User's Manual

16

RL78 family

User's Manual: Software

16-Bit Single-Chip Microcontrollers

Renesas Electronics
www.renesas.com

Rev.1.00 Jan 2011

Notice

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

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

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

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

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

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

NOTES FOR CMOS DEVICES

(1) VOLTAGE APPLICATION WAVEFORM AT INPUT PIN: Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between VIL (MAX) and VIH (MIN) due to noise, etc., the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between VIL (MAX) and VIH (MIN).

(2) HANDLING OF UNUSED INPUT PINS: Unconnected CMOS device inputs can be cause of malfunction. If an input pin is unconnected, it is possible that an internal input level may be generated due to noise, etc., causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using pull-up or pull-down circuitry. Each unused pin should be connected to VDD or GND via a resistor if there is a possibility that it will be an output pin. All handling related to unused pins must be judged separately for each device and according to related specifications governing the device.

(3) PRECAUTION AGAINST ESD: A 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 when it has occurred. Environmental control must be adequate. When it is dry, a humidifier should be used. It is recommended to avoid using insulators that easily build up 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 benches and floors should be grounded. The operator should be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with mounted semiconductor devices.

(4) STATUS BEFORE INITIALIZATION: Power-on does not necessarily define the initial status of a MOS device. Immediately after the power source is turned ON, devices with reset functions have not yet been initialized. Hence, power-on does not guarantee output pin levels, I/O settings or contents of registers. A device is not initialized until the reset signal is received. A reset operation must be executed immediately after power-on for devices with reset functions.

(5) POWER ON/OFF SEQUENCE: In the case of a device that uses different power supplies for the internal operation and external interface, as a rule, switch on the external power supply after switching on the internal power supply. When switching the power supply off, as a rule, switch off the external power supply and then the internal power supply. Use of the reverse power on/off sequences may result in the application of an overvoltage to the internal elements of the device, causing malfunction and degradation of internal elements due to the passage of an abnormal current. The correct power on/off sequence must be judged separately for each device and according to related specifications governing the device.

(6) INPUT OF SIGNAL DURING POWER OFF STATE : Do not input signals or an I/O pull-up power supply while the device is not powered. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Input of signals during the power off state must be judged separately for each device and according to related specifications governing the device.

How to Use This Manual

Target Readers

This manual is intended for users who wish to understand the functions of RL78 microcontrollers and to design and develop its application systems and programs.

Purpose

This manual is intended to give users an understanding of the various kinds of instruction functions of RL78 microcontrollers.

Organization

This manual is broadly divided into the following sections.

- CPU functions
- Instruction set
- Explanation of instructions

How to Read This Manual

It is assumed that readers of this manual have general knowledge in the fields of electrical engineering, logic circuits, and microcontrollers.

- To check the details of the functions of an instruction whose mnemonic is known:
  →Refer to **APPENDICES A** and **B**.

- To check an instruction whose mnemonic is not known but whose general function is known:
  →Find the mnemonic in **CHAPTER 5 INSTRUCTION SET** and then check the detailed functions in **CHAPTER 6 EXPLANATION OF INSTRUCTIONS**.

- To learn about the various kinds of RL78 microcontroller instructions in general:
  →Read this manual in the order of **CONTENTS**.

- To learn about the hardware functions of RL78 microcontrollers:
  →See the user's manual for each microcontroller.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numeric representation:	Binary ……XXXX or XXXXB
	Decimal ……..XXXX
	Hexadecimal …. XXXXH

CONTENTS

CHAPTER 1 OVERVIEW

1.1 Differences from 78K0 Microcontrollers (for Assembler Users)

(1) Use of pipeline processing reduces the number of processing clock cycles for all instructions. Existing programs must be re-evaluated.

(2) All instruction code maps have been modified. Reassemble them using the assembler. When reassembling, the code size is likely to increase as new instructions are added, but in some cases the overall code size may shrink if old instructions are replaced with new ones.

(3) The memory space was changed from 64 KB to 1 MB, and the total stack area was also increased. Within the assembler's program, the address must be changed whenever RAM contents within the stack pointer are manipulated. The stack size should be increased slightly to accommodate the depth of multiple CALLs or multiple interrupts.

(4) The CALLT table's address range has been changed from "0040H to 007FH" to "0080H to 00BFH". Consequently, the CALLT table's address should be changed.

(5) Among the programs used for the 78K0 microcontroller's bank switching, the assembler program must be rewritten.

(6) Address changes are made when using the expansion RAM. Be sure to change these addresses.

(7) If instructions are executed from expansion RAM, since memory space addresses have been changed, change BR !addr16 to BR !!addr20, and CALL !addr16 to CALL !!addr20.

(8) There are no IMS or IXS registers (registers used to set memory space). The programs that use these registers should be deleted if external memory is not being used. If external memory is being used, the specifications for the MM/MEM register (external memory setting register) have been changed, so check the user's manual for each product and change the settings accordingly.

(9) The following instructions are deleted and the alternative code is output, resulting in code size increases. Even when these instructions are used, they are automatically replaced during assembly.

Instruction	Operand	Remarks
DIVUW	C	The alternative instruction executes a division with shifting, so the execution time is longer than DIVUW. It is recommended to change this instruction to the added shift instruction.
ROR4	[HL]	The execution time of the alternative instruction is longer than ROR4. It is recommended to change this instruction to the added shift instruction.
ROL4	[HL]	The execution time of the alternative instruction is longer than ROL4. It is recommended to change this instruction to the added shift instruction.
ADJBA	None	The execution time of the alternative instruction is longer than ADJBA. No instruction is added for substitution.
ADJBS	None	The execution time of the alternative instruction is longer than ADJBS. No instruction is added for substitution.
CALLF	!addr11	CALLF is automatically changed to a 3-byte instruction CALL !addr16. This can be used without modification.
DBNZ	B, $addr16 C, $addr16 saddr, $addr16	This instruction is divided into two: DEC B/DEC C/DEC saddr and BNZ $addr20. These can be used without modification.

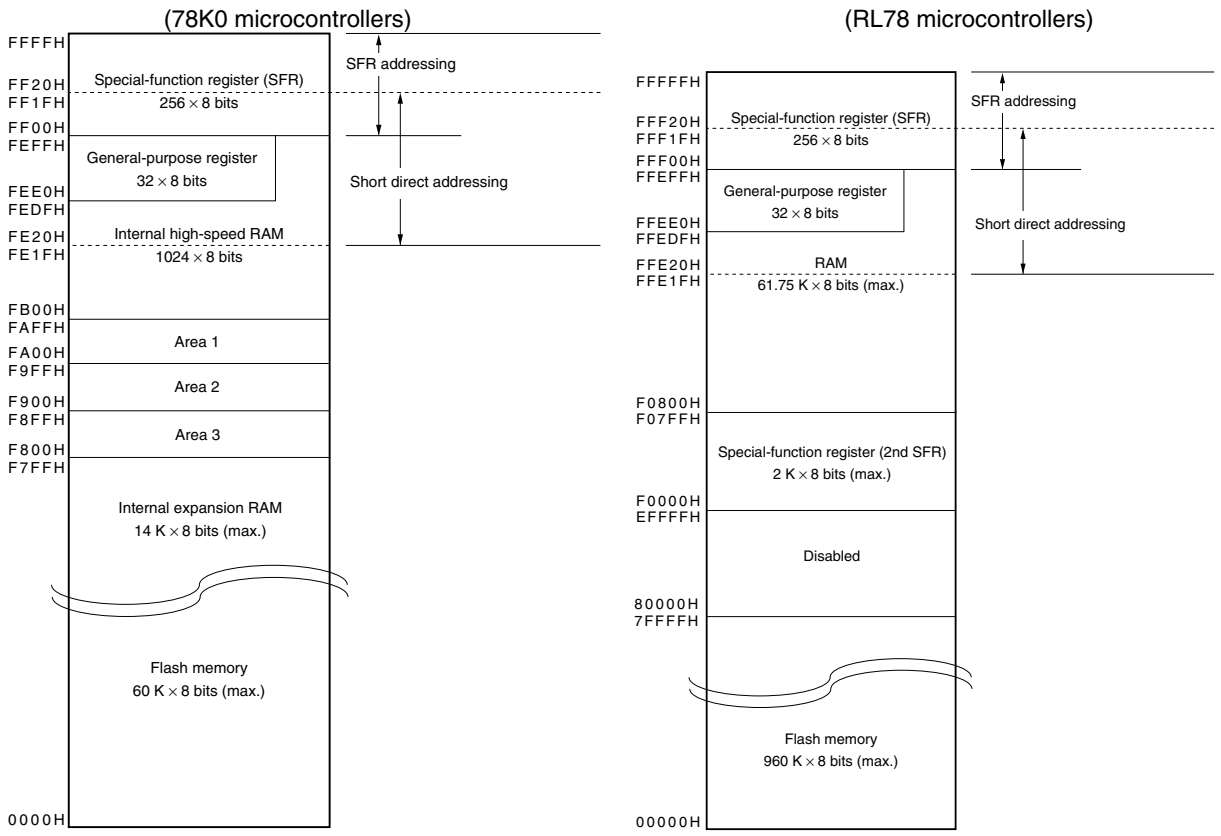
(10) Memory space has changed from 64KB to 1MB, and addressing by using ES register and addressing of word [BC], etc. are added. Be sure not to assign any address over maximum memory space. Especially in based addressing and based indexed addressing, an added value must not exceed FFFFH (without ES register) or FFFFFH (with ES register).

CHAPTER 2 MEMORY SPACE

2.1 Memory Space

While the 78K0 microcontroller's memory space is only 64 KB, this has been expanded to 1 MB in the RL78 microcontroller.

Figure 2-1. Memory Maps of 78K0 Microcontrollers and RL78 Microcontrollers



- Program memory space is 60 KB (max.).
- Internal high-speed RAM area is 1 KB (max.) (stack enabled).
  Internal expansion RAM area is 14 KB (max.) (fetch enabled).
- Area 1, area 2, and area 3 are from F800H to FAFFH (fixed).

- Program memory space is 960 KB (max.).
- RAM space is 61.75 KB (max.) (stack enabled, fetch enabled).
- Second SFR area (name changed) is 2 KB (max.), from F0000H to F07FFH.

2.2 Internal Program Memory Space

In the RL78 microcontrollers, the program memory space's address range is from 00000H to EFFFFH.
For description of the internal ROM (flash memory) maximum size, refer to the user's manual for each product.

Caution Do not use relative addressing in branch instructions from internal program memory space to RAM space or external memory space.

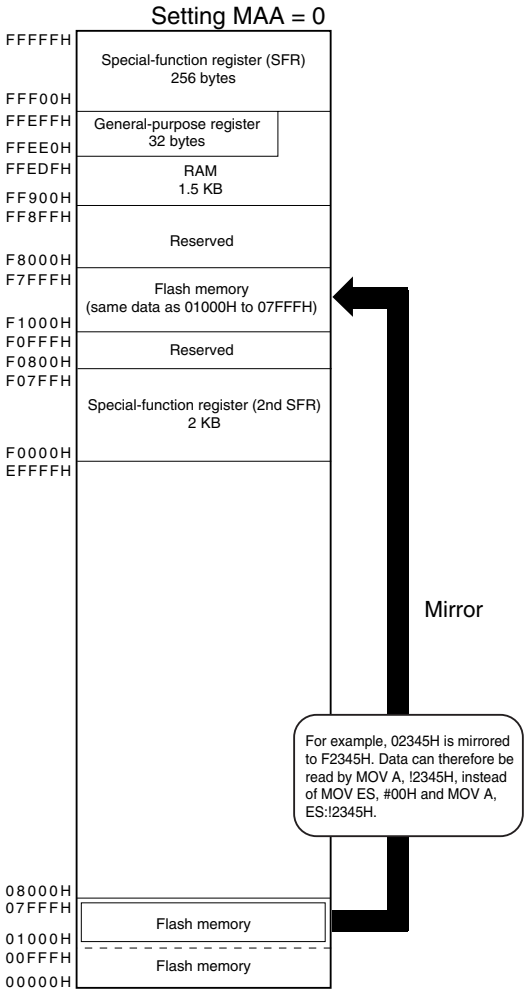
2.2.1 Mirror area

In the RL78 microcontrollers, the data flash areas from 00000H to 0FFFFH (when MAA = 0) and from 10000H to 1FFFFH (when MAA = 1) are mirrored to the addresses from F0000H to FFFFFH. By reading data from F0000H to FFFFFH, an instruction that does not have the ES registers as an operand can be used, and thus the contents of the data flash can be read with the shorter code. However, in this case the data flash area is not mirrored to the SFR, extended SFR (second SFR), RAM, and use prohibited areas.

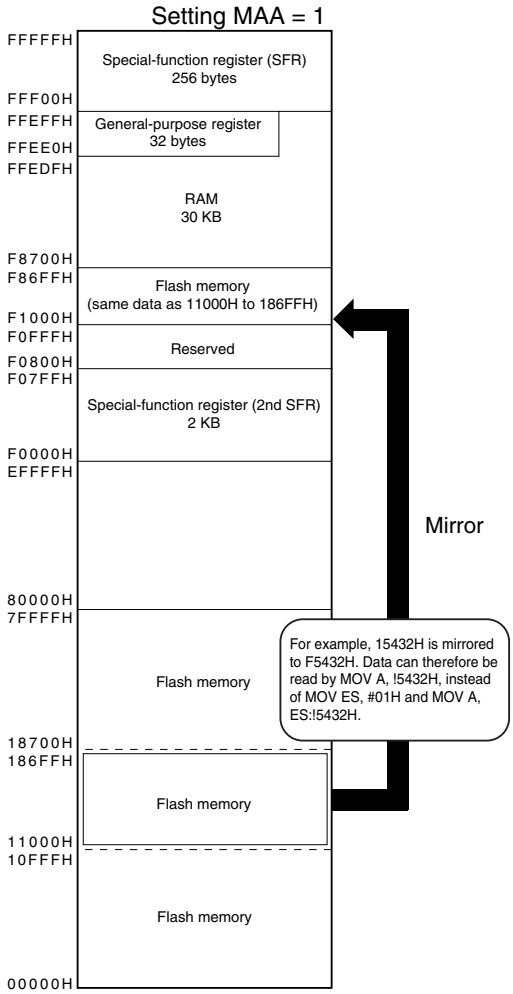
Mirror areas can only be read, and instruction fetch is not enabled.

The following show examples. Specifications vary for each product, so refer to the user's manual for each product.

Example 1 (Flash memory: 32 KB, RAM: 1.5 KB)



Example 2 (Flash memory: 512 KB, RAM: 30 KB)



Remark MAA: Bit 0 of the processor mode control register (PMC) (for details, refer to **3.4.1 Processor mode control register (PMC)**).

2.2.2 Vector table area

In the RL78 microcontrollers, the 128-byte area from 0000H to 007FH is reserved as the vector table area. The number of interrupts is calculated as 61 (maximum) + RESET vector + on-chip debugging vector + software break vector. Since there are only 2 bytes of vector code, the interrupt branch destination start address is 64 KB from 00000H to 0FFFFH. While in the 78K0 microcontrollers, addresses from 0040H to 007FH are used for the CALLT table, in the RL78 microcontrollers, these have been changed to vector addresses.

2.2.3 CALLT instruction table area

In the RL78 microcontrollers, the 64-byte area from 0080H to 00BFH is reserved as the CALLT instruction table area.

While single-byte CALL instructions are used in the 78K0 microcontrollers, the RL78 microcontrollers use 2-byte CALL instructions. Addresses have also been changed accordingly.

Since the address code is only 2 bytes long, the interrupt branch destination start address is 64 KB from 00000H to 0FFFFH.

2.3 Internal Data Memory (Internal RAM) Space

The 78K0 microcontrollers include internal high-speed RAM and internal expansion RAM, whereby the internal high-speed RAM is stack-enabled while the internal expansion RAM is fetch-enabled. By contrast, the RL78 microcontrollers have just one RAM area that enables both stack and fetch.

The higher limit of the address range is fixed to FFEFFH, and the range can be extended downward according to the product's mounted RAM size. The maximum size is 61.75 KB. For a description of the range's lower limit, refer to the user's manual for each product.

The saddr space and general-purpose register area (from FFEE0H to FFEFFH) have the same addresses in the 78K0 and RL78 microcontrollers.

Cautions1. Specify the address other than the general-purpose register area address as a stack area. It is prohibited to use the general-purpose register area for fetching instructions or as a stack area.

2. Do not use relative addressing in branch instructions from RAM space to internal program memory space or external memory space.

2.4 Special Function Register (SFR) Area

SFRs have specific functions, unlike general-purpose registers.

The SFR space is allocated to the area from FFF00H to FFFFFH.

SFRs can be manipulated like general-purpose registers, using operation, transfer, and bit manipulation instructions. The manipulable bit units, 1, 8, and 16, depend on the SFR type.

Each manipulation bit unit can be specified as follows.

- 1-bit manipulation
  Describe the symbol reserved by the assembler for the 1-bit manipulation instruction operand (sfr.bit). This manipulation can also be specified with an address.
- 8-bit manipulation
  Describe the symbol reserved by the assembler for the 8-bit manipulation instruction operand (sfr). This manipulation can also be specified with an address.
- 16-bit manipulation
  Describe the symbol reserved by the assembler for the 16-bit manipulation instruction operand (sfrp). When specifying an address, describe an even address.

Although the RL78 microcontrollers' SFR has the same specifications as in the 78K0 microcontrollers, some registers differ from the 78K0 in cases where addresses are fixed. Refer to the user's manual for each product for details.

2.5 Extended SFR (Second SFR) Area

Unlike a general-purpose register, each extended SFR (2nd SFR) has a special function.

Extended SFRs are allocated to the F0000H to F07FFH area. SFRs other than those in the SFR area (FFF00H to FFFFFH) are allocated to this area. An instruction that accesses the extended SFR area, however, is 1 byte longer than an instruction that accesses the SFR area.

Extended SFRs can be manipulated like general-purpose registers, using operation, transfer, and bit manipulation instructions. The manipulable bit units, 1, 8, and 16, depend on the SFR type.

Each manipulation bit unit can be specified as follows..

- 1-bit manipulation
  Describe the symbol reserved by the assembler for the 1-bit manipulation instruction operand (!addr16.bit). This manipulation can also be specified with an address.
- 8-bit manipulation
  Describe the symbol reserved by the assembler for the 8-bit manipulation instruction operand (!addr16). This manipulation can also be specified with an address.
- 16-bit manipulation
  Describe the symbol reserved by the assembler for the 16-bit manipulation instruction operand (!addr16). When specifying an address, describe an even address.

2.6 External Memory Space

The external memory space that can be accessed by setting the memory expansion mode register. This memory space is allocated from flash memory to EDFFFH.

As the external pins in separate mode, 28 pins (A19 to A0 and D7 to D0) are available. In multiplexed mode, 20 pins (A19 to A8 and AD7 to AD0) are available.

For pin settings when using external memory, refer to the chapter describing port functions in the user's manual for each product.

Cautions1. When fetching the instructions in an external memory area, start the execution by the branch instructions (CALL or BR) in flash memory or RAM memory areas and end the execution by return instructions (RET, RETB or RETI) in an external memory area.
While flash memory area is adjacent to an external memory area, serial program can not be executed.

2. Do not use relative addressing in branch instructions from external memory space to flash memory space or RAM space.

CHAPTER 3 REGISTERS

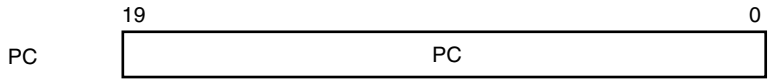
3.1 Control Registers

The control registers control the program sequence, statuses and stack memory. A program counter (PC), a program status word (PSW), and a stack pointer (SP) are the control registers.

3.1.1 Program counter (PC)

The program counter is a 20-bit register that holds the address information of the next program to be executed.

Figure 3-1. Program Counter Configuration



3.1.2 Program status word (PSW)

The program status word is an 8-bit register consisting of various flags to be set/reset by instruction execution.

The ISP1 flag is added as bit 2 in products that support interrupt level 4.

The contents of the program status word are automatically stacked when an interrupt request occurs and a PUSH PSW instruction is executed, and are automatically restored when an RETB or RETI instruction and a POP PSW instruction is executed.

The PSW value becomes 06H when a reset signal is input.

Figure 3-2. Program Status Word Configuration

	<7>	<6>	<5>	<4>	<3>	<2>	<1>	0
PSW	IE	Z	RBS1	AC	RBS0	ISP1	ISP0	CY

(1) Interrupt enable flag (IE)

This flag controls the interrupt request acknowledgement operations of the CPU.

When IE = 0, the IE flag is set to interrupt disable (DI), and interrupts other than non-maskable interrupts are all disabled.

When IE = 1, the IE flag is set to interrupt enable (EI), and interrupt request acknowledgement is controlled by an interrupt mask flag for various interrupt sources, and a priority specification flag.

This flag is reset (0) upon DI instruction execution or interrupt request acknowledgment and is set (1) upon execution of the EI instruction.

(2) Zero flag (Z)

When the operation result is zero, this flag is set (1). It is reset (0) in all other cases.

(3) Register bank select flags (RBS0 and RBS1)

These are 2-bit flags used to select one of the four register banks.

In these flags, the 2-bit information that indicates the register bank selected by SBL RBn instruction execution is stored.

(4) Auxiliary carry flag (AC)

If the operation result has a carry from bit 3 or a borrow at bit 3, this flag is set (1). It is reset (0) in all other cases.

(5) In-service priority flags (ISP0 and ISP1)

This flag manages the priority of acknowledgeable maskable vectored interrupts. The vectored interrupt requests specified as lower than the ISP0 and ISP1 values by the priority specification flag register (PR) are disabled for acknowledgment. Actual acknowledgment for interrupt requests is controlled by the state of the interrupt enable flag (IE).

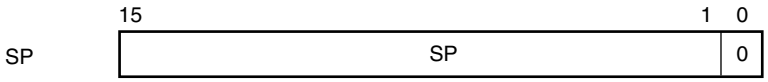
(6) Carry flag (CY)

This flag stores an overflow or underflow upon add/subtract instruction execution. It stores the shift-out value upon rotate instruction execution and functions as a bit accumulator during bit manipulation instruction execution.

3.1.3 Stack pointer (SP)

This is a 16-bit register that holds the start address of the memory stack area. Only the internal RAM area can be set as the stack area.

