a result of the subtraction, and cleared (0) otherwise.
- The AC flag is set (1) if a borrow is generated out of bit 4 into bit 3 as a result of the subtraction, and cleared (0) otherwise.
- The P/V flag is set (1) if a borrow is generated out of bit 6 into bit 7 and a borrow is not generated in bit 7 as a result of the subtraction (when underflow is generated by a two's complement type operation), or if a borrow is not generated out of bit 6 into bit 7 and a borrow is generated in bit 7 (when overflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a borrow is generated in bit 7 as a result of the subtraction, and cleared (0) otherwise.

[Coding example]

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

CMP

Compare
Byte Data Comparison

[Instruction format] CMP dst, src

[Operation] dst – src

[Operands]

Mnemonic	Operands (dst, src)
CMP	A, #byte
	r, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, saddr2
	r, saddr
	saddr, r
	r, sfr
	sfr, r
	saddr, saddr'

Mnemonic	Operands (dst, src)
CMP	A, [saddrp]
	A, [%saddrg]
	[saddrp], A
	[%saddrg], A
	A, !addr16
	A, !!addr24
	!addr16, A
	!!addr24, A
	A, mem
	mem, A

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The source operand (src) specified by the 2nd operand is subtracted from the destination operand (dst) specified by the 1st operand.
  The result of the subtraction is not stored anywhere, and only the S, Z, AC, P/V, and CY flags are changed.
- The S flag is set (1) if bit 7 is set (1) as a result of the subtraction, and cleared (0) otherwise.
- The Z flag is set (1) if dst is 0 as a result of the subtraction, and cleared (0) otherwise.
- The AC flag is set (1) if a borrow is generated out of bit 4 into bit 3 as a result of the subtraction, and cleared (0) otherwise.
- The P/V flag is set (1) if a borrow is generated in bit 7 and a borrow is not generated in bit 6 as a result of the subtraction (when underflow is generated by a two's complement type operation), or if a borrow is not generated in bit 7 and a borrow is generated in bit 6 (when overflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a borrow is generated in bit 7 as a result of the subtraction, and cleared (0) otherwise.

[Coding example]

CMP SADG1, SADG2 ; Subtracts the contents of address SADG2 that can be accessed by short direct addressing from the contents of address SADG1 that can be accessed by short direct addressing, and changes the flags only (comparison of the contents of address SADG1 and the contents of address SADG2)

AND

And
Byte Data Logical Product

[Instruction format] AND dst, src

[Operation] dst ← dst ∧ src

[Operands]

Mnemonic	Operands (dst, src)
AND	A, #byte
	r, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, saddr2
	r, saddr
	saddr, r
	r, sfr
	sfr, r
	saddr, saddr'

Mnemonic	Operands (dst, src)
AND	A, [saddrp]
	A, [%saddrg]
	[saddrp], A
	[%saddrg], A
	A, !addr16
	A, !!addr24
	!addr16, A
	!!addr24, A
	A, mem
	mem, A

[Flags]

S	Z	AC	P/V	CY
×	×		P	

[Description]

- The bit-wise logical sum of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand is found, and the result is stored in the destination operand (dst).
- The S flag is set (1) if bit 7 of dst is set (1) as a result of the logical product operation, and cleared (0) otherwise.
- The Z flag is set (1) if all bits are 0 as a result of the logical product operation, and cleared (0) otherwise.
- The P/V flag is set (1) if the number of bits set (1) in dst as a result of the logical product operation is even, and cleared (0) otherwise.

[Coding example]

AND SADG, #11011100B ; Finds the bit-wise logical product of the contents of address SADG that can be accessed by short direct addressing and 11011100B, and stores the result in SADG

OR

Or
Byte Data Logical Sum

[Instruction format] OR dst, src

[Operation] dst ← dst ∨ src

[Operands]

Mnemonic	Operands (dst, src)
OR	A, #byte
	r, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, saddr2
	r, saddr
	saddr, r
	r, sfr
	sfr, r
	saddr, saddr'

Mnemonic	Operands (dst, src)
OR	A, [saddrp]
	A, [%saddrg]
	[saddrp], A
	[%saddrg], A
	A, !addr16
	A, !!addr24
	!addr16, A
	!!addr24, A
	A, mem
	mem, A

[Flags]

S	Z	AC	P/V	CY
×	×		P	

[Description]

- The bit-wise logical sum of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand is found, and the result is stored in the destination operand (dst).
- The S flag is set (1) if bit 7 of dst is set (1) as a result of the logical sum operation, and cleared (0) otherwise.
- The Z flag is set (1) if all bits are 0 as a result of the logical sum operation, and cleared (0) otherwise.
- The P/V flag is set (1) if the number of bits set (1) in dst as a result of the logical sum operation is even, and cleared (0) otherwise.

[Coding example]

OR A, !!12345H ; Finds the bit-wise logical sum of the contents of the A register and address 12345H, and stores the result in the A register

XOR

Exclusive Or
Byte Data Exclusive Logical Sum

[Instruction format] XOR dst, src

[Operation] dst ← dst $\forall$ src

[Operands]

Mnemonic	Operands (dst, src)
XOR	A, #byte
	r, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, saddr2
	r, saddr
	saddr, r
	r, sfr
	sfr, r
	saddr, saddr'

Mnemonic	Operands (dst, src)
XOR	A, [saddrp]
	A, [%saddrg]
	[saddrp], A
	[%saddrg], A
	A, !addr16
	A, !!addr24
	!addr16, A
	!!addr24, A
	A, mem
	mem, A

[Flags]

S	Z	AC	P/V	CY
×	×		P	

[Description]
- The bit-wise exclusive logical sum of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand is found, and the result is stored in the destination operand (dst).
- Selecting #0FFH as the source operand (src) in this instruction results in logical negation of all the bits of the destination operand (dst).
- The S flag is set (1) if bit 7 of dst is set (1) as a result of the exclusive logical sum operation, and cleared (0) otherwise.
- The Z flag is set (1) if all bits are 0 as a result of the exclusive logical sum operation, and cleared (0) otherwise.
- The P/V flag is set (1) if the number of bits set (1) in dst as a result of the exclusive logical sum operation is even, and cleared (0) otherwise.

[Coding example]
XOR C, P2 ; Finds the bit-wise exclusive logical sum of the C register and P2 register, and stores the result in the C register

7.7 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
	rp, #word
	rp, rp'
	AX, saddrp2
	rp, saddrp
	saddrp, rp
	rp, sfrp
	sfrp, rp
	saddrp, #word
	sfrp, #word
	saddrp, saddrp'

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The source operand (src) specified by the 2nd operand is added to the destination operand (dst) specified by the 1st operand, and the result is stored in the destination operand (dst).
- The S flag is set (1) if bit 15 of dst is set (1) as a result of the addition, and cleared (0) otherwise.
- The Z flag is set (1) if dst is 0 as a result of the addition, and cleared (0) otherwise.
- The AC flag is undefined as a result of the addition.
- The P/V flag is set (1) if a carry is generated out of bit 14 into bit 15 and a carry is not generated out of bit 15 as a result of the addition (when overflow is generated by a two's complement type operation), or if a carry is not generated out of bit 14 into bit 15 and a carry is generated out of bit 15 (when underflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a carry is generated out of bit 15 as a result of the addition, and cleared (0) otherwise.
- The following caution should be noted if all the following conditions apply when a 78K/0, 78K/I, 78K/II, or 78K/III Series program is used.

[Conditions]

- An instruction in which rp is specified as an operand is used
- DE, HL, VP, or UP is actually written for rp
- DE, HL, VP, or UP is used as an address pointer

[Caution]

Ensure that the contents of the T, W, V, or U register that indicates the high-order 8 bits of the address pointer are coordinated with DE, HL, VP, or UP that indicates the low-order 16 bits, and if program amendment is possible, change the program so that a 24-bit manipulation instruction is used.

[Coding example]

ADDW BC, #0ABCDH ; Adds 0ABCDH to the BC register, and stores the result in the BC 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
	rp, #word
	rp, rp’
	AX, saddrp2
	rp, saddrp
	saddrp, rp
	rp, sfrp
	sfrp, rp
	saddrp, #word
	sfrp, #word
	saddrp, saddrp’

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The source operand (src) specified by the 2nd operand is subtracted from the destination operand (dst) specified by the 1st operand, and the result is stored in the destination operand (dst) and the CY flag.
- The destination operand (dst) can be cleared to 0 by making the source operand (src) and destination operand (dst) the same.
- The S flag is set (1) if bit 15 of dst is set (1) as a result of the subtraction, and cleared (0) otherwise.
- The Z flag is set (1) if dst is 0 as a result of the subtraction, and cleared (0) otherwise.
- The AC flag is undefined as a result of the subtraction.
- The P/V flag is set (1) if a borrow is generated out of bit 14 into bit 15 and a borrow is not generated in bit 15 as a result of the subtraction (when underflow is generated by a two’s complement type operation), or if a borrow is not generated out of bit 14 into bit 15 and a borrow is generated in bit 15 (when overflow is generated by a two’s complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a borrow is generated in bit 15 as a result of the subtraction, and cleared (0) otherwise.
- The following caution should be noted if all the following conditions apply when a 78K/0, 78K/I, 78K/II, or 78K/III Series program is used.

[Conditions]

- An instruction in which rp is specified as an operand is used
- DE, HL, VP, or UP is actually written for rp
- DE, HL, VP, or UP is used as an address pointer

[Caution]

Ensure that the contents of the T, W, V, or U register that indicates the high-order 8 bits of the address pointer are coordinated with DE, HL, VP, or UP that indicates the low-order 16 bits, and if program amendment is possible, change the program so that a 24-bit manipulation instruction is used.

[Coding example]

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

CMPW

Compare Word
Word Data Comparison

[Instruction format] CMPW dst, src

[Operation] dst – src

[Operands]

Mnemonic	Operands (dst, src)
CMPW	AX, #word
	rp, #word
	rp, rp'
	AX, saddrp2
	rp, saddrp
	saddrp, rp
	rp, sfrp
	sfrp, rp
	saddrp, #word
	sfrp, #word
	saddrp, saddrp'

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The source operand (src) specified by the 2nd operand is subtracted from the destination operand (dst) specified by the 1st operand. The result of the subtraction is not stored anywhere, and only the Z, AC, and CY flags are changed.
- The S flag is set (1) if bit 15 is set (1) as a result of the subtraction, and cleared (0) otherwise.
- The Z flag is set (1) if dst is 0 as a result of the subtraction, and cleared (0) otherwise.
- The AC flag is undefined as a result of the subtraction.
- The P/V flag is set (1) if a borrow is generated out of bit 14 into bit 15 and a borrow is not generated in bit 15 as a result of the subtraction (when underflow is generated by a two's complement type operation), or if a borrow is not generated out of bit 14 into bit 15 and a borrow is generated in bit 15 (when overflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a borrow is generated in bit 15 as a result of the subtraction, and cleared (0) otherwise.
- The following caution should be noted if all the following conditions apply when a 78K/0, 78K/I, 78K/II, or 78K/III Series program is used.

[Conditions]

- An instruction in which rp is specified as an operand is used
- DE, HL, VP, or UP is actually written for rp
- DE, HL, VP, or UP is used as an address pointer

[Caution]

Ensure that the contents of the T, W, V, or U register that indicates the high-order 8 bits of the address pointer are coordinated with DE, HL, VP, or UP that indicates the low-order 16 bits, and if program amendment is possible, change the program so that a 24-bit manipulation instruction is used.

[Coding example]

CMPW AX, SADG ; Subtracts the word data in address SADG that can be accessed by short direct addressing from the AX register, and changes the flags only (comparison of AX register and address SADG word data)

7.8 24-bit Operation Instructions

24-bit operation instructions are as follows:

ADDG

Add G Note
24-Bit Data Addition

[Instruction format] ADDG dst, src

Note G is a character that indicates that 24-bit data is to be manipulated.

[Operation] dst ← dst + src

[Operands]

Mnemonic	Operands (dst, src)
ADDG	rg, rg'
	rg, #imm24
	WHL, saddrg

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The source operand (src) specified by the 2nd operand is added to the destination operand (dst) specified by the 1st operand. The result of the addition is stored in dst, and the S, Z, AC, P/V, and CY flags are changed.
- The S flag is set (1) if bit 23 of dst is set (1) as a result of the addition, and cleared (0) otherwise.
- The Z flag is set (1) if the result of the addition is 0, and cleared (0) otherwise.
- The AC flag is undefined as a result of the addition.
- The P/V flag is set (1) if a carry is generated out of bit 22 into bit 23 and a carry is not generated out of bit 23 as a result of the addition (when overflow is generated by a two's complement type operation), or if a carry is not generated out of bit 22 into bit 23 and a carry is generated out of bit 23 (when underflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a carry is generated out of bit 23 as a result of the addition, and cleared (0) otherwise.

[Coding example]

ADDG TDE, VVP ; Adds the VVP register to the TDE register, and stores the result in the TDE register

SUBG

Subtract G Note
24-Bit Data Subtraction

[Instruction format] SUBG dst, src

Note G is a character that indicates that 24-bit data is to be manipulated.

[Operation] dst ← dst – src

[Operands]

Mnemonic	Operands (dst, src)
SUBG	rg, rg'
	rg, #imm24
	WHL, saddrg

[Flags]

S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The source operand (src) specified by the 2nd operand is subtracted from the destination operand (dst) specified by the 1st operand. The result of the subtraction is stored in dst, and the S, Z, AC, P/V, and CY flags are changed.
- The S flag is set (1) if bit 23 of dst is set (1) as a result of the subtraction, and cleared (0) otherwise.
- The Z flag is set (1) if the result of the subtraction is 0, and cleared (0) otherwise.
- The AC flag is undefined as a result of the subtraction.
- The P/V flag is set (1) if a borrow is generated out of bit 23 into bit 22 and a borrow is not generated in bit 23 as a result of the subtraction (when underflow is generated by a two's complement type operation), or if a borrow is not generated out of bit 23 into bit 22 and a borrow is generated in bit 23 (when overflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a borrow is generated in bit 23 as a result of the subtraction, and cleared (0) otherwise.

[Coding example]

SUBG UUP, #543210H ; Subtracts 543210H from the contents of the UUP register and stores the result in the UUP register

7.9 Multiplication/Division Instructions

Multiplication/division instructions are as follows:

MULU

Multiply Unsigned
Unsigned Data Multiplication

[Instruction format] MULU src

[Operation] AX ← A × src

[Operands]

Mnemonic	Operands (src)
MULU	r

[Flags]

S	Z	AC	P/V	CY

[Description]

- The contents of the A register and the source operand (src) data are multiplied as unsigned data, and the result is stored in the AX register.

[Coding example]

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

MULUW

Multiply Unsigned Word
Unsigned Word Data Multiplication

[Instruction format] MULUW src

[Operation] AX (upper half), src (lower half) ← AX × src

[Operands]

Mnemonic	Operands (src)
MULUW	rp

[Flags]

S	Z	AC	P/V	CY

[Description]

- The contents of the AX register and the source operand (src) data are multiplied as unsigned data, and the high-order 16 bits of the result are stored in the AX register, and the low-order 16 bits in the source operand.
- When the AX register is specified as the source operand (src), the high-order 16 bits of the multiplied result are stored in the AX register, and the low-order 16 bits are not stored anywhere.

[Coding example]

MULUW HL ; Multiplies the contents of the AX register by the contents of the HL register, and stores the result in the AX register and HL register

MULW

Multiply Signed Word
Signed Word Data Multiplication

[Instruction format] MULW src

[Operation] AX (upper half), src (lower half) ← AX × src

[Operands]

Mnemonic	Operands (src)
MULW	rp

[Flags]

S	Z	AC	P/V	CY

[Description]

- The contents of the AX register and the source operand (src) data are multiplied as signed data, and the high-order 16 bits of the result are stored in the AX register, and the low-order 16 bits in the source operand.
- When the AX register is specified as the source operand (src), the high-order 16 bits of the multiplied result are stored in the AX register, and the low-order 16 bits are not stored anywhere.

[Coding example]

MULW HL ; Multiplies the contents of the AX register by the contents of the HL register, and stores the result in the AX register and HL 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]

S	Z	AC	P/V	CY

[Description]

- The contents of the AX register are divided by the contents of the destination operand (dst), and the quotient is stored in the AX register, and the remainder in the destination operand (dst).
  The division is performed with the AX register and destination operand (dst) contents as unsigned data.
- If division by 0 is performed, the following will result:
  – AX (quotient) = FFFFH
  – dst (remainder) = Original X register value
- When the A register is specified as the destination operand (dst), the remainder is stored in the A register, and the low-order 8 bits of the quotient are stored in the X register.
- When the X register is specified as the destination operand (dst), the high-order 8 bits of the quotient are stored in the A register, and the remainder is stored in the X register.

[Coding example]

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

DIVUX

Divide Unsigned Word Expansion Word
Unsigned Doubleword Data Division

[Instruction format] DIVUX dst

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

[Operands]

Mnemonic	Operands (dst)
DIVUX	rp

[Flags]

S	Z	AC	P/V	CY

[Description]

- 32-bit data with the contents of the AX register as the high-order 16 bits and the contents of the DE register as the low-order 16 bits is divided by the contents of the destination operand (dst), the high-order 16 bits of the quotient are stored in the AX register, the low-order 16 bits in the DE register, and the remainder in the destination operand (dst).
  The division is performed with the contents of the 32-bit data indicated by the AX register and DE register and the contents of the destination operand (dst) as unsigned data.
- If division by 0 is performed, the following will result:
  – AXDE (quotient) = FFFFFFFFH
  – dst (remainder) = Original DE register value
- When the AX register is specified as the destination operand (dst), the remainder is stored in the AX register, and the low-order 16 bits of the quotient are stored in the DE register.
- When the DE register is specified as the destination operand (dst), the high-order 8 bits of the quotient is stored in the AX register, and the remainder is stored in the DE register.

[Coding example]

DIVUX BC ; Divides the contents of the AXDE register by the contents of the BC register, and stores the high-order 16 bits of the quotient in the AX register, the low-order 16 bits in the DE register, and the remainder in the BC register

7.10 Special Operation Instructions

Special operation instructions are as follows:

MACW

Multiply and Accumulate Word
Word Data Sum of Products Operation

[Instruction format] MACW dst

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

[Operands]

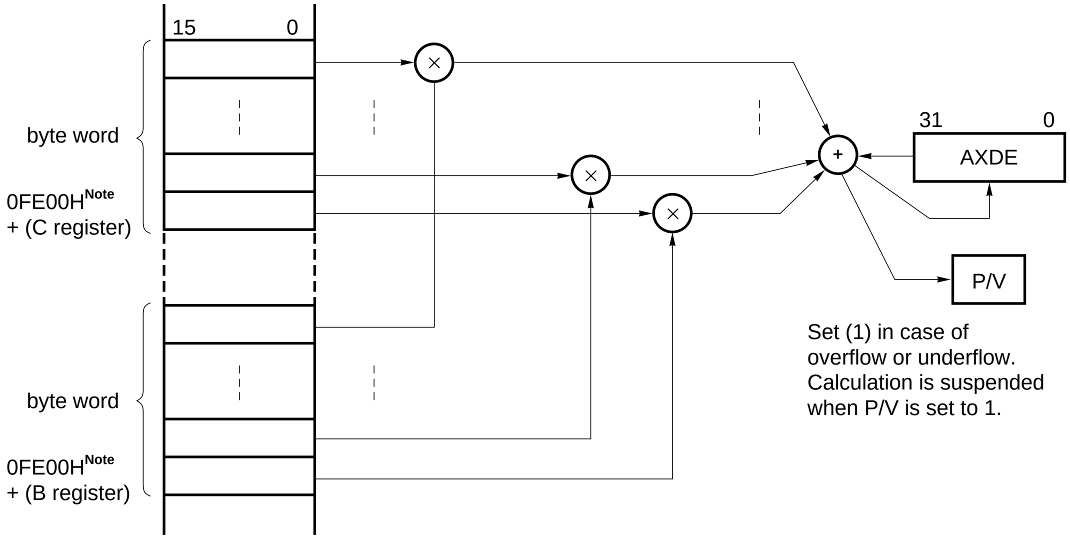
Mnemonic	Operands (dst, src)
MACW	byte

[Flags]

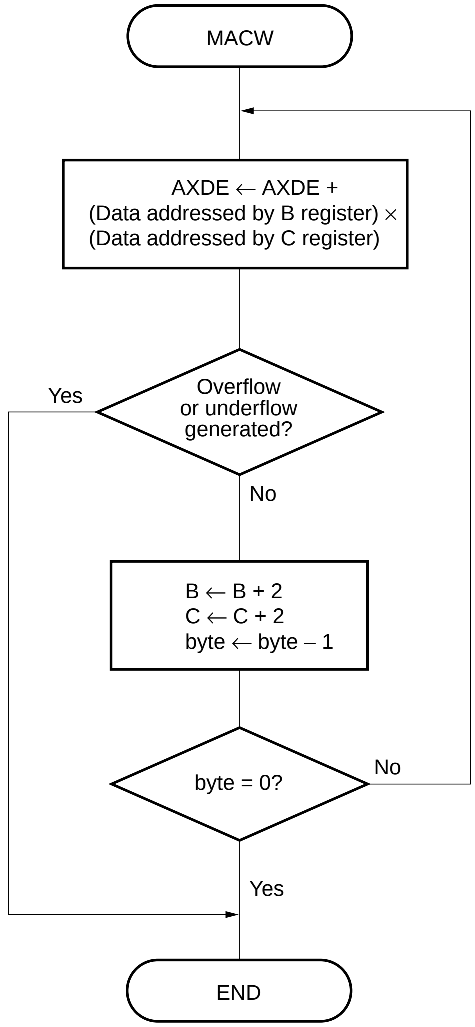
S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- Signed multiplication is performed of the contents of the 2-byte area addressed by the B register and the contents of the 2-byte area addressed by the C register, and binary addition is performed of the result and the contents of the AXDE register.
- After the result of the addition is stored in the AXDE register, the contents of the B register and C register are incremented by 2.
- The above operations are repeated the number of times equal to the 8-bit immediate data written in the operand.
- If overflow or underflow is generated as a result of the addition, the value of the AXDE register is undefined. Also, the B register and C register keep their values prior to overflow.
- The area addressed by the MACW instruction is limited to addresses 0FE00H to 0FEFFH when the LOCATION 0 instruction is executed, or addresses 0FFE00H to 0FFEFFH when the LOCATION 0FH instruction is executed. The lower byte of the address is specified by the B register and C register. Addresses FE80H to FEFFH (FFE80H to FFEFFH when the LOCATION 0FH instruction is executed) are also used as general registers.
- Interrupts and macro service requests are not acknowledged during execution of the MACW instruction.
- The MACW instruction does not clear the value of the AXDE register pair automatically, and therefore this should be cleared by the program if necessary.
- The S, Z, AC, and CY flags are undefined as a result of the operation.
- The P/V flag is set (1) if overflow or underflow occurs, and cleared (0) otherwise.



Note When a LOCATION 0 instruction is executed. 0FFE00H when a LOCATION 0FH instruction is executed.
The μPD784915 Subseries is fixed to the LOCATION 0 instruction.