Figure 3-3. Stack Pointer Configuration



The SP is decremented ahead of write (save) to the stack memory and is incremented after read (restored) from the stack memory.

Since reset signal generation makes the SP contents undefined, be sure to initialize the SP before using the stack. In addition, the values of the stack pointer must be set to even numbers. If odd numbers are specified, the least significant bit is automatically cleared to 0.

In the RL78 microcontrollers, since the memory space is expanded, the stack address used for a CALL instruction or interrupt is 1 byte longer, and 2-byte or 4-byte stack size is used because the RAM for the stack is 16 bits long (refer to **Table 3-1**).

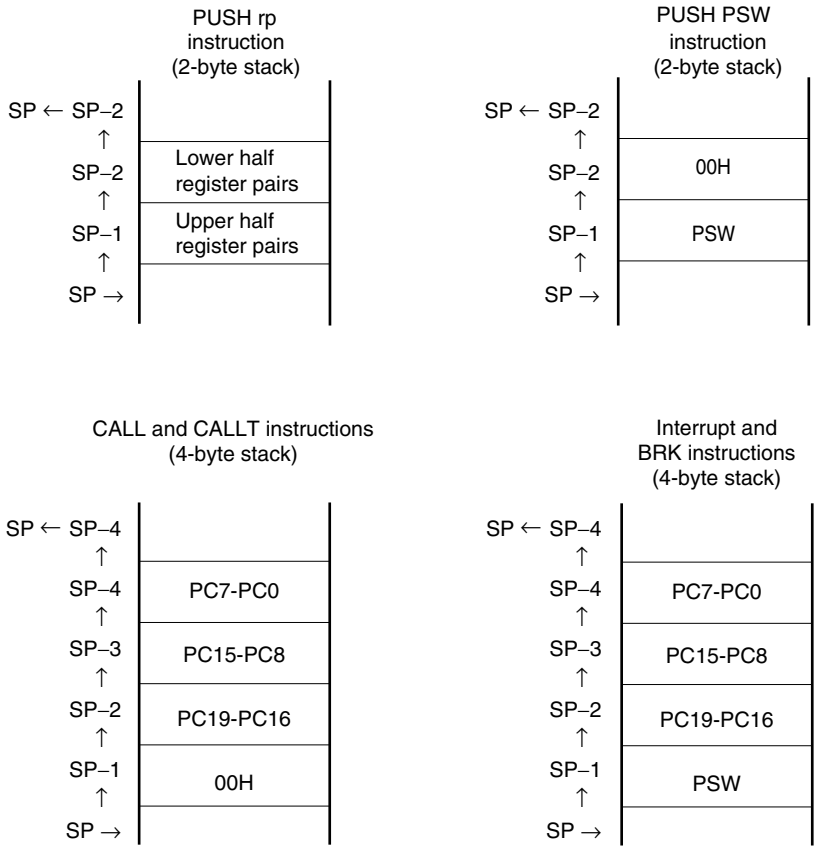
Caution It is prohibited to use the general-purpose register (FFEE0H to FFEFFH) space as a stack area.

Table 3-1. Stack Size Differences Between 78K0 Microcontrollers and RL78 Microcontrollers

Save Instruction	Restore Instruction	Stack Size of 78K0 Microcontrollers	Stack Size of RL78 Microcontrollers
PUSH rp	POP rp	2 bytes	2 bytes
PUSH PSW	POP PSW	1 byte	2 bytes
CALL, CALLT	RET	2 bytes	4 bytes
Interrupt	RETI	3 bytes	4 bytes
BRK	RETB	3 bytes	4 bytes

Figure 3-4 shows the data saved by various stack operations in the RL78 microcontrollers.

Figure 3-4. Data to Be Saved to Stack Memory



Stack pointers can be specified only within internal RAM. The target address range is from F0000H to FFFFFH; be sure not to exceed the internal RAM space. If an address outside the internal RAM space is specified, write operations to that address will be ignored and read operations will return undefined values.

3.2 General-Purpose Registers

On-chip general-purpose registers are mapped at addresses FFEE0H to FFEFFH of the RAM. These registers consist of 4 banks, each bank consisting of eight 8-bit registers (X, A, C, B, E, D, L and H). The bank to be used when an instruction is executed is set by the CPU control instruction “SEL RBn”.

In addition that each register can be used as an 8-bit register, two 8-bit registers in pairs can be used as a 16-bit register.

In programming, general-purpose registers can be described in terms of functional names (X, A, C, B, E, D, L, H, AX, BC, DE and HL) and absolute names (R0 to R7 and RP0 to RP3).

Caution Use of the general-purpose register space (FFEE0H to FFEFFH) as the instruction fetch area or stack area is prohibited.

Table 3-2. List of General-Purpose Registers (Common to 78K0 Microcontrollers)

Bank Name	Register				Absolute Address
	Functional Name		Absolute Name		
	16-bit Processing	8-bit Processing	16-bit Processing	8-bit Processing	
BANK0	HL	H	RP3	R7	FFEFFH
		L		R6	FFEFEH
	DE	D	RP2	R5	FFEFDH
		E		R4	FFEFCH
	BC	B	RP1	R3	FFEFBH
		C		R2	FFEFAH
	AX	A	RP0	R1	FFEF9H
		X		R0	FFEF8H
BANK1	HL	H	RP3	R7	FFEF7H
		L		R6	FFEF6H
	DE	D	RP2	R5	FFEF5H
		E		R4	FFEF4H
	BC	B	RP1	R3	FFEF3H
		C		R2	FFEF2H
	AX	A	RP0	R1	FFEF1H
		X		R0	FFEF0H
BANK2	HL	H	RP3	R7	FFEEFH
		L		R6	FFEEEH
	DE	D	RP2	R5	FFEEDH
		E		R4	FFEECH
	BC	B	RP1	R3	FFEEBH
		C		R2	FFEEAH
	AX	A	RP0	R1	FFEE9H
		X		R0	FFEE8H
BANK3	HL	H	RP3	R7	FFEE7H
		L		R6	FFEE6H
	DE	D	RP2	R5	FFEE5H
		E		R4	FFEE4H
	BC	B	RP1	R3	FFEE3H
		C		R2	FFEE2H
	AX	A	RP0	R1	FFEE1H
		X		R0	FFEE0H

3.3 ES and CS Registers

The RL78 microcontrollers have additional ES and CS registers. Data access can be specified via the ES register and higher addresses for execution of branch instructions can be specified via the CS register. For description of how these registers are used, refer to **CHAPTER 4 ADDRESSING**.

After reset, the initial value of ES is 0FH and the initial value of CS is 00H.

Figure 3-5. Configuration of ES and CS Registers

	7	6	5	4	3	2	1	0
ES	0	0	0	0	ES3	ES2	ES1	ES0

	7	6	5	4	3	2	1	0
CS	0	0	0	0	CS3	CP2	CP1	CP0

3.4 Special Function Registers (SFRs)

Table 3-3 describes fixed-address SFRs in the RL78 microcontrollers.

Table 3-3. List of Fixed SFRs

Address	Register Name
FFFF8H	SPL
FFFF9H	SPH
FFFFAH	PSW
FFFFBH	Reserve
FFFFCH	CS
FFFFDH	ES
FFFFEH	PMC
FFFFFH	MEM

3.4.1 Processor mode control register (PMC)

This is an 8-bit register that is used to control the processor modes. For details, refer to **2.2 Internal Program Memory Space.**

PMC's initial value after reset is 00H.

Figure 3-6. Configuration of Processor Mode Control Register

Address: FFFFEH After reset: 00H R/W

Symbol	7	6	5	4	3	2	1	<0>
PMC	0	0	0	0	0	0	0	MAA

MAA	Selection of flash memory space for mirroring to area from F0000H to FFFFFH[Note]
0	00000H to 0FFFFH is mirrored to F0000H to FFFFFH
1	10000H to 1FFFFH is mirrored to F0000H to FFFFFH

Note SFR and RAM areas are also allocated to the range from F0000H to FFFFFH, and take priority over other items for the overlapping areas.

Cautions 1. Set the PMC register only once for initial settings. Rewriting PMC is prohibited except for initial settings.

2. After setting PMC, wait for at least one instruction and access the mirror area.

CHAPTER 4 ADDRESSING

Addressing is divided into two types: addressing for processing data addresses and addressing for program addresses. The addressing modes corresponding to each type are described below.

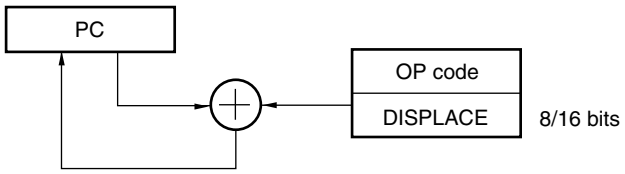
4.1 Instruction Address Addressing

4.1.1 Relative addressing

[Function]

Relative addressing stores in the program counter (PC) the result of adding a displacement value included in the instruction word (signed complement data: −128 to +127 or −32768 to +32767) to the program counter (PC)'s value (the start address of the next instruction), and specifies the program address to be used as the branch destination. Relative addressing is applied only to branch instructions.

Figure 4-1. Outline of Relative Addressing



Caution Do not use relative addressing in the following branch instructions:
- Branching from internal program memory space to RAM space or external memory space
- Branching from RAM space to internal program memory space or external memory space
- Branching from external memory space to internal program memory space or RAM space

4.1.2 Immediate addressing

[Function]

Immediate addressing stores immediate data of the instruction word in the program counter, and specifies the program address to be used as the branch destination.

For immediate addressing, CALL !!addr20 or BR !!addr20 is used to specify 20-bit addresses and CALL !addr16 or BR !addr16 is used to specify 16-bit addresses. 0000 is set to the higher 4 bits when specifying 16-bit addresses.

Figure 4-2. Example of CALL !!addr20/BR !!addr20

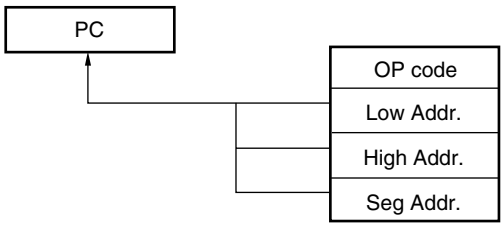
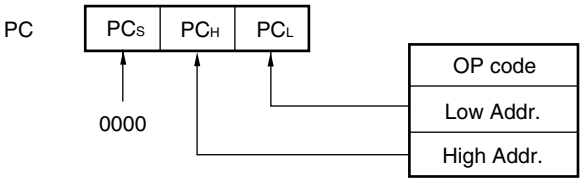


Figure 4-3. Example of CALL !addr16/BR !addr16



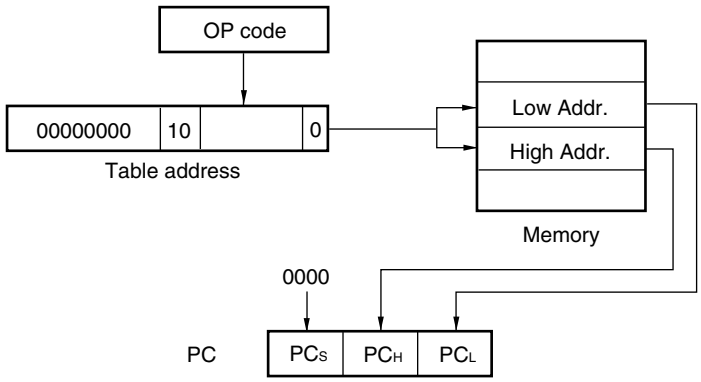
4.1.3 Table indirect addressing

[Function]

Table indirect addressing specifies a table address in the CALLT table area (0080H to 00BFH) with the 5-bit immediate data in the instruction word, stores the contents at that table address and the next address in the program counter (PC) as 16-bit data, and specifies the program address. Table indirect addressing is applied only for CALLT instructions.

In the RL78 microcontrollers, branching is enabled only to the 64 KB space from 00000H to 0FFFFH.

Figure 4-4. Outline of Table Indirect Addressing

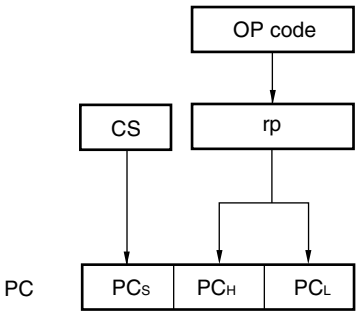


4.1.4 Register direct addressing

[Function]

Register direct addressing stores in the program counter (PC) the contents of a general-purpose register pair (AX/BC/DE/HL) and CS register of the current register bank specified with the instruction word as 20-bit data, and specifies the program address. Register direct addressing can be applied only to the CALL AX, BC, DE, HL, and BR AX instructions.

Figure 4-5. Outline of Register Direct Addressing



4.2 Addressing for Processing Data Addresses

4.2.1 Implied addressing

[Function]

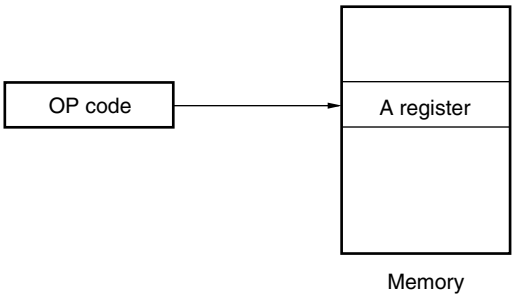
Instructions for accessing registers (such as accumulators) that have special functions are directly specified with the instruction word, without using any register specification field in the instruction word.

[Operand format]

Because implied addressing can be automatically employed with an instruction, no particular operand format is necessary.

Implied addressing can be applied only to MULU X.

Figure 4-6. Outline of Implied Addressing



4.2.2 Register addressing

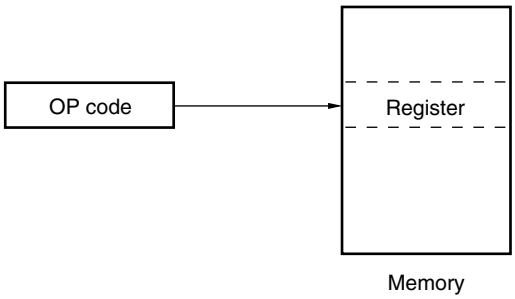
[Function]

Register addressing accesses a general-purpose register as an operand. The instruction word of 3-bit long is used to select an 8-bit register and the instruction word of 2-bit long is used to select a 16-bit register.

[Operand format]

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

Figure 4-7. Outline of Register Addressing



4.2.3 Direct addressing

[Function]

Direct addressing uses immediate data in the instruction word as an operand address to directly specify the target address.

[Operand format]

Identifier	Description
ADDR16	Label or 16-bit immediate data (only the space from F0000H to FFFFFH is specifiable)
ES: ADDR16	Label or 16-bit immediate data (higher 4-bit addresses are specified by the ES register)

Figure 4-8. Example of ADDR16

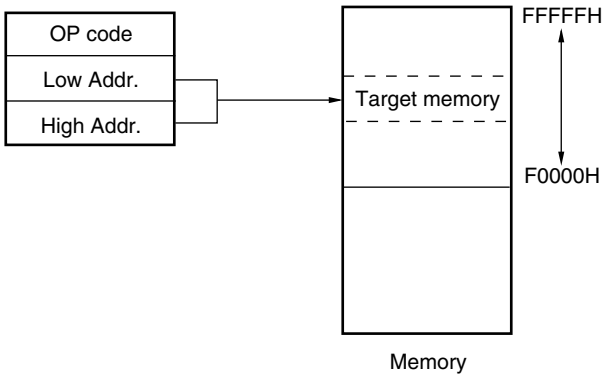
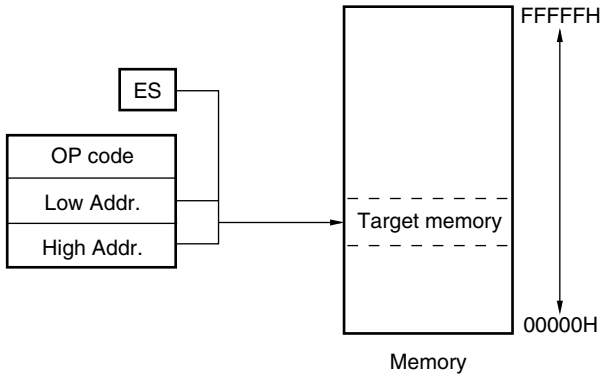


Figure 4-9. Example of ES:ADDR16



4.2.4 Short direct addressing

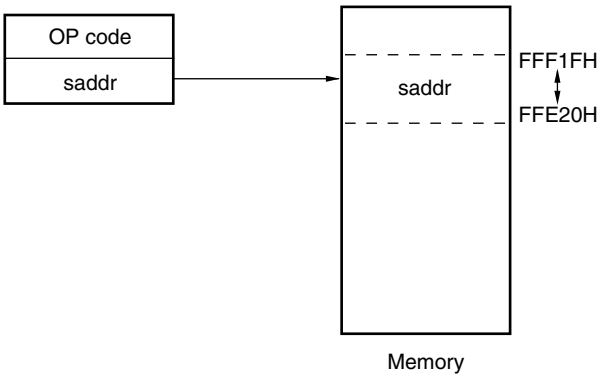
[Function]

Short direct addressing directly specifies the target addresses using 8-bit data in the instruction word. This type of addressing is applied only to the space from FFE20H to FFF1FH.

[Operand format]

Identifier	Description
SADDR	Label, FFE20H to FFF1FH immediate data or 0FE20H to 0FF1FH immediate data (only the space from FFE20H to FFF1FH is specifiable)
SADDRP	Label, FFE20H to FFF1FH immediate data or 0FE20H to 0FF1FH immediate data (even address only) (only the space from FFE20H to FFF1FH is specifiable)

Figure 4-10. Outline of Short Direct Addressing



Remark SADDR and SADDRP are used to describe the values of addresses FE20H to FF1FH with 16-bit immediate data (higher 4 bits of actual address are omitted), and the values of addresses FFE20H to FFF1FH with 20-bit immediate data.
Regardless of whether SADDR or SADDRP is used, addresses within the space from FFE20H to FFF1FH are specified for the memory.

4.2.5 SFR addressing

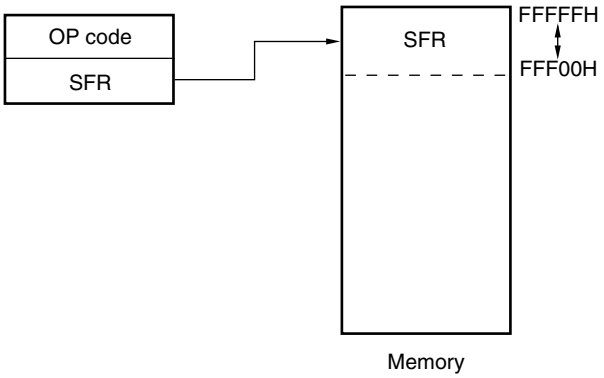
[Function]

SFR addressing directly specifies the target SFR addresses using 8-bit data in the instruction word. This type of addressing is applied only to the space from FFF00H to FFFFFH.

[Operand format]

Identifier	Description
SFR	SFR name
SFRP	16-bit-manipulatable SFR name (even address only)

Figure 4-11. Outline of SFR Addressing



4.2.6 Register indirect addressing

[Function]

Register indirect addressing directly specifies the target addresses using the contents of the register pair specified with the instruction word as an operand address.

[Operand format]

Identifier	Description
–	[DE], [HL] (only the space from F0000H to FFFFFH is specifiable)
–	ES:[DE], ES:[HL] (higher 4-bit addresses are specified by the ES register)

Figure 4-12. Example of [DE], [HL]

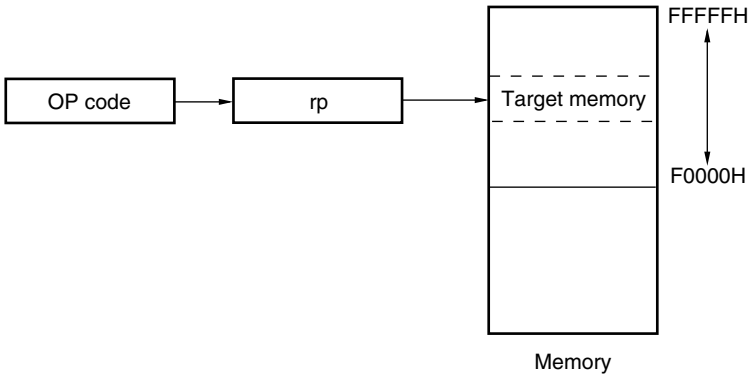
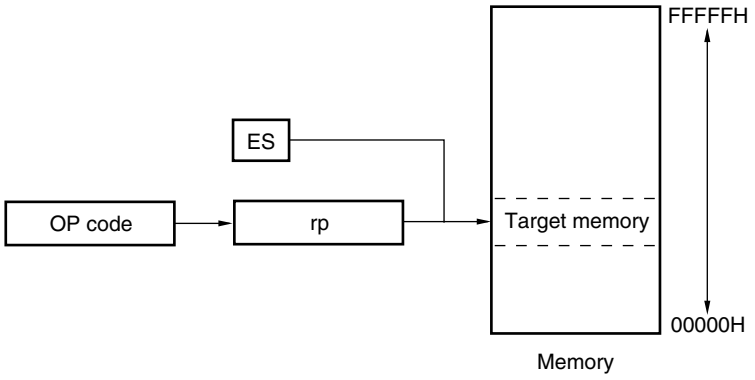


Figure 4-13. Example of ES:[DE], ES:[HL]



4.2.7 Based addressing

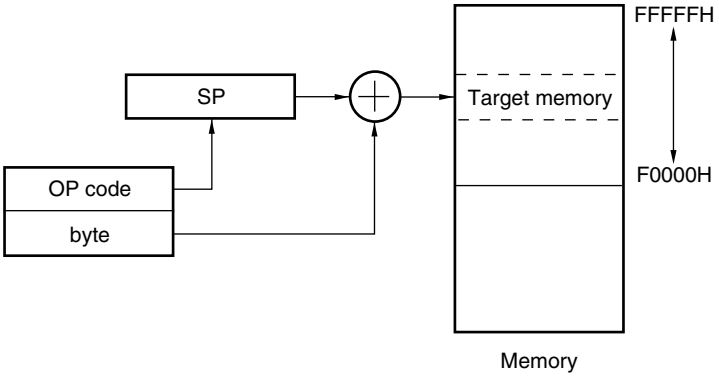
[Function]

Based addressing uses the contents of a register pair specified with the instruction word as a base address, and 8-bit immediate data or 16-bit immediate data as offset data. The sum of these values is used to specify the target address.

[Operand format]

Identifier	Description
–	[HL + byte], [DE + byte], [SP + byte] (only the space from F0000H to FFFFFH is specifiable)
–	word[B], word[C] (only the space from F0000H to FFFFFH is specifiable)
–	word[BC] (only the space from F0000H to FFFFFH is specifiable)
–	ES:[HL + byte], ES:[DE + byte] (higher 4-bit addresses are specified by the ES register)
–	ES:word[B], ES:word[C] (higher 4-bit addresses are specified by the ES register)
–	ES:word[BC] (higher 4-bit addresses are specified by the ES register)

Figure 4-14. Example of [SP+byte]



Caution In [HL+byte], [DE+byte], word[B], word[C], and word[BC], an added value must not exceed FFFFH.
In ES:[HL+byte], ES:[DE+byte], ES:word[B], ES:word[C], and ES:word[BC], an added value must not exceed FFFFFH.
For [SP+byte], an SP value must be within RAM space and the added value of SP+byte must be FFEDFH or less in RAM space.

Figure 4-15. Example of [HL + byte], [DE + byte]

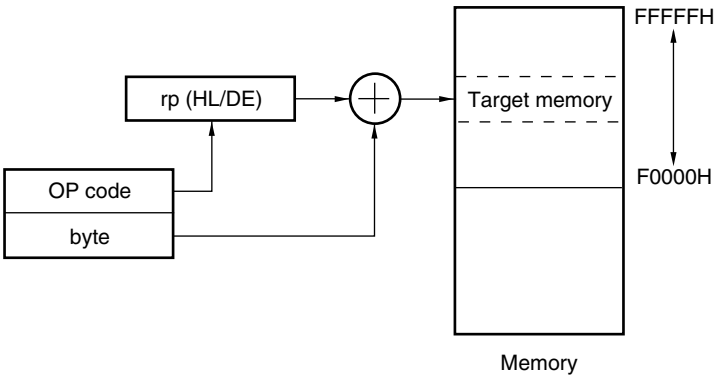


Figure 4-16. Example of word[B], word[C]

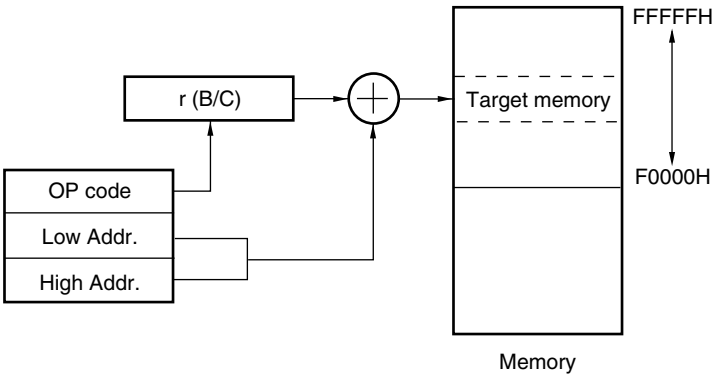


Figure 4-17. Example of word[BC]

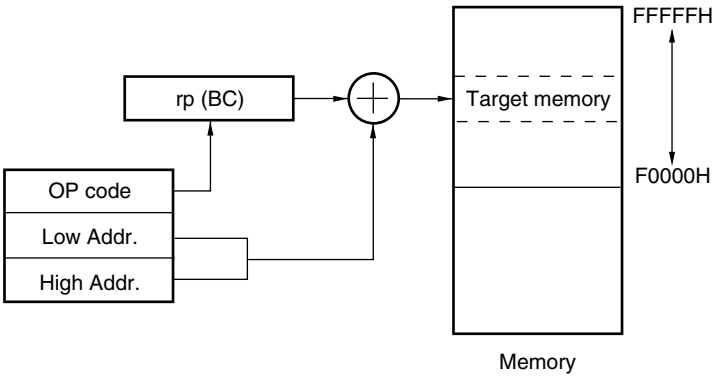


Figure 4-18. Example of ES:[HL + byte], ES:[DE + byte]

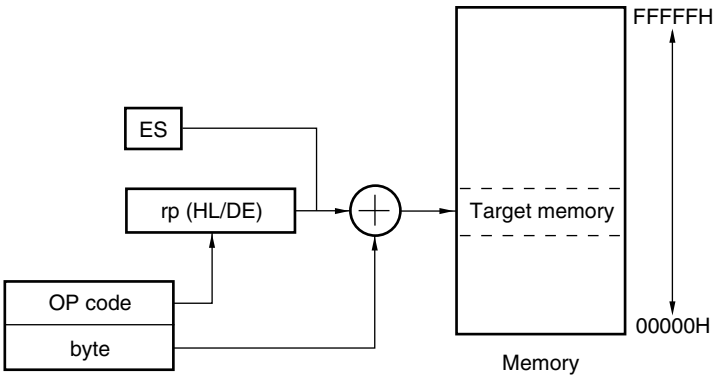


Figure 4-19. Example of ES:word[B], ES:word[C]

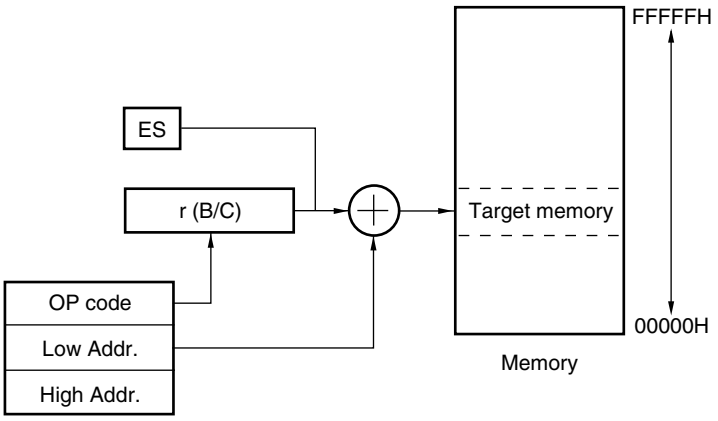
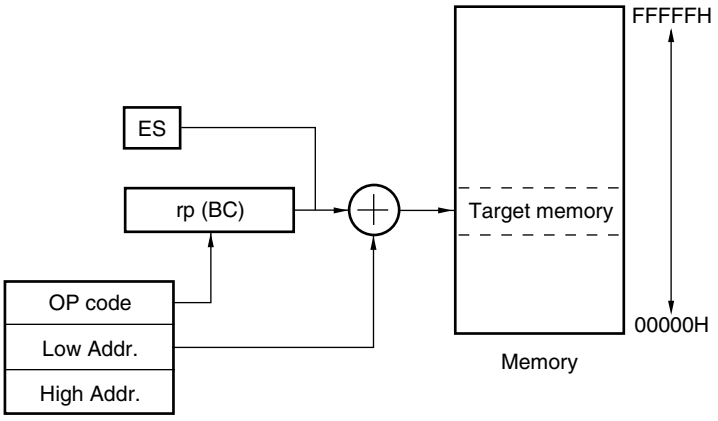


Figure 4-20. Example of ES:word[BC]



4.2.8 Based indexed addressing

[Function]

Based indexed addressing uses the contents of a register pair specified with the instruction word as the base address, and the content of the B register or C register similarly specified with the instruction word as offset address. The sum of these values is used to specify the target address.

[Operand format]

Identifier	Description
–	[HL+B], [HL+C] (only the space from F0000H to FFFFFH is specifiable)
–	ES:[HL+B], ES:[HL+C] (higher 4-bit addresses are specified by the ES register)

Figure 4-21. Example of [HL+B], [HL+C]

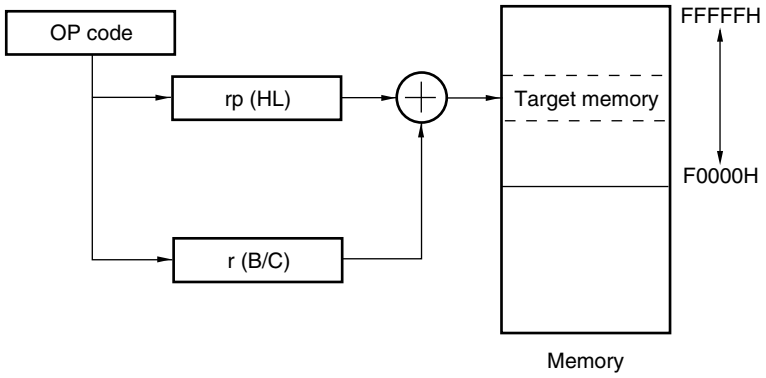
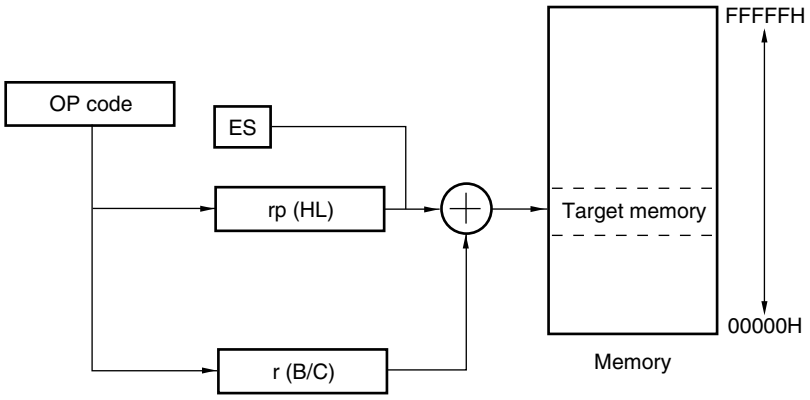


Figure 4-22. Example of ES:[HL+B], ES:[HL+C]



Caution In [HL+ B] and [HL+C], an added value must not exceed FFFFH.
In ES:[HL+ B] and ES:[HL+C], an added value must not exceed FFFFFH.

4.2.9 Stack addressing

[Function]

The stack area is indirectly addressed with the stack pointer (SP) contents. This addressing is automatically employed when the PUSH, POP, subroutine call, and return instructions are executed or the register is saved/restored upon generation of an interrupt request.

Stack addressing is applied only to the internal RAM area.

[Operand format]

Identifier	Description
–	PUSH AX/BC/DE/HL POP AX/BC/DE/HL CALL/CALLT RET BRK RETB (Interrupt request generated) RETI

CHAPTER 5 INSTRUCTION SET

This chapter lists the instructions in the RL78 microcontroller instruction set. The instructions are common to all RL78 microcontrollers. However, the following multiply/divide/multiply & accumulate instructions are expanded instructions and mounted or not mounted by product. For details, refer to user's manual of each product.

- MULHU (16-bit multiplication unsigned)
- MULH (16-bit multiplication signed)
- DIVHU (16-bit division unsigned)
- DIVWU (32-bit division unsigned)
- MACHU (16-bit multiplication and accumulation unsigned (16 bits × 16 bits) + 32 bits)
- MACH (16-bit multiplication and accumulation signed (16 bits × 16 bits) + 32 bits)

Remark The shaded parts of the tables in **5.5 List of Operations** and **5.6 List of Instruction Formats** indicate the operation or instruction format that is newly added for the RL78 microcontrollers.

5.1 Operand Identifiers and Description Methods

Operands are described in the "Operand" column of each instruction in accordance with the description method of the instruction operand identifier (refer to the assembler specifications for details). When there are two or more description methods, select one of them. Alphabetic letters in capitals and the symbols, #, !, !!, $, $!, [ ], and ES: are keywords and are described as they are. Each symbol has the following meaning.

- #: Immediate data specification
- !: 16-bit absolute address specification
- !!: 20-bit absolute address specification
- $: 8-bit relative address specification
- $!: 16-bit relative address specification
- [ ]: Indirect address specification
- ES:: Extension address specification

In the case of immediate data, describe an appropriate numeric value or a label. When using a label, be sure to describe the #, !, !!, $, $!, [ ], and ES: symbols.

For operand register identifiers, r and rp, either function names (X, A, C, etc.) or absolute names (names in parentheses in the table below, R0, R1, R2, etc.) can be used for description.

Table 5-1. Operand Identifiers and Description Methods

Identifier	Description Method
r rp sfr sfrp	X (R0), A (R1), C (R2), B (R3), E (R4), D (R5), L (R6), H (R7) AX (RP0), BC (RP1), DE (RP2), HL (RP3) Special-function register symbol (SFR symbol) FFF00H to FFFFFH Special-function register symbols (16-bit manipulatable SFR symbol. Even addresses only[Note]) FFF00H to FFFFFH
saddr saddrp	FFE20H to FFF1FH Immediate data or labels FFE20H to FF1FH Immediate data or labels (even addresses only[Note])
addr20 addr16 addr5	00000H to FFFFFH Immediate data or labels 0000H to FFFFH Immediate data or labels (only even addresses for 16-bit data transfer instructions[Note]) 0080H to 00BFH Immediate data or labels (even addresses only)
word byte bit	16-bit immediate data or label 8-bit immediate data or label 3-bit immediate data or label
RBn	RB0 to RB3

Note Bit 0 = 0 when an odd address is specified.

Remark The special function registers can be described to operand sfr as symbols.
The extended special function registers can be described to operand !addr16 as symbols.

5.2 Symbols in “Operation” Column

The operation when the instruction is executed is shown in the “Operation” column using the following symbols.

Table 5-2. Symbols in “Operation” Column

Symbol	Function
A	A register; 8-bit accumulator
X	X register
B	B register
C	C register
D	D register
E	E register
H	H register
L	L register
ES	ES register
CS	CS register
AX	AX register pair; 16-bit accumulator
BC	BC register pair
DE	DE register pair
HL	HL register pair
PC	Program counter
SP	Stack pointer
PSW	Program status word
CY	Carry flag
AC	Auxiliary carry flag
Z	Zero flag
RBS	Register bank select flag
IE	Interrupt request enable flag
()	Memory contents indicated by address or register contents in parentheses
X_H, X_L X_S, X_H, X_L	16-bit registers: X_H = higher 8 bits, X_L = lower 8 bits 20-bit registers: X_S = (bits 19 to 16), X_H = (bits 15 to 8), X_L = (bits 7 to 0)
∧	Logical product (AND)
∨	Logical sum (OR)
∀	Exclusive logical sum (exclusive OR)
¯	Inverted data
addr5	16-bit immediate data (even addresses only in 0080H to 00BFH)
addr16	16-bit immediate data
addr20	20-bit immediate data
jdisp8	Signed 8-bit data (displacement value)
jdisp16	Signed 16-bit data (displacement value)

5.3 Symbols in “Flag” Column

The change of the flag value when the instruction is executed is shown in the “Flag” column using the following symbols.

Table 5-3. Symbols in “Flag” Column

Symbol	Change of Flag Value
(Blank)	Unchanged
0	Cleared to 0
1	Set to 1
×	Set/cleared according to the result
R	Previously saved value is restored

5.4 PREFIX Instruction

Instructions with “ES:” have a PREFIX operation code as a prefix to extend the accessible data area to the 1 MB space (00000H to FFFFFH), by adding the ES register value to the 64 KB space from F0000H to FFFFFH. When a PREFIX operation code is attached as a prefix to the target instruction, only one instruction immediately after the PREFIX operation code is executed as the addresses with the ES register value added.

A interrupt and DMA transfer are not acknowledged between a PREFIX instruction code and the instruction immediately after.

Table 5-4. Use Example of PREFIX Operation Code

Instruction	Opcode				
	1	2	3	4	5
MOV !addr16, #byte	CFH	!addr16		#byte	–
MOV ES:!addr16, #byte	11H	CFH	!addr16		#byte
MOV A, [HL]	8BH	–	–	–	–
MOV A, ES:[HL]	11H	8BH	–	–	–

Caution Set the ES register value with MOV ES, A, etc., before executing the PREFIX instruction.

5.5 Operation List

Table 5-5. Operation List (1/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag		
				Note 1	Note 2		Z	AC	CY
8-bit data transfer	MOV	r, #byte	2	1	–	r ← byte			
		saddr, #byte	3	1	–	(saddr) ← byte			
		sfr, #byte	3	1	–	sfr ← byte			
		!addr16, #byte	4	1	–	(addr16) ← byte			
		A, r Note 3	1	1	–	A ← r			
		r, A Note 3	1	1	–	r ← A			
		A, saddr	2	1	–	A ← (saddr)			
		saddr, A	2	1	–	(saddr) ← A			
		A, sfr	2	1	–	A ← sfr			
		sfr, A	2	1	–	sfr ← A			
		A, !addr16	3	1	4	A ← (addr16)			
		!addr16, A	3	1	–	(addr16) ← A			
		PSW, #byte	3	3	–	PSW ← byte	×	×	×
		A, PSW	2	1	–	A ← PSW			
		PSW, A	2	3	–	PSW ← A	×	×	×
		ES, #byte	2	1	–	ES ← byte			
		ES, saddr	3	1	–	ES ← (saddr)			
		A, ES	2	1	–	A ← ES			
		ES, A	2	1	–	ES ← A			
		CS, #byte	3	1	–	CS ← byte			
		A, CS	2	1	–	A ← CS			
		CS, A	2	1	–	CS ← A			
		A, [DE]	1	1	4	A ← (DE)			
		[DE], A	1	1	–	(DE) ← A			
		[DE + byte], #byte	3	1	–	(DE + byte) ← byte			
		A, [DE + byte]	2	1	4	A ← (DE + byte)			
		[DE + byte], A	2	1	–	(DE + byte) ← A			
		A, [HL]	1	1	4	A ← (HL)			
		[HL], A	1	1	–	(HL) ← A			
		[HL + byte], #byte	3	1	–	(HL + byte) ← byte			

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.

2. When the program memory area is accessed.

3. Except r = A

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).

2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).

3. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (2/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag
				Note 1	Note 2		Z AC CY
8-bit data transfer	MOV	A, [HL + byte]	2	1	4	A ← (HL + byte)	
		[HL + byte], A	2	1	–	(HL + byte) ← A	
		A, [HL + B]	2	1	4	A ← (HL + B)	
		[HL + B], A	2	1	–	(HL + B) ← A	
		A, [HL + C]	2	1	4	A ← (HL + C)	
		[HL + C], A	2	1	–	(HL + C) ← A	
		word[B], #byte	4	1	–	(B + word) ← byte	
		A, word[B]	3	1	4	A ← (B + word)	
		word[B], A	3	1	–	(B + word) ← A	
		word[C], #byte	4	1	–	(C + word) ← byte	
		A, word[C]	3	1	4	A ← (C + word)	
		word[C], A	3	1	–	(C + word) ← A	
		word[BC], #byte	4	1	–	(BC + word) ← byte	
		A, word[BC]	3	1	4	A ← (BC + word)	
		word[BC], A	3	1	–	(BC + word) ← A	
		[SP + byte], #byte	3	1	–	(SP + byte) ← byte	
		A, [SP + byte]	2	1	–	A ← (SP + byte)	
		[SP + byte], A	2	1	–	(SP + byte) ← A	
		B, saddr	2	1	–	B ← (saddr)	
		B, !addr16	3	1	4	B ← (addr16)	
		C, saddr	2	1	–	C ← (saddr)	
		C, !addr16	3	1	4	C ← (addr16)	
		X, saddr	2	1	–	X ← (saddr)	
		X, !addr16	3	1	4	X ← (addr16)	
		ES:!addr16, #byte	5	2	–	(ES, addr16) ← byte	
		A, ES:!addr16	4	2	5	A ← (ES, addr16)	
		ES:!addr16, A	4	2	–	(ES, addr16) ← A	
		A, ES:[DE]	2	2	5	A ← (ES, DE)	
		ES:[DE], A	2	2	–	(ES, DE) ← A	
		ES:[DE + byte],#byte	4	2	–	((ES, DE) + byte) ← byte	
		A, ES:[DE + byte]	3	2	5	A ← ((ES, DE) + byte)	
		ES:[DE + byte], A	3	2	–	((ES, DE) + byte) ← A	

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.

2. When the program memory area is accessed.

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).

2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).

3. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (3/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag
				Note 1	Note 2		Z AC CY
8-bit data transfer	MOV	A, ES:[HL]	2	2	5	A ← (ES, HL)	
		ES:[HL], A	2	2	–	(ES, HL) ← A	
		ES:[HL + byte],#byte	4	2	–	((ES, HL) + byte) ← byte	
		A, ES:[HL + byte]	3	2	5	A ← ((ES, HL) + byte)	
		ES:[HL + byte], A	3	2	–	((ES, HL) + byte) ← A	
		A, ES:[HL + B]	3	2	5	A ← ((ES, HL) + B)	
		ES:[HL + B], A	3	2	–	((ES, HL) + B) ← A	
		A, ES:[HL + C]	3	2	5	A ← ((ES, HL) + C)	
		ES:[HL + C], A	3	2	–	((ES, HL) + C) ← A	
		ES:word[B], #byte	5	2	–	((ES, B) + word) ← byte	
		A, ES:word[B]	4	2	5	A ← ((ES, B) + word)	
		ES:word[B], A	4	2	–	((ES, B) + word) ← A	
		ES:word[C], #byte	5	2	–	((ES, C) + word) ← byte	
		A, ES:word[C]	4	2	5	A ← ((ES, C) + word)	
		ES:word[C], A	4	2	–	((ES, C) + word) ← A	
		ES:word[BC], #byte	5	2	–	((ES, BC) + word) ← byte	
		A, ES:word[BC]	4	2	5	A ← ((ES, BC) + word)	
		ES:word[BC], A	4	2	–	((ES, BC) + word) ← A	
		B, ES:!addr16	4	2	5	B ← (ES, addr16)	
		C, ES:!addr16	4	2	5	C ← (ES, addr16)	
		X, ES:!addr16	4	2	5	X ← (ES, addr16)	
	XCH	A, r Note 3	1 (r = X) 2 (other than r = X)	1	–	A ←→ r	
		A, saddr	3	2	–	A ←→ (saddr)	
		A, sfr	3	2	–	A ←→ sfr	
		A, !addr16	4	2	–	A ←→ (addr16)	
		A, [DE]	2	2	–	A ←→ (DE)	
		A, [DE + byte]	3	2	–	A ←→ (DE + byte)	
		A, [HL]	2	2	–	A ←→ (HL)	
		A, [HL + byte]	3	2	–	A ←→ (HL + byte)	
		A, [HL + B]	2	2	–	A ←→ (HL + B)	
		A, [HL + C]	2	2	–	A ←→ (HL + C)	

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.