[Coding example]

MACW 5 ; Executes sum of products operation 5 times

MACSW

Multiply and Accumulate with Saturation Word
Sum of Products Operation with Saturation Function

[Instruction format] MACSW byte

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

[Operands]

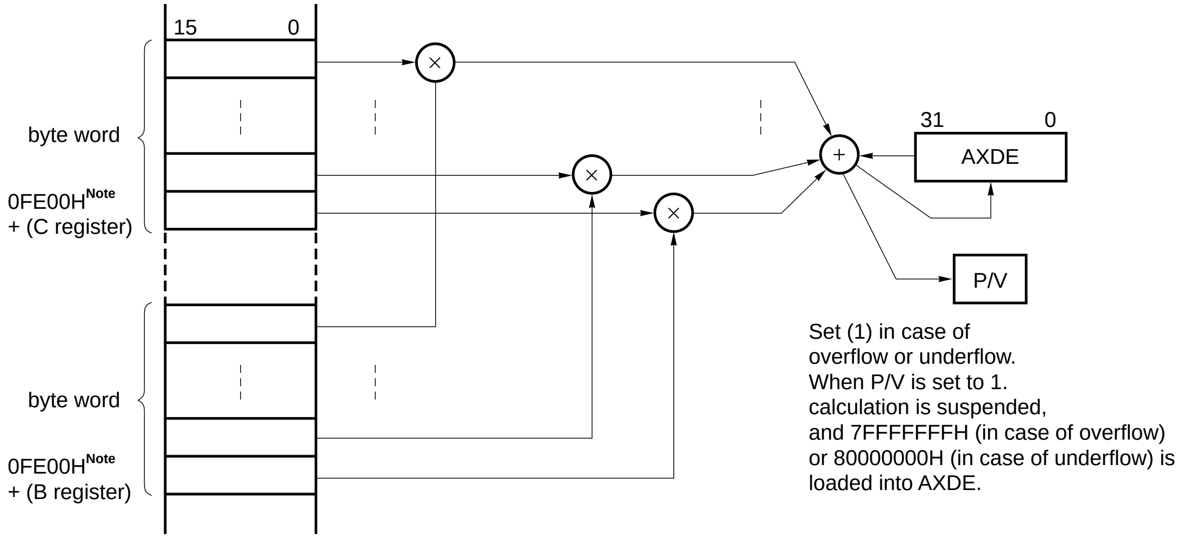
Mnemonic	Operands ($addr16)
MACSW	byte

[Flags]

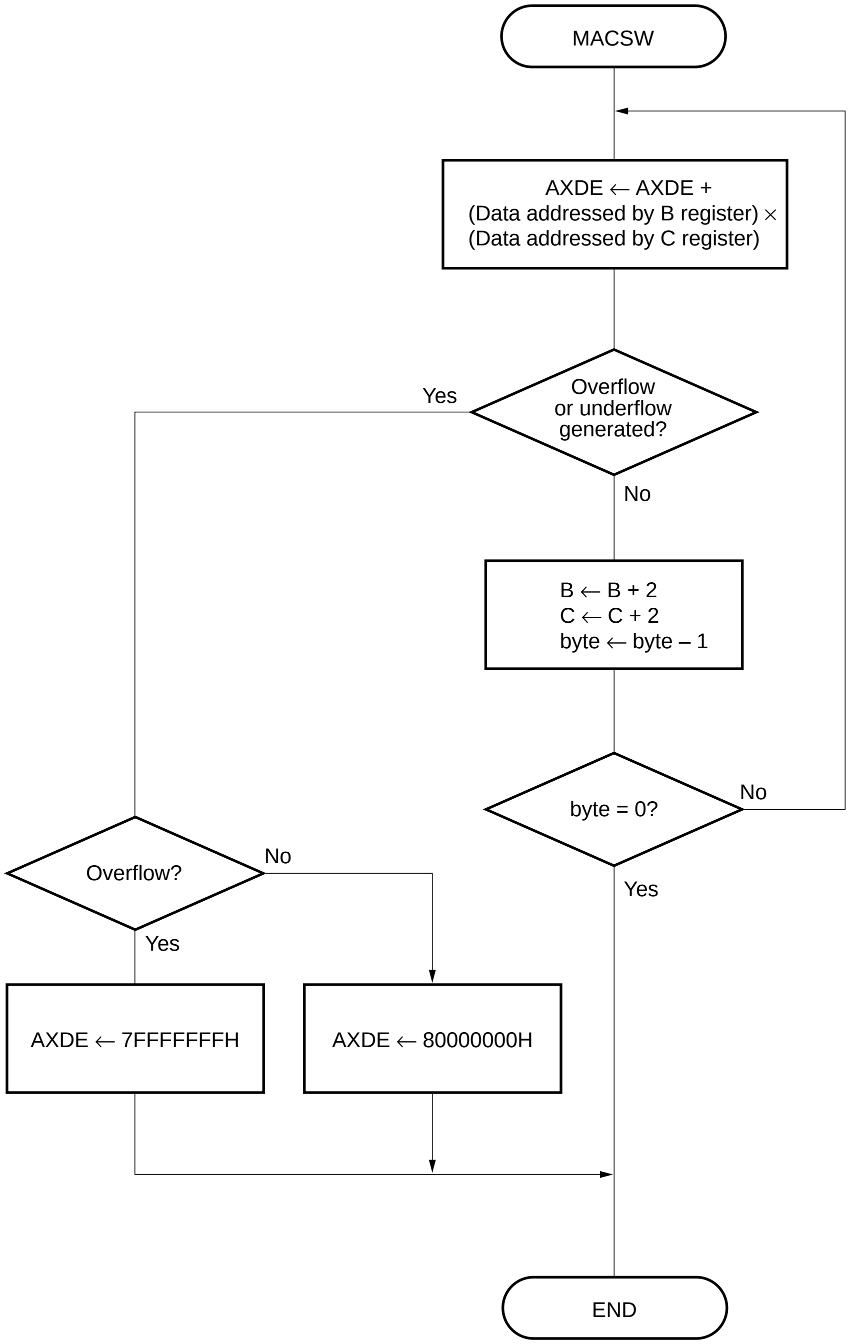
S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- Signed multiplication is performed of the contents of the 2-byte area addressed by the B register and the contents of the 2-byte area addressed by the C register, and binary addition is performed of the result and the contents of the AXDE register.
- After the result of the addition is stored in the AXDE register, the contents of the B register and C register are incremented by 2.
- The above operations are repeated the number of times equal to the 8-bit immediate data written in the operand.
- If overflow is generated as a result of the addition, the P/V flag is set (1) and the value of the AXDE register is 7FFFFFFFH. If underflow is generated, the P/V flag is set (1) and the AXDE register value is 80000000H. The B register and C register keep their values prior to overflow or underflow.
- The area addressed by the MACSW instruction is limited to addresses 0FE00H to 0FEFFH when the LOCATION 0 instruction is executed, or addresses 0FFE00H to 0FFEFFH when the LOCATION 0FH instruction is executed. The lower byte of the address is specified by the B register and C register. Addresses FE80H to FEFFH (FFE80H to FFEFFH when the LOCATION 0FH instruction is executed) are also used as general registers.
- Interrupts and macro service requests are not acknowledged during execution of the MACSW instruction.
- The MACSW instruction does not clear the value of the AXDE register pair automatically, and therefore this should be cleared by the program if necessary.
- The S, Z, AC, and CY flags are undefined as a result of the operation.
- The P/V flag is set (1) if overflow or underflow occurs, and cleared (0) otherwise.



Note When a LOCATION 0 instruction is executed. 0FFE00H when a LOCATION 0FH instruction is executed.
The μPD784915 Subseries is fixed to the LOCATION 0 instruction.



[Coding example]

MACSW 6 ; Executes sum of products operation 6 times

SACW

Subtract, Absolute and Accumulate Word Correlation Instruction

[Instruction format] SACW [TDE +], [WHL +]

[Operation] AX ← | (TDE) – (WHL) | + AX, TDE ← TDE + 2, WHL ← WHL + 2, C ← C – 1, end if (C = 0 or CY = 1)

[Operands]

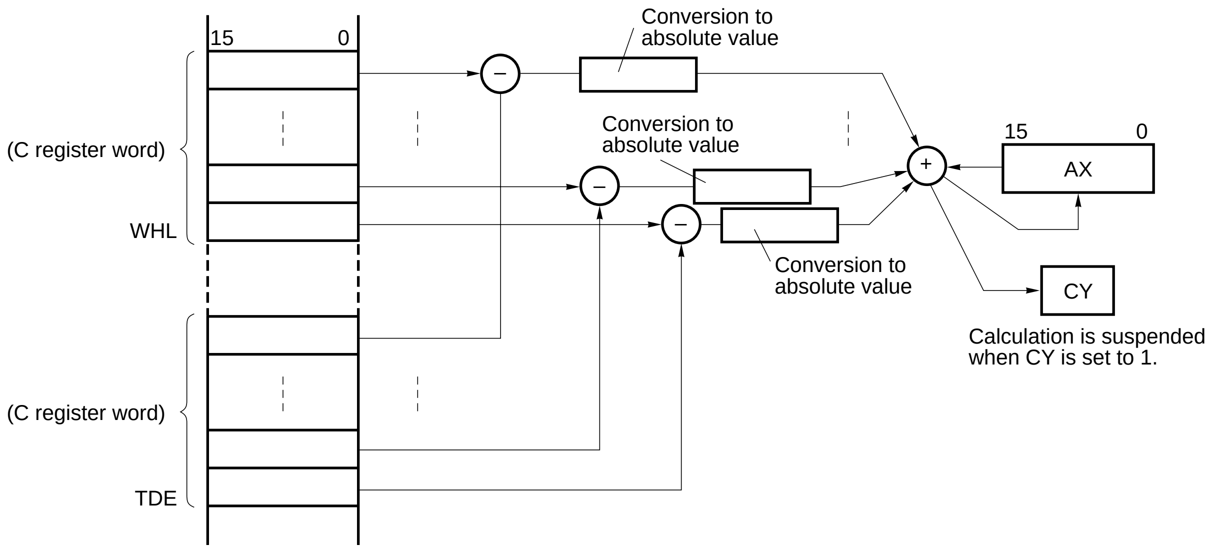
Mnemonic	Operands ($addr16)
SACW	[TDE +], [WHL +]

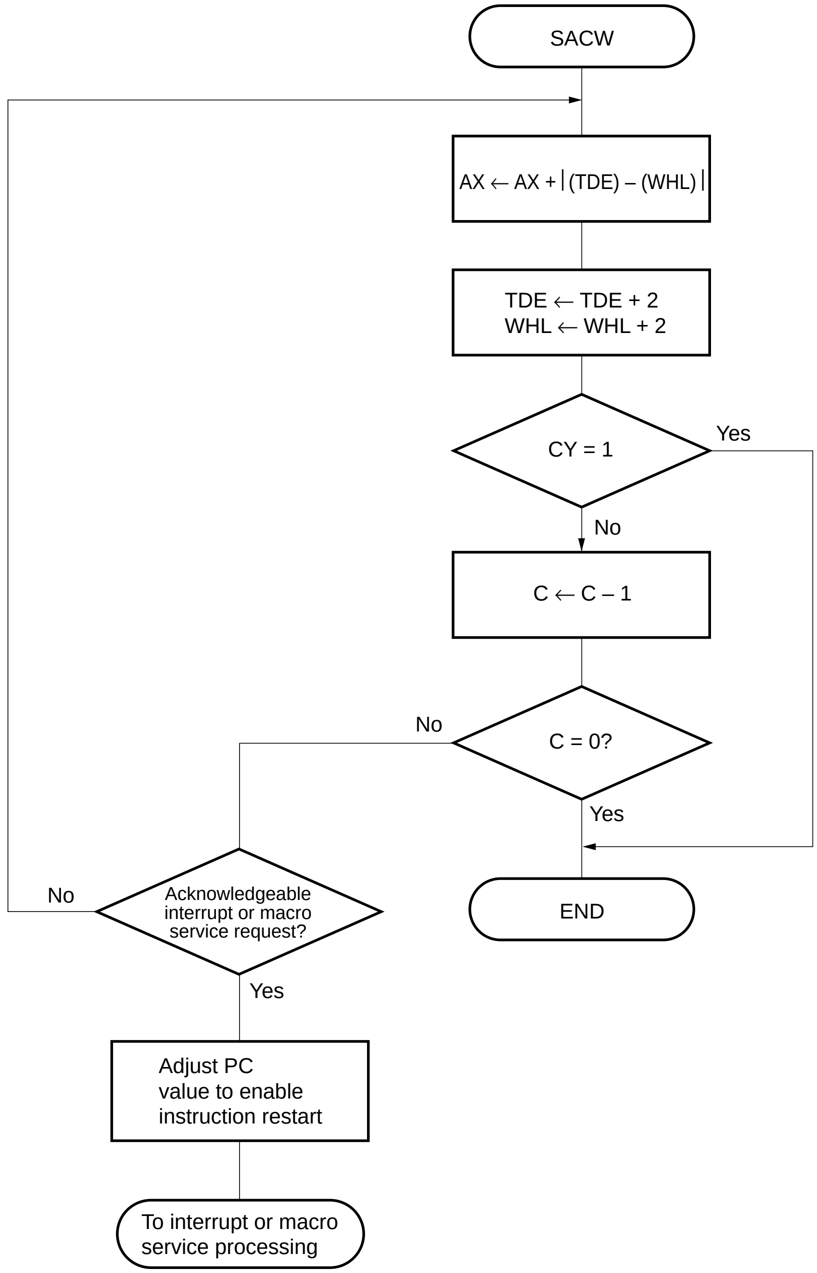
[Flags]

S	Z	AC	P/V	CY
×	×	×	×	×

[Description]

- Subtraction is performed on the 16-bit data items addressed by the TDE register and WHL register, the absolute value of the result is added to the contents of the AX register, and the result is stored in the AX register.
- Each time the above operation is performed, the contents of the TDE and WHL registers are incremented by 2, and the contents of the C register are decremented by 1.
- The above operations are repeated until the C register value is 0 or a carry is generated out of bit 15 as a result of the addition.
- If a carry occurs from bit 15 as a result of addition, therefore stopping repetition, the TDE and WHL registers retain the values immediately before the carry has occurred plus 2. The C register retains the value immediately before the carry has occurred.
- If an interrupt or macro service request that can be acknowledged during execution of the SACW instruction is generated, the interrupt or macro service processing is performed before the series of operation processing. When an interrupt is acknowledged, the program counter (PC) value saved to the stack is the SACW instruction start address.
  Therefore, after returning from the interrupt, continuation of the interrupted SACW instruction is executed.
- The CY flag is set (1) if a carry is generated out of bit 15 as a result of the final addition, and cleared (0) otherwise.
- The contents of the S, Z, AC, and P/V flags are undefined.
- The SACW instruction does not clear the contents of the AX register pair automatically, and therefore this should be done by the program if necessary.


15
0
(C register word)
WHL
(C register word)
TDE
Conversion to
absolute value
Conversion to
absolute value
Conversion to
absolute value
15
0
AX
CY
Calculation is suspended
when CY is set to 1.



[Coding example]

SACW [TDE+], [WHL+] ; Executes a correlation instruction

7.11 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]

S	Z	AC	P/V	CY
×	×	×	V	

[Description]
- The contents of the destination operand (dst) are incremented by 1.
- The Z flag is set (1) if the result of the increment is 0, and cleared (0) otherwise.
- The AC flag is set (1) if a carry is generated out of bit 3 into bit 4 as a result of the increment, and cleared (0) otherwise.
- The CY flag value does not change since this is often used for repeat processing counter or indexed address offset register incrementing (as the CY flag value is retained in a multiple-byte operation).
- The S flag is set (1) if bit 7 of dst is set (1) as a result of the increment, and cleared (0) otherwise.
- The P/V flag is set (1) if a carry is generated out of bit 6 into bit 7 and a carry is not generated out of bit 7 as a result of the increment (when overflow is generated by a two's complement type operation), and is cleared (0) otherwise.

[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]

S	Z	AC	P/V	CY
×	×	×	V	

[Description]

- The contents of the destination operand (dst) are decremented by 1.
- The Z flag is set (1) if the result of the decrement is 0, and cleared (0) otherwise.
- The AC flag is set (1) if a carry is generated out of bit 4 into bit 3 as a result of the decrement, and cleared (0) otherwise.
- The CY flag value does not change since this is often used for repeat processing counter or indexed address offset register decrementing (as the CY flag value is retained in a multiple-byte operation).
- The S flag is set (1) if bit 7 of dst is set (1) as a result of the decrement, and cleared (0) otherwise.
- The P/V flag is set (1) if a borrow is generated out of bit 6 into bit 7 and a borrow is not generated in bit 7 as a result of the decrement (when underflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- If dst is the B register, C register or saddr, and you do not want to change the S, Z, AC, or P/V flag, the DBNZ instruction can be used.

[Coding example]

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

INCW

Increment Word
Word Data Increment

[Instruction format] INCW dst

[Operation] dst ← dst + 1

[Operands]

Mnemonic	Operands (dst)
INCW	rp
	saddrp

[Flags]

S	Z	AC	P/V	CY

[Description]

- The contents of the destination operand (dst) are incremented by 1.
- The S, Z, AC, P/V, and CY flags are not changed since this is often used for incrementing the register used in addressing that uses a register.
- If the HL, DE, VP, or UP register (VP and UP: 78K/III Series only) is used as the base register in register indirect addressing, base addressing or based index addressing (78K/0 and 78K/III Series only) when rp is specified as the operand and a 78K/0, 78K/I, 78K/II, or 78K/III Series program is used, ensure that the contents of the T, W, V, or U register that indicates the high-order 8 bits of the address are coordinated with the DE, HL, VP, or UP register that indicates the low-order 16 bits. Also, if program amendment is possible, the program should be changed so that a 24-bit manipulation instruction (INCG instruction) is used.

[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
	saddrp

[Flags]

S	Z	AC	P/V	CY

[Description]

- The contents of the destination operand (dst) are decremented by 1.
- The S, Z, AC, P/V, and CY flags are not changed since this is often used for decrementing the register used in addressing that uses a register.
- If the HL, DE, VP, or UP register (VP and UP: 78K/III Series only) is used as the base register in register indirect addressing, base addressing or based index addressing (78K/0 and 78K/III Series only) when rp is specified as the operand and a 78K/0, 78K/I, 78K/II, or 78K/III Series program is used, ensure that the contents of the T, W, V, or U register that indicates the high-order 8 bits of the address are coordinated with the DE, HL, VP, or UP register that indicates the low-order 16 bits. Also, if program amendment is possible, the program should be changed so that a 24-bit manipulation instruction (INCG instruction) is used.

[Coding example]

DECW DE ; Decrements the DE register

INCG

Increment G Note
24-Bit Data Increment

[Instruction format] INCG dst

Note G is a character that indicates that 24-bit data is to be manipulated.

[Operation] dst ← dst + 1

[Operands]

Mnemonic	Operands (dst)
INCG	rg

[Flags]

S	Z	AC	P/V	CY

[Description]

- The contents of the destination operand (dst) are incremented by 1.
- The S, Z, AC, P/V, and CY flags are not changed since this is often used to decrement the register (pointer) used in addressing that uses a register.

[Coding example]

INCG UUP ; Increments the UUP register

DECG

Decrement G Note
24-Bit Data Decrement

[Instruction format] DECG dst

Note G is a character that indicates that 24-bit data is to be manipulated.

[Operation] dst ← dst – 1

[Operands]

Mnemonic	Operands (dst)
DECG	rg
	SP

[Flags]

S	Z	AC	P/V	CY

[Description]

- The contents of the destination operand (dst) are decremented by 1.
- The S, Z, AC, P/V, and CY flags are not changed since this is often used to decrement the register (pointer) used in addressing that uses a register.

[Coding example]

DECG VVP ; Decrements the VVP register

7.12 Adjustment Instructions

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]

S	Z	AC	P/V	CY
×	×	×	P	×

[Description]

- Decimal adjustment of the A register, CY flag and AC flag is performed from the A register, CY flag and AC flag contents. This instruction only performs a meaningful operation when the addition result is stored in the A register after BCD (binary-code decimal) data has been added (in other cases, a meaningless operation is performed). The adjustment method is shown in the table below.

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

- The Z flag is set (1) if the contents of the A register are 0 as a result of the adjustment, and cleared (0) otherwise.
- The S flag is set (1) if bit 7 of the A register is 1 as a result of the adjustment, and cleared (0) otherwise.
- The P/V flag is set (1) if the number of bits set (1) in the A register as a result of the adjustment is even, and cleared (0) otherwise.

[Coding example]

ADJBA ; Performs decimal adjustment of the contents of the A register

ADJBS

Decimal Adjust Register for Subtraction
Decimal Adjustment of Subtraction Result

[Instruction format] ADJBS

[Operation] Decimal Adjust Accumulator for Subtraction

[Operands]

None

[Flags]

S	Z	AC	P/V	CY
×	×	×	P	×

[Description]

- Decimal adjustment of the A register, CY flag and AC flag is performed from the A register, CY flag and AC flag contents. This instruction only performs a meaningful operation when the subtraction result is stored in the A register after BCD (binary-code decimal) data has been subtracted (in other cases, a meaningless operation is performed). The adjustment method is shown in the table below.

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

- The Z flag is set (1) if the contents of the A register are 0 as a result of the adjustment, and cleared (0) otherwise.
- The S flag is set (1) if bit 7 of the A register is 1 as a result of the adjustment, and cleared (0) otherwise.
- The P/V flag is set (1) if the number of bits set (1) in the A register as a result of the adjustment is even, and cleared (0) otherwise.

[Coding example]

ADJBS ; Performs decimal adjustment of the contents of the A register

CVTBW

Convert Byte to Word
Conversion from Byte Data to Word Data

[Instruction format] CVTBW

[Operation] When $A_7 = 0$, $X \leftarrow A$, $A \leftarrow 00H$
When $A_7 = 1$, $X \leftarrow A$, $A \leftarrow FFH$

[Operands]

None

[Flags]

S	Z	AC	P/V	CY

[Description]

- The signed 8-bit data in the A register is extended to signed 16-bit data in the AX register.
- The S, Z, AC, P/V, and CY flags are not changed by this instruction.

[Coding example]

CVTBW ; Extends the signed 8-bit data in the A register to signed 16-bit data and stores it in the AX register

7.13 Shift/Rotate Instructions

Shift/rotate instructions are as follows:

ROR

Rotate Right
Right Rotation of Byte Data

[Instruction format] ROR dst, cnt

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

[Operands]

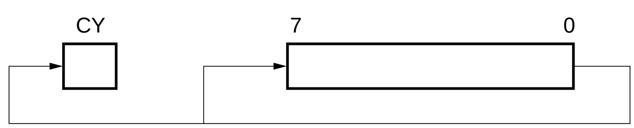
Mnemonic	Operands (dst, cnt)
ROR	r, n

[Flags]

S	Z	AC	P/V	CY
			P	×

[Description]

- The contents of the destination operand (dst) specified by the 1st operand are rotated to the right cnt times specified by the 2nd operand.
- The contents of the LSB (bit 0) are rotated into the MSB (bit 7) and are also transferred to the CY flag.
- If the 2nd operand (cnt) is 0, no processing is performed (and the S, Z, AC, P/V, and CY flags do not change).
- The P/V flag is set (1) if the number of bits set (1) in dst as a result of the right rotation is even, and cleared (0) otherwise.
- The S, Z, and AC flags do not change irrespective of the result of the rotate operation.



[Coding example]

ROR R5, 4 ; Rotates the contents of the R5 register 4 bits to the right

ROL

Rotate Left
Left Rotation of Byte Data

[Instruction format] ROL dst, cnt

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

[Operands]

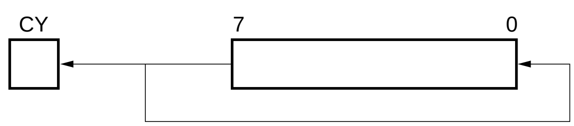
Mnemonic	Operands (dst, cnt)
ROL	r, n

[Flags]

S	Z	AC	P/V	CY
			P	×

[Description]

- The contents of the destination operand (dst) specified by the 1st operand are rotated to the left cnt times specified by the 2nd operand.
- The contents of the MSB (bit 7) are rotated into the LSB (bit 0) and are also transferred to the CY flag.
- If the 2nd operand (cnt) is 0, no processing is performed (and the S, Z, AC, P/V, and CY flags do not change).
- The P/V flag is set (1) if the number of bits set (1) in dst as a result of the left rotation is even, and cleared (0) otherwise.
- The S, Z, and AC flags do not change irrespective of the result of the rotate operation.



[Coding example]

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

RORC

Rotate Right with Carry
Right Rotation of Byte Data including Carry

[Instruction format] RORC dst, cnt

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

[Operands]

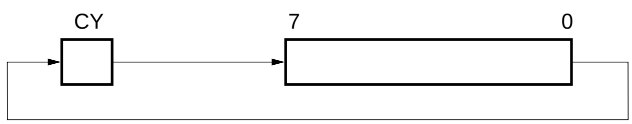
Mnemonic	Operands (dst, cnt)
RORC	r, n

[Flags]

S	Z	AC	P/V	CY
			P	×

[Description]

- The contents of the destination operand (dst) specified by the 1st operand, and the CY flag, are rotated to the right cnt times specified by the 2nd operand.
- If the 2nd operand (cnt) is 0, no processing is performed (and the S, Z, AC, P/V, and CY flags do not change).
- The P/V flag is set (1) if the number of bits set (1) in dst as a result of the right rotation is even, and cleared (0) otherwise.
- The S, Z, and AC flags do not change irrespective of the result of the rotate operation.



[Coding example]

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

ROLC

Rotate Left with Carry
Left Rotation of Byte Data including Carry

[Instruction format] ROLC dst, cnt

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

[Operands]

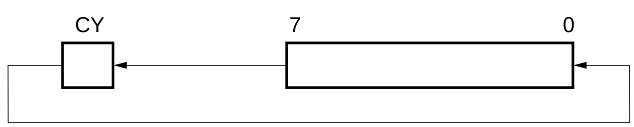
Mnemonic	Operands (dst, cnt)
ROLC	r, n

[Flags]

S	Z	AC	P/V	CY
			P	×

[Description]

- The contents of the destination operand (dst) specified by the 1st operand, and the CY flag, are rotated to the left cnt times specified by the 2nd operand.
- If the 2nd operand (cnt) is 0, no processing is performed (and the S, Z, AC, P/V, and CY flags do not change).
- If you wish to perform a left rotation of 1 bit only, the execution time can be reduced by using ADDC r, r.
- The P/V flag is set (1) if the number of bits set (1) in dst as a result of the left rotation is even, and cleared (0) otherwise.
- The S, Z, and AC flags do not change irrespective of the result of the rotate operation.



[Coding example]

ROLC R7, 3 ; Rotates the contents of the R7 register and the CY flag 3 bits to the left

SHR

Shift Right (Logical)
Logical Right Shift of Byte Data

[Instruction format] SHR dst, cnt

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

[Operands]

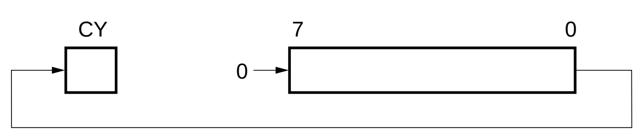
Mnemonic	Operands (dst, cnt)
SHR	r, n

[Flags]

S	Z	AC	P/V	CY
×	×	0	P	×

[Description]

- The contents of the destination operand (dst) specified by the 1st operand are shifted to the right cnt times specified by the 2nd operand.
  0 is shifted into the MSB (bit 7) each time a 1-bit shift is performed.
- The S flag is cleared (0) if cnt is 1 or more.
- The Z flag is set (1) if the result of the shift operation is 0, and cleared (0) otherwise.
- The AC flag is always 0 irrespective of the result of the shift operation.
- The P/V flag is set (1) if the number of bits set (1) in dst as a result of the shift operation is even, and cleared (0) otherwise.
- The last data shifted out of the LSB (bit 0) as a result of the shift operation is set in the CY flag.
- If cnt is 0, no processing is performed (and the S, Z, AC, P/V, and CY flags do not change).
- This instruction gives the same result as division of the destination operand (dst) as unsigned data by 2^{cnt}.



[Coding example]

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

SHL

Shift Left (Logical)
Logical Left Shift of Byte Data

[Instruction format] SHL dst, cnt

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

[Operands]

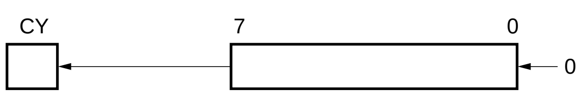
Mnemonic	Operands (dst, cnt)
SHL	r, n

[Flags]

S	Z	AC	P/V	CY
×	×	0	P	×

[Description]

- The contents of the destination operand (dst) specified by the 1st operand are shifted to the left cnt times specified by the 2nd operand.
- 0 is shifted into the LSB (bit 0) each time a 1-bit shift is performed.
- The S flag is set (1) if bit 7 of dst is 1 as a result of the shift operation, and cleared (0) if 0.
- The Z flag is set (1) if the result of the shift operation is 0, and cleared (0) otherwise.
- The AC flag is always 0 irrespective of the result of the shift operation.
- The P/V flag is set (1) if the number of bits set (1) in dst as a result of the shift operation is even, and cleared (0) otherwise.
- The last data shifted out of the LSB (bit 0) as a result of the shift operation is set in the CY flag.
- If cnt is 0, no processing is performed (and the S, Z, AC, P/V, and CY flags do not change).
- If you wish to perform a left shift of 1 bit only, the execution time can be reduced by using ADD r, r.
- This instruction gives the same result as multiplication of the destination operand (dst) by 2^{cnt} (if the multiplication result is 8 bits or less).



[Coding example]

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

SHRW

Shift Right (Logical) Word
Logical Right Shift of Word Data

[Instruction format] SHRW dst, cnt

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

[Operands]

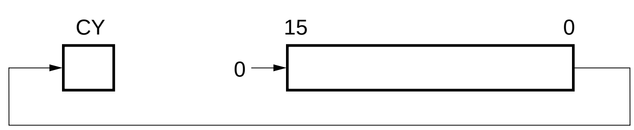
Mnemonic	Operands (dst, cnt)
SHRW	rp, n

[Flags]

S	Z	AC	P/V	CY
×	×	0	P	×

[Description]

- The contents of the destination operand (dst) specified by the 1st operand are shifted to the right cnt times specified by the 2nd operand.
  0 is shifted into the MSB (bit 15) each time a 1-bit shift is performed.
- The S flag is cleared (0) if cnt is 1 or more.
- The Z flag is set (1) if the result of the shift operation is 0, and cleared (0) otherwise.
- The AC flag is always 0 irrespective of the result of the shift operation.
- The P/V flag is set (1) if the number of bits set (1) in the low-order 8 bits of dst as a result of the shift operation is even, and cleared (0) otherwise.
- The last data shifted out of the LSB (bit 0) as a result of the shift operation is set in the CY flag.
- If cnt is 0, no processing is performed (and the S, Z, AC, P/V, and CY flags do not change).
- This instruction gives the same result as division of the destination operand (dst) as unsigned data by 2^{cnt}.



[Coding example]

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

SHLW

Shift Left (Logical) Word
Logical Left Shift of Word Data

[Instruction format] SHLW dst, cnt

[Operation] (CY $\leftarrow$ dst_{15}, dst_0 $\leftarrow$ 0, dst_{m+1} $\leftarrow$ dst_m) $\times$ cnt times cnt = 0 to 7

[Operands]

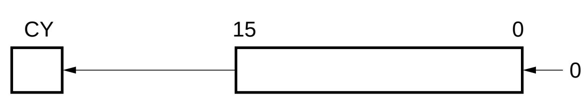
Mnemonic	Operands (dst, cnt)
SHLW	rp, n

[Flags]

S	Z	AC	P/V	CY
×	×	0	P	×

[Description]

- The contents of the destination operand (dst) specified by the 1st operand are shifted to the right cnt times specified by the 2nd operand.
- 0 is shifted into the LSB (bit 0) each time a 1-bit shift is performed.
- The S flag is set (1) if bit 15 of dst is 1 as a result of the shift operation, and cleared (0) if 0.
- The Z flag is set (1) if the result of the shift operation is 0, and cleared (0) otherwise.
- The AC flag is always 0 irrespective of the result of the shift operation.
- The P/V flag is set (1) if the number of bits set (1) in the low-order 8 bits of dst as a result of the shift operation is even, and cleared (0) otherwise.
- The last data shifted out of the LSB (bit 0) as a result of the shift operation is set in the CY flag.
- If cnt is 0, no processing is performed (and the S, Z, AC, P/V, and CY flags do not change).



[Coding example]

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

ROR4

Rotate Right Digit
Right Digit Rotation

[Instruction format] ROR4 dst

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

[Operands]

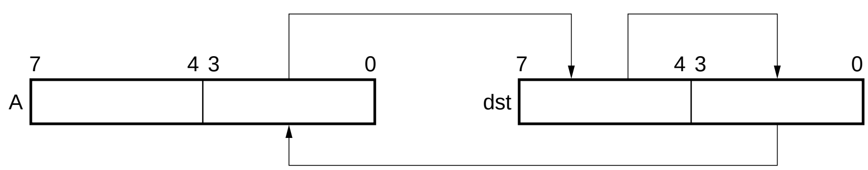
Mnemonic	Operands (dst)
ROR4	mem3

[Flags]

S	Z	AC	P/V	CY

[Description]

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



[Coding example]

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

	A (7 4 3 0)	(WHL) (7 4 3 0)
Before Execution	1 0 1 0 | 0 0 1 1	1 1 0 0 | 0 1 0 1
After Execution	1 0 1 0 | 0 1 0 1	0 0 1 1 | 1 1 0 0

ROL4

Rotate Left Digit
Left Digit Rotation

[Instruction format] ROL4 dst

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

[Operands]

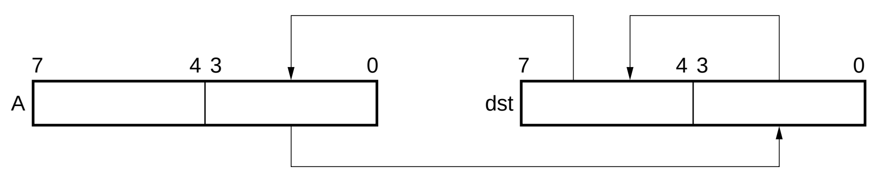
Mnemonic	Operands (dst)
ROL4	mem3

[Flags]

S	Z	AC	P/V	CY

[Description]

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



[Coding example]

ROL4 [TDE] ; Performs digit rotation to the left of the A register and memory contents specified by the TDE register.

	A	(TDE)
	7 4 3 0	7 4 3 0
Before Execution	0 0 0 1 0 0 1 0	0 1 0 0 1 0 0 0
After Execution	0 0 0 1 0 1 0 0	1 0 0 0 0 0 1 0

7.14 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, X.bit
	CY, A.bit
	CY, PSWL.bit
	CY, PSWH.bit
	CY, mem2.bit
	CY, !addr16.bit
	CY, !!addr24.bit
	saddr.bit, CY
	sfr.bit, CY
	X.bit, CY
	A.bit, CY
	PSWL.bit, CY
	PSWH.bit, CY
	mem2.bit, CY
	!addr16.bit, CY
	!!addr24.bit, CY

[Flags]

dst is PSWL.bit

S	Z	AC	P/V	CY
×	×	×	×	×

dst is CY

S	Z	AC	P/V	CY
				×

Other cases

S	Z	AC	P/V	CY

[Description]

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

[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, /saddr.bit
	CY, sfr.bit
	CY, /sfr.bit
	CY, X.bit
	CY, /X.bit
	CY, A.bit
	CY, /A.bit
	CY, PSWL.bit
	CY, /PSWL.bit
	CY, PSWH.bit
	CY, /PSWH.bit
	CY, mem2.bit
	CY, /mem2.bit
	CY, !addr16.bit
	CY, /!addr16.bit
	CY, !!addr24.bit
	CY, /!!addr24.bit

[Flags]

S	Z	AC	P/V	CY
				×

[Description]

- The logical product of the destination operand (dst) specified by the 1st operand and the source operand (src) bit data specified by the 2nd operand is found, and the result is stored in the destination operand (dst).
- If the 2nd operand is immediately preceded by "/", the logical product operation is performed on the logical NOT of the source operand (src).
- The CY flag stores the operation result (as it is the destination operand (dst)).

[Coding examples]

AND1 CY, SADR.3 ; Finds the logical product of bit 3 of address SADR which can be accessed by short direct addressing and the CY flag, and stores the result in the CY flag

AND1 CY, /PSW.6 ; Finds the logical product of the logical NOT of bit 6 of the PSW (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 $\leftarrow$ dst $\vee$ src dst $\leftarrow$ dst $\vee$ $\overline{\text{src}}$

[Operands]

Mnemonic	Operands (dst, src)
OR1	CY, saddr.bit
	CY, /saddr.bit
	CY, sfr.bit
	CY, /sfr.bit
	CY, X.bit
	CY, /X.bit
	CY, A.bit
	CY, /A.bit
	CY, PSWL.bit
	CY, /PSWL.bit
	CY, PSWH.bit
	CY, /PSWH.bit
	CY, mem2.bit
	CY, /mem2.bit
	CY, !addr16.bit
	CY, /!addr16.bit
	CY, !!addr24.bit
	CY, /!!addr24.bit

[Flags]

S	Z	AC	P/V	CY
				×

[Description]

- The logical sum of the destination operand (dst) specified by the 1st operand and the source operand (src) bit data specified by the 2nd operand is found, and the result is stored in the destination operand (dst).
- If the 2nd operand is immediately preceded by "/", the logical sum operation is performed on the logical NOT of the source operand (src).
- The CY flag stores the operation result (as it is the destination operand (dst)).

[Coding examples]

OR1 CY, P2.5 ; Finds 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 ; Finds the logical sum of the logical NOT 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, X.bit
	CY, A.bit
	CY, PSWL.bit
	CY, PSWH.bit
	CY, mem2.bit
	CY, !addr16.bit
	CY, !!addr24.bit

[Flags]

S	Z	AC	P/V	CY
				×

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

[Coding example]
XOR1 CY, A.7 ; Finds the exclusive logical sum of bit 7 of the A register and the CY flag, and stores the result in the CY flag

NOT1

Not Single Bit (Carry Flag)
1-Bit Data Logical NOT

[Instruction format] NOT1 dst

[Operation] dst $\leftarrow \overline{\text{dst}}$

[Operands]

Mnemonic	Operands (dst)
NOT1	saddr.bit
	sfr.bit
	X.bit
	A.bit
	PSWL.bit
	PSWH.bit
	mem2.bit
	!addr16.bit
	!!addr24.bit
	CY

[Flags]

dst is PSWL.bit

S	Z	AC	P/V	CY
×	×	×	×	×

dst is CY

S	Z	AC	P/V	CY
				×

Other cases

S	Z	AC	P/V	CY

[Description]

- The logical NOT of the bit specified by the destination operand (dst) is found, and the result is stored in the destination operand (dst).
- If the destination operand (dst) is CY or PSW.bit, only the relevant flag changes.

[Coding examples]

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

SET1

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

[Instruction format] SET1 dst

[Operation] dst ← 1

[Operands]

Mnemonic	Operands (dst)
SET1	saddr.bit
	sfr.bit
	X.bit
	A.bit
	PSWL.bit
	PSWH.bit
	mem2.bit
	!addr16.bit
	!!addr24.bit
	CY

[Flags]

dst is PSWL.bit

S	Z	AC	P/V	CY
×	×	×	×	×

dst is CY

S	Z	AC	P/V	CY
				1

Other cases

S	Z	AC	P/V	CY

[Description]

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

[Coding example]

SET1 BITSYM ; Sets (1) the contents of a bit located in an area that can be accessed by short direct addressing

CLR1

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

[Instruction format] CLR1 dst

[Operation] dst $\leftarrow$ 0

[Operands]

Mnemonic	Operands (dst)
CLR1	saddr.bit
	sfr.bit
	X.bit
	A.bit
	PSWL.bit
	PSWH.bit
	mem2.bit
	!addr16.bit
	!!addr24.bit
	CY

[Flags]

dst is PSWL.bit

S	Z	AC	P/V	CY
×	×	×	×	×

dst is CY

S	Z	AC	P/V	CY
				0

Other cases

S	Z	AC	P/V	CY

[Description]

- The destination operand (dst) is cleared (0).
- 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

7.15 Stack Manipulation Instructions

Stack manipulation instructions are as follows:

PUSH

Push
Push

[Instruction format] PUSH src

[Operation] Note

(1) When src is PSW, sfrp
(SP – 2) ← src,
SP ← SP – 2

(2) When src is sfr
(SP – 1) ← src,
SP ← SP – 1

(3) When src is rg
(SP – 3) ← src,
SP ← SP – 3

(4) When src is post
{(SP – 2) ← post, SP ← SP – 2} × n times

Note For details, refer to **CHAPTER 3, Figure 3-4 Data Saved to Stack Area**, and **Figure 3-5 Data Restored from Stack Area.**

[Operands]

Mnemonic	Operands (src)
PUSH	PSW
	sfrp
	sfr
	post
	rg

[Flags]

S	Z	AC	P/V	CY

[Description]

- The data in the registers specified by the source operand (src) is saved to the stack.
- If post is specified as the source operand, any combination of the following registers can be saved to the stack by the instruction.

  AX (RP0), BC (RP1), RP2, RP3, UP, VP, DE, HL

  The save order at this time is from the rightmost of the above registers.
- The VP, UP, DE, and HL registers should only be used when a 78K/0, 78K/I, 78K/II, or 78K/III Series program is used. In other cases, saving to the stack should be specified individually as the UUP, VVP, TDE, and WHL registers.

  Moreover, saving to the stack should also be specified individually as the UUP, VVP, TDE, and WHL registers when a 78K/0, 78K/I, 78K/II, or 78K/III Series program is used.
- After the source operand (src) save, the stack pointer (SP) is decremented by the number of bytes of data saved.

[Coding example]

PUSH AX, BC, RP2, RP3 ; Saves the contents of the AX, BC, RP2, and RP3 registers to the stack

PUSHU

Push to User Stack
Register Push to User Stack

[Instruction format] PUSHU src

[Operation] [Note] {(UUP – 1) ← post, UUP ← UUP – 2} × n times
(n = number of register pairs written as post)

Note For details, refer to **CHAPTER 3, Figure 3-4 Data Saved to Stack Area**, and **Figure 3-5 Data Restored from Stack Area.**

[Operands]

Mnemonic	Operands (src)
PUSHU	post

[Flags]

S	Z	AC	P/V	CY

[Description]

- The contents of the 16-bit register pair specified by the source operand (src) are saved to the memory addressed by the user stack pointer (UUP), and then the UUP is decremented.
- Any combination of the following registers can be written in post as the source operand (src).
  AX (RP0), BC (RP1), RP2, RP3, VP, PSW, DE, HL
  The save order at this time is from the rightmost of the above registers.

[Coding example]

PUSHU BC, PSW ; Saves the contents of the BC register and PSW to the stack

POP

Pop
Pop

[Instruction format] POP dst

[Operation] Note

(1) When dst is PSW, sfrp
dst ← (SP)
SP ← SP + 2

(2) When dst is sfr
dst ← (SP),
SP ← SP + 1

(3) When dst is rg
dst ← (SP),
SP ← SP + 3

(4) When dst is post
{post ← (SP), SP ← SP+2} × n times

Note For details, refer to **CHAPTER 3, Figure 3-4 Data Saved to Stack Area,** and **Figure 3-5 Data Restored from Stack Area.**

[Operands]

Mnemonic	Operands (dst)
POP	PSW
	sfrp
	sfr
	post
	rg

[Flags]

dst is PSW

S	Z	AC	P/V	CY
R	R	R	R	R

In other cases

S	Z	AC	P/V	CY

[Description]

- Data is restored from the stack to the registers specified by the destination operand (dst).
- If the destination operand (dst) is the PSW, each flag is replaced with stack data.
- If post is specified as the destination operand (dst), data can be restored to any combination of the following registers by one instruction.
  AX (RP0), BC (RP1), RP2, RP3, VP (RP4), UP (RP5), DE (RP6), HL (RP7)
  The restoration order at this time is from the leftmost of the above registers.
- The UP, VP, DE, and HL registers should only be used when a 78K/0, 78K/I, 78K/II, or 78K/III Series program is used. In other cases, restoration from the stack should be specified individually as the UUP, VVP, TDE, and WHL registers.
  Moreover, saving to the stack should also be specified individually as the UUP, VVP, TDE, and WHL registers when a 78K/0, 78K/I, 78K/II, or 78K/III Series program is used.
- After data has been restored from the stack, the stack pointer (SP) is incremented by the number of bytes of data restored.

[Coding example]

POP IMK0L ; Restores stack data to the IMK0L register

POPU

Pop from User Stack
Register Pop from User Stack

[Instruction format] POPU dst

[Operation] Note {post ← (UUP), UUP ← UUP + 2} × n times
(n = number of register pairs written as post)

Note For details, refer to **CHAPTER 3, Figure 3-4 Data Saved to Stack Area**, and **Figure 3-5 Data Restored from Stack Area.**

[Operands]

Mnemonic	Operands (dst)
POPU	post

[Flags]

S	Z	AC	P/V	CY

[Description]

- The contents of the memory (stack) addressed by the user stack pointer (UUP) are restored to the registers specified by the destination operand (dst), and then the UUP is incremented.
- Any combination of the following registers can be written in post as the destination operand (dst).
  AX (RP0), BC (RP1), RP2, RP3, VP (RP4), PSW, DE (RP6), HL (RP7)
  The restoration order at this time is from the leftmost of the above registers.

[Coding example]

POPU AX, BC ; Restores stack data to the AX and BC registers

MOVG

Move G Note
24-Bit Data Transfer

[Instruction format] MOVG dst, src

Note G is a character that indicates that 24-bit data is to be manipulated.

[Operation]

When dst is SP	When dst is WHL
SP ← src	WHL ← SP

[Operands]

Mnemonic	Operands (dst, src)
MOVG	SP, #imm24
	SP, WHL
	WHL, SP

[Flags]

S	Z	AC	P/V	CY

[Description]

- The contents of the source operand (src) specified by the 2nd operand are transferred to the destination operand (dst) specified by the 1st operand.
- After reset release, SP initialization must always be performed with an MOVG SP, #imm24 instruction after executing the LOCATION instruction.

[Coding example]

MOVG SP, #0FFD20H ; Sets 0FFD20H in the SP

ADDWG

Add Word to G Note
24-Bit Word Data Addition

[Instruction format] ADDWG dst, src

Note G is a character that indicates that 24-bit data is to be manipulated.

[Operation] SP ← SP + word

[Operands]

Mnemonic	Operands (dst, src)
ADDWG	SP, #word

[Flags]

S	Z	AC	P/V	CY

[Description]

- Unsigned 16-bit immediate data is added to the contents of the stack pointer (SP), and the result is stored in the stack pointer (SP).
- This instruction is used to release a memory area reserved as a temporary variable storage location in a high-level language, etc.

[Coding example]

ADDWG SP, #30H ; Adds 30H to the SP and stores the result in the SP

SUBWG

Subtract Word from G Note
24-Bit Word Data Subtraction

[Instruction format] SUBWG dst, src

Note G is a character that indicates that 24-bit data is to be manipulated.

[Operation] SUBWG SP ← SP – 1

[Operands]

Mnemonic	Operands (dst, src)
SUBWG	SP, #word

[Flags]

S	Z	AC	P/V	CY

[Description]

- Unsigned 16-bit immediate data is subtracted from the contents of the stack pointer (SP), and the result is stored in the SP.
- This instruction is used to reserve a temporary variable area in a high-level language, etc.

[Coding example]

SUBWG SP, #50H ;Subtracts 50H from the SP and stores the result in the SP. This reserves a 50H-byte temporary variable area.

INCG SP

Increment G Note
Stack Pointer 24-Bit Data Increment

[Instruction format] INCG SP

[Operation] SP ← SP + 1

[Operands]

None

[Flags]

S	Z	AC	P/V	CY

[Description]

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

[Coding example]

INCG SP

Note G is a character that indicates that 24-bit data is to be manipulated.

DECG SP

Decrement G Note
Stack Pointer 24-Bit Data Decrement

[Instruction format] DECG SP

[Operation] SP ← SP – 1

[Operands]

None

[Flags]

S	Z	AC	P/V	CY

[Description]

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

[Coding example]

DECG SP

Note G is a character that indicates that 24-bit data is to be manipulated.

7.16 Call/Return Instructions

Call/return instructions are as follows:

CALL

Call
Subroutine Call

[Instruction format] CALL target

[Operation] Note
(SP – 3) ← (PC + n),
SP ← SP – 3
PC ← target
n: Number of instruction bytes

Note For details, refer to **CHAPTER 3, Figure 3-4 Data Saved to Stack Area**, and **Figure 3-5 Data Restored from Stack Area.**

[Operands]

Mnemonic	Operands (target)
CALL	!addr16
	!!addr20
	rp
	rg
	[rp]
	[rg]
	$!addr20

[Flags]

S	Z	AC	P/V	CY

[Description]

- This is a subroutine call using a 16-bit or 20-bit absolute address, 16-bit relative address, register direct address, or register indirect address.
- The start address of the next instruction (PC + n) is saved to the stack, and the program branches to the address specified by the target operand (target).
- If !addr16, rp or [rp] is specified as the operand, the branch destination address is limited to the base area (0 to FFFFH) (in the case of [rp], the branch destination table is also limited to the base area). This should only be used when it is absolutely essential to reduce the execution time or object size, and when 78K/0, 78K/I, 78K/II, or 78K/III Series software is used and program amendment is difficult.
  Amendments may be necessary in order to make further use of a program that uses these instructions.
- With the NEC assembler (RA78K4), if CALL addr is written, the object code that can be assumed to be most appropriate can be selected and generated automatically from CALL !addr16, CALL !!addr20, and CALL $!addr20.

[Coding example]

CALL !!13059H ; Subroutine call to 013059H

CALLF

Call Flag
Subroutine Call (11-Bit Direct Specification)

[Instruction format] CALLF target

[Operation] Note

(SP – 3) ← (PC + 2),
SP ← SP – 3
PC ← target

Note For details, refer to **CHAPTER 3, Figure 3-4 Data Saved to Stack Area**, and **Figure 3-5 Data Restored from Stack Area.**

[Operands]

Mnemonic	Operands (target)
CALLF	!addr11

[Flags]

S	Z	AC	P/V	CY

[Description]

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

[Coding example]

CALLF !0C2AH ; Subroutine call to 00C2AH

CALLT

Call Table
Subroutine Call (Call Table Reference)

[Instruction format] CALLT [addr5]

[Operation] Note

$(SP - 3) \leftarrow (PC + 1)$,
$SP \leftarrow SP - 3$,
$PC_{HW} \leftarrow 0$
$PC_H \leftarrow (addr5 + 1)$
$PC_L \leftarrow (addr5)$

Note For details, refer to **CHAPTER 3, Figure 3-4 Data Saved to Stack Area**, and **Figure 3-5 Data Restored from Stack Area.**

[Operands]

Mnemonic	Operands ([addr5])
CALLT	[addr5]

[Flags]

S	Z	AC	P/V	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 the program branches to the address indicated by call table (high-order bits of the address fixed at 0000000001B, next 5 bit specified by addr5, LSB fixed at 0) word data.
- Subroutine start addresses that can be branched to by this instruction are limited to the base area (0 to FFFFH).

[Coding Example]

CALLT [60H] ; Uses the word data in addresses 00060H and 00061H as the address, and makes a subroutine call to that address

BRK

Break
Software Vectored Interrupt

[Instruction format] BRK

[Operation]
$(SP - 2) \leftarrow PSW$,
$(SP - 4) \leftarrow PC + 1$,
$IE \leftarrow 0$
$SP \leftarrow SP - 4$,
$PC_{HW} \leftarrow 0$,
$PC_{LW} \leftarrow (003EH)$

[Operands]

None

[Flags]

S	Z	AC	P/V	CY

[Description]

- This is a 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 specified by the vector address (0003EH) word data (the branch destination address is limited to the base area (0 to FFFFH)).
- The RETB instruction is used to return from a software vectored interrupt generated by this instruction.

[Coding Example]

BRK

BRKCS

Break Context Switch
Software Context Switch

[Instruction format] BRKCS RBn

[Operation]

$PC_{LW} \leftrightarrow RP2$,
$RP3 \leftarrow PSW, PC_{15-19}$
$PC_{15-19} \leftarrow 0$
$RBS2 - 0 \leftarrow n$,
$RSS \leftarrow 0$,
$IE \leftarrow 0$ (n = 0 to 7)

[Operands]

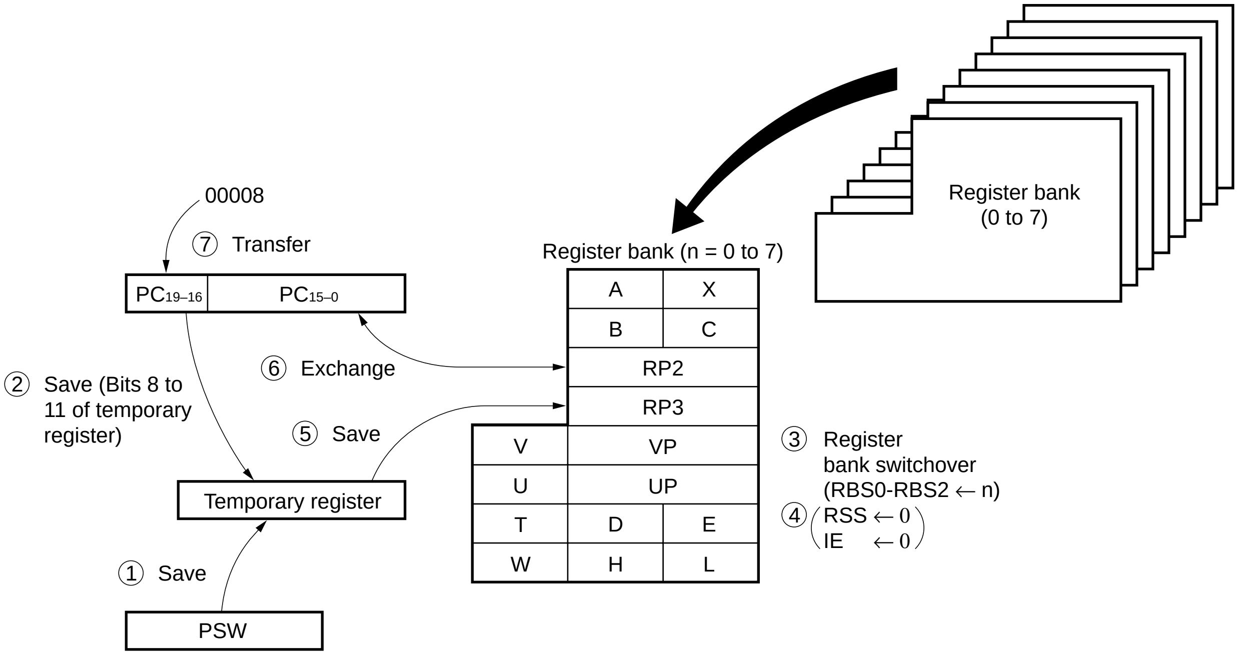
Mnemonic	Operands
BRKCS	RBn

[Flags]

S	Z	AC	P/V	CY

[Description]

- This is a software interrupt instruction.
- Register bank n written in the operand is selected, the contents of RP2 in that register bank and the contents of the low-order 16 bits of the program counter (PC) are exchanged, the contents of the program status word (PSW) and the high-order 4 bits of the PC are saved to the stack, the high-order 4 bits of the PC are set to 0, and a branch is made to that address. The RSS flag and IE flag are then cleared to 0.
- Only addresses in the base area (0 to FFFFH) can be branched to by this instruction.
- The RETCSB instruction is used to return from a software interrupt generated by this instruction.
- The contents of RP2 and RP3 must not be changed in the software interrupt program initiated by this instruction. If RP2 and RP3 are used, they must be saved to the stack, etc., and returned to their original value before the RETCSB instruction is executed.



[Coding Example]

BRKCS RB3 ; Selects register bank 3, and executes instructions from the address indicated by RP2 in register bank 3

RET	**Return** **Return from Subroutine**

[Instruction format] RET

[Operation] Note PC ← (SP),
SP ← SP + 3

Note For details, refer to **CHAPTER 3, Figure 3-4 Data Saved to Stack Area**, and **Figure 3-5 Data Restored from Stack Area.**

[Operands]

None

[Flags]

S	Z	AC	P/V	CY

[Description]

- This is the instruction for returning from a subroutine call made by a CALL, CALLF, or CALLT instruction.
- The 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] Note

PC ← (SP),
PSW ← (SP + 2),
PC ← SP + 4
The bit set (1) in ISPR with the highest priority is cleared (0).