2. When the program memory area is accessed.

3. Except r = A

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).

2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).

3. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (4/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag		
				Note 1	Note 2		Z	AC	CY
8-bit data transfer	XCH	A, ES:!addr16	5	3	–	A ←→ (ES, addr16)			
		A, ES:[DE]	3	3	–	A ←→ (ES, DE)			
		A, ES:[DE + byte]	4	3	–	A ←→ ((ES, DE) + byte)			
		A, ES:[HL]	3	3	–	A ←→ (ES, HL)			
		A, ES:[HL + byte]	4	3	–	A ←→ ((ES, HL) + byte)			
		A, ES:[HL + B]	3	3	–	A ←→ ((ES, HL) + B)			
		A, ES:[HL + C]	3	3	–	A ←→ ((ES, HL) + C)			
	ONEB	A	1	1	–	A ← 01H			
		X	1	1	–	X ← 01H			
		B	1	1	–	B ← 01H			
		C	1	1	–	C ← 01H			
		saddr	2	1	–	(saddr) ← 01H			
		!addr16	3	1	–	(addr16) ← 01H			
		ES:!addr16	4	2	–	(ES, addr16) ← 01H			
	CLRB	A	1	1	–	A ← 00H			
		X	1	1	–	X ← 00H			
		B	1	1	–	B ← 00H			
		C	1	1	–	C ← 00H			
		saddr	2	1	–	(saddr) ← 00H			
		!addr16	3	1	–	(addr16) ← 00H			
		ES:!addr16	4	2	–	(ES,addr16) ← 00H			
	MOVS	[HL + byte], X	3	1	–	(HL + byte) ← X	×		×
		ES:[HL + byte], X	4	2	–	(ES, HL + byte) ← X	×		×
16-bit data transfer	MOVW	rp, #word	3	1	–	rp ← word			
		saddrp, #word	4	1	–	(saddrp) ← word			
		sfrp, #word	4	1	–	sfrp ← word			
		AX, saddrp	2	1	–	AX ← (saddrp)			
		saddrp, AX	2	1	–	(saddrp) ← AX			
		AX, sfrp	2	1	–	AX ← sfrp			
		sfrp, AX	2	1	–	sfrp ← AX			
		AX, rp Note 3	1	1	–	AX ← rp			
		rp, AX Note 3	1	1	–	rp ← AX			

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.

2. When the program memory area is accessed.

3. Except rp = AX

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).

2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).

3. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (5/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag
				Note 1	Note 2		Z AC CY
16-bit data transfer	MOVW	AX, !addr16	3	1	4	AX ← (addr16)	
		!addr16, AX	3	1	–	(addr16) ← AX	
		AX, [DE]	1	1	4	AX ← (DE)	
		[DE], AX	1	1	–	(DE) ← AX	
		AX, [DE + byte]	2	1	4	AX ← (DE + byte)	
		[DE + byte], AX	2	1	–	(DE + byte) ← AX	
		AX, [HL]	1	1	4	AX ← (HL)	
		[HL], AX	1	1	–	(HL) ← AX	
		AX, [HL + byte]	2	1	4	AX ← (HL + byte)	
		[HL + byte], AX	2	1	–	(HL + byte) ← AX	
		AX, word[B]	3	1	4	AX ← (B + word)	
		word[B], AX	3	1	–	(B + word) ← AX	
		AX, word[C]	3	1	4	AX ← (C + word)	
		word[C], AX	3	1	–	(C + word) ← AX	
		AX, word[BC]	3	1	4	AX ← (BC + word)	
		word[BC], AX	3	1	–	(BC + word) ← AX	
		AX, [SP + byte]	2	1	–	AX ← (SP + byte)	
		[SP + byte], AX	2	1	–	(SP + byte) ← AX	
		BC, saddrp	2	1	–	BC ← (saddrp)	
		BC, !addr16	3	1	4	BC ← (addr16)	
		DE, saddrp	2	1	–	DE ← (saddrp)	
		DE, !addr16	3	1	4	DE ← (addr16)	
		HL, saddrp	2	1	–	HL ← (saddrp)	
		HL, !addr16	3	1	4	HL ← (addr16)	
		AX, ES:!addr16	4	2	5	AX ← (ES, addr16)	
		ES:!addr16, AX	4	2	–	(ES, addr16) ← AX	
		AX, ES:[DE]	2	2	5	AX ← (ES, DE)	
		ES:[DE], AX	2	2	–	(ES, DE) ← AX	
		AX, ES:[DE + byte]	3	2	5	AX ← ((ES, DE) + byte)	
		ES:[DE + byte], AX	3	2	–	((ES, DE) + byte) ← AX	
		AX, ES:[HL]	2	2	5	AX ← (ES, HL)	
		ES:[HL], AX	2	2	–	(ES, HL) ← AX	

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.

2. When the program memory area is accessed.

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).

2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).

3. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (6/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag		
				Note 1	Note 2		Z	AC	CY
16-bit data transfer	MOVW	AX, ES:[HL + byte]	3	2	5	AX ← ((ES, HL) + byte)			
		ES:[HL + byte], AX	3	2	–	((ES, HL) + byte) ← AX			
		AX, ES:word[B]	4	2	5	AX ← ((ES, B) + word)			
		ES:word[B], AX	4	2	–	((ES, B) + word) ← AX			
		AX, ES:word[C]	4	2	5	AX ← ((ES, C) + word)			
		ES:word[C], AX	4	2	–	((ES, C) + word) ← AX			
		AX, ES:word[BC]	4	2	5	AX ← ((ES, BC) + word)			
		ES:word[BC], AX	4	2	–	((ES, BC) + word) ← AX			
		BC, ES:!addr16	4	2	5	BC ← (ES, addr16)			
		DE, ES:!addr16	4	2	5	DE ← (ES, addr16)			
		HL, ES:!addr16	4	2	5	HL ← (ES, addr16)			
	XCHW	AX, rp Note 3	1	1	–	AX ←→ rp			
	ONEW	AX	1	1	–	AX ← 0001H			
		BC	1	1	–	BC ← 0001H			
	CLRW	AX	1	1	–	AX ← 0000H			
		BC	1	1	–	BC ← 0000H			
8-bit operation	ADD	A, #byte	2	1	–	A, CY ← A + byte	×	×	×
		saddr, #byte	3	2	–	(saddr), CY ← (saddr) + byte	×	×	×
		A, r Note 4	2	1	–	A, CY ← A + r	×	×	×
		r, A	2	1	–	r, CY ← r + A	×	×	×
		A, saddr	2	1	–	A, CY ← A + (saddr)	×	×	×
		A, !addr16	3	1	4	A, CY ← A + (addr16)	×	×	×
		A, [HL]	1	1	4	A, CY ← A + (HL)	×	×	×
		A, [HL + byte]	2	1	4	A, CY ← A + (HL + byte)	×	×	×
		A, [HL + B]	2	1	4	A, CY ← A + (HL + B)	×	×	×
		A, [HL + C]	2	1	4	A, CY ← A + (HL + C)	×	×	×
		A, ES:!addr16	4	2	5	A, CY ← A + (ES, addr16)	×	×	×
		A, ES:[HL]	2	2	5	A,CY ← A + (ES, HL)	×	×	×
		A, ES:[HL + byte]	3	2	5	A,CY ← A + ((ES, HL) + byte)	×	×	×
		A, ES:[HL + B]	3	2	5	A,CY ← A + ((ES, HL) + B)	×	×	×
		A, ES:[HL + C]	3	2	5	A,CY ← A + ((ES, HL) + C)	×	×	×

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.
2. When the program memory area is accessed.
3. Except rp = AX
4. Except r = A

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).
2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).
3. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (7/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag		
				Note 1	Note 2		Z	AC	CY
8-bit operation	ADDC	A, #byte	2	1	–	A, CY ← A + byte + CY	×	×	×
		saddr, #byte	3	2	–	(saddr), CY ← (saddr) + byte + CY	×	×	×
		A, r Note 3	2	1	–	A, CY ← A + r + CY	×	×	×
		r, A	2	1	–	r, CY ← r + A + CY	×	×	×
		A, saddr	2	1	–	A, CY ← A + (saddr) + CY	×	×	×
		A, !addr16	3	1	4	A, CY ← A + (addr16) + CY	×	×	×
		A, [HL]	1	1	4	A, CY ← A + (HL) + CY	×	×	×
		A, [HL + byte]	2	1	4	A, CY ← A + (HL + byte) + CY	×	×	×
		A, [HL + B]	2	1	4	A, CY ← A + (HL + B) + CY	×	×	×
		A, [HL + C]	2	1	4	A, CY ← A + (HL + C) + CY	×	×	×
		A, ES:!addr16	4	2	5	A, CY ← A + (ES, addr16) + CY	×	×	×
		A, ES:[HL]	2	2	5	A, CY ← A + (ES, HL) + CY	×	×	×
		A, ES:[HL + byte]	3	2	5	A, CY ← A + ((ES, HL) + byte) + CY	×	×	×
		A, ES:[HL + B]	3	2	5	A, CY ← A + ((ES, HL) + B) + CY	×	×	×
		A, ES:[HL + C]	3	2	5	A, CY ← A + ((ES, HL) + C) + CY	×	×	×
	SUB	A, #byte	2	1	–	A, CY ← A – byte	×	×	×
		saddr, #byte	3	2	–	(saddr), CY ← (saddr) – byte	×	×	×
		A, r Note 3	2	1	–	A, CY ← A – r	×	×	×
		r, A	2	1	–	r, CY ← r – A	×	×	×
		A, saddr	2	1	–	A, CY ← A – (saddr)	×	×	×
		A, !addr16	3	1	4	A, CY ← A – (addr16)	×	×	×
		A, [HL]	1	1	4	A, CY ← A – (HL)	×	×	×
		A, [HL + byte]	2	1	4	A, CY ← A – (HL + byte)	×	×	×
		A, [HL + B]	2	1	4	A, CY ← A – (HL + B)	×	×	×
		A, [HL + C]	2	1	4	A, CY ← A – (HL + C)	×	×	×
		A, ES:!addr16	4	2	5	A, CY ← A – (ES:addr16)	×	×	×
		A, ES:[HL]	2	2	5	A, CY ← A – (ES:HL)	×	×	×
		A, ES:[HL + byte]	3	2	5	A, CY ← A – ((ES:HL) + byte)	×	×	×
		A, ES:[HL + B]	3	2	5	A, CY ← A – ((ES:HL) + B)	×	×	×
		A, ES:[HL + C]	3	2	5	A, CY ← A – ((ES:HL) + C)	×	×	×

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.
2. When the program memory area is accessed.
3. Except r = A

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).
2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).
3. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (8/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag		
				Note 1	Note 2		Z	AC	CY
8-bit operation	SUBC	A, #byte	2	1	–	A, CY ← A – byte – CY	×	×	×
		saddr, #byte	3	2	–	(saddr), CY ← (saddr) – byte – CY	×	×	×
		A, r Note 3	2	1	–	A, CY ← A – r – CY	×	×	×
		r, A	2	1	–	r, CY ← r – A – CY	×	×	×
		A, saddr	2	1	–	A, CY ← A – (saddr) – CY	×	×	×
		A, !addr16	3	1	4	A, CY ← A – (addr16) – CY	×	×	×
		A, [HL]	1	1	4	A, CY ← A – (HL) – CY	×	×	×
		A, [HL + byte]	2	1	4	A, CY ← A – (HL + byte) – CY	×	×	×
		A, [HL + B]	2	1	4	A, CY ← A – (HL + B) – CY	×	×	×
		A, [HL + C]	2	1	4	A, CY ← A – (HL + C) – CY	×	×	×
		A, ES:!addr16	4	2	5	A, CY ← A – (ES:addr16) – CY	×	×	×
		A, ES:[HL]	2	2	5	A, CY ← A – (ES:HL) – CY	×	×	×
		A, ES:[HL + byte]	3	2	5	A, CY ← A – ((ES:HL) + byte) – CY	×	×	×
		A, ES:[HL + B]	3	2	5	A, CY ← A – ((ES:HL) + B) – CY	×	×	×
		A, ES:[HL + C]	3	2	5	A, CY ← A – ((ES:HL) + C) – CY	×	×	×
	AND	A, #byte	2	1	–	A ← A ∧ byte	×		
		saddr, #byte	3	2	–	(saddr) ← (saddr) ∧ byte	×		
		A, r Note 3	2	1	–	A ← A ∧ r	×		
		r, A	2	1	–	r ← r ∧ A	×		
		A, saddr	2	1	–	A ← A ∧ (saddr)	×		
		A, !addr16	3	1	4	A ← A ∧ (addr16)	×		
		A, [HL]	1	1	4	A ← A ∧ (HL)	×		
		A, [HL + byte]	2	1	4	A ← A ∧ (HL + byte)	×		
		A, [HL + B]	2	1	4	A ← A ∧ (HL + B)	×		
		A, [HL + C]	2	1	4	A ← A ∧ (HL + C)	×		
		A, ES:!addr16	4	2	5	A ← A ∧ (ES:addr16)	×		
		A, ES:[HL]	2	2	5	A ← A ∧ (ES:HL)	×		
		A, ES:[HL + byte]	3	2	5	A ← A ∧ ((ES:HL) + byte)	×		
		A, ES:[HL + B]	3	2	5	A ← A ∧ ((ES:HL) + B)	×		
		A, ES:[HL + C]	3	2	5	A ← A ∧ ((ES:HL) + C)	×		

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.

2. When the program memory area is accessed.

3. Except r = A

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).

2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).

3. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (9/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag
				Note 1	Note 2		Z AC CY
8-bit operation	OR	A, #byte	2	1	–	A ← A ∨ byte	×
		saddr, #byte	3	2	–	(saddr) ← (saddr) ∨ byte	×
		A, r Note 3	2	1	–	A ← A ∨ r	×
		r, A	2	1	–	r ← r ∨ A	×
		A, saddr	2	1	–	A ← A ∨ (saddr)	×
		A, !addr16	3	1	4	A ← A ∨ (addr16)	×
		A, [HL]	1	1	4	A ← A ∨ (HL)	×
		A, [HL + byte]	2	1	4	A ← A ∨ (HL + byte)	×
		A, [HL + B]	2	1	4	A ← A ∨ (HL + B)	×
		A, [HL + C]	2	1	4	A ← A ∨ (HL + C)	×
		A, ES:!addr16	4	2	5	A ← A ∨ (ES:addr16)	×
		A, ES:[HL]	2	2	5	A ← A ∨ (ES:HL)	×
		A, ES:[HL + byte]	3	2	5	A ← A ∨ ((ES:HL) + byte)	×
		A, ES:[HL + B]	3	2	5	A ← A ∨ ((ES:HL) + B)	×
		A, ES:[HL + C]	3	2	5	A ← A ∨ ((ES:HL) + C)	×
	XOR	A, #byte	2	1	–	A ← A ⩛ byte	×
		saddr, #byte	3	2	–	(saddr) ← (saddr) ⩛ byte	×
		A, r Note 3	2	1	–	A ← A ⩛ r	×
		r, A	2	1	–	r ← r ⩛ A	×
		A, saddr	2	1	–	A ← A ⩛ (saddr)	×
		A, !addr16	3	1	4	A ← A ⩛ (addr16)	×
		A, [HL]	1	1	4	A ← A ⩛ (HL)	×
		A, [HL + byte]	2	1	4	A ← A ⩛ (HL + byte)	×
		A, [HL + B]	2	1	4	A ← A ⩛ (HL + B)	×
		A, [HL + C]	2	1	4	A ← A ⩛ (HL + C)	×
		A, ES:!addr16	4	2	5	A ← A ⩛ (ES:addr16)	×
		A, ES:[HL]	2	2	5	A ← A ⩛ (ES:HL)	×
		A, ES:[HL + byte]	3	2	5	A ← A ⩛ ((ES:HL) + byte)	×
		A, ES:[HL + B]	3	2	5	A ← A ⩛ ((ES:HL) + B)	×
		A, ES:[HL + C]	3	2	5	A ← A ⩛ ((ES:HL) + C)	×

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.
2. When the program memory area is accessed.
3. Except r = A

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).
2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).
3. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (10/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag		
				Note 1	Note 2		Z	AC	CY
8-bit operation	CMP	A, #byte	2	1	–	A – byte	×	×	×
		saddr, #byte	3	1	–	(saddr) – byte	×	×	×
		A, r Note 3	2	1	–	A – r	×	×	×
		r, A	2	1	–	r – A	×	×	×
		A, saddr	2	1	–	A – (saddr)	×	×	×
		A, !addr16	3	1	4	A – (addr16)	×	×	×
		A, [HL]	1	1	4	A – (HL)	×	×	×
		A, [HL + byte]	2	1	4	A – (HL + byte)	×	×	×
		A, [HL + B]	2	1	4	A – (HL + B)	×	×	×
		A, [HL + C]	2	1	4	A – (HL + C)	×	×	×
		!addr16, #byte	4	1	4	(addr16) – byte	×	×	×
		A, ES:!addr16	4	2	5	A – (ES:addr16)	×	×	×
		A, ES:[HL]	2	2	5	A – (ES:HL)	×	×	×
		A, ES:[HL + byte]	3	2	5	A – ((ES:HL) + byte)	×	×	×
		A, ES:[HL + B]	3	2	5	A – ((ES:HL) + B)	×	×	×
		A, ES:[HL + C]	3	2	5	A – ((ES:HL) + C)	×	×	×
		ES:!addr16, #byte	5	2	5	(ES:addr16) – byte	×	×	×
	CMP0	A	1	1	–	A – 00H	×	×	×
		X	1	1	–	X – 00H	×	×	×
		B	1	1	–	B – 00H	×	×	×
		C	1	1	–	C – 00H	×	×	×
		saddr	2	1	–	(saddr) – 00H	×	×	×
		!addr16	3	1	4	(addr16) – 00H	×	×	×
		ES:!addr16	4	2	5	(ES:addr16) – 00H	×	×	×
	CMPS	X, [HL + byte]	3	1	4	X – (HL + byte)	×	×	×
		X, ES:[HL + byte]	4	2	5	X – ((ES:HL) + byte)	×	×	×

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.

2. When the program memory area is accessed.

3. Except r = A

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).

2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).

3. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (11/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag		
				Note 1	Note 2		Z	AC	CY
16-bit operation	ADDW	AX, #word	3	1	–	AX, CY ← AX + word	×	×	×
		AX, AX	1	1	–	AX, CY ← AX + AX	×	×	×
		AX, BC	1	1	–	AX, CY ← AX + BC	×	×	×
		AX, DE	1	1	–	AX, CY ← AX + DE	×	×	×
		AX, HL	1	1	–	AX, CY ← AX + HL	×	×	×
		AX, saddrp	2	1	–	AX, CY ← AX + (saddrp)	×	×	×
		AX, !addr16	3	1	4	AX, CY ← AX + (addr16)	×	×	×
		AX, [HL+byte]	3	1	4	AX, CY ← AX + (HL + byte)	×	×	×
		AX, ES:!addr16	4	2	5	AX, CY ← AX + (ES:addr16)	×	×	×
		AX, ES: [HL+byte]	4	2	5	AX, CY ← AX + ((ES:HL) + byte)	×	×	×
	SUBW	AX, #word	3	1	–	AX, CY ← AX – word	×	×	×
		AX, BC	1	1	–	AX, CY ← AX – BC	×	×	×
		AX, DE	1	1	–	AX, CY ← AX – DE	×	×	×
		AX, HL	1	1	–	AX, CY ← AX – HL	×	×	×
		AX, saddrp	2	1	–	AX, CY ← AX – (saddrp)	×	×	×
		AX, !addr16	3	1	4	AX, CY ← AX – (addr16)	×	×	×
		AX, [HL+byte]	3	1	4	AX, CY ← AX – (HL + byte)	×	×	×
		AX, ES:!addr16	4	2	5	AX, CY ← AX – (ES:addr16)	×	×	×
		AX, ES: [HL+byte]	4	2	5	AX, CY ← AX – ((ES:HL) + byte)	×	×	×
	CMPW	AX, #word	3	1	–	AX – word	×	×	×
		AX, BC	1	1	–	AX – BC	×	×	×
		AX, DE	1	1	–	AX – DE	×	×	×
		AX, HL	1	1	–	AX – HL	×	×	×
		AX, saddrp	2	1	–	AX – (saddrp)	×	×	×
		AX, !addr16	3	1	4	AX – (addr16)	×	×	×
		AX, [HL+byte]	3	1	4	AX – (HL + byte)	×	×	×
		AX, ES:!addr16	4	2	5	AX – (ES:addr16)	×	×	×
		AX, ES: [HL+byte]	4	2	5	AX – ((ES:HL) + byte)	×	×	×

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.

2. When the program memory area is accessed.

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).

2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).

3. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (12/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag		
				Note 1	Note 2		Z	AC	CY
Multiply/ Divide/ Multiply & Accumu-late	MULU	X	1	1	–	AX ← A × X			
	MULHU		3	2	–	BCAX ← AX × BC (unsigned)			
	MULH		3	2	–	BCAX ← AX × BC (signed)			
	DIVHU		3	9	–	AX (quotient), DE (remainder) ← AX ÷ DE (unsigned)			
	DIVWU		3	17	–	BCAX (quotient), HLDE (remainder) ← BCAX ÷ HLDE (unsigned)			
	MACHU		3	3	–	MARC ← MARC + AX × BC (unsigned)		×	×
	MACH		3	3	–	MARC ← MARC + AX × BC (signed)		×	×

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.

2. When the program memory area is accessed.

Caution The following multiply/divide/multiply & accumulate instructions are expanded instructions and mounted or not mounted by product. For details, refer to user's manual of each product.

- **MULHU (16-bit multiplication unsigned)**
- **MULH (16-bit multiplication signed)**
- **DIVHU (16-bit division unsigned)**
- **DIVWU (32-bit division unsigned)**
- **MACHU (16-bit multiplication and accumulation unsigned (16 bits × 16 bits) + 32 bits)**
- **MACH (16-bit multiplication and accumulation signed (16 bits × 16 bits) + 32 bits)**

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).

2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).

3. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (13/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag		
				Note 1	Note 2		Z	AC	CY
Increment/ decrement	INC	r	1	1	–	r ← r + 1	×	×	
		saddr	2	2	–	(saddr) ← (saddr) + 1	×	×	
		!addr16	3	2	–	(addr16) ← (addr16) + 1	×	×	
		[HL+byte]	3	2	–	(HL+byte) ← (HL+byte) + 1	×	×	
		ES:!addr16	4	3	–	(ES, addr16) ← (ES, addr16) + 1	×	×	
		ES: [HL+byte]	4	3	–	((ES:HL)+byte) ← ((ES:HL) + byte) + 1	×	×	
	DEC	r	1	1	–	r ← r – 1	×	×	
		saddr	2	2	–	(saddr) ← (saddr) – 1	×	×	
		!addr16	3	2	–	(addr16) ← (addr16) – 1	×	×	
		[HL+byte]	3	2	–	(HL+byte) ← (HL+byte) – 1	×	×	
		ES:!addr16	4	3	–	(ES, addr16) ← (ES, addr16) – 1	×	×	
		ES: [HL+byte]	4	3	–	((ES:HL)+byte) ← ((ES:HL) + byte) – 1	×	×	
	INCW	rp	1	1	–	rp ← rp + 1			
		saddrp	2	2	–	(saddrp) ← (saddrp) + 1			
		!addr16	3	2	–	(addr16) ← (addr16) + 1			
		[HL+byte]	3	2	–	(HL+byte) ← (HL+byte) + 1			
		ES:!addr16	4	3	–	(ES, addr16) ← (ES, addr16) + 1			
		ES: [HL+byte]	4	3	–	((ES:HL)+byte) ← ((ES:HL) + byte) + 1			
	DECW	rp	1	1	–	rp ← rp – 1			
		saddrp	2	2	–	(saddrp) ← (saddrp) – 1			
		!addr16	3	2	–	(addr16) ← (addr16) – 1			
		[HL+byte]	3	2	–	(HL+byte) ← (HL+byte) – 1			
		ES:!addr16	4	3	–	(ES, addr16) ← (ES, addr16) – 1			
		ES: [HL+byte]	4	3	–	((ES:HL)+byte) ← ((ES:HL) + byte) – 1			
Shift	SHR	A, cnt	2	1	–	(CY ← A_0, A_{m-1} ← A_m, A_7 ← 0) × cnt			×
	SHRW	AX, cnt	2	1	–	(CY ← AX_0, AX_{m-1} ← AX_m, AX_{15} ← 0) × cnt			×
	SHL	A, cnt	2	1	–	(CY ← A_7, A_m ← A_{m-1}, A_0 ← 0) × cnt			×
		B, cnt	2	1	–	(CY ← B_7, B_m ← B_{m-1}, B_0 ← 0) × cnt			×
		C, cnt	2	1	–	(CY ← C_7, C_m ← C_{m-1}, C_0 ← 0) × cnt			×
	SHLW	AX, cnt	2	1	–	(CY ← AX_{15}, AX_m ← AX_{m-1}, AX_0 ← 0) × cnt			×
		BC, cnt	2	1	–	(CY ← BC_{15}, BC_m ← BC_{m-1}, BC_0 ← 0) × cnt			×
	SAR	A, cnt	2	1	–	(CY ← A_0, A_{m-1} ← A_m, A_7 ← A_7) × cnt			×
	SARW	AX, cnt	2	1	–	(CY ← AX_0, AX_{m-1} ← AX_m, AX_{15} ← AX_{15}) × cnt			×

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.

2. When the program memory area is accessed.

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).

2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).

3. cnt indicates the bit shift count.

4. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (14/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag		
				Note 1	Note 2		Z	AC	CY
Rotate	ROR	A, 1	2	1	–	(CY, $A_7 \leftarrow A_0$, $A_{m-1} \leftarrow A_m$) × 1			×
	ROL	A, 1	2	1	–	(CY, $A_0 \leftarrow A_7$, $A_{m+1} \leftarrow A_m$) × 1			×
	RORC	A, 1	2	1	–	(CY $\leftarrow A_0$, $A_7 \leftarrow$ CY, $A_{m-1} \leftarrow A_m$) × 1			×
	ROLC	A, 1	2	1	–	(CY $\leftarrow A_7$, $A_0 \leftarrow$ CY, $A_{m+1} \leftarrow A_m$) × 1			×
	ROLWC	AX,1	2	1	–	(CY $\leftarrow AX_{15}$, $AX_0 \leftarrow$ CY, $AX_{m+1} \leftarrow AX_m$) × 1			×
		BC,1	2	1	–	(CY $\leftarrow BC_{15}$, $BC_0 \leftarrow$ CY, $BC_{m+1} \leftarrow BC_m$) × 1			×
Bit manipulate	MOV1	CY, saddr.bit	3	1	–	CY ← (saddr).bit			×
		CY, sfr.bit	3	1	–	CY ← sfr.bit			×
		CY, A.bit	2	1	–	CY ← A.bit			×
		CY, PSW.bit	3	1	–	CY ← PSW.bit			×
		CY,[HL].bit	2	1	4	CY ← (HL).bit			×
		saddr.bit, CY	3	2	–	(saddr).bit ← CY			
		sfr.bit, CY	3	2	–	sfr.bit ← CY			
		A.bit, CY	2	1	–	A.bit ← CY			
		PSW.bit, CY	3	4	–	PSW.bit ← CY	×	×	
		[HL].bit, CY	2	2	–	(HL).bit ← CY			
		CY, ES:[HL].bit	3	2	5	CY ← (ES, HL).bit			×
		ES:[HL].bit, CY	3	3	–	(ES, HL).bit ← CY			
	AND1	CY, saddr.bit	3	1	–	CY ← CY ∧ (saddr).bit			×
		CY, sfr.bit	3	1	–	CY ← CY ∧ sfr.bit			×
		CY, A.bit	2	1	–	CY ← CY ∧ A.bit			×
		CY, PSW.bit	3	1	–	CY ← CY ∧ PSW.bit			×
		CY,[HL].bit	2	1	4	CY ← CY ∧ (HL).bit			×
		CY, ES:[HL].bit	3	2	5	CY ← CY ∧ (ES, HL).bit			×
	OR1	CY, saddr.bit	3	1	–	CY ← CY ∨ (saddr).bit			×
		CY, sfr.bit	3	1	–	CY ← CY ∨ sfr.bit			×
		CY, A.bit	2	1	–	CY ← CY ∨ A.bit			×
		CY, PSW.bit	3	1	–	CY ← CY ∨ PSW.bit			×
		CY, [HL].bit	2	1	4	CY ← CY ∨ (HL).bit			×
		CY, ES:[HL].bit	3	2	5	CY ← CY ∨ (ES, HL).bit			×

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.

2. When the program memory area is accessed.

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).

2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).

3. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (15/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag		
				Note 1	Note 2		Z	AC	CY
Bit manipulate	XOR1	CY, saddr.bit	3	1	–	CY ← CY ⊻ (saddr).bit			×
		CY, sfr.bit	3	1	–	CY ← CY ⊻ sfr.bit			×
		CY, A.bit	2	1	–	CY ← CY ⊻ A.bit			×
		CY, PSW.bit	3	1	–	CY ← CY ⊻ PSW.bit			×
		CY, [HL].bit	2	1	4	CY ← CY ⊻ (HL).bit			×
		CY, ES:[HL].bit	3	2	5	CY ← CY ⊻ (ES, HL).bit			×
	SET1	saddr.bit	3	2	–	(saddr).bit ← 1			
		sfr.bit	3	2	–	sfr.bit ← 1			
		A.bit	2	1	–	A.bit ← 1			
		!addr16.bit	4	2	–	(addr16).bit ← 1			
		PSW.bit	3	4	–	PSW.bit ← 1	×	×	×
		[HL].bit	2	2	–	(HL).bit ← 1			
		ES:!addr16.bit	5	3	–	(ES, addr16).bit ← 1			
		ES:[HL].bit	3	3	–	(ES, HL).bit ← 1			
	CLR1	saddr.bit	3	2	–	(saddr.bit) ← 0			
		sfr.bit	3	2	–	sfr.bit ← 0			
		A.bit	2	1	–	A.bit ← 0			
		!addr16.bit	4	2	–	(addr16).bit ← 0			
		PSW.bit	3	4	–	PSW.bit ← 0	×	×	×
		[HL].bit	2	2	–	(HL).bit ← 0			
		ES:!addr16.bit	5	3	–	(ES, addr16).bit ← 0			
		ES:[HL].bit	3	3	–	(ES, HL).bit ← 0			
	SET1	CY	2	1	–	CY ← 1			1
	CLR1	CY	2	1	–	CY ← 0			0
	NOT1	CY	2	1	–	CY ← $\overline{CY}$			×

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.

2. When the program memory area is accessed.

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by system clock control register (CKC).

2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).

3. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (16/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag		
				Note 1	Note 2		Z	AC	CY
Call/ return	CALL	rp	2	3	–	(SP – 2) ← (PC + 2)S, (SP – 3) ← (PC + 2)H, (SP – 4) ← (PC + 2)L, PC ← CS, rp, SP ← SP – 4			
		$!addr20	3	3	–	(SP – 2) ← (PC + 3)S, (SP – 3) ← (PC + 3)H, (SP – 4) ← (PC + 3)L, PC ← PC + 3 + jdisp16, SP ← SP – 4			
		!addr16	3	3	–	(SP – 2) ← (PC + 3)S, (SP – 3) ← (PC + 3)H, (SP – 4) ← (PC + 3)L, PC ← 0000, addr16, SP ← SP – 4			
		!!addr20	4	3	–	(SP – 2) ← (PC + 4)S, (SP – 3) ← (PC + 4)H, (SP – 4) ← (PC + 4)L, PC ← addr20, SP ← SP – 4			
	CALLT	[addr5]	2	5	–	(SP – 2) ← (PC + 2)S, (SP – 3) ← (PC + 2)H, (SP – 4) ← (PC + 2)L , PCS ← 0000, PCH ← (0000, addr5 + 1), PCL ← (0000, addr5), SP ← SP – 4			
	BRK	–	2	5	–	(SP – 1) ← PSW, (SP – 2) ← (PC + 2)S, (SP – 3) ← (PC + 2)H, (SP – 4) ← (PC + 2)L, PCS ← 0000, PCH ← (0007FH), PCL ← (0007EH), SP ← SP – 4, IE ← 0			
	RET	–	1	6	–	PCL ← (SP), PCH ← (SP + 1), PCS ← (SP + 2), SP ← SP + 4			
	RETI	–	2	6	–	PCL ← (SP), PCH ← (SP + 1), PCS ← (SP + 2), PSW ← (SP + 3), SP ← SP + 4	R	R	R
	RETB	–	2	6	–	PCL ← (SP), PCH ← (SP + 1), PCS ← (SP + 2), PSW ← (SP + 3), SP ← SP + 4	R	R	R

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.

2. When the program memory area is accessed.

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).

2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).

3. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (17/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag
				Note 1	Note 2		Z AC CY
Stack manipulate	PUSH	PSW	2	1	–	(SP – 1) ← PSW, (SP – 2) ← 00H, SP ← SP – 2	
		rp	1	1	–	(SP – 1) ← rp_H, (SP – 2)← rp_L, SP ← SP – 2	
	POP	PSW	2	3	–	PSW ← (SP + 1), SP ← SP + 2	R R R
		rp	1	1	–	rp_L ← (SP), rp_H ← (SP + 1), SP ← SP + 2	
	MOVW	SP, #word	4	1	–	SP ← word	
		SP, AX	2	1	–	SP ← AX	
		AX, SP	2	1	–	AX ← SP	
		HL, SP	3	1	–	HL ← SP	
		BC, SP	3	1	–	BC ← SP	
		DE, SP	3	1	–	DE ← SP	
	ADDW	SP, #byte	2	1	–	SP ← SP + byte	
	SUBW	SP, #byte	2	1	–	SP ← SP – byte	
Unconditional branch	BR	AX	2	3	–	PC ← CS, AX	
		$addr20	2	3	–	PC ← PC + 2 + jdisp8	
		$!addr20	3	3	–	PC ← PC + 3 + jdisp16	
		!addr16	3	3	–	PC ← 0000, addr16	
		!!addr20	4	3	–	PC ← addr20	
Conditional branch	BC	$addr20	2	2/4[Note 3]	–	PC ← PC + 2 + jdisp8 if CY = 1	
	BNC	$addr20	2	2/4[Note 3]	–	PC ← PC + 2 + jdisp8 if CY = 0	
	BZ	$addr20	2	2/4[Note 3]	–	PC ← PC + 2 + jdisp8 if Z = 1	
	BNZ	$addr20	2	2/4[Note 3]	–	PC ← PC + 2 + jdisp8 if Z = 0	
	BH	$addr20	3	2/4[Note 3]	–	PC ← PC+3+jdisp8 if (Z ∨ CY) =0	
	BNH	$addr20	3	2/4[Note 3]	–	PC ← PC+3+jdisp8 if (Z ∨ CY) =1	
	BT	saddr.bit, $addr20	4	3/5[Note 3]	–	PC ← PC + 4 + jdisp8 if (saddr).bit = 1	
		sfr.bit, $addr20	4	3/5[Note 3]	–	PC ← PC + 4 + jdisp8 if sfr.bit = 1	
		A.bit, $addr20	3	3/5[Note 3]	–	PC ← PC + 3 + jdisp8 if A.bit = 1	
		PSW.bit, $addr20	4	3/5[Note 3]	–	PC ← PC + 4 + jdisp8 if PSW.bit = 1	
		[HL].bit, $addr20	3	3/5[Note 3]	6/7	PC ← PC + 3 + jdisp8 if (HL).bit = 1	
		ES:[HL].bit, $addr20	4	4/6[Note 3]	7/8	PC ← PC + 4 + jdisp8 if (ES, HL).bit = 1	

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.

2. When the program memory area is accessed.

3. This indicates the number of clocks "when condition is not met/when condition is met".

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).

2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).

3. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

Table 5-5. Operation List (18/18)

Instruction Group	Mnemonic	Operands	Bytes	Clocks		Operation	Flag		
				Note 1	Note 2		Z	AC	CY
Conditional branch	BF	saddr.bit, $addr20	4	3/5[Note 3]	–	PC ← PC + 4 + jdisp8 if (saddr).bit = 0			
		sfr.bit, $addr20	4	3/5[Note 3]	–	PC ← PC + 4 + jdisp8 if sfr.bit = 0			
		A.bit, $addr20	3	3/5[Note 3]	–	PC ← PC + 3 + jdisp8 if A.bit = 0			
		PSW.bit, $addr20	4	3/5[Note 3]	–	PC ← PC + 4 + jdisp8 if PSW.bit = 0			
		[HL].bit, $addr20	3	3/5[Note 3]	6/7	PC ← PC + 3 + jdisp8 if (HL).bit = 0			
		ES:[HL].bit, $addr20	4	4/6[Note 3]	7/8	PC ← PC + 4 + jdisp8 if (ES, HL).bit = 0			
	BTCLR	saddr.bit, $addr20	4	3/5[Note 3]	–	PC ← PC + 4 + jdisp8 if (saddr).bit = 1 then reset (saddr).bit			
		sfr.bit, $addr20	4	3/5[Note 3]	–	PC ← PC + 4 + jdisp8 if sfr.bit = 1 then reset sfr.bit			
		A.bit, $addr20	3	3/5[Note 3]	–	PC ← PC + 3 + jdisp8 if A.bit = 1 then reset A.bit			
		PSW.bit, $addr20	4	3/5[Note 3]	–	PC ← PC + 4 + jdisp8 if PSW.bit = 1 then reset PSW.bit	×	×	×
		[HL].bit, $addr20	3	3/5[Note 3]	–	PC ← PC + 3 + jdisp8 if (HL).bit = 1 then reset (HL).bit			
		ES:[HL].bit, $addr20	4	4/6[Note 3]	–	PC ← PC + 4 + jdisp8 if (ES, HL).bit = 1 then reset (ES, HL).bit			
Conditional skip	SKC	–	2	1	–	Next instruction skip if CY = 1			
	SKNC	–	2	1	–	Next instruction skip if CY = 0			
	SKZ	–	2	1	–	Next instruction skip if Z = 1			
	SKNZ	–	2	1	–	Next instruction skip if Z = 0			
	SKH	–	2	1	–	Next instruction skip if (Z ∨ CY) = 0			
	SKNH	–	2	1	–	Next instruction skip if (Z ∨ CY) = 1			
CPU control	SEL	RBn	2	1	–	RBS[1:0] ← n			
	NOP	–	1	1	–	No Operation			
	EI	–	3	4	–	IE ← 1(Enable Interrupt)			
	DI	–	3	4	–	IE ← 0(Disable Interrupt)			
	HALT	–	2	3	–	Set HALT Mode			
	STOP	–	2	3	–	Set STOP Mode			

Notes 1. When the internal RAM area, SFR area, or extended SFR area is accessed, or for an instruction with no data access.

2. When the program memory area is accessed.

3. This indicates the number of clocks “when condition is not met/when condition is met”.

Remarks 1. One instruction clock cycle is one cycle of the CPU clock (f_{CLK}) selected by the system clock control register (CKC).

2. This number of clocks is for when the program is in the internal ROM (flash memory) area. When fetching an instruction from the internal RAM area, the number of clocks is twice the number of clocks plus 3, maximum (except when branching to the external memory area).

3. n indicates the number of register banks (n = 0 to 3)

4. In products where the external memory area is adjacent to the internal flash area, the number of waits is added to the number of instruction execution clocks placed in the last address (16-byte max.) in the flash memory, in order to use the external bus interface function. This should be done because, during pre-reading of the instruction code, an external memory wait being inserted due to an external memory area exceeding the flash space is accessed. For the number of waits, refer to **7.2.2 Access to external memory contents as data**.

5.6 Instruction Format

Instructions consist of fixed opcodes followed by operands. Their formats are listed below.

Table 5-6. List of Instruction Formats (1/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
MOV	X, #byte	50	data	–	–	–
	A, #byte	51	data	–	–	–
	C, #byte	52	data	–	–	–
	B, #byte	53	data	–	–	–
	E, #byte	54	data	–	–	–
	D, #byte	55	data	–	–	–
	L, #byte	56	data	–	–	–
	H, #byte	57	data	–	–	–
	saddr, #byte	CD	saddr	data	–	–
	sfr, #byte	CE	sfr	data	–	–
	!addr16,#byte	CF	adrl	adrh	data	–
	A, X	60	–	–	–	–
	A, C	62	–	–	–	–
	A, B	63	–	–	–	–
	A, E	64	–	–	–	–
	A, D	65	–	–	–	–
	A, L	66	–	–	–	–
	A, H	67	–	–	–	–
	X, A	70	–	–	–	–
	C, A	72	–	–	–	–
	B, A	73	–	–	–	–
	E, A	74	–	–	–	–
	D, A	75	–	–	–	–
	L, A	76	–	–	–	–
	H, A	77	–	–	–	–
	A, saddr	8D	saddr	–	–	–
	saddr, A	9D	saddr	–	–	–
	A, sfr	8E	sfr	–	–	–
	sfr, A	9E	sfr	–	–	–
	A, !addr16	8F	adrl	adrh	–	–
	!addr16, A	9F	adrl	adrh	–	–
	PSW, #byte	CE	FA	data	–	–
	A, PSW	8E	FA	–	–	–
	PSW, A	9E	FA	–	–	–
	ES, #byte	41	data	–	–	–
	ES, saddr	61	B8	saddr	–	–
	A, ES	8E	FD	–	–	–
	ES, A	9E	FD	–	–	–
	CS, #byte	CE	FC	data	–	–

Table 5-6. List of Instruction Formats (2/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
MOV	A, CS	8E	FC	–	–	–
	CS, A	9E	FC	–	–	–
	A, [DE]	89	–	–	–	–
	[DE], A	99	–	–	–	–
	[DE+byte],#byte	CA	adr	data	–	–
	A, [DE+byte]	8A	adr	–	–	–
	[DE+byte], A	9A	adr	–	–	–
	A, [HL]	8B	–	–	–	–
	[HL], A	9B	–	–	–	–
	[HL+byte],#byte	CC	adr	data	–	–
	A, [HL+byte]	8C	adr	–	–	–
	[HL+byte], A	9C	adr	–	–	–
	A, [HL+B]	61	C9	–	–	–
	[HL+B], A	61	D9	–	–	–
	A, [HL+C]	61	E9	–	–	–
	[HL+C], A	61	F9	–	–	–
	word[B], #byte	19	adrl	adrh	data	–
	A, word[B]	09	adrl	adrh	–	–
	word[B], A	18	adrl	adrh	–	–
	word[C], #byte	38	adrl	adrh	data	–
	A, word[C]	29	adrl	adrh	–	–
	word[C], A	28	adrl	adrh	–	–
	word[BC], #byte	39	adrl	adrh	data	–
	A, word[BC]	49	adrl	adrh	–	–
	word[BC], A	48	adrl	adrh	–	–
	[SP+byte], #byte	C8	adr	data	–	–
	A, [SP+byte]	88	adr	–	–	–
	[SP+byte], A	98	adr	–	–	–
	B, saddr	E8	saddr	–	–	–
	B, !addr16	E9	adrl	adrh	–	–
	C, saddr	F8	saddr	–	–	–
	C, !addr16	F9	adrl	adrh	–	–
	X, saddr	D8	saddr	–	–	–
	X, !addr16	D9	adrl	adrh	–	–
	ES:!addr16, #byte	11	CF	adrl	adrh	data
	A, ES:!addr16	11	8F	adrl	adrh	–
	ES:!addr16, A	11	9F	adrl	adrh	–
	A, ES:[DE]	11	89	–	–	–
	ES:[DE], A	11	99	–	–	–
	ES:[DE+byte], #byte	11	CA	adr	data	–
	A, ES:[DE+byte]	11	8A	adr	–	–
	ES:[DE+byte], A	11	9A	adr	–	–
	A, ES:[HL]	11	8B	–	–	–