Note For details, refer to **CHAPTER 3, Figure 3-4 Data Saved to Stack Area,** and **Figure 3-5 Data Restored from Stack Area.**

[Operands]

None

[Flags]

S	Z	AC	P/V	CY
R	R	R	R	R

[Description]

- This is the instruction for returning from a vectored interrupt.
- The data saved in the stack is restored in PC and PSW, and of the flags set (1) in the ISPR register, the flag with the highest priority is cleared (0), and operation then returns from the interrupt processing routine.
- This instruction cannot be used to return from a software interrupt generated by a BRK instruction, BRKCS instruction or operand error, or from an interrupt that uses context switching.

RETB

Return from Break
Return from Software Vectored Interrupt

[Instruction format] RETB

[Operation] Note

PC $\leftarrow$ (SP),
PSW $\leftarrow$ (SP + 2),
PC $\leftarrow$ SP + 4

Note For details, refer to **CHAPTER 3, Figure 3-4 Data Saved to Stack Area**, and **Figure 3-5 Data Restored from Stack Area.**

[Operands]

None

[Flags]

S	Z	AC	P/V	CY
R	R	R	R	R

[Description]

- This is the instruction for returning from a software interrupt generated by an BRK instructions operand error.
- The PC and PSW saved to the stack are restored, and a return is made from the interrupt service routine.
- This instruction cannot be used to return from a hardware interrupt caused by a BRKCS instruction or hardware interrupt

RETCS

Return from Context Switch
Return from Hardware Context Switch

[Instruction format] RETCS targer

[Operation]

$PC_{LW} \leftarrow$ RP2,
$PC_{15-19} \leftarrow RP2_{8-11}$
RP2 $\leftarrow$ addr16,
PSW $\leftarrow$ RP3
The bit set (1) in ISPR with the highest priority is cleared (0).

[Operands]

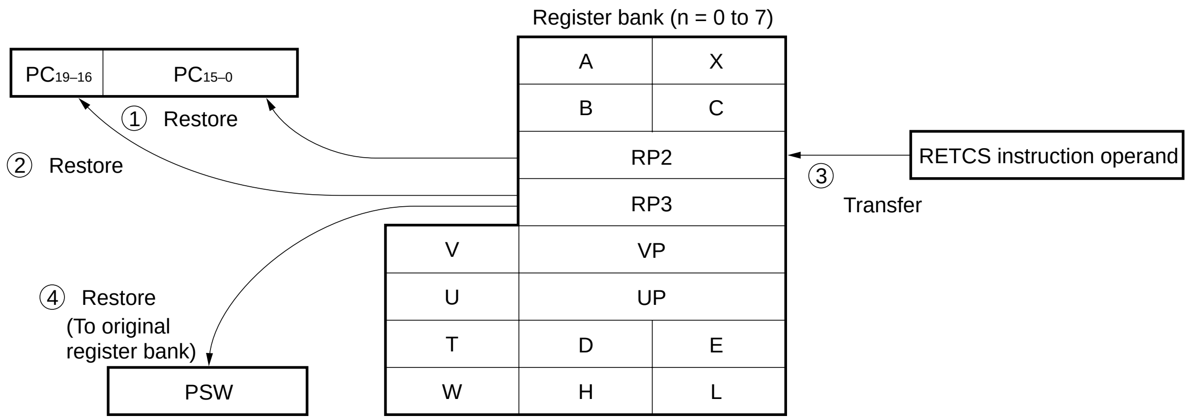
Mnemonic	Operands
RETCS	!addr16

[Flags]

S	Z	AC	P/V	CY
R	R	R	R	R

[Description]

- The contents of register banks RP2 and RP3 that are specified when this instruction is executed are transferred to the program counter (PC) and program status word (PSW), and of the flags set (1) in the ISPR register, the flag with the highest priority is cleared (0), and operation then returns from the interrupt processing routine. The data specified by the operand is then transferred to RP2.
- The RETCS instruction is valid for context switching associated with generation of an interrupt request, and is used to return from branch processing due to context switching. addr16 written in the operand is the branch address used if the same register bank is specified again by the context switching function (only an address in the base area can be specified as the branch destination address).
- This instruction cannot be used to return from a software interrupt generated by a BRK instruction, BRKCS instruction or operand error, or from a vectored interrupt.
- Before this instruction is executed, the contents of RP2 and RP3 must be the same as immediately after interrupt acknowledgment.



[Coding example]

RETCS !03456H ; Returns from a context switching interrupt, and sets the address for acknowledgment of the next interrupt to 03456H

RETCSB

Return from Context Switch Break
Return from Software Context Switch

[Instruction format] RETCSB targer

[Operation] $PC_{LW} \leftarrow RP2$,
$PC_{15-19} \leftarrow RP3_{8-11}$
$RP2 \leftarrow addr16$,
$PSW \leftarrow RP3$

[Operands]

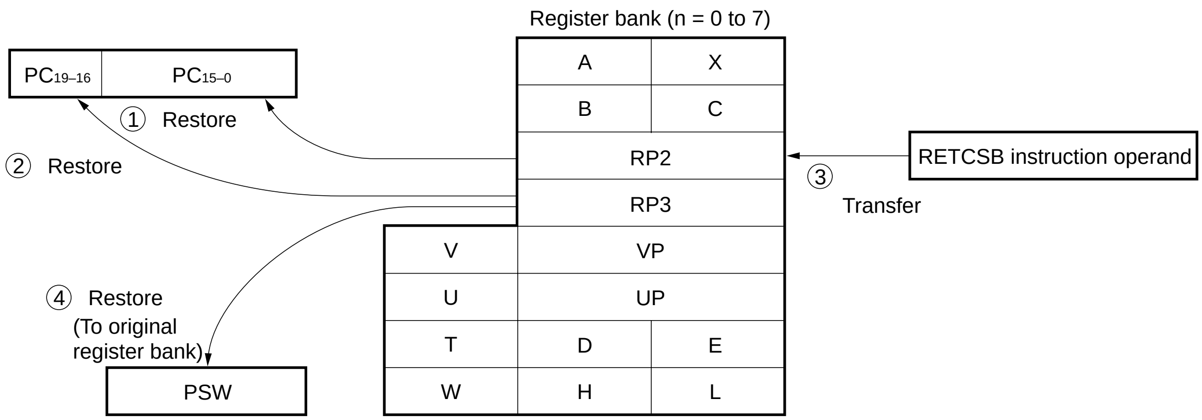
Mnemonic	Operands
RETCSB	!addr16

[Flags]

S	Z	AC	P/V	CY
R	R	R	R	R

[Description]

- The contents of RP2 and RP3 in the register bank specified when this instruction is executed are transferred to the program counter (PC) and program status word (PSW), and a return is made from the interrupt service routine.
  The data specified by the operand is then transferred to RP2.
- The RETCSB instruction is valid for context switching by means of the BRKCS instruction, and is used to return from branch processing due to context switching. addr16 written in the operand is the branch address used if the same register bank is specified again by the context switching function (only an address in the base area can be specified as the branch destination address).
- This instruction cannot be used to return from a software interrupt generated by a BRK instruction or operand error, or from a hardware interrupt.
- Before this instruction is executed, the contents of RP2 and RP3 must be the same as immediately after interrupt acknowledgment.



[Coding example]

RETCSB !0ABCDH ; Returns from an interrupt generated by a BRKCS instruction

7.17 Unconditional Branch Instruction

There is one unconditional branch instruction, as follows.

BR

Branch
Unconditional Branch

[Instruction format] BR target

[Operation] PC ← target

[Operands]

Mnemonic	Operands (target)
BR	!addr16
	!!addr20
	rp
	rg
	[rp]
	[rg]
	$addr20
	$!addr20

[Flags]

S	Z	AC	P/V	CY

[Description]
- Instruction that performs an unconditional branch.
- The target address operand (target) data is transferred to the PC, and a branch is made.
- If !addr16, rp or [rp] is specified as the operand, the branch destination address is limited to the base area (0 to FFFFH) (in the case of [rp], the branch destination table is also limited to the base area). This should only be used when it is absolutely essential to reduce the execution time or object size, and when a 78K/0, 78K/I, 78K/II, or 78K/III Series software is used and program amendment is difficult. Amendments may be necessary in order to make further use of a program that uses these instructions.
- With the NEC assembler RA78K4, if BR addr is written, the object code that can be assumed to be most appropriate can be selected and generated automatically from BR $addr20, BR $!addr20, BR !addr16, and BR!!addr20.

[Coding example]
BR TDE ; Branches using the contents of the TDE register as the address

7.18 Conditional Branch Instructions

Conditional branch instructions are as follows:

BNZ
BNE

Branch if Not Zero/Not Equal
Conditional Branch upon Zero Flag (Z = 0)

[Instruction format] BNZ $addr20
BNE $addr20

[Operation] PC ← PC + 2 + jdisp8 if Z = 0

[Operands]

Mnemonic	Operands ($addr20)
BNZ	$addr20
BNE	

[Flags]

S	Z	AC	P/V	CY

[Description]

- If Z = 0, the program branches to the address specified by the operand.
  If Z = 1, no processing is performed and the next instruction is executed.
- The operation of the BNZ instruction and the BNE instruction is the same. They are used as follows:
  - BNZ instruction: To check whether the result of an addition, subtraction or increment/decrement instruction, or an 8-bit logical operation or shift/rotate instruction is 0.
  - BNE instruction: Checks for a match after a compare instruction.
- If two –80H values are added together in the case of 8 bits when two's complement type data addition is performed, or two –8000H values in the case of 16 bits, Z is set to 1. When determining whether or not the result of a two's complement type data addition is 0, check for overflow beforehand using the overflow flag (V).

[Coding example]

```
CMP A, #55H
BNE $0A39H  ;  Branches to 00A39H if the A register is not 0055H
               The start address of the BNE instruction must be in the range 009B8H to 00AB7H
```

BZ
BE

Branch if Zero/Equal than
Conditional Branch upon Zero Flag (Z = 1)

[Instruction format] BZ $addr20
BE $addr20

[Operation] PC ← PC + 2 + jdisp8 if Z = 1

[Operands]

Mnemonic	Operands ($addr20)
BZ	$addr20
BE	

[Flags]

S	Z	AC	P/V	CY

[Description]

- If Z = 1, the program branches to the address specified by the operand.
  If 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. They are used as follows:
  – BZ instruction: To check whether the result of an addition, subtraction or increment/decrement instruction, or an 8-bit logical operation or shift/rotate instruction is 0.
  – BE instruction: Checks for a match after a compare instruction.
- If two –80H values are added together in the case of 8 bits when two's complement type data addition is performed, or two –8000H values in the case of 16 bits, Z is set to 1. When determining whether or not the result of a two's complement type data addition is 0, check for overflow beforehand using the overflow flag (V).

[Coding example]

```
DEC B
BZ $3C5H ; Branches to 003C5H if the B register is 0
           The start address of the BZ instruction must be in the range 00344H to 00443H
```

BNC
BNL

Branch if Not Carry/Less than
Conditional Branch upon Carry Flag (CY = 0)

[Instruction format] BNC $addr20
BNL $addr20

[Operation] PC ← PC + 2 + jdisp8 if CY = 0

[Operands]

Mnemonic	Operands ($addr20)
BNC	$addr20
BNL	

[Flags]

S	Z	AC	P/V	CY

[Description]

- If CY = 0, the program branches to the address specified by the operand.
  If 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 or shift/rotate instruction.
    Determines the result of bit manipulation.
  - BNL instruction: Checks whether a borrow has been generated after a subtraction instruction.
    After a compare instruction on unsigned data, checks whether or not the 1st operand of the compare instruction is smaller.

[Coding example]

```
CMP A, B     ; Compares the size of the A register contents and B register contents
BNL $1500H ; Branches to 01500H if the A register contents are smaller than the B register contents
               The start address of the BNL instruction must be in the range 0147FH to 0157EH
```

BC
BL

Branch if Carry/Less than
Conditional Branch upon Carry Flag (CY = 1)

[Instruction format] BC $addr20
BL $addr20

[Operation] PC ← PC + 2 + jdisp8 if CY = 1

[Operands]

Mnemonic	Operands ($addr20)
BC	$addr20
BL	

[Flags]

S	Z	AC	P/V	CY

[Description]

- If CY = 1, the program branches to the address specified by the operand.
  If 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. They are used as follows:
  – BC instruction : Checks whether a carry has been generated after an addition or shift/rotate instruction.
  Determines the result of bit manipulation.
  – BL instruction : Checks whether a borrow has been generated after a subtraction instruction.
  After a compare instruction on unsigned data, 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 address of the BC instruction must be in the range 0027FH to 0037EH

BNV
BPO

Branch if No Overflow/Branch if Parity Odd
Conditional Branch upon Parity/Overflow Flag (P/V = 0)

[Instruction format] BNV $addr20
BPO $addr20

[Operation] PC ← PC + 2 + jdisp8 if P/V = 0

[Operands]

Mnemonic	Operands ($addr20)
BNV	$addr20
BPO	

[Flags]

S	Z	AC	P/V	CY

[Description]

- If P/V = 0, the program branches to the address specified by the operand.
  If P/V = 1, no processing is performed and the next instruction is executed.
- The operation of the BNV instruction and the BPO instruction is the same. They are used as follows:
  - BNV instruction : Checks that the result has neither overflowed nor underflowed after an operation of two's complement format data, etc.
  - BPO instruction : Checks that the parity of the logical operation instruction or shift rotate instruction execution result is odd.

[Coding example]

```
ADD B, C    ; Adds together the contents of the B register and C register (two's complement type data)
BNV $560H   ; Branches to 560H if there is no overflow in the result of the addition
              The start address of the BNV instruction must be in the range 004DFH to 05DEH
```

BV
BPE

Branch if Overflow/Branch if Parity Even
Conditional Branch upon Parity/Overflow Flag (P/V = 1)

[Instruction format] BV $addr20
BPE $addr20

[Operation] PC ← PC + 2 + jdisp8 if P/V = 1

[Operands]

Mnemonic	Operands ($addr20)
BV	$addr20
BPE	

[Flags]

S	Z	AC	P/V	CY

[Description]
- If P/V = 1, the program branches to the address specified by the operand.
  If P/V = 0, no processing is performed and the next instruction is executed.
- The operation of the BV instruction and the BPE instruction is the same. They are used as follows:
  - BV instruction : Checks that the result has overflowed or underflowed after an operation of two's complement format data, etc.
  - BPE instruction : Checks that the parity of the logical operation instruction or shift rotate instruction execution result is even.

[Coding example]
OR D, #055H ; Finds the bit-wise logical sum of the D register contents and 055H
BPE $841EH ; Branches to 841EH if the parity is even as a result of finding the logical sum
The start address of this instruction must be in the range 839DH to 849CH

BP

Branch if Positive
Conditional Branch upon Sign Flag (S = 0)

[Instruction format] BP $addr20

[Operation] PC ← PC + 2 + jdisp8 if S = 0

[Operands]

Mnemonic	Operands ($addr20)
BP	$addr20

[Flags]

S	Z	AC	P/V	CY

[Description]

- If S = 0, the program branches to the address specified by the operand.
  If S = 1, no processing is performed and the next instruction is executed.
- This instruction is used to check whether the result is positive (including 0) after a two's complement type data operation. However, a correct judgment cannot be made if the operation result overflows or underflows; therefore, a BV instruction or BNV instruction should be used beforehand to check that there is no overflow or underflow, or the BGE instruction should be used.

[Coding example]

```
BV $OVER    ; Branches to address OVER, if the operation result overflows or underflows
BP $TARGET  ; Branches to address TARGET if the operation result is positive (including 0)
              Address TARGET must be within –126 to +129 of the start address of the BP instruction
```

BN

Branch if Negative
Conditional Branch upon Sign Flag (S = 1)

[Instruction format] BN \$addr20

[Operation] PC ← PC + 2 + jdisp8 if S = 1

[Operands]

Mnemonic	Operands (\$addr20)
BN	\$addr20

[Flags]

S	Z	AC	P/V	CY

[Description]

- If S = 1, the program branches to the address specified by the operand.
  If S = 0, no processing is performed and the next instruction is executed.
- This instruction is used to check whether the result is negative after a two's complement type data operation. However, a correct judgment cannot be made if the operation result overflows or underflows; therefore, a BV instruction or BNV instruction should be used beforehand to check that there is no overflow or underflow, or the BLT instruction should be used.

[Coding example]

BN #TARGET ; Branches to address TARGET if the operation result is negative

BLT

Branch if less than
Conditional Branch upon Size of Number (Less than ...)

[Instruction format] BLT $addr20

[Operation] PC ← PC + 3 + jdisp8 if $P/V \veebar S = 1$

[Operands]

Mnemonic	Operands ($addr20)
BLT	$addr20

[Flags]

S	Z	AC	P/V	CY

[Description]

- If $P/V \veebar S = 1$, the program branches to the address specified by the operand.
  If $P/V \veebar S = 0$, no processing is performed and the next instruction is executed.
- This instruction is used to determine the relative size of two's complement type data, or to check whether the result of an operation is negative. In relative size determination, the instruction checks whether the 1st operand of the CMP instruction executed immediately before is smaller than the 2nd operand. This instruction is also used to check whether the operation result is negative, including the case where underflow has occurred.

[Coding example]

```
CMPW AX, #3456H
BLT $8123H         ; Branches to address 8123H if the contents of the AX register are less than 3456H
                     The start address of the BLT instruction must be in the range 80A1H to 81A0H
```

BGE

Branch if Greater than/Equal
Conditional Branch upon Size of Number (Greater than or Equal to ...)

[Instruction format] BGE $addr20

[Operation] PC ← PC + 3 + jdisp8 if P/V $\forall$ S = 0

[Operands]

Mnemonic	Operands ($addr20)
BGE	$addr20

[Flags]

S	Z	AC	P/V	CY

[Description]

- If P/V $\forall$ S = 0, the program branches to the address specified by the operand.
  If P/V $\forall$ S = 1, no processing is performed and the next instruction is executed.
- This instruction is used to determine the relative size of two's complement type data, or to check whether the result of an operation is 0 or positive. In relative size determination, the instruction checks whether the 1st operand of the CMP instruction executed immediately before is greater than the 2nd operand. This instruction is also used to check whether the operation result is 0 or greater, including the case where overflow has occurred.

[Coding example]

ADDW AX, BC

BGE $23456H; Branches to address 23456H if the result of the immediately preceding addition instruction is 0 or greater

The start address of the BGE instruction must be in the range 233D4H to 234D3H

BLE

Branch if less than/Equal
Conditional Branch upon Size of Number (Less than or Equal to ...)

[Instruction format] BLE $addr20

[Operation] PC ← PC + 3 + jdisp8 if $(P/V \forall S) \vee Z = 1$

[Operands]

Mnemonic	Operands ($addr20)
BLE	$addr20

[Flags]

S	Z	AC	P/V	CY

[Description]

- If $(P/V \forall S) \vee Z = 1$, the program branches to the address specified by the operand.
  If $(P/V \forall S) \vee Z = 0$, no processing is performed and the next instruction is executed.
- This instruction is used to determine the relative size of two's complement type data, or to check whether the result of an operation is negative, including 0. In relative size determination, the instruction checks whether the 1st operand of the CMP instruction executed immediately before is smaller than the 2nd operand. This instruction is also used to check whether the operation result is negative, including the case where underflow has occurred.

[Coding example]

SUB H, L
BLE $789ABH ; Branches to 789ABH if the result of the immediately preceding subtraction instruction is 0 or less, including the case where underflow has occurred
The start address of the BL instruction must be in the range 78929H to 789ABH

BGT

Branch if Greater than
Conditional Branch upon Size of Number (Greater than ...)

[Instruction format] BGT $addr20

[Operation] PC ← PC + 3 + jdisp8 if $(P/V \forall S) \vee Z = 0$

[Operands]

Mnemonic	Operands ($addr20)
BGT	$addr20

[Flags]

S	Z	AC	P/V	CY

[Description]

- If $(P/V \forall S) \vee Z = 0$, the program branches to the address specified by the operand.
  If $(P/V \forall S) \vee Z = 1$, no processing is performed and the next instruction is executed.
- This instruction is used to determine the relative size of two's complement type data, or to check whether the result of an operation is greater than 0. In relative size determination, the instruction checks whether the 1st operand of the CMP instruction executed immediately before is greater than the 2nd operand. This instruction is also used to check whether the operation result is greater than 0, including the case where overflow has occurred.

[Coding example]

CMP A, E
BGT $0CFFEDH ;Branches to address 0CFFEDH if the contents of the A register are greater than the contents of the B register
The start address of the BGT instruction must be in the range 0CFF6BH to 0D006DH

BNH

Branch if Not Higher than
Conditional Branch upon Size of Number (Not Higher than ...)

[Instruction format] BNH $addr20

[Operation] PC ← PC + 3 + jdisp8 if Z ∨ CY = 1

[Operands]

Mnemonic	Operands ($addr20)
BNH	$addr20

[Flags]

S	Z	AC	P/V	CY

[Description]

- If Z ∨ CY = 1, the program branches to the address specified by the operand.
  If Z ∨ CY = 0, no processing is performed and the next instruction is executed.
- This instruction is used to determine the relative size of unsigned data. The instruction checks whether the 1st operand of the CMP instruction executed immediately before is not greater than the 2nd operand (i.e. the 1st operand is the same as or smaller than the 2nd operand).

[Coding example]

CMPW RP2, #8921H
BNH $TARGET ; Branches to address TARGET if the contents of the RP2 register are not greater than 8921H (equal to or less than 8912H)
The start address of the BNH instruction must be an address from which a branch can be made to address TARGET

BH

Branch if Higher than
Conditional Branch upon Size of Number (Higher than ...)

[Instruction format] BH $addr20

[Operation] PC $\leftarrow$ PC + 3 + jdisp8 if Z $\vee$ CY = 0

[Operands]

Mnemonic	Operands ($addr20)
BH	$addr20

[Flags]

S	Z	AC	P/V	CY

[Description]

- If Z $\vee$ CY = 0, the program branches to the address specified by the operand.
  If Z $\vee$ CY = 1, no processing is performed and the next instruction is executed.
- This instruction is used to determine the relative size of unsigned data. The instruction checks whether the 1st operand of the CMP instruction executed immediately before is greater than the 2nd operand.

[Coding example]

CMP B, C

BH $356H ; Branches to 356H if the contents of the B register are greater than the contents of the C register
The start address of the BH instruction must be in the range 2D4H to 3D3H

BF

Branch if False
Conditional Branch depending on Bit Test (Byte Data Bit = 0)

[Instruction format] BF bit, $addr20

[Operation] PC ← PC + b + jdisp8 if bit = 0

[Operands]

Mnemonic	Operands (bit, $addr20)	b (Number of Bytes)
BF	saddr.bit, $addr20	4/5
	sfr.bit, $addr20	4
	X.bit, $addr20	3
	A.bit, $addr20	3
	PSWL.bit, $addr20	3
	PSWH.bit, $addr20	3
	mem2.bit, $addr20	3
	!addr16.bit, $addr20	6
	!!addr24.bit, $addr20	7

[Flags]

S	Z	AC	P/V	CY

[Description]

- If the contents of the 1st operand (bit) are cleared (0), the program branches to the address specified by the 2nd operand ($addr20).
  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 address 01549H if bit 2 of port 2 is 0
The start address of the BF instruction must be in the range 014C6H to 015C5H

BT

Branch if True
Conditional Branch depending on Bit Test (Byte Data Bit = 1)

[Instruction format] BT bit, \$addr20

[Operation] PC ← PC + b + jdisp8 if bit = 1

[Operands]

Mnemonic	Operands (bit, \$addr20)	b (Number of Bytes)
BF	saddr.bit, \$addr20	3/4
	sfr.bit, \$addr20	4
	X.bit, \$addr20	3
	A.bit, \$addr20	3
	PSWL.bit, \$addr20	3
	PSWH.bit, \$addr20	3
	mem2.bit, \$addr20	3
	!addr16.bit, \$addr20	6
	!!addr24.bit, \$addr20	7

[Flags]

S	Z	AC	P/V	CY

[Description]

- If the contents of the 1st operand (bit) are set (1), the program 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
The start address of the BT instruction must be in the range 004D9H to 005D8H

BTCLR

Branch if True and Clear
Conditional Branch and Clear depending on Bit Test (Byte Data Bit = 1)

[Instruction format] BTCLR bit, $addr20

[Operation] PC ← PC + b + jdisp8 if bit = 1, then bit ← 0

[Operands]

Mnemonic	Operands (bit, $addr20)	b (Number of Bytes)
BTCLR	saddr.bit, $addr20	4/5
	sfr.bit, $addr20	4
	X.bit, $addr20	3
	A.bit, $addr20	3
	PSWL.bit, $addr20	3
	PSWH.bit, $addr20	3
	mem2.bit, $addr20	3
	!addr16.bit, $addr20	6
	!!addr24.bit, $addr20	7

[Flags]

bit is PSWL.bit

S	Z	AC	P/V	CY
×	×	×	×	×

In other cases

S	Z	AC	P/V	CY

[Description]

- If the contents of the 1st operand (bit) are set (1), the contents of the 1st operand (bit) are cleared (0), and the program 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, the contents of the relevant flag are cleared (0).

[Coding example]

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

BFSET

Branch if False and Set
Conditional Branch and Set depending on Bit Test (Byte Data Bit = 0)

[Instruction format] BFSET bit, $addr20

[Operation] PC ← PC + b + jdisp8 if bit = 0, then bit ← 1

[Operands]

Mnemonic	Operands (bit, $addr20)	b (Number of Bytes)
BFSET	saddr.bit, $addr20	4/5
	sfr.bit, $addr20	4
	X.bit, $addr20	3
	A.bit, $addr20	3
	PSWL.bit, $addr20	3
	PSWH.bit, $addr20	3
	mem2.bit, $addr20	3
	!addr16.bit, $addr20	6
	!!addr24.bit, $addr20	7

[Flags]

bit is PSWL.bit

S	Z	AC	P/V	CY
×	×	×	×	×

In other cases

S	Z	AC	P/V	CY

[Description]

- If the contents of the 1st operand (bit) are cleared (0), the contents of the 1st operand (bit) are set (1), and the program branches to the address specified by the 2nd operand.
  If the contents of the 1st operand (bit) are set (1), no processing is performed and the next instruction is executed.
- If the 1st operand (bit) is PSW.bit, the contents of the relevant flag are set (1).

[Coding example]

BFSET A.6, $3FFE1H ; If bit 6 of the A register is 0, sets (1) bit 6 of the A register and branches to address 3FFE1H
The start address of the BFSET instruction must be in the range 3FF5FH to 4005EH

DBNZ

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

[Instruction format] DBNZ dst, $addr20

[Operation] dst ← dst – 1,
then PC ← PC + b + jdisp8 if dst ≠ 0

[Operands]

Mnemonic	Operands (bit, $addr20)	b (Number of Bytes)
DBNZ	B, $addr20	2
	C, $addr20	2
	saddr, $addr20	3/4

[Flags]

S	Z	AC	P/V	CY

[Description]

- The contents of the destination operand (dst) specified by the 1st operand are decremented by 1, and the program branches to the destination operand (dst).
- If the result of decrementing the destination operand (dst) by 1 is not 0, the program branches to the address indicated by the 2nd operand ($addr20). 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 0, branches to 001215H
The start address of the DBNZ instruction must be in the range 001194H to 001293H

7.19 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]

S	Z	AC	P/V	CY

[Description]

- This a special instruction for writing to the standby control register (STBC). The immediate data specified by the 2nd operand is written to STBC. The STBC register can only be written to by means of this instruction.
- This instruction has a special format, and in addition to the immediate data used to perform the write, the logical NOT of that value must also be provided in the operation code (see figure below). (This is generated automatically by the NEC assembler (RA78K4).)

  – Operation code format

0 0 0 0 1 0 0 1
1 1 0 0 0 0 0 0
← $\overline{\text{byte}}$ →
← byte →

- The CPU checks the immediate data to be used for the write and the logical NOT data, and only performs the write if they are correct. If they are not correct, the write is not performed and an operand error interrupt is generated.

[Coding example]

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

MOV WDM, #byte

Move
Watchdog Timer Setting

[Instruction format] MOV WDM #byte

[Operation] WDM ← byte

[Operands]

Mnemonic	Operands
MOV	WDM, #byte

[Flags]

S	Z	AC	P/V	CY

[Description]

- This a special instruction for writing to the watchdog timer mode register (WDM). The immediate data specified by the 2nd operand is written to WDM. The WDM register can only be written to by means of this instruction.
- This instruction can only be used with a product that incorporates a watchdog timer. Please refer to the **User's Manual Hardware Volume** for the relevant product to see whether a watchdog timer is incorporated.
- This instruction has a special format, and in addition to the immediate data used to perform the write, the logical NOT of that value must also be provided in the operation code (see figure below). (This is generated automatically by the NEC assembler (RA78K4).)

  – Operation code format

0 0 0 0   1 0 0 1
1 1 0 0   0 0 1 0
← $\overline{\text{byte}}$ →
← byte →

- The CPU checks the immediate data to be used for the write and the logical NOT data, and only performs the write if they are correct. If they are not correct, the write is not performed and an operand error interrupt is generated.

[Coding example]

MOV WDM, #0C0H ; Writes 0C0H to WDM

LOCATION

Location
Location

[Instruction format] LOCATION locaddr

[Operation] SFR & internal data area location address upper word specification

[Operands]

Mnemonic	Operands
LOCATION	locaddr

[Flags]

S	Z	AC	P/V	CY

[Description]

- This instruction is used to specify the address of the internal data area (internal RAM and special function registers (SFRs)). If 0 is specified, the maximum address of the internal data area is 0FFFFH, and if 0FH is specified, the maximum address of the internal data area is 0FFFFFH.
- An interrupt or macro service request is not acknowledged between this instruction and the next instruction.
- This instruction must always be executed immediately after reset release. That is, this instruction must be located in the address specified by the reset vector. This instruction cannot be used more than once. If executed more than once, an address in the internal data area cannot be changed in the second or subsequent executions.
- The operand for this instruction is coded as shown below.

locaddr	Operand Code
0H	01FEH
0FH	00FFH

Execution of this instruction is ignored if a different value is specified. Also, an operand error interrupt is generated if the exclusive logical sum of the upper byte and lower byte of the operand is not 0FFH.

[Coding example]

LOCATION 0FH ; Sets the maximum address of the internal data area to 0FFFFFH

SEL RBn

Select Register Bank
Register Bank Selection

[Instruction format] SEL RBn

[Operation] RSS ← 0, RBS2 – 0 ← n ; (n = 0 – 3)

[Operands]

Mnemonic	Operands (RBn)
SEL	RBn

[Flags]

S	Z	AC	P/V	CY

[Description]

- Selects the register bank specified by the operand (RBn) as the register bank to be used from the next instruction onward.
- The range for RBn is RB0 to RB7.

[Coding example]

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

SEL RBn, ALT

Select Register Bank
Register Bank Selection

[Instruction format] SEL RBn, ALT

[Operation] RSS1 ← 1, RBS2 – 0 ← n ; (n = 0 – 3)

[Operands]

Mnemonic	Operands
SEL	RBn, ALT

[Flags]

S	Z	AC	P/V	CY

[Description]

- Selects the register bank specified by the 1st operand (RBn) as the register bank to be used from the next instruction onward, and also sets (1) the register selection flag (RSS).
- The range for RBn is RB0 to RB7.
- This instruction is provided to maintain compatibility with the 78K/III Series, and can only be used when a 78K/III Series program is used. It should not be used when using a program for a 78K Series other than the 78K/III Series or when using a newly written program.

SWRS

Switch Register Set
Register Bit Switching

[Instruction format] SWRS

[Operation] $RSS \leftarrow \overline{RSS}$

[Operands]

None

[Flags]

S	Z	AC	P/V	CY

[Description]

- Inverts the contents of the register set selection flag (RSS).
- This instruction is provided to maintain compatibility with the 78K/III Series, and can only be used when a 78K/III Series program is used. It should not be used when using a program for a 78K Series other than the 78K/III Series or when using a newly written program.

NOP

No Operation
No Operation

[Instruction format] NOP

[Operation] No Operation

[Operands]

None

[Flags]

S	Z	AC	P/V	CY

[Description]

- This instruction simply consumes time without performing any processing.

EI

Enable interrupt
Interrupt Enabling

[Instruction format] EI

[Operation] IE ← 1 (Enable interrupt)

[Operands]

None

[Flags]

S	Z	AC	P/V	CY

[Description]

- Sets the state in which maskable interrupts can be acknowledged (sets (1) the interrupt enable flag (IE)).
- No interrupts or macro service requests are acknowledged for a certain period after execution of this instruction. Please refer to the **User's Manual Hardware Volume** for the relevant product for details.
- It is possible to arrange for acknowledgment of vectored interrupts from other sources not to be performed even though this instruction is executed. Please refer to the **User's Manual Hardware Volume** for the individual products for details.

DI

Disable interrupt
Interrupt Disabling

[Instruction format] DI

[Operation] IE ← 0 (Disable interrupt)

[Operands]

None

[Flags]

S	Z	AC	P/V	CY

[Description]

- Disables acknowledgment by vectored interrupts among maskable interrupts (clears (0) the interrupt enable flag (IE)).
- No interrupts or macro service requests are acknowledged for a certain period after execution of this instruction. Please refer to the **User's Manual Hardware Volume** for the relevant product for details.
- Please refer to the **User's Manual Hardware Volume** for the individual products for details of interrupt servicing.

7.20 Special Instructions

Special instructions are as follows.

CHKL

Check Level
Pin Output Level Check

[Instruction format] CHKL sfr

[Operation] (Pin level) $\forall$ (output latch)

[Operands]

Mnemonic	Operands
CHKL	sfr

[Flags]

S	Z	AC	P/V	CY
×	×		P	

[Description]

- The exclusive logical sum of the output pin level and output buffer prestage signal level is found.
- The S flag is set (1) if bit 7 is set (1) as a result of the exclusive logical sum operation, and S flag is cleared (0) if bit 7 is cleared (0).
- The Z flag is set (1) if all bits are 0 as a result of the exclusive logical sum operation, and Z flag is cleared (0) if there are non-zero bits.
- The P/V flag is set (1) if the number of bits in the data set (1) as a result of the exclusive logical sum operation is even, and cleared (0) if the number is odd.
- This instruction is used to detect an abnormal state which has arisen for some reason or other in which the output pin level and the output buffer prestage signal level are different. In normal operation, the Z flag is always set (1).
- When this instruction is executed, with a product that has a port read control register (PRDC), the PRDC0 bit of the PRDC register must be cleared (0). An abnormal state cannot be detected if the PRDC0 bit is set (1).
- When this instruction is executed on a port that includes a pin used as a control output, the input/output mode for the port with a pin used as a control output must be set to input mode. If the input/output mode for a port with a pin used as a control output is set to output mode, operation may be judged to be abnormal even though it is normal.
- A pin for which the input/output mode as a port is specified as the input mode will always be judged to be normal by this instruction.

[Coding example]

CHKL P0
BNZ $ERROR ; Checks whether the port 0 pin level and output buffer prestage signal level match, and if they do not, branches to address ERROR

★ **Caution The CHKL instruction is not available in the μPD784216, 784216Y, 784218, 784218Y, 784225, 784225Y, 784937 Subseries. Do not execute this instruction. If this instruction is executed, the following condition will result.**

- **After the pin levels of output pins are read two times, they are exclusive-ORed. As a result, if the pins checked with this instruction are used in the port output mode, the exclusive-OR result is always 0 for all bits, and the Z flag is set to (1).**

CHKLA

Check Level and Transfer to Register
Pin Output Level Check and Transfer to Register

[Instruction format] CHKLA sfr

[Operation] A ← (Pin level) ∀ (output latch)

[Operands]

Mnemonic	Operands
CHKLA	sfr

[Flags]

S	Z	AC	P/V	CY
×	×		P	

[Description]

- The exclusive logical sum of the output pin level and output buffer prestage signal level is found, and the result is stored in the A register.
- The S flag is set (1) if bit 7 is set (1) as a result of the exclusive logical sum operation, and S flag is cleared (0) if bit 7 is cleared (0).
- The Z flag is set (1) if all bits are 0 as a result of the exclusive logical sum operation, and Z flag is cleared (0) if there are non-zero bits.
- The P/V flag is set (1) if the number of bits in the data set (1) as a result of the exclusive logical sum operation is even, and cleared (0) if the number is odd.
- This instruction is used to detect an abnormal state which has arisen for some reason or other in which the output pin level and the output buffer prestage signal level are different. In normal operation, the Z flag is always set (1).
- When this instruction is executed, with a product that has a port read control register (PRDC), the PRDC0 bit of the PRDC register must be cleared (0). An abnormal state cannot be detected if the PRDC0 bit is set (1).
- When this instruction is executed on a port that includes a pin used as a control output, the input/output mode for the port with a pin used as a control output must be set to input mode. If the input/output mode for a port with a pin used as a control output is set to output mode, operation may be judged to be abnormal even though it is normal.
- A pin for which the input/output mode as a port is specified as the input mode will always be judged to be normal by this instruction.

[Coding example]

CHKLA P3 ; Checks whether the port 3 pin level and output buffer prestage signal level match, and stores the result in the A register

★ **Caution The CHKLA instruction is not available in the μPD784216, 784216Y, 784218, 784218Y, 784225, 784225Y, 784937 Subseries. Do not execute this instruction. If this instruction is executed, the following condition will result.**

- **After the pin levels of output pins are read two times, they are Exclusive-ORed. As a result, if the pins checked with this instruction are used in the port output mode, the exclusive-OR result is always 0 for all bits, and the Z flag is set to (1) along with that the result is saved in the A register.**

7.21 String Instructions

String instructions are as follows.

MOVTBLW

Move Table Word
Table Word Transfer

[Instruction format] MOVTBLW !addr8, byte

[Operation] (addr8 + 2) ← (addr8),
byte ← byte – 1,
addr8 ← addr8 – 2,
End if byte = 0

[Operands]

Mnemonic	Operands
MOVTBLW	!addr16, byte

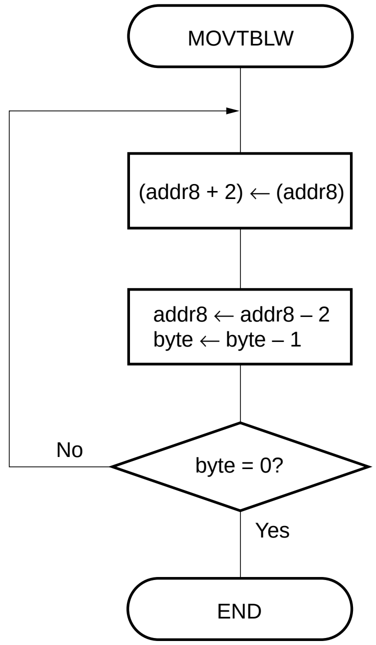
[Flags]

S	Z	AC	P/V	CY

[Description]

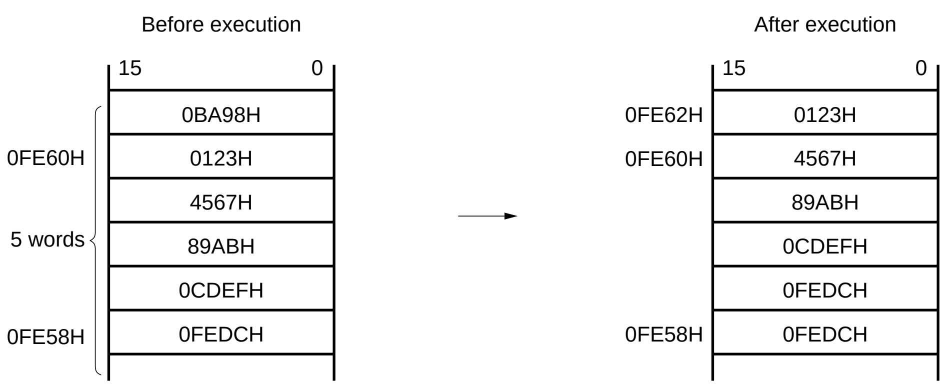
- The contents of the memory addressed by the 16 bits immediate data specified by the 1st operand are transferred to the address incremented by 2. addr8 is then decremented by 2. The above operations are repeated the number of times indicated by the 8 bits immediate data written as the 2nd operand.
- This instruction is used to shift the data table used by the MACW and MACSW instructions.
- The address of the most significant data of the data on which the transfer is to be performed is written directly in the 1st operand !addr8 as a label or number.
- The address written as the 1st operand must be in the range 00FE00H to 00FEFFH when a LOCATION 0 instruction is executed, or in the range 0FFE00H to 0FFEFFH when a LOCATION 0FH instruction is executed.

Remark The μPD784915 Subseries is fixed to the LOCATION 0 instruction.



[Coding example]

MOVTBLW !0FFE60H, 5 ; Transfers the data in 0FFE58H through 0FFE60H to 0FFE5AH through 0FFE62H



MOVM

Move Multiple Byte
Block Transfer of Fixed Byte Data

[Instruction format] MOVM [TDE +], A
MOVM [TDE –], A

[Operation] (TDE) ← A, TDE ← TDE + 1, C ← C – 1 End if C = 0
(TDE) ← A, TDE ← TDE – 1, C ← C – 1 End if C = 0

[Operands]

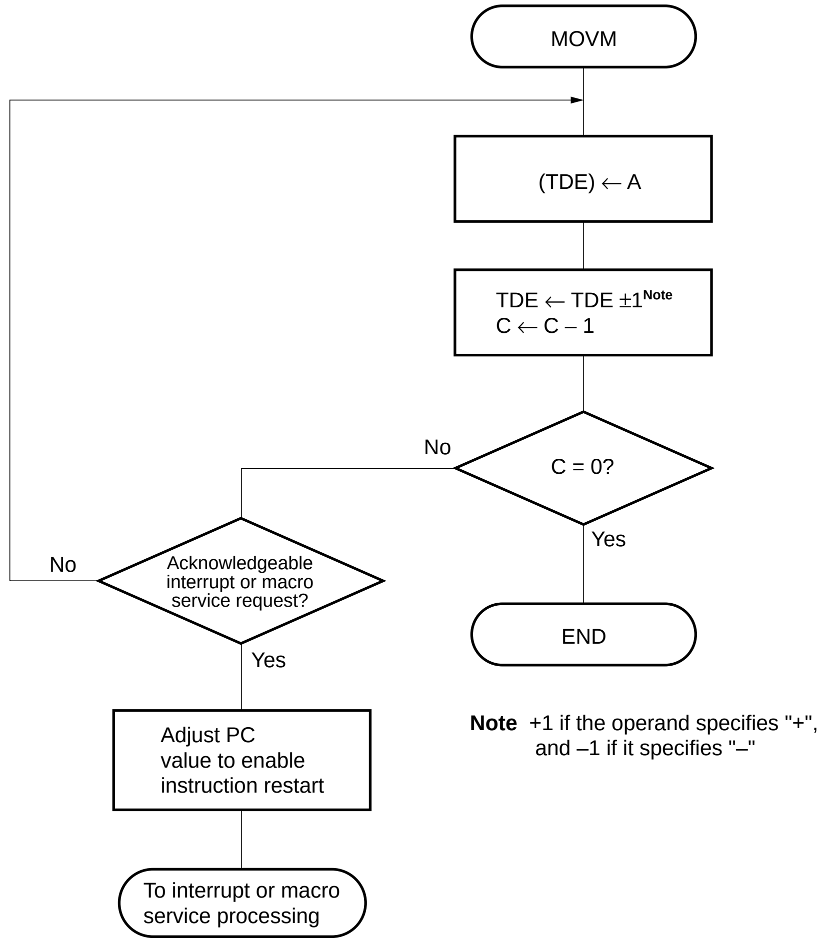
Mnemonic	Operands
MOVM	[TDE +], A
	[TDE –], A

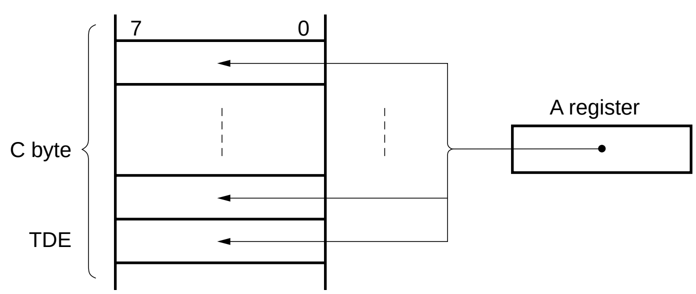
[Flags]

S	Z	AC	P/V	CY

[Description]

- The contents of the A register are transferred to the memory addressed by the TDE register, and the contents of the TDE register are incremented/decremented. The contents of the C register are then decremented, and the above operations are repeated until the contents of the C register are 0.
- If an acknowledgeable interrupt or macro service request is generated during execution of this instruction, execution of this instruction is interrupted and the interrupt or macro service request is acknowledged. When an interrupt is acknowledged, if the return address and the contents of the TDE and C registers used by this instruction which have been saved to the stack or to RP2 and R7 are not changed, execution of the interrupted instruction is resumed upon returning from the interrupt. When a macro service request is acknowledged, execution of this instruction is resumed after completion of the macro service.
- This instruction is mainly used to initialize a certain area of memory with a specific value. The MOVBK instruction is used to perform initialization with multi-byte data.





[Coding example]

```
MOV C, #00H          ;  C ← 00H
MOV A, #00H          ;  A ← 00H
MOVG TDE, #0FE00H    ;  TDE ← FE00H
MOVM [TDE +], A      ;  Clears RAM FE00H to FEFFH
```

XCHM

Exchange Multiple Byte
Block Exchange of Fixed Byte Data

[Instruction format] XCHM [TDE +], A
XCHM [TDE –], A

[Operation] (TDE) ↔ A, TDE ← TDE + 1, C ← C – 1 End if C = 0
(TDE) ↔ A, TDE ← TDE – 1, C ← C – 1 End if C = 0

[Operands]

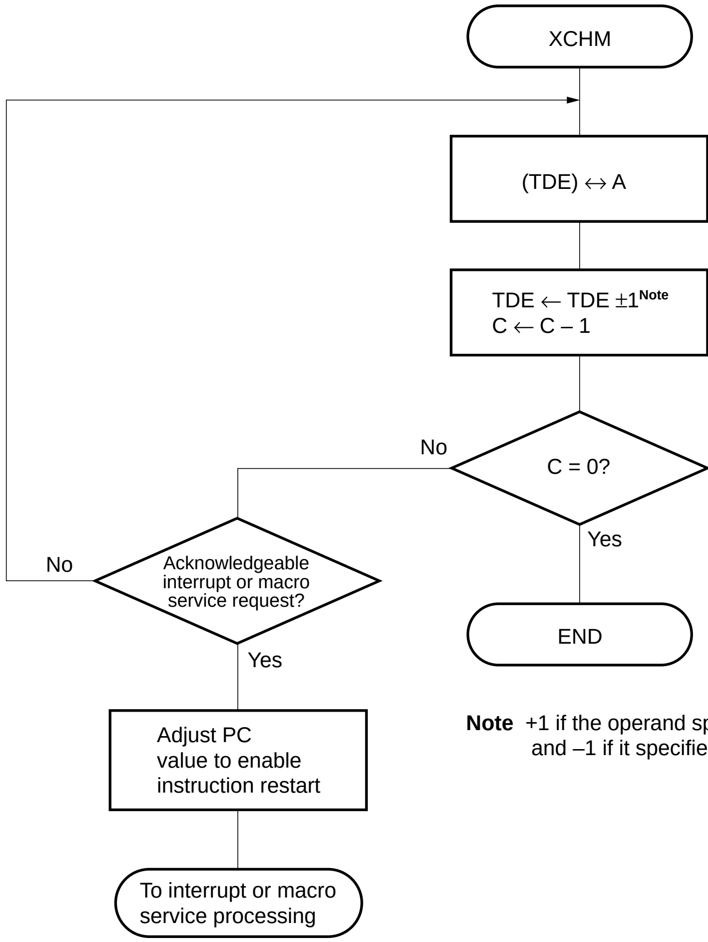
Mnemonic	Operands
XCHM	[TDE +], A
	[TDE –], A

[Flags]

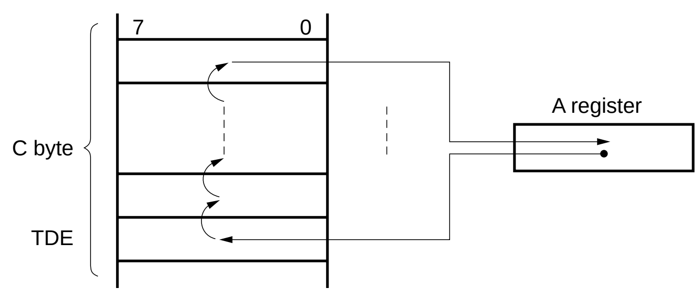
S	Z	AC	P/V	CY

[Description]

- The contents of the A register are exchanged with the contents of the memory addressed by the TDE register, and the contents of the TDE register are incremented/decremented. The contents of the C register are then decremented, and the above operations are repeated until the contents of the C register are 0.
- If an acknowledgeable interrupt or macro service request is generated during execution of this instruction, execution of this instruction is interrupted and the interrupt or macro service request is acknowledged. When an interrupt is acknowledged, if the return address and the contents of the TDE and C registers used by this instruction which have been saved to the stack or to RP2 and R7 are not changed, execution of the interrupted instruction is resumed upon returning from the interrupt. When a macro service request is acknowledged, execution of this instruction is resumed after completion of the macro service.
- This instruction is mainly used to perform a one-byte move of data in memory. XCHM [TDE +], A is used for a move in the upper address direction, and XCHM [TDE –], A for a move in the low-order address direction. The MOVBK instruction is used to move two or more bytes.



Note +1 if the operand specifies "+", and –1 if it specifies "–"



[Coding example]

MOV C, #10H	; C ← 10H
MOV A, #00H	; A ← 00H
MOVG TDE, #3050H	; TDE ← 3050H
XCHM [TDE +], A	; Shifts the contents of memory 3050H through 305FH one address at a time into the addresses behind (the contents of address 3050H become 0)

MOVBK

Move Block Byte
Byte Data Block Transfer

[Instruction format] MOVBK [TDE +], [WHL +]
MOVBK [TDE –], [WHL –]

[Operation] (TDE) ← (WHL), TDE ← TDE + 1, WHL ← WHL + 1 C ← C – 1
End if C = 0
(TDE) ← (WHL), TDE ← TDE – 1, WHL ← WHL – 1 C ← C – 1
End if C = 0

[Operands]

Mnemonic	Operands
MOVBK	[TDE +], [WHL +]
	[TDE –], [WHL –]

[Flags]

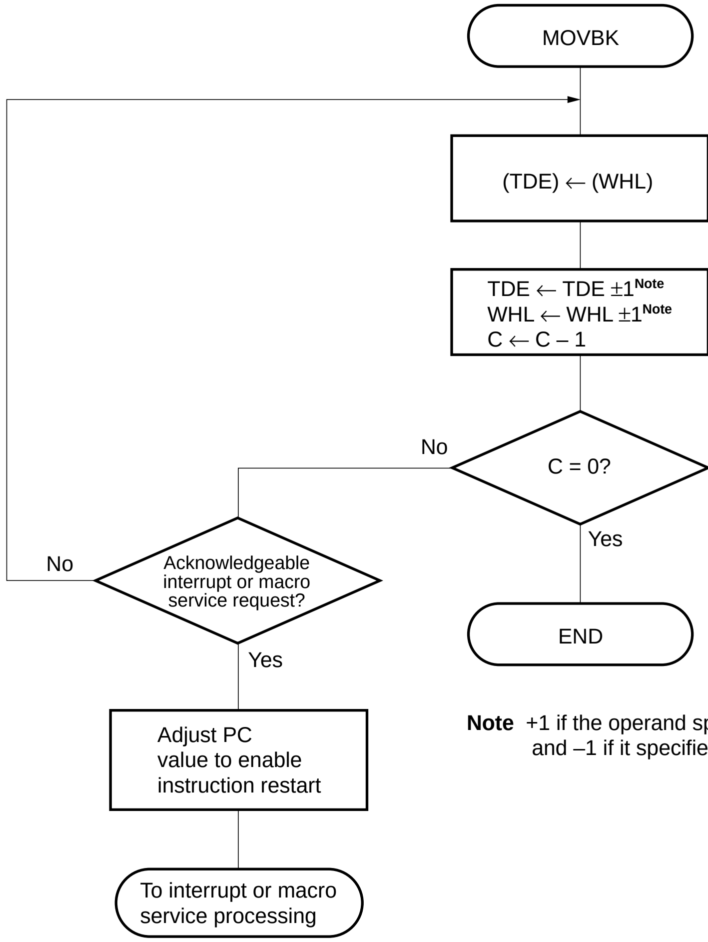
S	Z	AC	P/V	CY

[Description]

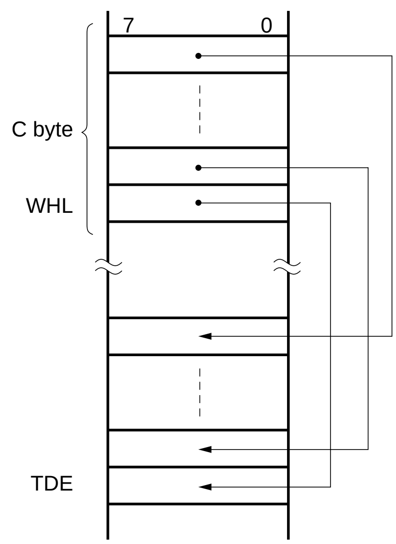
- The contents of the memory addressed by the WHL register are transferred to the memory addressed by the TDE register, and the contents of the TDE and WHL registers are incremented/decremented. The contents of the C register are then decremented, and the above operations are repeated until the contents of the C register are 0.
- If an acknowledgeable interrupt or macro service request is generated during execution of this instruction, execution of this instruction is interrupted and the interrupt or macro service request is acknowledged. When an interrupt is acknowledged, if the return address and the contents of the TDE, WHL, and C registers used by this instruction which have been saved to the stack or to RP2 and R7 are not changed, execution of the interrupted instruction is resumed upon returning from the interrupt. When a macro service request is acknowledged, execution of this instruction is resumed after completion of the macro service.
- If the transfer source data area and transfer destination data area overlap, the operation is as follows.