Table 5-6. List of Instruction Formats (3/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
MOV	ES:[HL], A	11	9B	–	–	–
	ES:[HL+byte], #byte	11	CC	adr	data	–
	A, ES:[HL+byte]	11	8C	adr	–	–
	ES:[HL+byte], A	11	9C	adr	–	–
	A, ES:[HL+B]	11	61	C9	–	–
	ES:[HL+B], A	11	61	D9	–	–
	A, ES:[HL+C]	11	61	E9	–	–
	ES:[HL+C], A	11	61	F9	–	–
	ES:word[B], #byte	11	19	adrl	adrh	data
	A, ES:word[B]	11	09	adrl	adrh	–
	ES:word[B], A	11	18	adrl	adrh	–
	ES:word[C], #byte	11	38	adrl	adrh	data
	A, ES:word[C]	11	29	adrl	adrh	–
	ES:word[C], A	11	28	adrl	adrh	–
	ES:word[BC], #byte	11	39	adrl	adrh	data
	A, ES:word[BC]	11	49	adrl	adrh	–
	ES:word[BC], A	11	48	adrl	adrh	–
	B, ES:!addr16	11	E9	adrl	adrh	–
	C, ES:!addr16	11	F9	adrl	adrh	–
	X, ES:!addr16	11	D9	adrl	adrh	–
XCH	A, X	08	–	–	–	–
	A, C	61	8A	–	–	–
	A, B	61	8B	–	–	–
	A, E	61	8C	–	–	–
	A, D	61	8D	–	–	–
	A, L	61	8E	–	–	–
	A, H	61	8F	–	–	–
	A, saddr	61	A8	saddr	–	–
	A, sfr	61	AB	sfr	–	–
	A, !addr16	61	AA	adrl	adrh	–
	A, [DE]	61	AE	–	–	–
	A, [DE+byte]	61	AF	adr	–	–
	A, [HL]	61	AC	–	–	–
	A, [HL+byte]	61	AD	adr	–	–
	A, [HL+B]	61	B9	–	–	–
	A, [HL+C]	61	A9	–	–	–
	A, ES:!addr16	11	61	AA	adrl	adrh
	A, ES: [DE]	11	61	AE	–	–
	A, ES: [DE+byte]	11	61	AF	adr	–
	A, ES: [HL]	11	61	AC	–	–
	A, ES: [HL+byte]	11	61	AD	adr	–
	A, ES: [HL+B]	11	61	B9	–	–
	A, ES: [HL+C]	11	61	A9	–	–

Table 5-6. List of Instruction Formats (4/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
ONEB	A	E1	–	–	–	–
	X	E0	–	–	–	–
	B	E3	–	–	–	–
	C	E2	–	–	–	–
	saddr	E4	saddr	–	–	–
	!addr16	E5	adrl	adrh	–	–
	ES:!addr16	11	E5	adrl	adrh	–
CLRB	A	F1	–	–	–	–
	X	F0	–	–	–	–
	B	F3	–	–	–	–
	C	F2	–	–	–	–
	saddr	F4	saddr	–	–	–
	!addr16	F5	adr1	adrh	–	–
	ES:!addr16	11	F5	adr1	adrh	–
MOVS	[HL+byte], X	61	CE	adr	–	–
	ES: [HL+byte], X	11	61	CE	adr	–
MOVW	AX, #word	30	datal	datah	–	–
	BC, #word	32	datal	datah	–	–
	DE, #word	34	datal	datah	–	–
	HL, #word	36	datal	datah	–	–
	saddrp,#word	C9	saddr	datal	datah	–
	sfrp,#word	CB	sfr	datal	datah	–
	AX, saddrp	AD	saddr	–	–	–
	saddrp, AX	BD	saddr	–	–	–
	AX, sfrp	AE	sfr	–	–	–
	sfrp, AX	BE	sfr	–	–	–
	AX, BC	13	–	–	–	–
	AX, DE	15	–	–	–	–
	AX, HL	17	–	–	–	–
	BC, AX	12	–	–	–	–
	DE, AX	14	–	–	–	–
	HL, AX	16	–	–	–	–
	AX, !addr16	AF	adrl	adrh	–	–
	!addr16, AX	BF	adrl	adrh	–	–
	AX, [DE]	A9	–	–	–	–
	[DE], AX	B9	–	–	–	–
	AX, [DE+byte]	AA	adr	–	–	–
	[DE+byte], AX	BA	adr	–	–	–
	AX, [HL]	AB	–	–	–	–
	[HL], AX	BB	–	–	–	–
	AX, [HL+byte]	AC	adr	–	–	–
	[HL+byte], AX	BC	adr	–	–	–
	AX,word[B]	59	adrl	adrh	–	–

Table 5-6. List of Instruction Formats (5/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
MOVW	word[B], AX	58	adrl	adrh	–	–
	AX,word[C]	69	adrl	adrh	–	–
	word[C], AX	68	adrl	adrh	–	–
	AX,word[BC]	79	adrl	adrh	–	–
	word[BC], AX	78	adrl	adrh	–	–
	AX, [SP+byte]	A8	adr	–	–	–
	[SP+byte], AX	B8	adr	–	–	–
	BC, saddrp	DA	saddr	–	–	–
	BC, !addr16	DB	adrl	adrh	–	–
	DE, saddrp	EA	saddr	–	–	–
	DE, !addr16	EB	adrl	adrh	–	–
	HL, saddrp	FA	saddr	–	–	–
	HL, !addr16	FB	adrl	adrh	–	–
	AX, ES:!addr16	11	AF	adrl	adrh	–
	ES:!addr16, AX	11	BF	adrl	adrh	–
	AX, ES:[DE]	11	A9	–	–	–
	ES:[DE], AX	11	B9	–	–	–
	AX, ES:[DE+byte]	11	A4	adr	–	–
	ES:[DE+byte], AX	11	BA	adr	–	–
	AX, ES:[HL]	11	AB	–	–	–
	ES:[HL], AX	11	BB	–	–	–
	AX, ES:[HL+byte]	11	AC	adr	–	–
	ES:[HL+byte], AX	11	BC	adr	–	–
	AX, ES:word[B]	11	59	adrl	adrh	–
	ES:word[B], AX	11	58	adrl	adrh	–
	AX, ES:word[C]	11	69	adrl	adrh	–
	ES:word[C], AX	11	68	adrl	adrh	–
	AX, ES:word[BC]	11	79	adrl	adrh	–
	ES:word[BC], AX	11	78	adrl	adrh	–
	BC, ES:!addr16	11	DB	adrl	adrh	–
	DE, ES:!addr16	11	EB	adrl	adrh	–
	HL, ES:!addr16	11	FB	adrl	adrh	–
XCHW	AX, BC	33	–	–	–	–
	AX, DE	35	–	–	–	–
	AX, HL	37	–	–	–	–
ONEW	AX	E6	–	–	–	–
	BC	E7	–	–	–	–
CLRW	AX	F6	–	–	–	–
	BC	F7	–	–	–	–

Table 5-6. List of Instruction Formats (6/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
ADD	A, #byte	0C	data	–	–	–
	saddr, #byte	0A	saddr	data	–	–
	A, X	61	08	–	–	–
	A, C	61	0A	–	–	–
	A, B	61	0B	–	–	–
	A, E	61	0C	–	–	–
	A, D	61	0D	–	–	–
	A, L	61	0E	–	–	–
	A, H	61	0F	–	–	–
	X, A	61	00	–	–	–
	A, A	61	01	–	–	–
	C, A	61	02	–	–	–
	B, A	61	03	–	–	–
	E, A	61	04	–	–	–
	D, A	61	05	–	–	–
	L, A	61	06	–	–	–
	H, A	61	07	–	–	–
	A, saddr	0B	saddr	–	–	–
	A, !addr16	0F	adrl	adrh	–	–
	A, [HL]	0D	–	–	–	–
	A, [HL+byte]	0E	adr	–	–	–
	A, [HL+B]	61	80	–	–	–
	A, [HL+C]	61	82	–	–	–
	A, ES:!addr16	11	0F	adrl	adrh	–
	A, ES:[HL]	11	0D	–	–	–
	A, ES:[HL+byte]	11	0E	adr	–	–
	A, ES:[HL+B]	11	61	80	–	–
	A, ES:[HL+C]	11	61	82	–	–
ADDC	A, #byte	1C	data	–	–	–
	saddr, #byte	1A	saddr	data	–	–
	A, X	61	18	–	–	–
	A, C	61	1A	–	–	–
	A, B	61	1B	–	–	–
	A, E	61	1C	–	–	–
	A, D	61	1D	–	–	–
	A, L	61	1E	–	–	–
	A, H	61	1F	–	–	–
	X, A	61	10	–	–	–
	A, A	61	11	–	–	–
	C, A	61	12	–	–	–
	B, A	61	13	–	–	–
	E, A	61	14	–	–	–
	D, A	61	15	–	–	–

Table 5-6. List of Instruction Formats (7/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
ADDC	L, A	61	16	–	–	–
	H, A	61	17	–	–	–
	A, saddr	1B	saddr	–	–	–
	A, !addr16	1F	adrl	adrh	–	–
	A, [HL]	1D	–	–	–	–
	A, [HL+byte]	1E	adr	–	–	–
	A, [HL+B]	61	90	–	–	–
	A, [HL+C]	61	92	–	–	–
	A, ES:!addr16	11	1F	adrl	adrh	–
	A, ES:[HL]	11	1D	–	–	–
	A, ES:[HL+byte]	11	1E	adr	–	–
	A, ES:[HL+B]	11	61	90	–	–
	A, ES:[HL+C]	11	61	92	–	–
SUB	A, #byte	2C	data	–	–	–
	saddr, #byte	2A	saddr	data	–	–
	A, X	61	28	–	–	–
	A, C	61	2A	–	–	–
	A, B	61	2B	–	–	–
	A, E	61	2C	–	–	–
	A, D	61	2D	–	–	–
	A, L	61	2E	–	–	–
	A, H	61	2F	–	–	–
	X, A	61	20	–	–	–
	A, A	61	21	–	–	–
	C, A	61	30	–	–	–
	B, A	61	23	–	–	–
	E, A	61	24	–	–	–
	D, A	61	25	–	–	–
	L, A	61	26	–	–	–
	H, A	61	27	–	–	–
	A, saddr	2B	saddr	–	–	–
	A, !addr16	2F	adrl	adrh	–	–
	A, [HL]	2D	–	–	–	–
	A, [HL+byte]	2E	adr	–	–	–
	A, [HL+B]	61	A0	–	–	–
	A, [HL+C]	61	A2	–	–	–
	A, ES:!addr16	11	2F	adrl	adrh	–
	A, ES:[HL]	11	2D	–	–	–
	A, ES:[HL+byte]	11	2E	adr	–	–
	A, ES:[HL+B]	11	61	A0	–	–
	A, ES:[HL+C]	11	61	A2	–	–

Table 5-6. List of Instruction Formats (8/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
SUBC	A, #byte	3C	data	–	–	–
	saddr, #byte	3A	saddr	data	–	–
	A, X	61	38	–	–	–
	A, C	61	3A	–	–	–
	A, B	61	3B	–	–	–
	A, E	61	3C	–	–	–
	A, D	61	3D	–	–	–
	A, L	61	3E	–	–	–
	A, H	61	3F	–	–	–
	X, A	61	30	–	–	–
	A, A	61	31	–	–	–
	C, A	61	32	–	–	–
	B, A	61	33	–	–	–
	E, A	61	34	–	–	–
	D, A	61	35	–	–	–
	L, A	61	36	–	–	–
	H, A	61	37	–	–	–
	A, saddr	3B	saddr	–	–	–
	A, !addr16	3F	adrl	adrh	–	–
	A, [HL]	3D	–	–	–	–
	A, [HL+byte]	3E	adr	–	–	–
	A, [HL+B]	61	B0	–	–	–
	A, [HL+C]	61	B2	–	–	–
	A, ES:!addr16	11	3F	adrl	adrh	–
	A, ES:[HL]	11	3D	–	–	–
	A, ES:[HL+byte]	11	3E	adr	–	–
	A, ES:[HL+B]	11	61	B0	–	–
	A, ES:[HL+C]	11	61	B2	–	–
AND	A, #byte	5C	data	–	–	–
	saddr, #byte	5A	saddr	data	–	–
	A, X	61	58	–	–	–
	A, C	61	5A	–	–	–
	A, B	61	5B	–	–	–
	A, E	61	5C	–	–	–
	A, D	61	5D	–	–	–
	A, L	61	5E	–	–	–
	A, H	61	5F	–	–	–
	X, A	61	50	–	–	–
	A, A	61	51	–	–	–
	C, A	61	52	–	–	–
	B, A	61	53	–	–	–
	E, A	61	54	–	–	–
	D, A	61	55	–	–	–

Table 5-6. List of Instruction Formats (9/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
AND	L, A	61	56	–	–	–
	H, A	61	57	–	–	–
	A, saddr	5B	saddr	–	–	–
	A, !addr16	5F	adrl	adrh	–	–
	A, [HL]	5D	–	–	–	–
	A, [HL+byte]	5E	adr	–	–	–
	A, [HL+B]	61	D0	–	–	–
	A, [HL+C]	61	D2	–	–	–
	A, ES:!addr16	11	5F	adrl	adrh	–
	A, ES:[HL]	11	5D	–	–	–
	A, ES:[HL+byte]	11	5E	adr	–	–
	A, ES:[HL+B]	11	61	D0	–	–
	A, ES:[HL+C]	11	61	D2	–	–
OR	A, #byte	6C	data	–	–	–
	saddr, #byte	6A	saddr	data	–	–
	A, X	61	68	–	–	–
	A, C	61	6A	–	–	–
	A, B	61	6B	–	–	–
	A, E	61	6C	–	–	–
	A, D	61	6D	–	–	–
	A, L	61	6E	–	–	–
	A, H	61	6F	–	–	–
	X, A	61	60	–	–	–
	A, A	61	61	–	–	–
	C, A	61	62	–	–	–
	B, A	61	63	–	–	–
	E, A	61	64	–	–	–
	D, A	61	65	–	–	–
	L, A	61	66	–	–	–
	H, A	61	67	–	–	–
	A, saddr	6B	saddr	–	–	–
	A, !addr16	6F	adrl	adrh	–	–
	A, [HL]	6D	–	–	–	–
	A, [HL+byte]	6E	adr	–	–	–
	A, [HL+B]	61	E0	–	–	–
	A, [HL+C]	61	E2	–	–	–
	A, ES:!addr16	11	6F	adrl	adrh	–
	A, ES:[HL]	11	6D	–	–	–
	A, ES:[HL+byte]	11	6E	adr	–	–
	A, ES:[HL+B]	11	61	E0	–	–
	A, ES:[HL+C]	11	61	E2	–	–

Table 5-6. List of Instruction Formats (10/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
XOR	A, #byte	7C	data	–	–	–
	saddr, #byte	7A	saddr	data	–	–
	A, X	61	78	–	–	–
	A, C	61	7A	–	–	–
	A, B	61	7B	–	–	–
	A, E	61	7C	–	–	–
	A, D	61	7D	–	–	–
	A, L	61	7E	–	–	–
	A, H	61	7F	–	–	–
	X, A	61	70	–	–	–
	A, A	61	71	–	–	–
	C, A	61	72	–	–	–
	B, A	61	73	–	–	–
	E, A	61	74	–	–	–
	D, A	61	75	–	–	–
	L, A	61	76	–	–	–
	H, A	61	77	–	–	–
	A, saddr	7B	saddr	–	–	–
	A, !addr16	7F	adrl	adrh	–	–
	A, [HL]	7D	–	–	–	–
	A, [HL+byte]	7E	adr	–	–	–
	A, [HL+B]	61	F0	–	–	–
	A, [HL+C]	61	F2	–	–	–
	A, ES:!addr16	11	7F	adrl	adrh	–
	A, ES:[HL]	11	7D	–	–	–
	A, ES:[HL+byte]	11	7E	adr	–	–
	A, ES:[HL+B]	11	61	F0	–	–
	A, ES:[HL+C]	11	61	F2	–	–
CMP	A, #byte	4C	data	–	–	–
	saddr, #byte	4A	saddr	data	–	–
	A, X	61	48	–	–	–
	A, C	61	4A	–	–	–
	A, B	61	4B	–	–	–
	A, E	61	4C	–	–	–
	A, D	61	4D	–	–	–
	A, L	61	4E	–	–	–
	A, H	61	4F	–	–	–
	X, A	61	40	–	–	–
	A, A	61	41	–	–	–
	C, A	61	42	–	–	–
	B, A	61	43	–	–	–
	E, A	61	44	–	–	–
	D, A	61	45	–	–	–

Table 5-6. List of Instruction Formats (11/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
CMP	L, A	61	46	–	–	–
	H, A	61	47	–	–	–
	A, saddr	4B	saddr	–	–	–
	A, !addr16	4F	adrl	adrh	–	–
	A, [HL]	4D	–	–	–	–
	A, [HL+byte]	4E	adr	–	–	–
	A, [HL+B]	61	C0	–	–	–
	A, [HL+C]	61	C2	–	–	–
	!addr16, #byte	40	adrl	adrh	data	–
	A, ES:!addr16	11	4F	adrl	adrh	–
	A, ES:[HL]	11	4D	–	–	–
	A, ES:[HL+byte]	11	4E	adr	–	–
	A, ES:[HL+B]	11	61	C0	–	–
	A, ES:[HL+C]	11	61	C2	–	–
	ES:!addr16, #byte	11	40	adrl	adrh	data
CMP0	A	D1	–	–	–	–
	X	D0	–	–	–	–
	B	D3	–	–	–	–
	C	D2	–	–	–	–
	saddr	D4	saddr	–	–	–
	!addr16	D5	adrl	adrh	–	–
	ES:!addr16	11	D5	adrl	adrh	–
CMPS	X, [HL+byte]	61	DE	adr	–	–
	X, ES:[HL+byte]	11	61	DE	adr	–
ADDW	AX, #word	04	datal	datah	–	–
	AX, AX	01	–	–	–	–
	AX, BC	03	–	–	–	–
	AX, DE	05	–	–	–	–
	AX, HL	07	–	–	–	–
	AX, saddrp	06	saddr	–	–	–
	AX, !addr16	02	adrl	adrh	–	–
	AX, [HL+byte]	61	09	adr	–	–
	AX, ES:!addr16	11	02	adrl	adrh	–
	AX, ES:[HL+byte]	11	61	09	adr	–
SUBW	AX, #word	24	datal	datah	–	–
	AX, BC	23	–	–	–	–
	AX, DE	25	–	–	–	–
	AX, HL	27	–	–	–	–
	AX, saddrp	26	saddr	–	–	–
	AX, !addr16	22	adrl	adrh	–	–
	AX, [HL+byte]	61	29	adr	–	–
	AX, ES:!addr16	11	22	adrl	adrh	–
	AX, ES:[HL+byte]	11	61	29	adr	–

Table 5-6. List of Instruction Formats (12/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
CMPW	AX, #word	44	datal	datah	–	–
	AX, BC	43	–	–	–	–
	AX, DE	45	–	–	–	–
	AX, HL	47	–	–	–	–
	AX, saddrp	46	saddr	–	–	–
	AX, !addr16	42	adrl	adrh	–	–
	AX, [HL+byte]	61	49	adr	–	–
	AX, ES:!addr16	11	42	adrl	adrh	–
	AX, ES:[HL+byte]	11	61	49	adr	–
MULU	X	D6	–	–	–	–
MULHU		CEH	FBH	01H	–	–
MULH		CEH	FBH	02H	–	–
DIVHU		CEH	FBH	03H	–	–
DIVWU		CEH	FBH	0BH	–	–
MACHU		CEH	FBH	05H	–	–
MACH		CEH	FBH	06H	–	–
INC	X	80	–	–	–	–
	A	81	–	–	–	–
	C	82	–	–	–	–
	B	83	–	–	–	–
	E	84	–	–	–	–
	D	85	–	–	–	–
	L	86	–	–	–	–
	H	87	–	–	–	–
	saddr	A4	saddr	–	–	–
	!addr16	A0	adrl	adrh	–	–
	[HL+byte]	61	59	adr	–	–
	ES:!addr16	11	A0	adrl	adrh	–
	ES:[HL+byte]	11	61	59	adr	–
DEC	X	90	–	–	–	–
	A	91	–	–	–	–
	C	92	–	–	–	–
	B	93	–	–	–	–
	E	94	–	–	–	–
	D	95	–	–	–	–
	L	96	–	–	–	–
	H	97	–	–	–	–
	saddr	B4	saddr	–	–	–
	!addr16	B0	adrl	adrh	–	–
	[HL+byte]	61	69	adr	–	–
	ES:!addr16	11	B0	adrl	adrh	–
	ES:[HL+byte]	11	61	69	adr	–

Table 5-6. List of Instruction Formats (13/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
INCW	AX	A1	–	–	–	–
	BC	A3	–	–	–	–
	DE	A5	–	–	–	–
	HL	A7	–	–	–	–
	saddrp	A6	saddr	–	–	–
	!addr16	A2	adrl	adrh	–	–
	[HL+byte]	61	79	adr	–	–
	ES:!addr16	11	A2	adrl	adrh	–
	ES:[HL+byte]	11	61	79	adr	–
DECW	AX	B1	–	–	–	–
	BC	B3	–	–	–	–
	DE	B5	–	–	–	–
	HL	B7	–	–	–	–
	saddrp	B6	saddr	–	–	–
	!addr16	B2	adrl	adrh	–	–
	[HL+byte]	61	89	adr	–	–
	ES:!addr16	11	B2	adrl	adrh	–
	ES:[HL+byte]	11	61	89	adr	–
SHR	A, 1	31	1A	–	–	–
	A, 2	31	2A	–	–	–
	A, 3	31	3A	–	–	–
	A, 4	31	4A	–	–	–
	A, 5	31	5A	–	–	–
	A, 6	31	6A	–	–	–
	A, 7	31	7A	–	–	–
SHRW	AX, 1	31	1E	–	–	–
	AX, 2	31	2E	–	–	–
	AX, 3	31	3E	–	–	–
	AX, 4	31	4E	–	–	–
	AX, 5	31	5E	–	–	–
	AX, 6	31	6E	–	–	–
	AX, 7	31	7E	–	–	–
	AX, 8	31	8E	–	–	–
	AX, 9	31	9E	–	–	–
	AX, 10	31	AE	–	–	–
	AX, 11	31	BE	–	–	–
	AX, 12	31	CE	–	–	–
	AX, 13	31	DE	–	–	–
	AX, 14	31	EE	–	–	–
	AX, 15	31	FE	–	–	–