  – If the minimum address of the transfer source is smaller than the maximum address of the transfer destination, the respective minimum addresses are used as the initial values for both the TDE and the WHL register, and MOVBK [TDE +], [WHL +] is used.

  – If the maximum address of the transfer source is greater than the minimum address of the transfer destination, the respective maximum addresses are used as the initial values for both the TDE and the WHL register, and MOVBK [TDE –], [WHL –] is used.



Note +1 if the operand specifies "+", and –1 if it specifies "–"



[Coding example]

```
MOV C, #10H                 ; C ← 10H
MOVG TDE, #3000H            ; TDE ← 3000H
MOVG WHL, #5000H            ; WHL ← 5000H
MOVBK [TDE + ], [WHL + ]    ; Transfers the contents of memory 5000H through 500FH to memory 3000H through
                              300FH
```

XCHBK

Exchange Block Byte
Byte Data Block Exchange

[Instruction format] XCHBK [TDE +], [WHL +]
XCHBK [TDE –], [WHL –]

[Operation] (TDE) ↔ (WHL), TDE ← TDE + 1,
WHL ← WHL + 1 C ← C – 1 End if C = 0
(TDE) ↔ (WHL), TDE ← TDE – 1,
WHL ← WHL – 1 C ← C – 1 End if C = 0

[Operands]

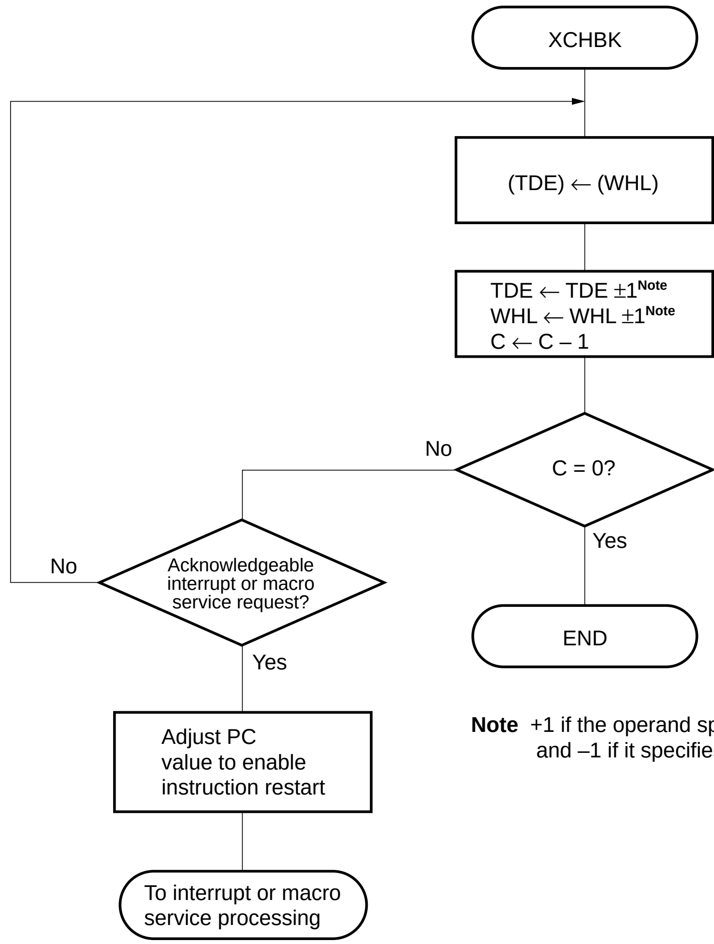
Mnemonic	Operands
XCHBK	[TDE +], [WHL +]
	[TDE –], [WHL –]

[Flags]

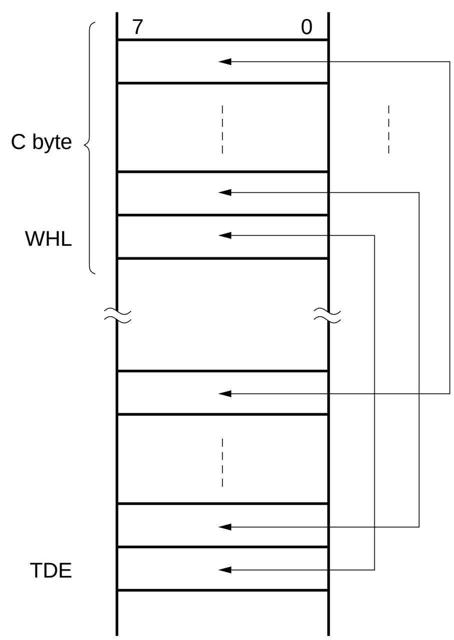
S	Z	AC	P/V	CY

[Description]

- The contents of the memory addressed by the WHL register are exchanged with the contents of the memory addressed by the TDE register, and the contents of the WHL and TDE registers are incremented/decremented. The contents of the C register are then decremented, and the above operations are repeated until the contents of the C register are 0.
- If an acknowledgeable interrupt or macro service request is generated during execution of this instruction, execution of this instruction is interrupted and the interrupt or macro service request is acknowledged. When an interrupt is acknowledged, if the return address and the contents of the TDE, WHL, and C registers used by this instruction which have been saved to the stack or to RP2 and R7 are not changed, execution of the interrupted instruction is resumed upon returning from the interrupt. When a macro service request is acknowledged, execution of this instruction is resumed after completion of the macro service.



Note +1 if the operand specifies "+", and –1 if it specifies "–"



[Coding example]

```
MOV C, #80H
MOVG TDE, #3456H
MOVG WHL, #1FF96H
XCHBK [TDE + ], [WHL + ] ; Exchanges the 80H-byte data from address 3456H with the data from address 1FF96H
```

CMPME

Compare Multiple Equal Byte
Block Comparison with Fixed Byte Data (Match Detection)

[Instruction format] CMPME [TDE +], A
CMPME [TDE –], A

[Operation] (TDE) – A, TDE ← TDE + 1, C ← C – 1 End if C = 0 or Z = 0
(TDE) – A, TDE ← TDE – 1, C ← C – 1 End if C = 0 or Z = 0

[Operands]

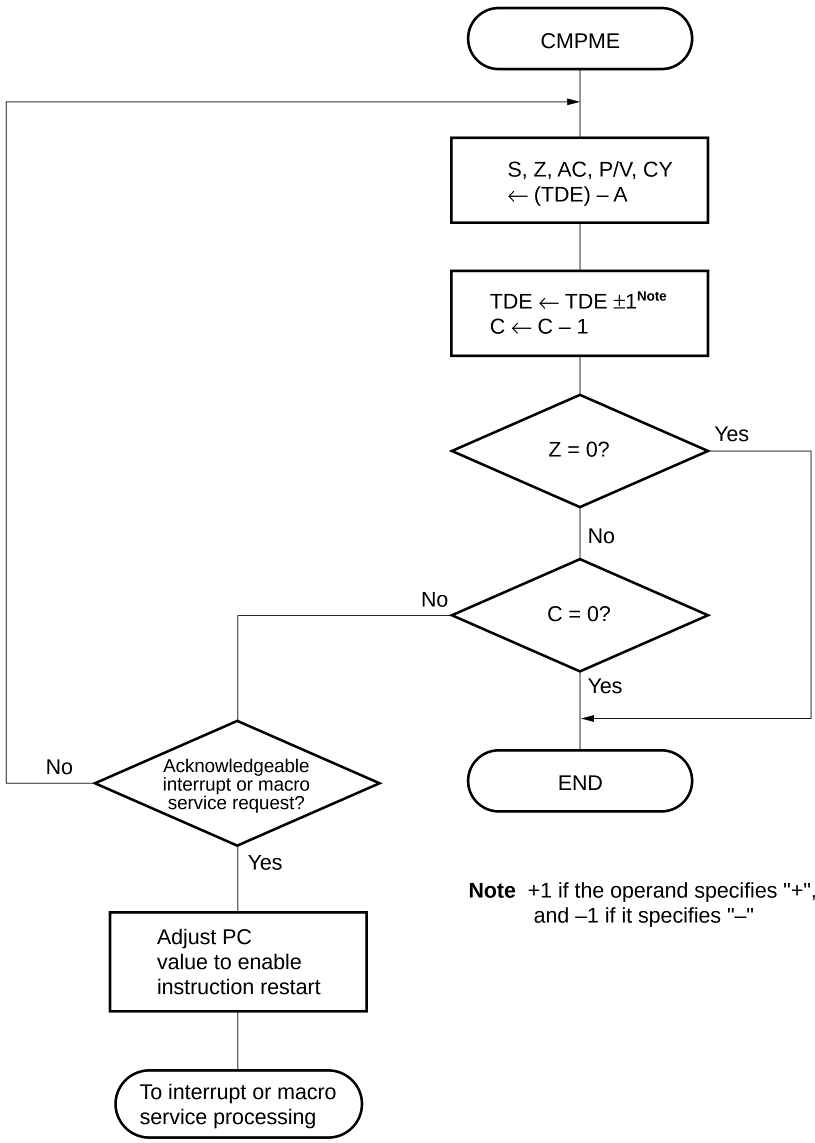
Mnemonic	Operands
CMPME	[TDE +], A
	[TDE –], A

[Flags]

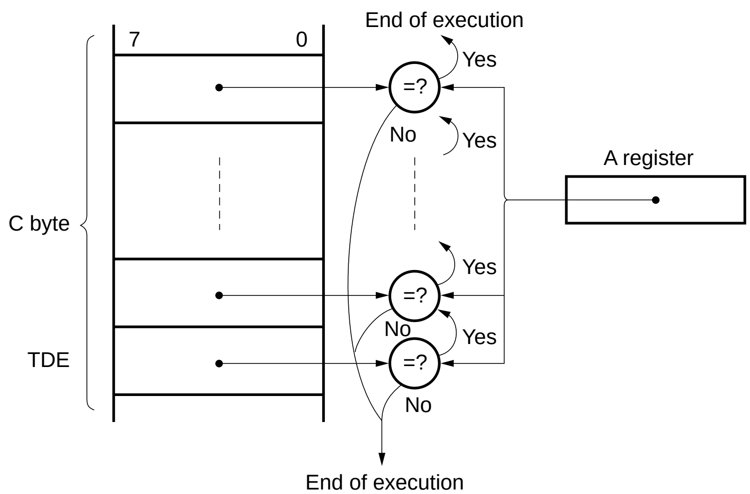
S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The contents of the A register are compared with the contents of the memory addressed by the TDE register, the contents of the TDE register are incremented/decremented, and the contents of the C register are decremented. The above operations are repeated until the result of the comparison is a mismatch, or the contents of the C register are 0.
- Execution of this instruction does not change the contents of the A register or of the memory addressed by the TDE register.
- If an acknowledgeable interrupt or macro service request is generated during execution of this instruction, execution of this instruction is interrupted and the interrupt or macro service request is acknowledged. When an interrupt is acknowledged, if the return address and the contents of the TDE and C registers used by this instruction which have been saved to the stack or to RP2 and R7 are not changed, execution of the interrupted instruction is resumed upon returning from the interrupt. When a macro service request is acknowledged, execution of this instruction is resumed after completion of the macro service.
- The S, Z, AC, P/V, and CY flags are changed in accordance with the last compare operation (subtraction) executed by this instruction.
- The S flag is set (1) if bit 7 is set (1) as a result of the subtraction, and cleared (0) otherwise.
- The Z flag is set (1) if the result of the subtraction is 0, and z flag is cleared (0) otherwise.
- The AC flag is set (1) if a borrow is generated out of bit 4 into bit 3 as a result of the subtraction, and cleared (0) otherwise.
- The P/V flag is set (1) if a borrow is generated in bit 6 and a borrow is not generated in bit 7 as a result of the subtraction (when underflow is generated by a two's complement type operation), or if a borrow is not generated in bit 6 and a borrow is generated in bit 7 (when overflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a borrow is generated in bit 7 as a result of the subtraction, and cleared (0) otherwise.



Note +1 if the operand specifies "+", and –1 if it specifies "–"



[Coding example]

MOV C, #20H
MOVG TDE, #56283H
MOV A, #00H
CMPME [TDE +], A ; Indicates whether the 20H-byte data from address 56283H is all 00H
BNZ $JMP ; Branches to address JMP if there is data that is not 00H

CMPMNE

Compare Multiple Not Equal Byte
Block Comparison with Fixed Byte Data (Mismatch Detection)

[Instruction format] CMPMNE [TDE +], A
CMPMNE [TDE –], A

[Operation] (TDE) – A, TDE ← TDE + 1, C ← C – 1 End if C = 0 or Z = 1
(TDE) – A, TDE ← TDE – 1, C ← C – 1 End if C = 0 or Z = 1

[Operands]

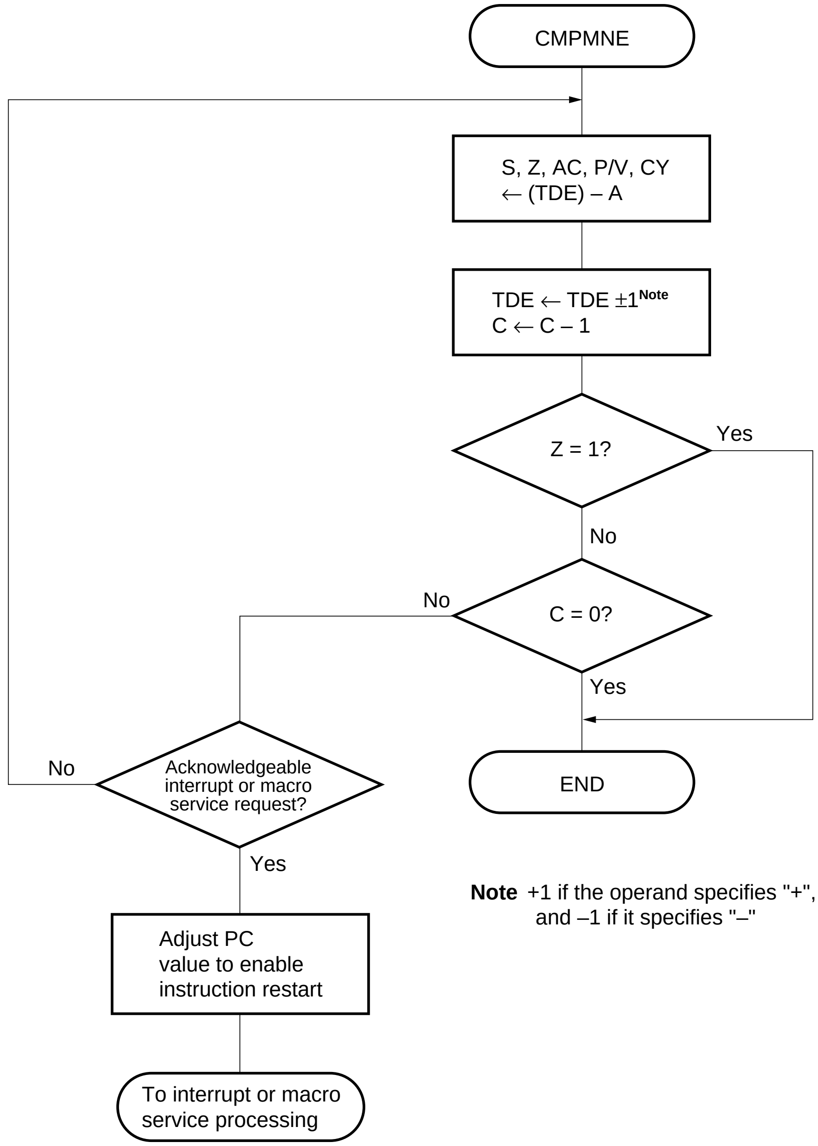
Mnemonic	Operands
CMPMNE	[TDE +], A
	[TDE –], A

[Flags]

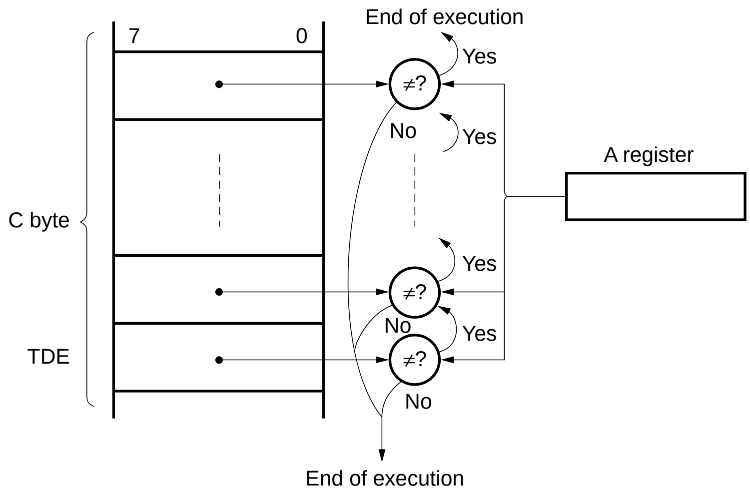
S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The contents of the A register are compared with the contents of the memory addressed by the TDE register, the contents of the TDE register are incremented/decremented, and the contents of the C register are decremented. The above operations are repeated until the result of the comparison is a match or the contents of the C register are 0.
- Execution of this instruction does not change the contents of the A register or of the memory addressed by the TDE register.
- If an acknowledgeable interrupt or macro service request is generated during execution of this instruction, execution of this instruction is interrupted and the interrupt or macro service request is acknowledged. When an interrupt is acknowledged, if the return address and the contents of the TDE and C registers used by this instruction which have been saved to the stack or to RP2 and R7 are not changed, execution of the interrupted instruction is resumed upon returning from the interrupt. When a macro service request is acknowledged, execution of this instruction is resumed after completion of the macro service.
- The S, Z, AC, P/V, and CY flags are changed in accordance with the last compare operation (subtraction) executed by this instruction.
- The S flag is set (1) if bit 7 is set (1) as a result of the subtraction, and cleared (0) otherwise.
- The Z flag is set (1) if the result of the subtraction is 0, and z flag is cleared (0) otherwise.
- The AC flag is set (1) if a borrow is generated out of bit 4 into bit 3 as a result of the subtraction, and cleared (0) otherwise.
- The P/V flag is set (1) if a borrow is generated in bit 6 and a borrow is not generated in bit 7 as a result of the subtraction (when underflow is generated by a two's complement type operation), or if a borrow is not generated in bit 6 and a borrow is generated in bit 7 (when overflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a borrow is generated in bit 7 as a result of the subtraction, and cleared (0) otherwise.



Note +1 if the operand specifies "+", and –1 if it specifies "–"



[Coding example]

```
MOV C, #00H          ; C ← 00H
MOVG TDE, #3000H     ; TDE ← 3000H
CMPMNE [TDE +], A
BZ $IMP              ; Branches to the address indicated by label IMP if the same value as that of the A register
                       is in 3000H to 30FFH
```

CMPMC

Compare Multiple Carry Byte
Block Comparison with Fixed Byte Data (Size Comparison)

[Instruction format] CMPMC [TDE +], A
CMPMC [TDE –], A

[Operation] (TDE) – A, TDE ← TDE + 1, C ← C – 1 End if C = 0 or CY = 0
(TDE) – A, TDE ← TDE – 1, C ← C – 1 End if C = 0 or CY = 0

[Operands]

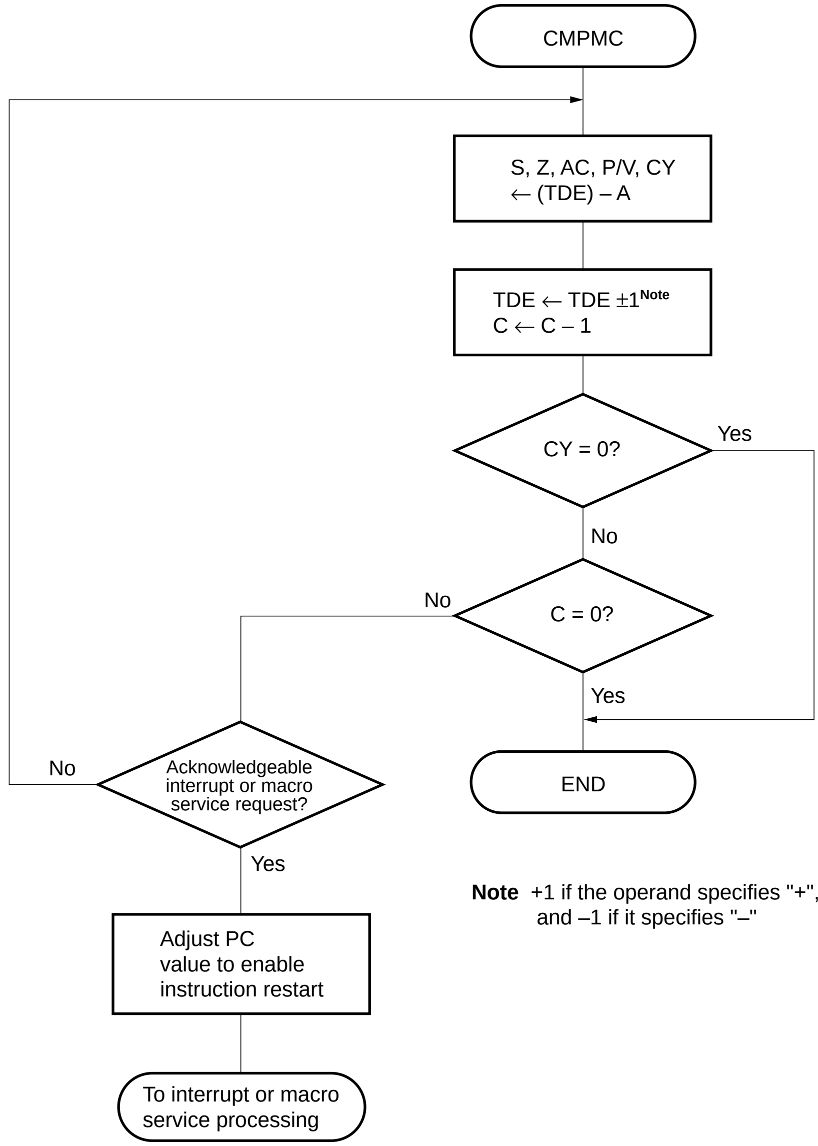
Mnemonic	Operands
CMPMC	[TDE +], A
	[TDE –], A

[Flags]

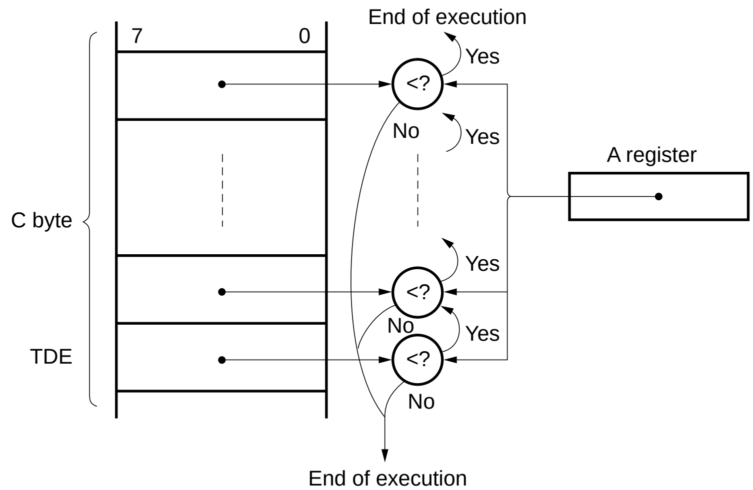
S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The contents of the A register are compared with the contents of the memory addressed by the TDE register, the contents of the TDE register are incremented/decremented, and the contents of the C register are decremented. The above operations are repeated until the result of the comparison is that the contents of the memory addressed by the TDE register are equal to or greater than the contents of the A register, or the contents of the C register are 0.
- Execution of this instruction does not change the contents of the A register or of the memory addressed by the TDE register.
- If an acknowledgeable interrupt or macro service request is generated during execution of this instruction, execution of this instruction is interrupted and the interrupt or macro service request is acknowledged. When an interrupt is acknowledged, if the return address and the contents of the TDE and C registers used by this instruction which have been saved to the stack or to RP2 and R7 are not changed, execution of the interrupted instruction is resumed upon returning from the interrupt. When a macro service request is acknowledged, execution of this instruction is resumed after completion of the macro service.
- The S, Z, AC, P/V, and CY flags are changed in accordance with the last compare operation (subtraction) executed by this instruction.
- The S flag is set (1) if bit 7 is set (1) as a result of the subtraction, and cleared (0) otherwise.
- The Z flag is set (1) if the result of the subtraction is 0, and z flag is cleared (0) otherwise.
- The AC flag is set (1) if a borrow is generated out of bit 4 into bit 3 as a result of the subtraction, and cleared (0) otherwise.
- The P/V flag is set (1) if a borrow is generated in bit 6 and a borrow is not generated in bit 7 as a result of the subtraction (when underflow is generated by a two's complement type operation), or if a borrow is not generated in bit 6 and a borrow is generated in bit 7 (when overflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a borrow is generated in bit 7 as a result of the subtraction, and cleared (0) otherwise.



Note +1 if the operand specifies "+", and –1 if it specifies "–"



[Coding example]

MOV C, #10H

MOV A, #80H

MOVG TDE, #567800H

CMPMC [TDE +], A

BNC $BIG ; Branches to address BIG if data of 80H or above is present in the 10H-byte data from address 567800H

CMPMNC

Compare Multiple Not Carry Byte
Block Comparison with Fixed Byte Data (Size Comparison)

[Instruction format] CMPMNC [TDE +], A
CMPMNC [TDE –], A

[Operation] (TDE) – A, TDE ← TDE + 1, C ← C – 1 End if C = 0 or CY = 1
(TDE) – A, TDE ← TDE – 1, C ← C – 1 End if C = 0 or CY = 1

[Operands]

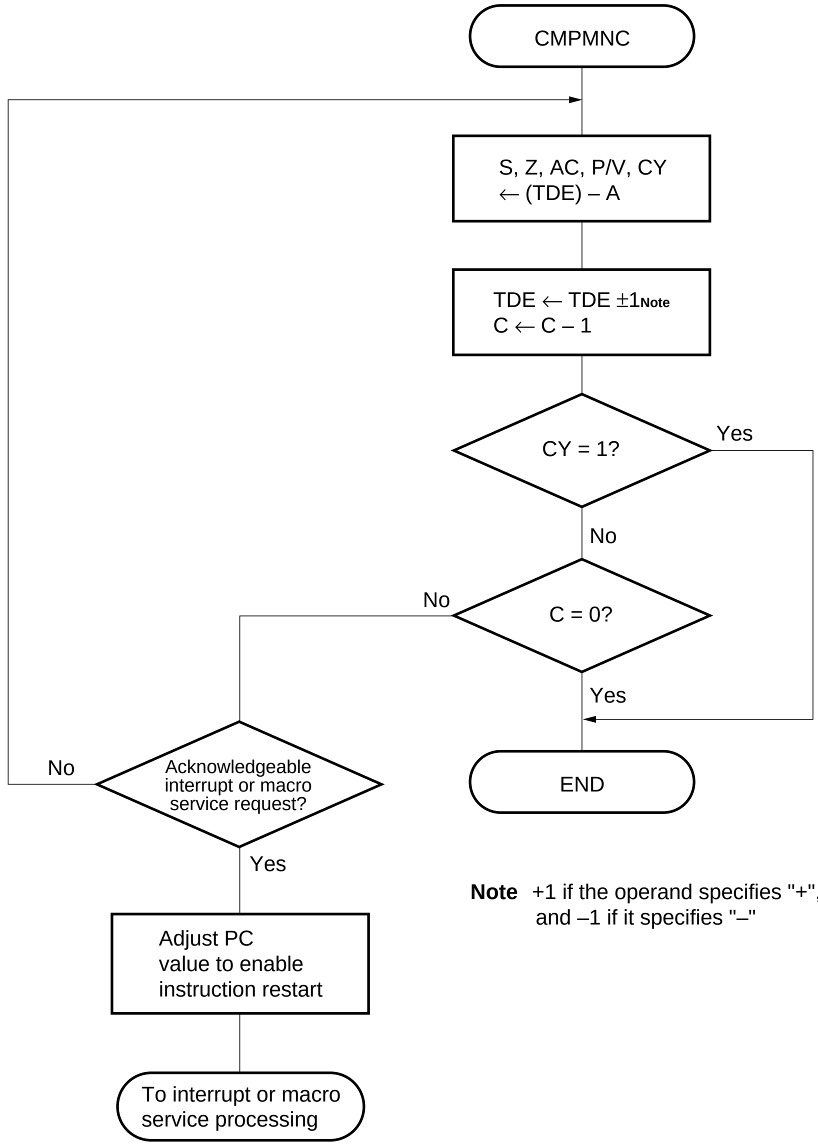
Mnemonic	Operands
CMPMNC	[TDE +], A
	[TDE –], A

[Flags]

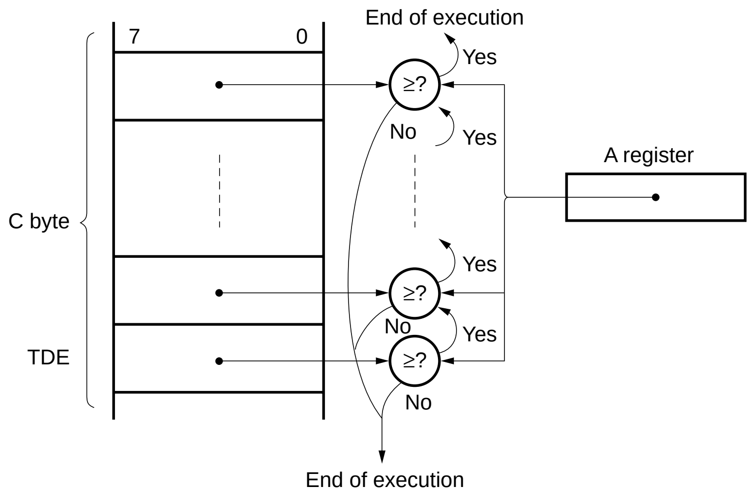
S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The contents of the A register are compared with the contents of the memory addressed by the TDE register, the contents of the TDE register are incremented/decremented, and the contents of the C register are decremented. The above operations are repeated until the result of the comparison is that the contents of the A register are greater, or the contents of the C register are 0.
- Execution of this instruction does not change the contents of the A register or of the memory addressed by the TDE register.
- If an acknowledgeable interrupt or macro service request is generated during execution of this instruction, execution of this instruction is interrupted and the interrupt or macro service request is acknowledged. When an interrupt is acknowledged, if the return address and the contents of the TDE and C registers used by this instruction which have been saved to the stack or to RP2 and R7 are not changed, execution of the interrupted instruction is resumed upon returning from the interrupt. When a macro service request is acknowledged, execution of this instruction is resumed after completion of the macro service.
- The S, Z, AC, P/V, and CY flags are changed in accordance with the last compare operation (subtraction) executed by this instruction.
- The S flag is set (1) if bit 7 is set (1) as a result of the subtraction, and cleared (0) otherwise.
- The Z flag is set (1) if the result of the subtraction is 0, and z flag is cleared (0) otherwise.
- The AC flag is set (1) if a borrow is generated out of bit 4 into bit 3 as a result of the subtraction, and cleared (0) otherwise.
- The P/V flag is set (1) if a borrow is generated in bit 6 and a borrow is not generated in bit 7 as a result of the subtraction (when underflow is generated by a two's complement type operation), or if a borrow is not generated in bit 6 and a borrow is generated in bit 7 (when overflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a borrow is generated in bit 7 as a result of the subtraction, and cleared (0) otherwise.



Note +1 if the operand specifies "+", and –1 if it specifies "–"



[Coding example]

MOV C, #00H	; C ← 00H
MOVG TDE, #8000H	; TDE ← 8000H
CMPMNC [TDE +], A	
BC $JMP	; Branches to the address indicated by label JMP if there is a value greater than the contents of the A register in 8000H to 80FFH

CMPBKE

Compare Block Equal Byte
Block Comparison with Byte Data (Match Detection)

[Instruction format] CMPBKE [TDE +], [WHL +]
CMPBKE [TDE –], [WHL –]

[Operation] (TDE) – (WHL), TDE ← TDE + 1, WHL ← WHL + 1, C ← C – 1
End if C = 0 or Z = 0
(TDE) – (WHL), TDE ← TDE – 1, WHL ← WHL – 1, C ← C – 1
End if C = 0 or Z = 0

[Operands]

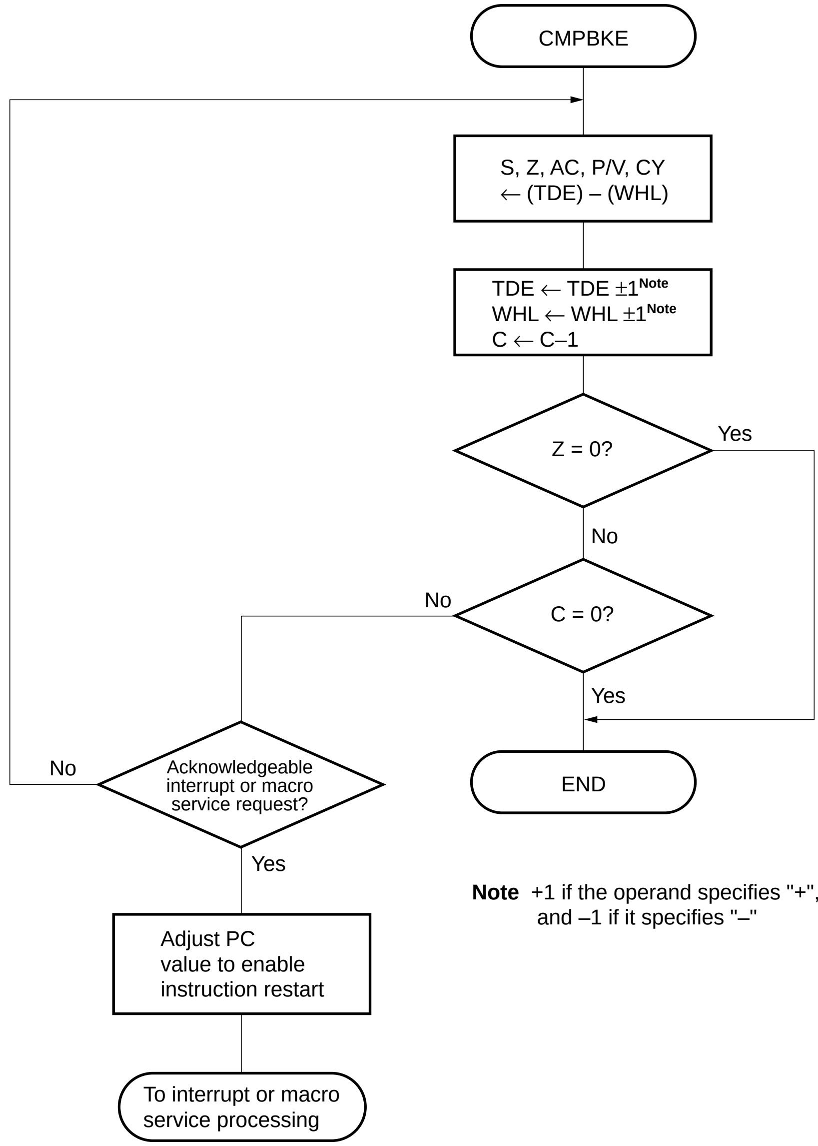
Mnemonic	Operands
CMPBKE	[TDE +], [WHL +]
	[TDE –], [WHL –]

[Flags]

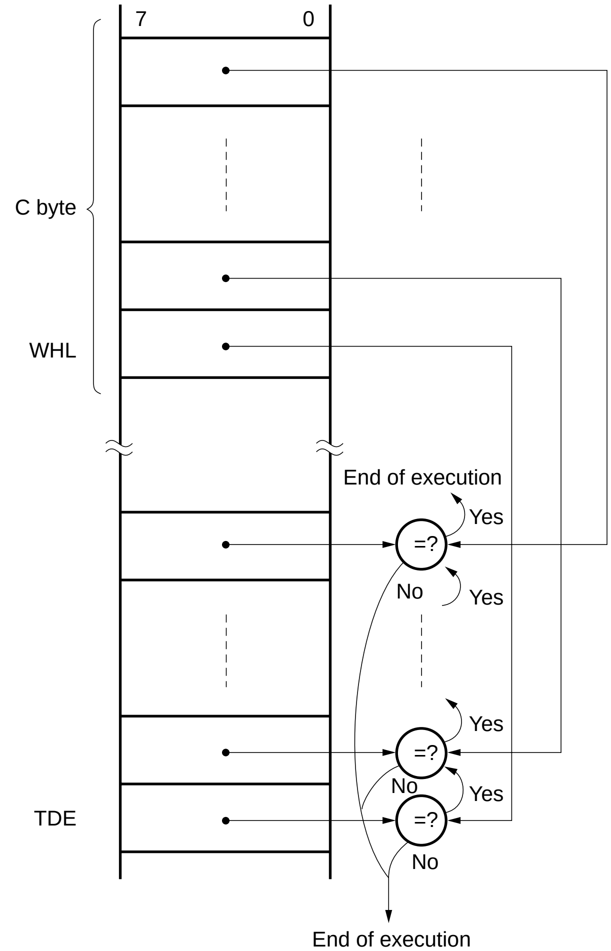
S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The contents of the memory addressed by the WHL register are compared with the contents of the memory addressed by the TDE register, the contents of the TDE and WHL registers are incremented/decremented, and the contents of the C register are decremented. The above operations are repeated until the result of the comparison is a mismatch, or the contents of the C register are 0.
- Execution of this instruction does not change the contents of the memory addressed by the TDE and WHL registers.
- If an acknowledgeable interrupt or macro service request is generated during execution of this instruction, execution of this instruction is interrupted and the interrupt or macro service request is acknowledged. When an interrupt is acknowledged, if the return address and the contents of the TDE, WHL, and C registers used by this instruction which have been saved to the stack or to RP2 and R7 are not changed, execution of the interrupted instruction is resumed upon returning from the interrupt. When a macro service request is acknowledged, execution of this instruction is resumed after completion of the macro service.
- The S, Z, AC, P/V, and CY flags are changed in accordance with the last compare operation (subtraction) executed by this instruction.
- The S flag is set (1) if bit 7 is set (1) as a result of the subtraction, and cleared (0) otherwise.
- The Z flag is set (1) if the result of the subtraction is 0, and z flag is cleared (0) otherwise.
- The AC flag is set (1) if a borrow is generated out of bit 4 into bit 3 as a result of the subtraction, and cleared (0) otherwise.
- The P/V flag is set (1) if a borrow is generated in bit 6 and a borrow is not generated in bit 7 as a result of the subtraction (when underflow is generated by a two's complement type operation), or if a borrow is not generated in bit 6 and a borrow is generated in bit 7 (when overflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a borrow is generated in bit 7 as a result of the subtraction, and cleared (0) otherwise.



Note +1 if the operand specifies "+", and –1 if it specifies "–"



[Coding example]

MOV C, #40H
MOVG TDE, #342156H
MOVG WHL, #3421AAH
CMPBKE [TDE +], [WHL +]
BNE $DIFF ; Compares the 40H-byte data from address 342156H with the data from address 3421AAH, and branches to address DIFF if there is different data

CMPBKNE

Compare Block Not Equal Byte
Block Comparison with Byte Data (Mismatch Detection)

[Instruction format] CMPBKNE [TDE +], [WHL +]
CMPBKNE [TDE –], [WHL –]

[Operation] (TDE) – (WHL), TDE ← TDE + 1, WHL ← WHL + 1, C ← C – 1
End if C = 0 or Z = 1
(TDE) – (WHL), TDE ← TDE – 1, WHL ← WHL – 1, C ← C – 1
End if C = 0 or Z = 1

[Operands]

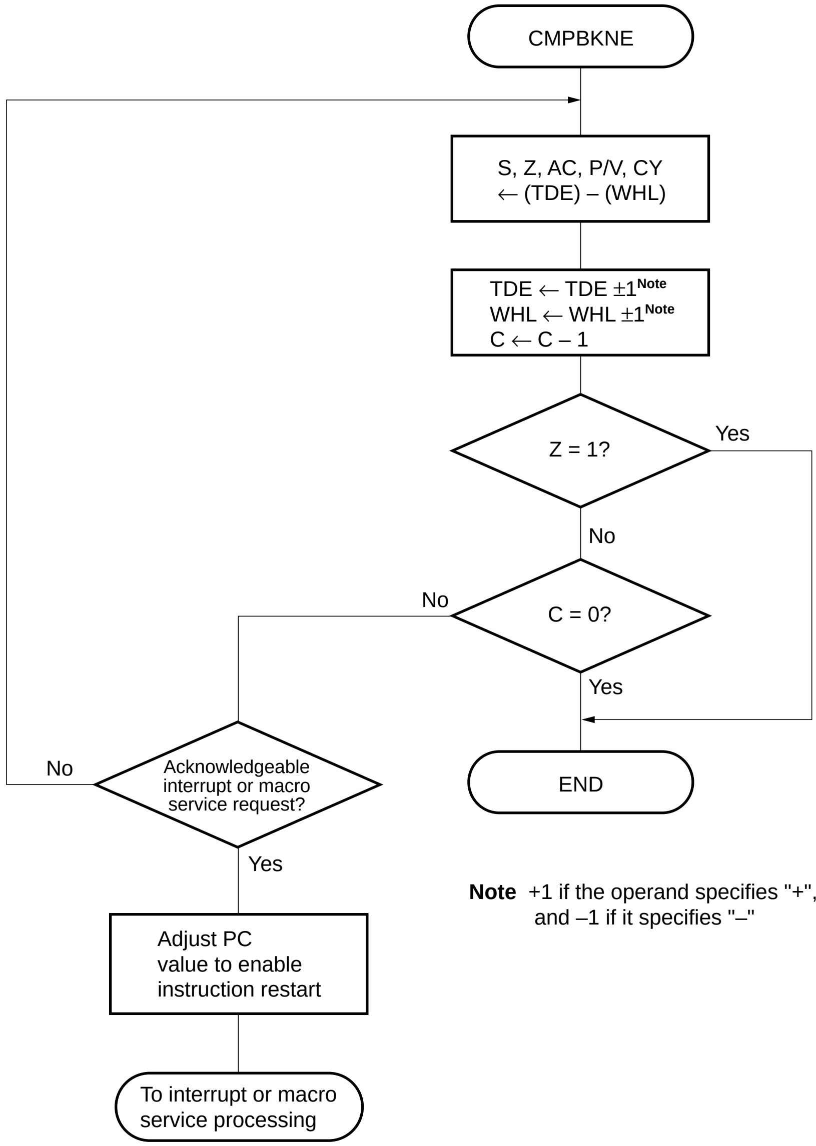
Mnemonic	Operands
CMPBKNE	[TDE +], [WHL +]
	[TDE –], [WHL –]

[Flags]

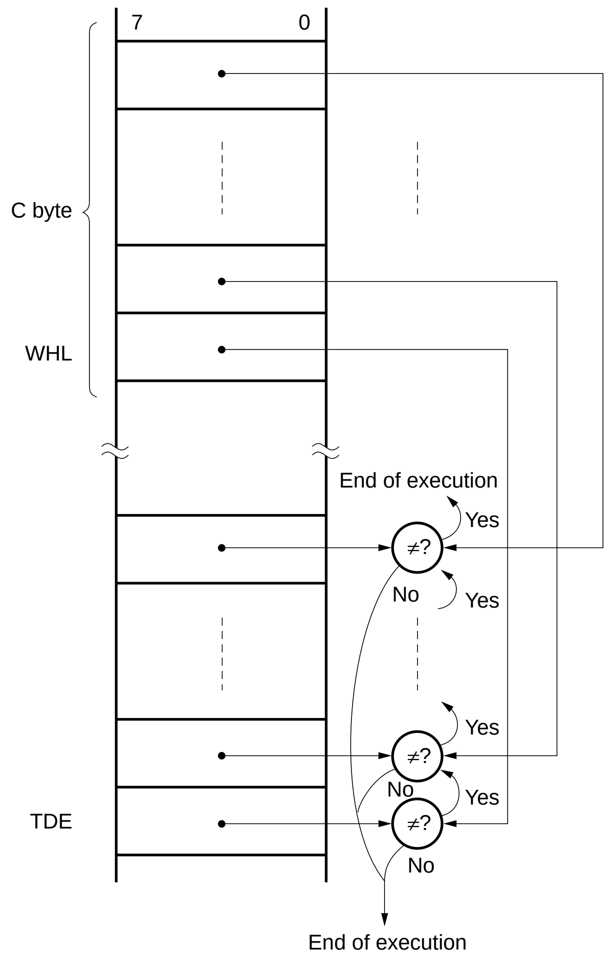
S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The contents of the memory addressed by the WHL register are compared with the contents of the memory addressed by the TDE register, the contents of the TDE and WHL registers are incremented/decremented, and the contents of the C register are decremented. The above operations are repeated until the result of the comparison is a match, or the contents of the C register are 0.
- Execution of this instruction does not change the contents of the memory addressed by the TDE and WHL registers.
- If an acknowledgeable interrupt or macro service request is generated during execution of this instruction, execution of this instruction is interrupted and the interrupt or macro service request is acknowledged. When an interrupt is acknowledged, if the return address and the contents of the TDE, WHL, and C registers used by this instruction which have been saved to the stack or to RP2 and R7 are not changed, execution of the interrupted instruction is resumed upon returning from the interrupt. When a macro service request is acknowledged, execution of this instruction is resumed after completion of the macro service.
- The S, Z, AC, P/V, and CY flags are changed in accordance with the last compare operation (subtraction) executed by this instruction.
- The S flag is set (1) if bit 7 is set (1) as a result of the subtraction, and cleared (0) otherwise.
- The Z flag is set (1) if the result of the subtraction is 0, and z flag is cleared (0) otherwise.
- The AC flag is set (1) if a borrow is generated out of bit 4 into bit 3 as a result of the subtraction, and cleared (0) otherwise.
- The P/V flag is set (1) if a borrow is generated in bit 6 and a borrow is not generated in bit 7 as a result of the subtraction (when underflow is generated by a two's complement type operation), or if a borrow is not generated in bit 6 and a borrow is generated in bit 7 (when overflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a borrow is generated in bit 7 as a result of the subtraction, and cleared (0) otherwise.



Note +1 if the operand specifies "+", and –1 if it specifies "–"



[Coding example]

MOV C, #5H
MOVG TDE, #0FFC50H
MOVG WHL, #0FC50H
CMPBKNE [TDE +], [WHL +]
BE $FIND ; Compares the 5-byte data from address 0FFC50H with the data from address 0FC50H, and branches to address FIND if there is matching data

CMPBKC

Compare Block Carry Byte
Block Comparison with Byte Data (Size Detection)

[Instruction format] CMPBKC [TDE +], [WHL +]
CMPBKC [TDE –], [WHL –]

[Operation] (TDE) – (WHL), TDE ← TDE + 1, WHL ← WHL + 1, C ← C – 1
End if C = 0 or CY = 0
(TDE) – (WHL), TDE ← TDE – 1, WHL ← WHL – 1, C ← C – 1
End if C = 0 or CY = 0

[Operands]

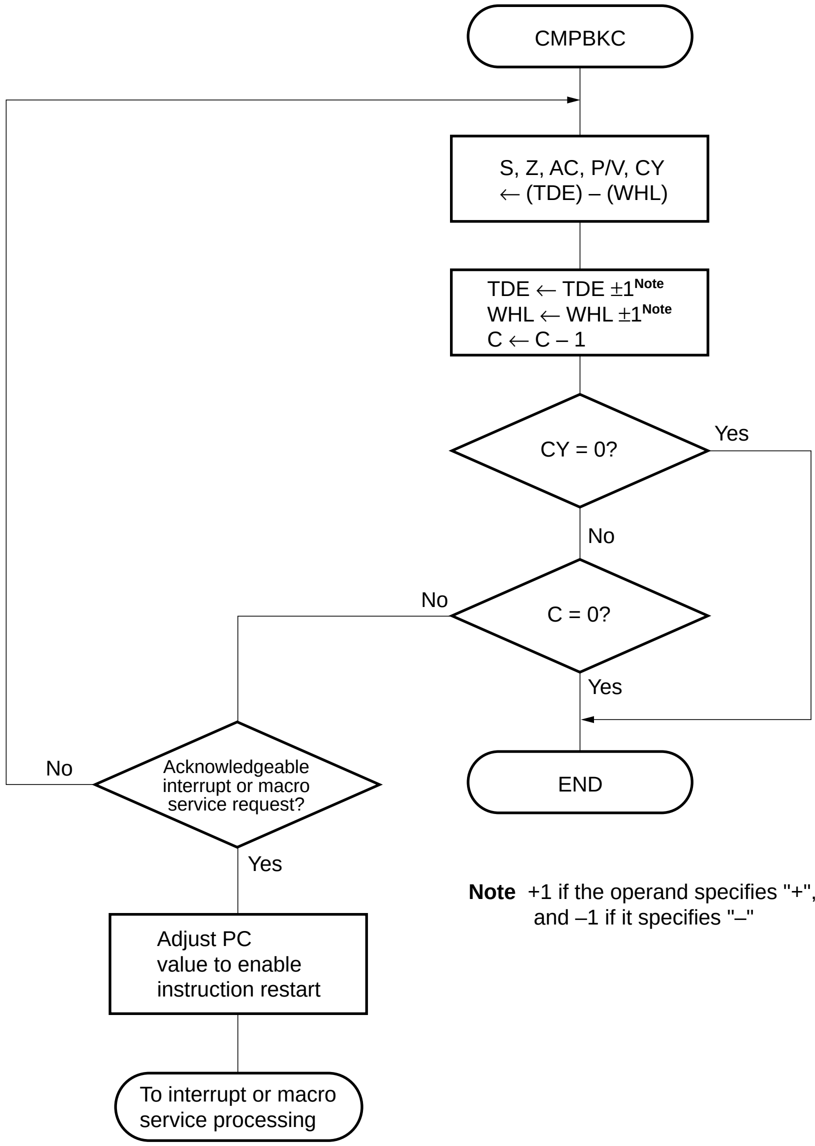
Mnemonic	Operands
CMPBKC	[TDE +], [WHL +]
	[TDE –], [WHL –]

[Flags]

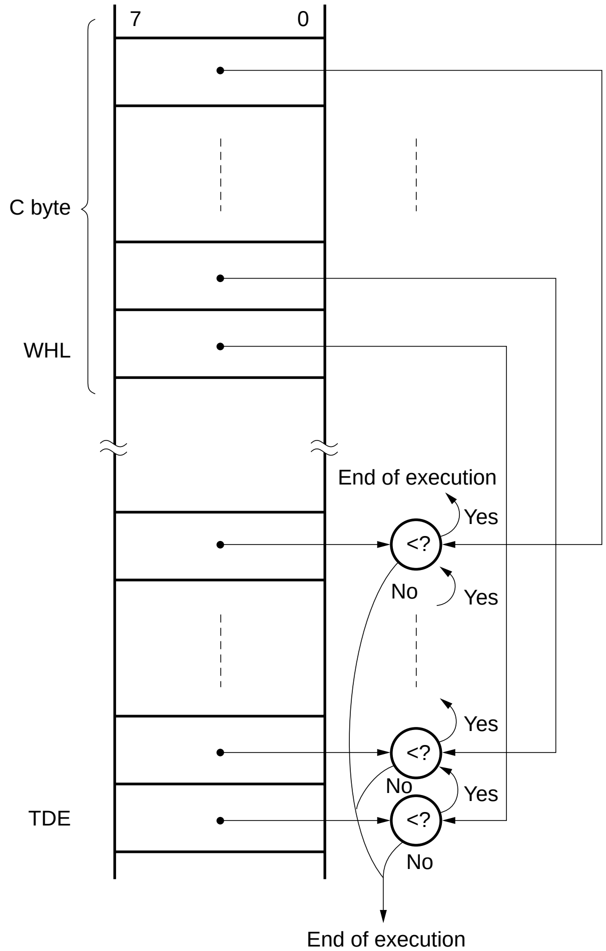
S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The contents of the memory addressed by the WHL register are compared with the contents of the memory addressed by the TDE register, the contents of the TDE and WHL registers are incremented/decremented, and the contents of the C register are decremented. The above operations are repeated until the result of the comparison is that the contents of the memory addressed by the TDE register are equal to or greater than the contents of the memory addressed by the WHL register, or the contents of the C register are 0.
- Execution of this instruction does not change the contents of the memory addressed by the TDE and WHL registers.
- If an acknowledgeable interrupt or macro service request is generated during execution of this instruction, execution of this instruction is interrupted and the interrupt or macro service request is acknowledged. When an interrupt is acknowledged, if the return address and the contents of the TDE, WHL, and C registers used by this instruction which have been saved to the stack or to RP2 and R7 are not changed, execution of the interrupted instruction is resumed upon returning from the interrupt. When a macro service request is acknowledged, execution of this instruction is resumed after completion of the macro service.
- The S, Z, AC, P/V, and CY flags are changed in accordance with the last compare operation (subtraction) executed by this instruction.
- The S flag is set (1) if bit 7 is set (1) as a result of the subtraction, and cleared (0) otherwise.
- The Z flag is set (1) if the result of the subtraction is 0, and z flag is cleared (0) otherwise.
- The AC flag is set (1) if a borrow is generated out of bit 4 into bit 3 as a result of the subtraction, and cleared (0) otherwise.
- The P/V flag is set (1) if a borrow is generated in bit 6 and a borrow is not generated in bit 7 as a result of the subtraction (when underflow is generated by a two's complement type operation), or if a borrow is not generated in bit 6 and a borrow is generated in bit 7 (when overflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a borrow is generated in bit 7 as a result of the subtraction, and cleared (0) otherwise.