Table 5-6. List of Instruction Formats (14/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
SHL	A, 1	31	19	–	–	–
	A, 2	31	29	–	–	–
	A, 3	31	39	–	–	–
	A, 4	31	49	–	–	–
	A, 5	31	59	–	–	–
	A, 6	31	69	–	–	–
	A, 7	31	79	–	–	–
	B, 1	31	18	–	–	–
	B, 2	31	28	–	–	–
	B, 3	31	38	–	–	–
	B, 4	31	48	–	–	–
	B, 5	31	58	–	–	–
	B, 6	31	68	–	–	–
	B, 7	31	78	–	–	–
	C, 1	31	17	–	–	–
	C, 2	31	27	–	–	–
	C, 3	31	37	–	–	–
	C, 4	31	47	–	–	–
	C, 5	31	57	–	–	–
	C, 6	31	67	–	–	–
	C, 7	31	77	–	–	–
SHLW	AX, 1	31	1D	–	–	–
	AX, 2	31	2D	–	–	–
	AX, 3	31	3D	–	–	–
	AX, 4	31	4D	–	–	–
	AX, 5	31	5D	–	–	–
	AX, 6	31	6D	–	–	–
	AX, 7	31	7D	–	–	–
	AX, 8	31	8D	–	–	–
	AX, 9	31	9D	–	–	–
	AX, 10	31	AD	–	–	–
	AX, 11	31	BD	–	–	–
	AX, 12	31	CD	–	–	–
	AX, 13	31	DD	–	–	–
	AX, 14	31	ED	–	–	–
	AX, 15	31	FD	–	–	–
	BC, 1	31	1C	–	–	–
	BC, 2	31	2C	–	–	–
	BC, 3	31	3C	–	–	–
	BC, 4	31	4C	–	–	–
	BC, 5	31	5C	–	–	–
	BC, 6	31	6C	–	–	–
	BC, 7	31	7C	–	–	–

Table 5-6. List of Instruction Formats (15/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
SHLW	BC, 8	31	8C	–	–	–
	BC, 9	31	9C	–	–	–
	BC, 10	31	AC	–	–	–
	BC, 11	31	BC	–	–	–
	BC, 12	31	CC	–	–	–
	BC, 13	31	DC	–	–	–
	BC, 14	31	EC	–	–	–
	BC, 15	31	FC	–	–	–
SAR	A, 1	31	1B	–	–	–
	A, 2	31	2B	–	–	–
	A, 3	31	3B	–	–	–
	A, 4	31	4B	–	–	–
	A, 5	31	5B	–	–	–
	A, 6	31	6B	–	–	–
	A, 7	31	7B	–	–	–
SARW	AX, 1	31	1F	–	–	–
	AX, 2	31	2F	–	–	–
	AX, 3	31	3F	–	–	–
	AX, 4	31	4F	–	–	–
	AX, 5	31	5F	–	–	–
	AX, 6	31	6F	–	–	–
	AX, 7	31	7F	–	–	–
	AX, 8	31	8F	–	–	–
	AX, 9	31	9F	–	–	–
	AX, 10	31	AF	–	–	–
	AX, 11	31	BF	–	–	–
	AX, 12	31	CF	–	–	–
	AX, 13	31	DF	–	–	–
	AX, 14	31	EF	–	–	–
	AX, 15	31	FF	–	–	–
ROR	A, 1	61	DB	–	–	–
ROL	A, 1	61	EB	–	–	–
RORC	A, 1	61	FB	–	–	–
ROLC	A, 1	61	DC	–	–	–
ROLWC	AX, 1	61	EE	–	–	–
	BC, 1	61	FE	–	–	–
MOV1	CY, saddr.0	71	04	saddr	–	–
	CY, saddr.1	71	14	saddr	–	–
	CY, saddr.2	71	24	saddr	–	–
	CY, saddr.3	71	34	saddr	–	–
	CY, saddr.4	71	44	saddr	–	–
	CY, saddr.5	71	54	saddr	–	–
	CY, saddr.6	71	64	saddr	–	–

Table 5-6. List of Instruction Formats (16/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
MOV1	CY, saddr.7	71	74	saddr	–	–
	CY, sfr.0	71	0C	sfr	–	–
	CY, sfr.1	71	1C	sfr	–	–
	CY, sfr.2	71	2C	sfr	–	–
	CY, sfr.3	71	3C	sfr	–	–
	CY, sfr.4	71	4C	sfr	–	–
	CY, sfr.5	71	5C	sfr	–	–
	CY, sfr.6	71	6C	sfr	–	–
	CY, sfr.7	71	7C	sfr	–	–
	CY, A.0	71	8C	–	–	–
	CY, A.1	71	9C	–	–	–
	CY, A.2	71	AC	–	–	–
	CY, A.3	71	BC	–	–	–
	CY, A.4	71	CC	–	–	–
	CY, A.5	71	DC	–	–	–
	CY, A.6	71	EC	–	–	–
	CY, A.7	71	FC	–	–	–
	CY, PSW.0	71	0C	FA	–	–
	CY, PSW.1	71	1C	FA	–	–
	CY, PSW.2	71	2C	FA	–	–
	CY, PSW.3	71	3C	FA	–	–
	CY, PSW.4	71	4C	FA	–	–
	CY, PSW.5	71	5C	FA	–	–
	CY, PSW.6	71	6C	FA	–	–
	CY, PSW.7	71	7C	FA	–	–
	CY, [HL].0	71	84	–	–	–
	CY, [HL].1	71	94	–	–	–
	CY, [HL].2	71	A4	–	–	–
	CY, [HL].3	71	B4	–	–	–
	CY, [HL].4	71	C4	–	–	–
	CY, [HL].5	71	D4	–	–	–
	CY, [HL].6	71	E4	–	–	–
	CY, [HL].7	71	F4	–	–	–
	saddr.0, CY	71	01	saddr	–	–
	saddr.1, CY	71	11	saddr	–	–
	saddr.2, CY	71	21	saddr	–	–
	saddr.3, CY	71	31	saddr	–	–
	saddr.4, CY	71	41	saddr	–	–
	saddr.5, CY	71	51	saddr	–	–
	saddr.6, CY	71	61	saddr	–	–
	saddr.7, CY	71	71	saddr	–	–

Table 5-6. List of Instruction Formats (17/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
MOV1	sfr.0. CY	71	09	sfr	–	–
	sfr.1. CY	71	19	sfr	–	–
	sfr.2. CY	71	29	sfr	–	–
	sfr.3. CY	71	39	sfr	–	–
	sfr.4. CY	71	49	sfr	–	–
	sfr.5. CY	71	59	sfr	–	–
	sfr.6. CY	71	69	sfr	–	–
	sfr.7. CY	71	79	sfr	–	–
	A.0, CY	71	89	–	–	–
	A.1, CY	71	99	–	–	–
	A.2, CY	71	A9	–	–	–
	A.3, CY	71	B9	–	–	–
	A.4, CY	71	C9	–	–	–
	A.5, CY	71	D9	–	–	–
	A.6, CY	71	E9	–	–	–
	A.7, CY	71	F9	–	–	–
	PSW.0, CY	71	09	FA	–	–
	PSW.1, CY	71	19	FA	–	–
	PSW.2, CY	71	29	FA	–	–
	PSW.3, CY	71	39	FA	–	–
	PSW.4, CY	71	49	FA	–	–
	PSW.5, CY	71	59	FA	–	–
	PSW.6, CY	71	69	FA	–	–
	PSW.7, CY	71	79	FA	–	–
	[HL].0, CY	71	81	–	–	–
	[HL].1, CY	71	91	–	–	–
	[HL].2, CY	71	A1	–	–	–
	[HL].3, CY	71	B1	–	–	–
	[HL].4, CY	71	C1	–	–	–
	[HL].5, CY	71	D1	–	–	–
	[HL].6, CY	71	E1	–	–	–
	[HL].7, CY	71	F1	–	–	–
	CY, ES:[HL].0	11	71	84	–	–
	CY, ES:[HL].1	11	71	94	–	–
	CY, ES:[HL].2	11	71	A4	–	–
	CY, ES:[HL].3	11	71	B4	–	–
	CY, ES:[HL].4	11	71	C4	–	–
	CY, ES:[HL].5	11	71	D4	–	–
	CY, ES:[HL].6	11	71	E4	–	–
	CY, ES:[HL].7	11	71	F4	–	–

Table 5-6. List of Instruction Formats (18/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
MOV1	ES:[HL].0, CY	11	71	81	–	–
	ES:[HL].1, CY	11	71	91	–	–
	ES:[HL].2, CY	11	71	A1	–	–
	ES:[HL].3, CY	11	71	B1	–	–
	ES:[HL].4, CY	11	71	C1	–	–
	ES:[HL].5, CY	11	71	D1	–	–
	ES:[HL].6, CY	11	71	E1	–	–
	ES:[HL].7, CY	11	71	F1	–	–
AND1	CY, saddr.0	71	05	saddr	–	–
	CY, saddr.1	71	15	saddr	–	–
	CY, saddr.2	71	25	saddr	–	–
	CY, saddr.3	71	35	saddr	–	–
	CY, saddr.4	71	45	saddr	–	–
	CY, saddr.5	71	55	saddr	–	–
	CY, saddr.6	71	65	saddr	–	–
	CY, saddr.7	71	75	saddr	–	–
	CY, sfr.0	71	0D	sfr	–	–
	CY, sfr.1	71	1D	sfr	–	–
	CY, sfr.2	71	2D	sfr	–	–
	CY, sfr.3	71	3D	sfr	–	–
	CY, sfr.4	71	4D	sfr	–	–
	CY, sfr.5	71	5D	sfr	–	–
	CY, sfr.6	71	6D	sfr	–	–
	CY, sfr.7	71	7D	sfr	–	–
	CY, A.0	71	8D	–	–	–
	CY, A.1	71	9D	–	–	–
	CY, A.2	71	AD	–	–	–
	CY, A.3	71	BD	–	–	–
	CY, A.4	71	CD	–	–	–
	CY, A.5	71	DD	–	–	–
	CY, A.6	71	ED	–	–	–
	CY, A.7	71	FD	–	–	–
	CY, PSW.0	71	0D	FA	–	–
	CY, PSW.1	71	1D	FA	–	–
	CY, PSW.2	71	2D	FA	–	–
	CY, PSW.3	71	3D	FA	–	–
	CY, PSW.4	71	4D	FA	–	–
	CY, PSW.5	71	5D	FA	–	–
	CY, PSW.6	71	6D	FA	–	–
	CY, PSW.7	71	7D	FA	–	–

Table 5-6. List of Instruction Formats (19/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
AND1	CY, [HL].0	71	85	–	–	–
	CY, [HL].1	71	95	–	–	–
	CY, [HL].2	71	A5	–	–	–
	CY, [HL].3	71	B5	–	–	–
	CY, [HL].4	71	C5	–	–	–
	CY, [HL].5	71	D5	–	–	–
	CY, [HL].6	71	E5	–	–	–
	CY, [HL].7	71	F5	–	–	–
	CY, ES:[HL].0	11	71	85	–	–
	CY, ES:[HL].1	11	71	95	–	–
	CY, ES:[HL].2	11	71	A5	–	–
	CY, ES:[HL].3	11	71	B5	–	–
	CY, ES:[HL].4	11	71	C5	–	–
	CY, ES:[HL].5	11	71	D5	–	–
	CY, ES:[HL].6	11	71	E5	–	–
	CY, ES:[HL].7	11	71	F5	–	–
OR1	CY, saddr.0	71	06	saddr	–	–
	CY, saddr.1	71	16	saddr	–	–
	CY, saddr.2	71	26	saddr	–	–
	CY, saddr.3	71	36	saddr	–	–
	CY, saddr.4	71	46	saddr	–	–
	CY, saddr.5	71	56	saddr	–	–
	CY, saddr.6	71	66	saddr	–	–
	CY, saddr.7	71	76	saddr	–	–
	CY, sfr.0	71	0E	sfr	–	–
	CY, sfr.1	71	1E	sfr	–	–
	CY, sfr.2	71	2E	sfr	–	–
	CY, sfr.3	71	3E	sfr	–	–
	CY, sfr.4	71	4E	sfr	–	–
	CY, sfr.5	71	5E	sfr	–	–
	CY, sfr.6	71	6E	sfr	–	–
	CY, sfr.7	71	7E	sfr	–	–
	CY, A.0	71	8E	–	–	–
	CY, A.1	71	9E	–	–	–
	CY, A.2	71	AE	–	–	–
	CY, A.3	71	BE	–	–	–
	CY, A.4	71	CE	–	–	–
	CY, A.5	71	DE	–	–	–
	CY, A.6	71	EE	–	–	–
	CY, A.7	71	FE	–	–	–

Table 5-6. List of Instruction Formats (20/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
OR1	CY, PSW.0	71	0E	FA	–	–
	CY, PSW.1	71	1E	FA	–	–
	CY, PSW.2	71	2E	FA	–	–
	CY, PSW.3	71	3E	FA	–	–
	CY, PSW.4	71	4E	FA	–	–
	CY, PSW.5	71	5E	FA	–	–
	CY, PSW.6	71	6E	FA	–	–
	CY, PSW.7	71	7E	FA	–	–
	CY, [HL].0	71	86	–	–	–
	CY, [HL].1	71	96	–	–	–
	CY, [HL].2	71	A6	–	–	–
	CY, [HL].3	71	B6	–	–	–
	CY, [HL].4	71	C6	–	–	–
	CY, [HL].5	71	D6	–	–	–
	CY, [HL].6	71	E6	–	–	–
	CY, [HL].7	71	F6	–	–	–
	CY, ES:[HL].0	11	71	86	–	–
	CY, ES:[HL].1	11	71	96	–	–
	CY, ES:[HL].2	11	71	A6	–	–
	CY, ES:[HL].3	11	71	B6	–	–
	CY, ES:[HL].4	11	71	C6	–	–
	CY, ES:[HL].5	11	71	D6	–	–
	CY, ES:[HL].6	11	71	E6	–	–
	CY, ES:[HL].7	11	71	F6	–	–
XOR1	CY, saddr.0	71	07	saddr	–	–
	CY, saddr.1	71	17	saddr	–	–
	CY, saddr.2	71	27	saddr	–	–
	CY, saddr.3	71	37	saddr	–	–
	CY, saddr.4	71	47	saddr	–	–
	CY, saddr.5	71	57	saddr	–	–
	CY, saddr.6	71	67	saddr	–	–
	CY, saddr.7	71	77	saddr	–	–
	CY, sfr.0	71	0F	sfr	–	–
	CY, sfr.1	71	1F	sfr	–	–
	CY, sfr.2	71	2F	sfr	–	–
	CY, sfr.3	71	3F	sfr	–	–
	CY, sfr.4	71	4F	sfr	–	–
	CY, sfr.5	71	5F	sfr	–	–
	CY, sfr.6	71	6F	sfr	–	–
	CY, sfr.7	71	7F	sfr	–	–

Table 5-6. List of Instruction Formats (21/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
XOR1	CY, A.0	71	8F	–	–	–
	CY, A.1	71	9F	–	–	–
	CY, A.2	71	AF	–	–	–
	CY, A.3	71	BF	–	–	–
	CY, A.4	71	CF	–	–	–
	CY, A.5	71	DF	–	–	–
	CY, A.6	71	EF	–	–	–
	CY, A.7	71	FF	–	–	–
	CY, PSW.0	71	0F	FA	–	–
	CY, PSW.1	71	1F	FA	–	–
	CY, PSW.2	71	2F	FA	–	–
	CY, PSW.3	71	3F	FA	–	–
	CY, PSW.4	71	4F	FA	–	–
	CY, PSW.5	71	5F	FA	–	–
	CY, PSW.6	71	6F	FA	–	–
	CY, PSW.7	71	7F	FA	–	–
	CY, [HL].0	71	87	–	–	–
	CY, [HL].1	71	97	–	–	–
	CY, [HL].2	71	A7	–	–	–
	CY, [HL].3	71	B7	–	–	–
	CY, [HL].4	71	C7	–	–	–
	CY, [HL].5	71	D7	–	–	–
	CY, [HL].6	71	E7	–	–	–
	CY, [HL].7	71	F7	–	–	–
	CY, ES:[HL].0	11	71	87	–	–
	CY, ES:[HL].1	11	71	97	–	–
	CY, ES:[HL].2	11	71	A7	–	–
	CY, ES:[HL].3	11	71	B7	–	–
	CY, ES:[HL].4	11	71	C7	–	–
	CY, ES:[HL].5	11	71	D7	–	–
	CY, ES:[HL].6	11	71	E7	–	–
	CY, ES:[HL].7	11	71	F7	–	–
SET1	saddr.0	71	02	saddr	–	–
	saddr.1	71	12	saddr	–	–
	saddr.2	71	22	saddr	–	–
	saddr.3	71	32	saddr	–	–
	saddr.4	71	42	saddr	–	–
	saddr.5	71	52	saddr	–	–
	saddr.6	71	62	saddr	–	–
	saddr.7	71	72	saddr	–	–

Table 5-6. List of Instruction Formats (22/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
SET1	sfr.0	71	0A	sfr	–	–
	sfr.1	71	1A	sfr	–	–
	sfr.2	71	2A	sfr	–	–
	sfr.3	71	3A	sfr	–	–
	sfr.4	71	4A	sfr	–	–
	sfr.5	71	5A	sfr	–	–
	sfr.6	71	6A	sfr	–	–
	sfr.7	71	7A	sfr	–	–
	A.0	71	8A	–	–	–
	A.1	71	9A	–	–	–
	A.2	71	AA	–	–	–
	A.3	71	BA	–	–	–
	A.4	71	CA	–	–	–
	A.5	71	DA	–	–	–
	A.6	71	EA	–	–	–
	A.7	71	FA	–	–	–
	!addr16.0	71	00	adrl	adrh	–
	!addr16.1	71	10	adrl	adrh	–
	!addr16.2	71	20	adrl	adrh	–
	!addr16.3	71	30	adrl	adrh	–
	!addr16.4	71	40	adrl	adrh	–
	!addr16.5	71	50	adrl	adrh	–
	!addr16.6	71	60	adrl	adrh	–
	!addr16.7	71	70	adrl	adrh	–
	PSW.0	71	0A	FA	–	–
	PSW.1	71	1A	FA	–	–
	PSW.2	71	2A	FA	–	–
	PSW.3	71	3A	FA	–	–
	PSW.4	71	4A	FA	–	–
	PSW.5	71	5A	FA	–	–
	PSW.6	71	6A	FA	–	–
	PSW.7	71	7A	FA	–	–
	[HL].0	71	82	–	–	–
	[HL].1	71	92	–	–	–
	[HL].2	71	A2	–	–	–
	[HL].3	71	B2	–	–	–
	[HL].4	71	C2	–	–	–
	[HL].5	71	D2	–	–	–
	[HL].6	71	E2	–	–	–
	[HL].7	71	F2	–	–	–

Table 5-6. List of Instruction Formats (23/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
SET1	ES:!addr16.0	11	71	00	adrl	adrh
	ES:!addr16.1	11	71	10	adrl	adrh
	ES:!addr16.2	11	71	20	adrl	adrh
	ES:!addr16.3	11	71	30	adrl	adrh
	ES:!addr16.4	11	71	40	adrl	adrh
	ES:!addr16.5	11	71	50	adrl	adrh
	ES:!addr16.6	11	71	60	adrl	adrh
	ES:!addr16.7	11	71	70	adrl	adrh
	ES:[HL].0	11	71	82	–	–
	ES:[HL].1	11	71	92	–	–
	ES:[HL].2	11	71	A2	–	–
	ES:[HL].3	11	71	B2	–	–
	ES:[HL].4	11	71	C2	–	–
	ES:[HL].5	11	71	D2	–	–
	ES:[HL].6	11	71	E2	–	–
	ES:[HL].7	11	71	F2	–	–
CLR1	saddr.0	71	03	saddr	–	–
	saddr.1	71	13	saddr	–	–
	saddr.2	71	23	saddr	–	–
	saddr.3	71	33	saddr	–	–
	saddr.4	71	43	saddr	–	–
	saddr.5	71	53	saddr	–	–
	saddr.6	71	63	saddr	–	–
	saddr.7	71	73	saddr	–	–
	sfr.0	71	0B	sfr	–	–
	sfr.1	71	1B	sfr	–	–
	sfr.2	71	2B	sfr	–	–
	sfr.3	71	3B	sfr	–	–
	sfr.4	71	4B	sfr	–	–
	sfr.5	71	5B	sfr	–	–
	sfr.6	71	6B	sfr	–	–
	sfr.7	71	7B	sfr	–	–
	A.0	71	8B	–	–	–
	A.1	71	9B	–	–	–
	A.2	71	AB	–	–	–
	A.3	71	BB	–	–	–
	A.4	71	CB	–	–	–
	A.5	71	DB	–	–	–
	A.6	71	EB	–	–	–
	A.7	71	FB	–	–	–

Table 5-6. List of Instruction Formats (24/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
CLR1	!addr16.0	71	08	adrl	adrh	–
	!addr16.1	71	18	adrl	adrh	–
	!addr16.2	71	28	adrl	adrh	–
	!addr16.3	71	38	adrl	adrh	–
	!addr16.4	71	48	adrl	adrh	–
	!addr16.5	71	58	adrl	adrh	–
	!addr16.6	71	68	adrl	adrh	–
	!addr16.7	71	78	adrl	adrh	–
	PSW.0	71	0B	FA	–	–
	PSW.1	71	1B	FA	–	–
	PSW.2	71	2B	FA	–	–
	PSW.3	71	3B	FA	–	–
	PSW.4	71	4B	FA	–	–
	PSW.5	71	5B	FA	–	–
	PSW.6	71	6B	FA	–	–
	PSW.7	71	7B	FA	–	–
	[HL].0	71	83	–	–	–
	[HL].1	71	93	–	–	–
	[HL].2	71	A3	–	–	–
	[HL].3	71	B3	–	–	–
	[HL].4	71	C3	–	–	–
	[HL].5	71	D3	–	–	–
	[HL].6	71	E3	–	–	–
	[HL].7	71	F3	–	–	–
	ES:!addr16.0	11	71	08	adrl	adrh
	ES:!addr16.1	11	71	18	adrl	adrh
	ES:!addr16.2	11	71	28	adrl	adrh
	ES:!addr16.3	11	71	38	adrl	adrh
	ES:!addr16.4	11	71	48	adrl	adrh
	ES:!addr16.5	11	71	58	adrl	adrh
	ES:!addr16.6	11	71	68	adrl	adrh
	ES:!addr16.7	11	71	78	adrl	adrh
	ES:[HL].0	11	71	83	–	–
	ES:[HL].1	11	71	93	–	–
	ES:[HL].2	11	71	A3	–	–
	ES:[HL].3	11	71	B3	–	–
	ES:[HL].4	11	71	C3	–	–
	ES:[HL].5	11	71	D3	–	–
	ES:[HL].6	11	71	E3	–	–
	ES:[HL].7	11	71	F3	–	–
SET1	CY	71	80	–	–	–
CLR1	CY	71	88	–	–	–
NOT1	CY	71	C0	–	–	–