Note +1 if the operand specifies "+", and –1 if it specifies "–"



[Coding example]

MOV C, #3H

MOVG TDE, #0E8762H

MOVG WHL, #03502H

CMPBKC [TDE –], [WHL –]

BNC $BIG ; Compares the 3-byte data from address 0E8760H with the 3-byte data from address 03500H, and branches to address BIG if the result of the comparison is that the values are the same or the 3-byte data from address 0E8760H is greater

CMPBKNC

Compare Block Not Carry Byte
Fixed Byte Data Block Comparison (Size Comparison)

[Instruction format] CMPBKNC [TDE +], [WHL +]
CMPBKNC [TDE –], [WHL –]

[Operation] (TDE) – (WHL), TDE ← TDE + 1, WHL ← WHL + 1, C ← C – 1
End if C = 0 or CY = 1
(TDE) – (WHL), TDE ← TDE – 1, WHL ← WHL – 1, C ← C – 1
End if C = 0 or CY = 1

[Operands]

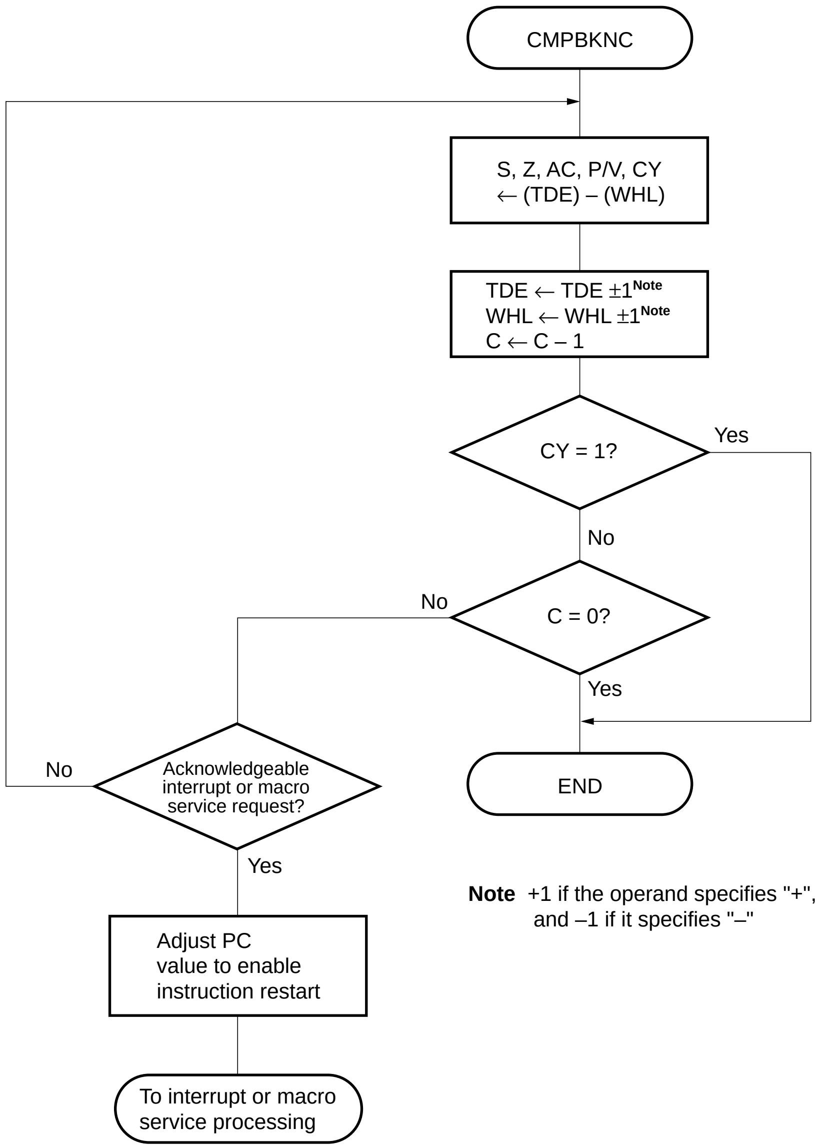
Mnemonic	Operands
CMPBKNC	[TDE +], [WHL +]
	[TDE –], [WHL –]

[Flags]

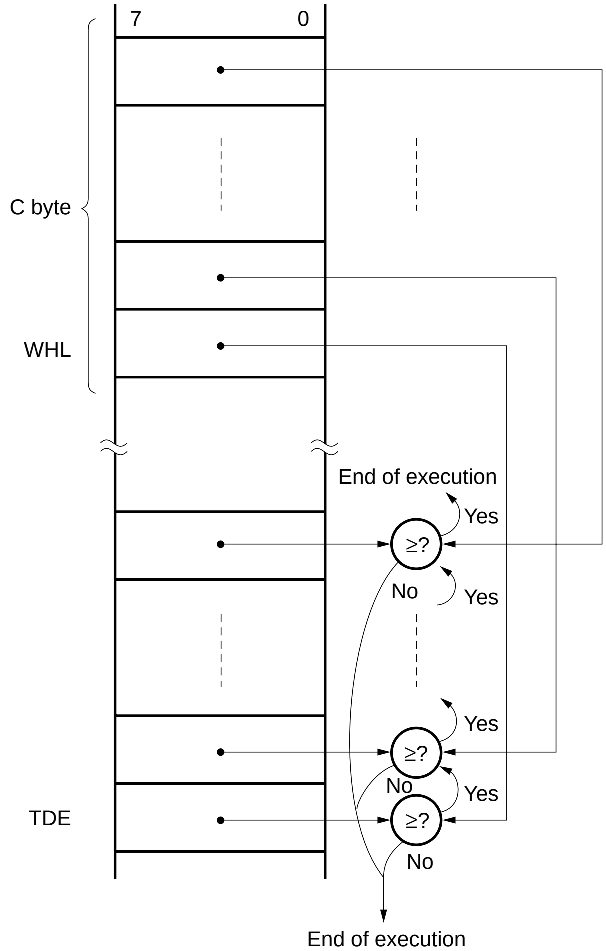
S	Z	AC	P/V	CY
×	×	×	V	×

[Description]

- The contents of the memory addressed by the WHL register are compared with the contents of the memory addressed by the TDE register, the contents of the TDE and WHL registers are incremented/decremented, and the contents of the C register are decremented. The above operations are repeated until the result of the comparison is that the contents of the memory addressed by the WHL register are greater, or the contents of the C register are 0.
- Execution of this instruction does not change the contents of the memory addressed by the TDE and WHL registers.
- If an acknowledgeable interrupt or macro service request is generated during execution of this instruction, execution of this instruction is interrupted and the interrupt or macro service request is acknowledged. When an interrupt is acknowledged, if the return address and the contents of the TDE, WHL, and C registers used by this instruction which have been saved to the stack or to RP2 and R7 are not changed, execution of the interrupted instruction is resumed upon returning from the interrupt. When a macro service request is acknowledged, execution of this instruction is resumed after completion of the macro service.
- The S, Z, AC, P/V, and CY flags are changed in accordance with the last compare operation (subtraction) executed by this instruction.
- The S flag is set (1) if bit 7 is set (1) as a result of the subtraction, and cleared (0) otherwise.
- The Z flag is set (1) if the result of the subtraction is 0, and z flag is cleared (0) otherwise.
- The AC flag is set (1) if a borrow is generated out of bit 4 into bit 3 as a result of the subtraction, and cleared (0) otherwise.
- The P/V flag is set (1) if a borrow is generated in bit 6 and a borrow is not generated in bit 7 as a result of the subtraction (when underflow is generated by a two's complement type operation), or if a borrow is not generated in bit 6 and a borrow is generated in bit 7 (when overflow is generated by a two's complement type operation), and is cleared (0) otherwise.
- The CY flag is set (1) if a borrow is generated in bit 7 as a result of the subtraction, and cleared (0) otherwise.



Note +1 if the operand specifies "+", and –1 if it specifies "–"



[Coding example]

MOV C, #4H

MOVG TDE, #05503H

MOVG WHL, #0FFC03H

CMPBKNC [TDE –], [WHL –]

BC $LITTLE ; Compares the 4-byte data from address 05500H with the data from address 0FFC00H, and branches to address LITTLE if the data from address 05500H is smaller

[MEMO]

★

CHAPTER 8 DEVELOPMENT TOOLS

Tools required for 78K/IV Series product development are shown in this chapter.

For details, refer to the **User's Manual — Hardware** of each device and the **Single-Chip Microcontroller Development Tool Selection Guide (U11069E)**.

8.1 Development Tools

The following development tools are provided to develop programs for application systems

Table 8-1. Types and Functions of Development Tools (1/2)

Development tools		Functions
Hardware	In-circuit emulator (IE-784000-R) (IE-78K4-NS)	This is a hardware tool used for program debugging for system development of 78K/IV Series. When a personal computer (PC-9800 Series or IBM PC/(IE-78K4-NS) AT™) is used as a host machine of this emulator, it is possible to perform more efficient debugging by means of functions such as the symbolic debugging and the object file and symbol file transfer. An on-chip serial interface RS-232-C enables connection to a PROM programmer (PG-1500).
	Emulation board (IE-78×××-R-EM) (IE-78×××-R-EM-A) (IE-78××××-R-EM)	This is a board to emulate peripheral hardware that is specific to the target device.
	I/O emulation board (IE-78×××-R-EM1) (IE-78××××-R-EM1) (IE-78××××-NS-EM1)	This is a board to emulate peripheral hardware that is specific to the target device. It is used in combination with an emulation board. The I/O emulation board required for the target device depends on the products.
	Emulation probe (EP-78×××......) (NP-×××......)	This probe connects the in-circuit emulator to the target device. It is provided each target device package.
	Conversion socket (EV-9200××-××)	This is a socket used to connect the emulation probe for QFP to the application system. It is a standard accessory for an emulation probe for QPF. Mount it on the circuit board for an application system.
	Programmer adapter (PA-78P×××......)	This is an adapter for the PROM programmer (PG-1500) that is used for programming on-chip PROM products.
Jig (EV-9900)		Jig for removing WQFP-package product from the EV-9200××-××.

Table 8-1. Types and Functions of Development Tools (2/2)

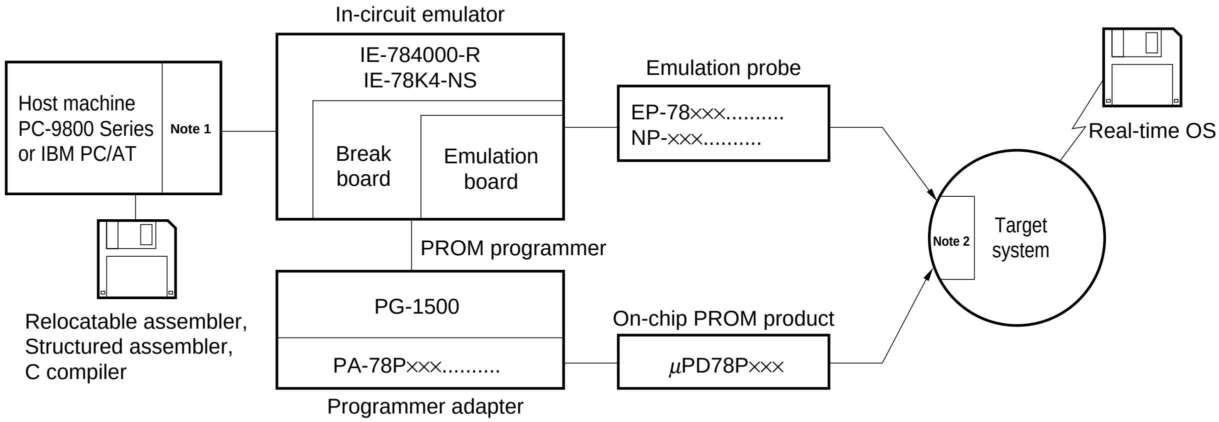
Development tools		Functions
Software	Integrated debugger (ID78K4)	This is a control program for in-circuit emulators for the 78K/IV Series. This debugger is used in combination with the device files. This debugger enables more effective debugging than previous IE controllers by offering the following features: source program level debugging written in C language, structured assembly language, or assembly language and display for a variety of simultaneous a variety of information by dividing the screen of the host machine.
	Device file	Used in combination with an integrated debugger. This file is required when debugging the 78K/IV Series.
	In-circuit emulator control program (IE controller)	This is software to perform efficient debugging by connecting the IE to the host machine. This software makes full use of the capabilities of the IE by means of file (object or symbol) transfer, on-line assembly, disassembly, break condition (event) setup, etc.
	Relocatable assembler**Note 1**	This is a program to convert a program written in mnemonic to an object code that can be executed by microcontrollers. In addition, an automatic function to perform a symbol table creation and branch instruction optimization processing is provided.
	Structured assembler preprocessor	This software introduces a structured programming method into the assembler. It enables writing functions with a C language-like control structure without sacrificing the size and speed of the assembler.
	C compiler**Note 1**	This is a program that translates a program written in the high-level C language into object code, which can be executed by a microcontroller.
	C library source	This source program is attached to the C compiler. This program is required when modifying a library (To better match user specifications).
	System simulator (SM78K4)**Note 2**	This is a software development support tool. C source level or assembler level debugging is possible while simulating the operation of the target system in the host machine. The SM78K4 enables the verification of the logic and performance of applications independently from the hardware development. Consequently, development efficiency and software quality can be improved.

Notes 1. Used in combination with the device files for 78K/IV Series.
2. Used in combination with the device files.

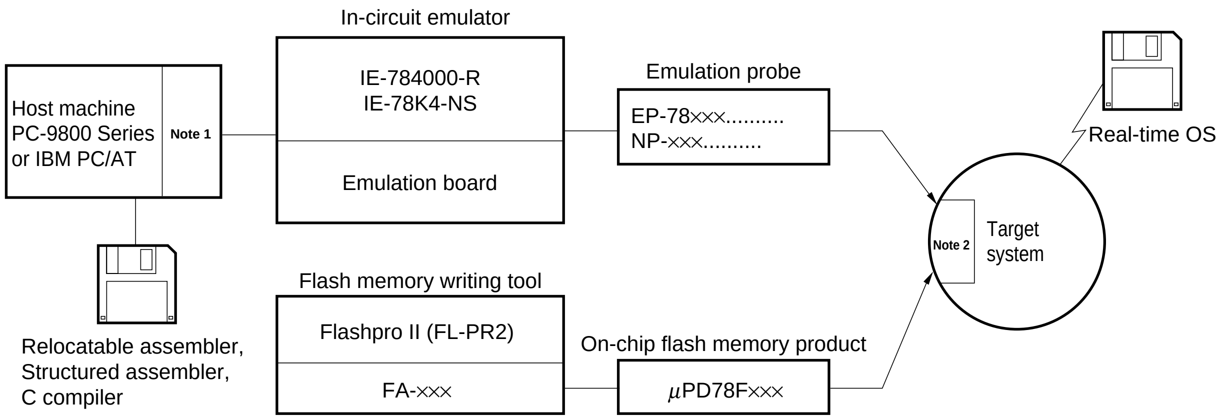
Remark All the software listed above runs under MS-DOS™ and PC DOS™.

Figure 8-1. Development Tools Structure

(On-chip PROM)



(On-chip Flash Memory)



Notes 1. Integrated debugger and device file

2. Conversion socket to connect emulation probe to the target system (Products whose prefix is EV-9200)

Remark The meaning of part number prefix are as follows.

IE : In-circuit emulator
EP : Emulation probe
NP : Emulation probe (Made by Naitou Densei Machidaseisakusho, Co., Ltd.)
PA : PROM programmer adapter
FA : Adapter for flash memory writing
×××...... : Varies depending on the target device or package.

8.2 PROM Programming Tools

(1) Hardware

PG-1500	PROM programmer which allows programming, in standalone mode or via operation from a host machine, of a single-chip microcontroller with on-chip PROM by connection of the board provided and a separately available PROM programmer adapter. It can also program typical 256-Kbit to 4-Mbit PROM.
PROM programmer adapter	Adapter which provided for each product with on-chip PROM. This adapter is used in combination with PROM programmer. For actual product names, refer to the **User's Manual — Hardware** for the relevant device.

(2) Software

PG-1500 controller	Controls the PG-1500 on the host machine by connecting PG-1500 to the host machine with parallel and serial interface.

8.3 Flash Memory Programming Tools

Flashpro II (FL-PR2)	This is flash programmer for a microcontroller in the flash memory.
Adapter for flash memory programming	This must be wired to match the objective product. For details about part names, refer to the hardware version of the user's manual for each device.

Remark This is a product of Naitou Densei Machidaseisakusho Co., Ltd. Consult with an NEC representative before buying this part.

9.1 Real-time OS

RX78K/IV Note Real-time OS	The aim of the RX78K/IV is to realize multi-task environments for real-time required control fields. The CPU idle time can be allotted to other processes to improve the overall performance of the system. The RX78K/IV provides system calls (31 kinds) conforming to the μITRON specification, and the tools (configurator) for creating the RX78K/IV nucleus and several information tables. The RX78K/IV should be used in combination with separately available assembler package (RA78K4) and device files. ★ **<Precaution when used under PC environment>** Real-time OS is a DOS-based application. When using this application on Windows, use the DOS prompt.
MX78K4 OS	μITRON specification subset OS. Nucleus of MX78K4 is attached. Task, event, and time management are performed. Task execution sequence is controlled in task management and with subsequent switch to the next execution task. ★ **<Precaution when used under PC environment>** MX78K4 is a DOS-based application. When using this application on Windows, use the DOS prompt.

Note When purchasing the RX78K/IV, the purchasing application must be filled in advance and a using conditions agreement signed.

[MEMO]

APPENDIX A INDEX OF INSTRUCTIONS (MNEMONICS: BY FUNCTION)

[Bit Manipulation Instructions]

[Stack Manipulation Instructions]

[Call/Return Instructions]

[Unconditional Branch Instruction]

[Conditional Branch Instructions]

[CPU Control Instructions]

[Special Instructions]

[String Instructions]

[MEMO]

APPENDIX B INDEX OF INSTRUCTIONS (MNEMONICS: ALPHABETICAL ORDER)

[L]

[M]

[N]

[O]

[P]

[R]

[S]

[X]

APPENDIX C REVISION HISTORY

Revisions through this document are listed in the following table. The column "Applicable Chapters" indicates the chapters in each edition.

(1/3)

Edition	Major Revisions from Previous Edition	Applicable Chapters
2nd Edition	• The following instructions are added to bit manipulation instructions. MOV1 CY, !addr16.bit CY, !!addr24, bit !addr16.bit, CY !!addr24.bit, CY AND1, OR1 CY, !addr16.bit CY, !!addr24.bit CY,/!addr16.bit CY,/!!addr24.bit XOR1 CY, !addr16.bit CY,!!addr24.bit NOT1, SET1, CLR1 !addr16.bit !!addr24.bit • The following instructions are added to conditional branch instructions. BF, BT, BFSET, BTCLR !addr16.bit, $addr20 !!addr24.bit, $addr20	CHAPTER 6 INSTRUCTION SET
3rd Edition	• Descriptions regarding μPD784915 Subseries are added. • μPD784020 is added to μPD784026 Subseries.	Throughout
	Notation used in section **5.2.10 Short direct 24-bit memory indirect addressing** changed as follows: [%saddrp] → [%saddrg]	CHAPTER 5 ADDRESSING
	• saddrg1 and saddrg2 are added to section 6.1 Legend, (1) Operand Identifiers and Description (2/2). • MOVG operand corrected as follows: [TDE+HL], WHL → [TDE+C], WHL • Section **6.5 Number of Instruction Clocks** is added	CHAPTER 6 INSTRUCTION SET
	• 3.5-inch 2HC or 3.5-inch 2HD is added as supply medium for IBM PC/AT • Part numbers for ordering integrated debuggers are changed as follows: μS5A10ID78K4 → μSAA10ID78K4 μS5A13ID78K4 → μSAA13ID78K4 μS7B10ID78K4 → μSBB10ID78K4	CHAPTER 8 DEVELOPMENT TOOLS

(2/3)

Edition	Major Revisions from Previous Edition	Applicable Chapters
4th Edition	• GK Package (80-pin plastic TQFP, fine pitch, 12 mm × 12 mm) is added to μPD784021. • Descriptions regarding μPD784038/784038Y Subseries are added. • Descriptions regarding μPD784046 Subseries are added. • Descriptions regarding μPD784208/784208Y Subseries are added. • A "Note" mark is appended to the RETCS instruction, which indicates that the μPD784208 and 784208Y Subseries do not have the RETCS instruction.	Throughout
	• Descriptions regarding flash memory are added.	CHAPTER 8 DEVELOPMENT TOOLS
	• Descriptions regarding the MX78K4 are added.	CHAPTER 9 SOFTWARE FOR EMBEDDING
5th Edition	• New products (μPD784031/Y) and new package (80-pin plastic QFP (14 mm square, 1.4 mm thick)) have been added to the μPD784038/Y Subseries. • Entries related to the new product (μPD784054) of the μPD784046 Subseries have been added. • Entries related to the μPD784208 Subseries have been deleted. • Entries related to the μPD784216/Y Subseries have been added. • Entries related to the new products (μPD784915A, 784916A) of the μPD784915 Subseries have been added. • Entries related to the μPD784908 Subseries have been added. • Entries related to the μPD78F4943 Subseries have been added.	Throughout
	• Note that there is no RETCS instruction in the μPD764208 and μPD784208Y Subseries has been deleted.	CHAPTER 6 INSTRUCTION SET
	• The entry, 'Highest-order/On Highest-order side' for RETI instructions has been changed to 'Highest Priority.' • Note that there is no RETCS instruction in the μPD764208 and μPD784208Y Subseries has been deleted, 'target' has been added to the instruction format, and the entry, 'Highest-order/On Highest-order side' has been changed to 'Highest Priority.' • 'target' has been added to the instruction format for RETCSB instructions.	CHAPTER 7 DESCRIPTION OF INSTRUCTIONS
	• Entries related to flash memory have been corrected.	CHAPTER 8 DEVELOPMENT TOOLS

(3/3)

Edition	Major Revisions from Previous Edition	Applicable Chapters
6th Edition	• Adds the μPD784218, 784218Y, 784225, 784225Y, 784928, and 784928Y Subseries and μPD784943. • The following products are in the development to completion stage: μPD784037, 784038, 78P4038 μPD784031Y, 784035Y, 784036Y, 784037Y, 784038Y, 78P4038Y μPD784215, 784216 μPD784215Y, 784216Y μPD784915A, 784916A • Changes the GC-7EA package to the GC-8EU package in the μPD784214, 784215, 784216, 784214Y, 784215Y, and 784216Y. • Describes that the μPD784915 Subseries provide the fixed LOCATION 0 instruction instead of the LOCATION 0FH instruction. • Adds Note describing that the special instructions (CHKL and CHKLA) are not available for the μPD784216, 784216Y, 784218, 784218Y, 784225, and 784215Y, • Changes the μPD78F4943 Subseries to the μPD784943 Subseries.	Throughout
7th Edition	• Addition of μPD784937 and 784955 Subseries. Deletion of μPD784943. • The following products changed from under development stage to completed. μPD784031(A), 784035(A), 784036(A), μPD784044(A), 784044(A1), 784044(A2), 784046(A), 784046(A1), 784046(A2), μPD784054(A), 784054(A1), 784054(A2), μPD784214, 784214Y, μPD784915B, 784916B, μPD784927, 78F4928, 784927Y, 78F4928Y • Modification of GC-7EA package to GC-8EU package for the μPD78F4216, 78F4216Y • Modification of power supply voltage in the μPD784908 Subseries. Mask ROM version (μPD784907, 784908) ... changed from (V_{DD} = 4.5 to 5.5 V) to (V_{DD} = 3.5 to 5.5 V) PROM version (μPD78P4908) ... changed from (V_{DD} = 4.5 to 5.5 V) to (V_{DD} = 4.0 to 5.5 V)	Throughout
	Modification of the Notes in the special instructions (CHKL, CHKLA).	CHAPTER 6 INSTRUCTION SET
	Modification of the operation sequence of the POP instruction. Addition of the Note in CHKL instruction. Addition of Note in CHKLA instruction.	CHAPTER 7 DESCRIPTION OF INSTRUCTIONS
	Modification of the format.	CHAPTER 8 DEVELOPMENT TOOLS
	Addition of the description on the PC environment.	CHAPTER 9 EMBEDDED SOFTWARE

[MEMO]

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288
1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6465-6829

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

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

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: 044-548-7900

I would like to report the following error/make the following suggestion:

Document title: ______________________________

Document number: ______________________ Page number: ____________

____

____

____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❏	❏	❏	❏
Technical Accuracy	❏	❏	❏	❏
Organization	❏	❏	❏	❏

CS 98.2