Table 5-6. List of Instruction Formats (25/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
CALL	AX	61	CA	–	–	–
	BC	61	DA	–	–	–
	DE	61	EA	–	–	–
	HL	61	FA	–	–	–
	$!addr20	FE	adrl	adrh	–	–
	!addr16	FD	adrl	adrh	–	–
	!!addr20	FC	adrl	adrh	adrs	–
CALLT	[0080h]	61	84	–	–	–
	[0082h]	61	94	–	–	–
	[0084h]	61	A4	–	–	–
	[0086h]	61	B4	–	–	–
	[0088h]	61	C4	–	–	–
	[008Ah]	61	D4	–	–	–
	[008Ch]	61	E4	–	–	–
	[008Eh]	61	F4	–	–	–
	[0090h]	61	85	–	–	–
	[0092h]	61	95	–	–	–
	[0094h]	61	A5	–	–	–
	[0096h]	61	B5	–	–	–
	[0098h]	61	C5	–	–	–
	[009Ah]	61	D5	–	–	–
	[009Ch]	61	E5	–	–	–
	[009Eh]	61	F5	–	–	–
	[00A0h]	61	86	–	–	–
	[00A2h]	61	96	–	–	–
	[00A4h]	61	A6	–	–	–
	[00A6h]	61	B6	–	–	–
	[00A8h]	61	C6	–	–	–
	[00AAh]	61	D6	–	–	–
	[00ACh]	61	E6	–	–	–
	[00AEh]	61	F6	–	–	–
	[00B0h]	61	87	–	–	–
	[00B2h]	61	97	–	–	–
	[00B4h]	61	A7	–	–	–
	[00B6h]	61	B7	–	–	–
	[00B8h]	61	C7	–	–	–
	[00BAh]	61	D7	–	–	–
	[00BCh]	61	E7	–	–	–
	[00BEh]	61	F7	–	–	–
BRK	–	61	CC	–	–	–
RET	–	D7	–	–	–	–
RETI	–	61	FC	–	–	–
RETB	–	61	EC	–	–	–

Table 5-6. List of Instruction Formats (26/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
PUSH	PSW	61	DD	–	–	–
	AX	C1	–	–	–	–
	BC	C3	–	–	–	–
	DE	C5	–	–	–	–
	HL	C7	–	–	–	–
POP	PSW	61	CD	–	–	–
	AX	C0	–	–	–	–
	BC	C2	–	–	–	–
	DE	C4	–	–	–	–
	HL	C6	–	–	–	–
MOVW	SP, #word	CB	F8	datal	datah	–
	SP, AX	BE	F8	–	–	–
	AX, SP	AE	F8	–	–	–
	BC, SP	DB	adrl	adrh	–	–
	DE, SP	EB	adrl	adrh	–	–
	HL, SP	FB	adrl	adrh	–	–
ADDW	SP, #byte	10	data	–	–	–
SUBW	SP, #byte	20	data	–	–	–
BR	AX	61	CB	–	–	–
	$addr20	EF	adr	–	–	–
	$!addr20	EE	adrl	adrh	–	–
	!addr16	ED	adrl	adrh	–	–
	!!addr20	EC	adrl	adrh	adrs	–
BC	$addr20	DC	adr	–	–	–
BNC	$addr20	DE	adr	–	–	–
BZ	$addr20	DD	adr	–	–	–
BNZ	$addr20	DF	adr	–	–	–
BH	$addr20	61	C3	adr	–	–
BNH	$addr20	61	D3	adr	–	–
BT	saddr.0, $addr20	31	02	saddr	adr	–
	saddr.1, $addr20	31	12	saddr	adr	–
	saddr.2, $addr20	31	22	saddr	adr	–
	saddr.3, $addr20	31	32	saddr	adr	–
	saddr.4, $addr20	31	42	saddr	adr	–
	saddr.5, $addr20	31	52	saddr	adr	–
	saddr.6, $addr20	31	62	saddr	adr	–
	saddr.7, $addr20	31	72	saddr	adr	–
	sfr.0, $addr20	31	82	sfr	adr	–
	sfr.1, $addr20	31	92	sfr	adr	–
	sfr.2, $addr20	31	A2	sfr	adr	–
	sfr.3, $addr20	31	B2	sfr	adr	–
	sfr.4, $addr20	31	C2	sfr	adr	–

Table 5-6. List of Instruction Formats (27/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
BT	sfr.5, $addr20	31	D2	sfr	adr	–
	sfr.6, $addr20	31	E2	sfr	adr	–
	sfr.7, $addr20	31	F2	sfr	adr	–
	A.0, $addr20	31	03	adr	–	–
	A.1, $addr20	31	13	adr	–	–
	A.2, $addr20	31	23	adr	–	–
	A.3, $addr20	31	33	adr	–	–
	A.4, $addr20	31	43	adr	–	–
	A.5, $addr20	31	53	adr	–	–
	A.6, $addr20	31	63	adr	–	–
	A.7, $addr20	31	73	adr	–	–
	PSW.0, $addr20	31	82	FA	adr	–
	PSW.1, $addr20	31	92	FA	adr	–
	PSW.2, $addr20	31	A2	FA	adr	–
	PSW.3, $addr20	31	B2	FA	adr	–
	PSW.4, $addr20	31	C2	FA	adr	–
	PSW.5, $addr20	31	D2	FA	adr	–
	PSW.6, $addr20	31	E2	FA	adr	–
	PSW.7, $addr20	31	F2	FA	adr	–
	[HL].0, $addr20	31	83	adr	–	–
	[HL].1, $addr20	31	93	adr	–	–
	[HL].2, $addr20	31	A3	adr	–	–
	[HL].3, $addr20	31	B3	adr	–	–
	[HL].4, $addr20	31	C3	adr	–	–
	[HL].5, $addr20	31	D3	adr	–	–
	[HL].6, $addr20	31	E3	adr	–	–
	[HL].7, $addr20	31	F3	adr	–	–
	ES:[HL].0, $addr20	11	31	83	adr	–
	ES:[HL].1, $addr20	11	31	93	adr	–
	ES:[HL].2, $addr20	11	31	A3	adr	–
	ES:[HL].3, $addr20	11	31	B3	adr	–
	ES:[HL].4, $addr20	11	31	C3	adr	–
	ES:[HL].5, $addr20	11	31	D3	adr	–
	ES:[HL].6, $addr20	11	31	E3	adr	–
	ES:[HL].7, $addr20	11	31	F3	adr	–
BF	saddr.0, $addr20	31	04	saddr	adr	–
	saddr.1, $addr20	31	14	saddr	adr	–
	saddr.2, $addr20	31	24	saddr	adr	–
	saddr.3, $addr20	31	34	saddr	adr	–
	saddr.4, $addr20	31	44	saddr	adr	–
	saddr.5, $addr20	31	54	saddr	adr	–
	saddr.6, $addr20	31	64	saddr	adr	–
	saddr.7,$addr20	31	74	saddr	adr	–

Table 5-6. List of Instruction Formats (28/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
BF	sfr.0, $addr20	31	84	sfr	adr	–
	sfr.1, $addr20	31	94	sfr	adr	–
	sfr.2, $addr20	31	A4	sfr	adr	–
	sfr.3,$addr20	31	B4	sfr	adr	–
	sfr.4, $addr20	31	C4	sfr	adr	–
	sfr.5, $addr20	31	D4	sfr	adr	–
	sfr.6,$addr20	31	E4	sfr	adr	–
	sfr.7, $addr20	31	F4	sfr	adr	–
	A.0, $addr20	31	05	adr	–	–
	A.1, $addr20	31	15	adr	–	–
	A.2, $addr20	31	25	adr	–	–
	A.3, $addr20	31	35	adr	–	–
	A.4, $addr20	31	45	adr	–	–
	A.5, $addr20	31	55	adr	–	–
	A.6, $addr20	31	65	adr	–	–
	A.7, $addr20	31	75	adr	–	–
	PSW.0, $addr20	31	84	FA	adr	–
	PSW.1, $addr20	31	94	FA	adr	–
	PSW.2, $addr20	31	A4	FA	adr	–
	PSW.3, $addr20	31	B4	FA	adr	–
	PSW.4, $addr20	31	C4	FA	adr	–
	PSW.5, $addr20	31	D4	FA	adr	–
	PSW.6, $addr20	31	E4	FA	adr	–
	PSW.7, $addr20	31	F4	FA	adr	–
	[HL].0, $addr20	31	85	adr	–	–
	[HL].1, $addr20	31	95	adr	–	–
	[HL].2, $addr20	31	A5	adr	–	–
	[HL].3, $addr20	31	B5	adr	–	–
	[HL].4, $addr20	31	C5	adr	–	–
	[HL].5, $addr20	31	D5	adr	–	–
	[HL].6, $addr20	31	E5	adr	–	–
	[HL].7, $addr20	31	F5	adr	–	–
	ES:[HL].0, $addr20	11	31	85	adr	–
	ES:[HL].1, $addr20	11	31	95	adr	–
	ES:[HL].2, $addr20	11	31	A5	adr	–
	ES:[HL].3, $addr20	11	31	B5	adr	–
	ES:[HL].4, $addr20	11	31	C5	adr	–
	ES:[HL].5, $addr20	11	31	D5	adr	–
	ES:[HL].6, $addr20	11	31	E5	adr	–
	ES:[HL].7, $addr20	11	31	F5	adr	–

Table 5-6. List of Instruction Formats (29/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
BTCLR	saddr.0, $addr20	31	00	saddr	adr	–
	saddr.1, $addr20	31	10	saddr	adr	–
	saddr.2, $addr20	31	20	saddr	adr	–
	saddr.3, $addr20	31	30	saddr	adr	–
	saddr.4, $addr20	31	40	saddr	adr	–
	saddr.5, $addr20	31	50	saddr	adr	–
	saddr.6, $addr20	31	60	saddr	adr	–
	saddr.7, $addr20	31	70	saddr	adr	–
	sfr.0, $addr20	31	80	sfr	adr	–
	sfr.1, $addr20	31	90	sfr	adr	–
	sfr.2, $addr20	31	A0	sfr	adr	–
	sfr.3, $addr20	31	B0	sfr	adr	–
	sfr.4, $addr20	31	C0	sfr	adr	–
	sfr.5, $addr20	31	D0	sfr	adr	–
	sfr.6, $addr20	31	E0	sfr	adr	–
	sfr.7, $addr20	31	F0	sfr	adr	–
	A.0, $addr20	31	01	adr	–	–
	A.1, $addr20	31	11	adr	–	–
	A.2, $addr20	31	21	adr	–	–
	A.3, $addr20	31	31	adr	–	–
	A.4, $addr20	31	41	adr	–	–
	A.5, $addr20	31	51	adr	–	–
	A.6, $addr20	31	61	adr	–	–
	A.7, $addr20	31	71	adr	–	–
	PSW.0, $addr20	31	80	FA	adr	–
	PSW.1, $addr20	31	90	FA	adr	–
	PSW.2, $addr20	31	A0	FA	adr	–
	PSW.3, $addr20	31	B0	FA	adr	–
	PSW.4, $addr20	31	C0	FA	adr	–
	PSW.5, $addr20	31	D0	FA	adr	–
	PSW.6, $addr20	31	E0	FA	adr	–
	PSW.7, $addr20	31	F0	FA	adr	–
	[HL].0, $addr20	31	81	adr	–	–
	[HL].1, $addr20	31	91	adr	–	–
	[HL].2, $addr20	31	A1	adr	–	–
	[HL].3, $addr20	31	B1	adr	–	–
	[HL].4, $addr20	31	C1	adr	–	–
	[HL].5, $addr20	31	D1	adr	–	–
	[HL].6, $addr20	31	E1	adr	–	–
	[HL].7, $addr20	31	F1	adr	–	–

Table 5-6. List of Instruction Formats (30/30)

Mnemonic	Operands	Opcode				
		1st	2nd	3rd	4th	5th
BTCLR	ES:[HL].0, $addr20	11	31	81	adr	–
	ES:[HL].1, $addr20	11	31	91	adr	–
	ES:[HL].2, $addr20	11	31	A1	adr	–
	ES:[HL].3, $addr20	11	31	B1	adr	–
	ES:[HL].4, $addr20	11	31	C1	adr	–
	ES:[HL].5, $addr20	11	31	D1	adr	–
	ES:[HL].6, $addr20	11	31	E1	adr	–
	ES:[HL].7, $addr20	11	31	F1	adr	–
SKC	–	61	C8	–	–	–
SKNC	–	61	D8	–	–	–
SKZ	–	61	E8	–	–	–
SKNZ	–	61	F8	–	–	–
SKH	–	61	E3	–	–	–
SKNH	–	61	F3	–	–	–
SEL	RB0	61	CF	–	–	–
	RB1	61	DF	–	–	–
	RB2	61	EF	–	–	–
	RB3	61	FF	–	–	–
NOP	–	00	–	–	–	–
EI	–	71	7A	FA	–	–
DI	–	71	7B	FA	–	–
HALT	–	61	ED	–	–	–
STOP	–	61	FD	–	–	–
PREFIX	–	11	–	–	–	–

5.7 Instruction Maps

Tables 5-7 to 5-10 show instruction maps.

Table 5-7. Instruction Map (1st MAP)

	0(low)	1(low)	2(low)	3(low)	4(low)	5(low)	6(low)	7(low)	8(low)	9(low)	a(low)	b(low)	c(low)	d(low)	e(low)	f(low)
0	NOP	ADDW AX,AX	ADDW AX,!addr16	ADDW AX,BC	ADDW AX,#word	ADDW AX,DE	ADDW AX,saddrp	ADDW AX,HL	XCH A,X	MOV A,word[B]	ADD saddr,#byte	ADD A,saddr	ADD A,#byte	ADD A,[HL]	ADD A,[HL+byte]	ADD A,!addr16
1	ADDW SP,#byte	PREFIX	MOVW BC,AX	MOVW AX,BC	MOVW DE,AX	MOVW AX,DE	MOVW HL,AX	MOVW AX,HL	MOV word[B],A	MOV word[B],#byte	ADDC saddr,#byte	ADDC A,saddr	ADDC A,#byte	ADDC A,[HL]	ADDC A,[HL+byte]	ADDC A,!addr16
2	SUBW SP,#byte		SUBW AX,!addr16	SUBW AX,BC	SUBW AX,#word	SUBW AX,DE	SUBW AX,saddrp	SUBW AX,HL	MOV word[C],A	MOV A,word[C]	SUB saddr,#byte	SUB A,saddr	SUB A,#byte	SUB A,[HL]	SUB A,[HL+byte]	SUB A,!addr16
3	MOVW AX,#word	4th MAP	MOVW BC,#word	XCHW AX,BC	MOVW DE,#word	XCHW AX,DE	MOVW HL,#word	XCHW AX,HL	MOV word[C],#byte	MOV word[BC],#byte	SUBC saddr,#byte	SUBC A,saddr	SUBC A,#byte	SUBC A,[HL]	SUBC A,[HL+byte]	SUBC A,!addr16
4	CMP !addr16,#byte	MOV ES,#byte	CMPW AX,!addr16	CMPW AX,BC	CMPW AX,#word	CMPW AX,DE	CMPW AX,saddrp	CMPW AX,HL	MOV word[BC],A	MOV A,word[BC]	CMP saddr,#byte	CMP A,saddr	CMP A,#byte	CMP A,[HL]	CMP A,[HL+byte]	CMP A,!addr16
5	MOV X,#byte	MOV A,#byte	MOV C,#byte	MOV B,#byte	MOV E,#byte	MOV D,#byte	MOV L,#byte	MOV H,#byte	MOVW word[B],AX	MOVW AX,word[B]	AND saddr,#byte	AND A,saddr	AND A,#byte	AND A,[HL]	AND A,[HL+byte]	AND A,!addr16
6	MOV A,X	2nd MAP	MOV A,C	MOV A,B	MOV A,E	MOV A,D	MOV A,L	MOV A,H	MOVW word[C],AX	MOVW AX,word[C]	OR saddr,#byte	OR A,saddr	OR A,#byte	OR A,[HL]	OR A,[HL+byte]	OR A,!addr16
7	MOV X,A	3rd MAP	MOV C,A	MOV B,A	MOV E,A	MOV D,A	MOV L,A	MOV H,A	MOVW word[BC],AX	MOVW AX,word[BC]	XOR saddr,#byte	XOR A,saddr	XOR A,#byte	XOR A,[HL]	XOR A,[HL+byte]	XOR A,!addr16
8	INC X	INC A	INC C	INC B	INC E	INC D	INC L	INC H	MOV A,[SP+byte]	MOV A,[DE]	MOV A,[DE+byte]	MOV A,[HL]	MOV A,[HL+byte]	MOV A,saddr	MOV A,sfr	MOV A,!addr16
9	DEC X	DEC A	DEC C	DEC B	DEC E	DEC D	DEC L	DEC H	MOV [SP+byte],A	MOV [DE],A	MOV [DE+byte],A	MOV [HL],A	MOV [HL+byte],A	MOV saddr,A	MOV sfr,A	MOV !addr16,A
a	INC !addr16	INCW AX	INCW !addr16	INCW BC	INC saddr	INCW DE	INCW saddrp	INCW HL	MOVW AX,[SP+byte]	MOVW AX,[DE]	MOVW AX,[DE+byte]	MOVW AX,[HL]	MOVW AX,[HL+byte]	MOVW AX,saddrp	MOVW AX,sfrp	MOVW AX,!addr16
b	DEC !addr16	DECW AX	DECW !addr16	DECW BC	DEC saddr	DECW DE	DECW saddrp	DECW HL	MOVW [SP+byte],AX	MOVW [DE],AX	MOVW [DE+byte],AX	MOVW [HL],AX	MOVW [HL+byte],AX	MOVW saddrp,AX	MOVW sfrp,AX	MOVW !addr16,AX
c	POP AX	PUSH AX	POP BC	PUSH BC	POP DE	PUSH DE	POP HL	PUSH HL	MOV [SP+byte],#byte	MOVW saddrp,#word	MOV [DE+byte],#byte	MOVW sfrp,#word	MOV [HL+byte],#byte	MOV saddr,#byte	NOTE MOV sfr,#byte	MOV !addr16,#byte
d	CMP0 X	CMP0 A	CMP0 C	CMP0 B	CMP0 saddr	CMP0 !addr16	MULU X	RET	MOV X,saddr	MOV X,!addr16	MOVW BC,saddrp	MOVW BC,!addr16	BC $addr20	BZ $addr20	BNC $addr20	BNZ $addr20
e	ONEB X	ONEB A	ONEB C	ONEB B	ONEB saddr	ONEB !addr16	ONEW AX	ONEW BC	MOV B,saddr	MOV B,!addr16	MOVW DE,saddrp	MOVW DE,!addr16	BR !!addr20	BR !addr16	BR $!addr20	BR $addr20
f	CLRB X	CLRB A	CLRB C	CLRB B	CLRB saddr	CLRB !addr16	CLRW AX	CLRW BC	MOV C,saddr	MOV C,!addr16	MOVW HL,saddrp	MOVW HL,!addr16	CALL !!addr20	CALL !addr16	CALL $!addr20	

Note MULHU, MULH, DIVHU, DIVWU, MACHU, and MACH of multiply/divide/multiply & accumulate instructions are also mapped here.

Table 5-8. Instruction Map (2nd MAP)

	0(low)	1(low)	2(low)	3(low)	4(low)	5(low)	6(low)	7(low)	8(low)	9(low)	a(low)	b(low)	c(low)	d(low)	e(low)	f(low)
0	ADD X,A	ADD A,A	ADD C,A	ADD B,A	ADD E,A	ADD D,A	ADD L,A	ADD H,A	ADD A,X	ADDW AX,[HL+byte]	ADD A,C	ADD A,B	ADD A,E	ADD A,D	ADD A,L	ADD A,H
1	ADDC X,A	ADDC A,A	ADDC C,A	ADDC B,A	ADDC E,A	ADDC D,A	ADDC L,A	ADDC H,A	ADDC A,X		ADDC A,C	ADDC A,B	ADDC A,E	ADDC A,D	ADDC A,L	ADDC A,H
2	SUB X,A	SUB A,A	SUB C,A	SUB B,A	SUB E,A	SUB D,A	SUB L,A	SUB H,A	SUB A,X	SUBW AX,[HL+byte]	SUB A,C	SUB A,B	SUB A,E	SUB A,D	SUB A,L	SUB A,H
3	SUBC X,A	SUBC A,A	SUBC C,A	SUBC B,A	SUBC E,A	SUBC D,A	SUBC L,A	SUBC H,A	SUBC A,X		SUBC A,C	SUBC A,B	SUBC A,E	SUBC A,D	SUBC A,L	SUBC A,H
4	CMP X,A	CMP A,A	CMP C,A	CMP B,A	CMP E,A	CMP D,A	CMP L,A	CMP H,A	CMP A,X	CMPW AX,[HL+byte]	CMP A,C	CMP A,B	CMP A,E	CMP A,D	CMP A,L	CMP A,H
5	AND X,A	AND A,A	AND C,A	AND B,A	AND E,A	AND D,A	AND L,A	AND H,A	AND A,X	INC [HL+byte]	AND A,C	AND A,B	AND A,E	AND A,D	AND A,L	AND A,H
6	OR X,A	OR A,A	OR C,A	OR B,A	OR E,A	OR D,A	OR L,A	OR H,A	OR A,X	DEC [HL+byte]	OR A,C	OR A,B	OR A,E	OR A,D	OR A,L	OR A,H
7	XOR X,A	XOR A,A	XOR C,A	XOR B,A	XOR E,A	XOR D,A	XOR L,A	XOR H,A	XOR A,X	INCW [HL+byte]	XOR A,C	XOR A,B	XOR A,E	XOR A,D	XOR A,L	XOR A,H
8	ADD A,[HL+B]		ADD A,[HL+C]		CALLT [0080h]	CALLT [0090h]	CALLT [00A0h]	CALLT [00B0h]		DECW [HL+byte]	XCH A,C	XCH A,B	XCH A,E	XCH A,D	XCH A,L	XCH A,H
9	ADDC A,[HL+B]		ADDC A,[HL+C]		CALLT [0082h]	CALLT [0092h]	CALLT [00A2h]	CALLT [00B2h]								
a	SUB A,[HL+B]		SUB A,[HL+C]		CALLT [0084h]	CALLT [0094h]	CALLT [00A4h]	CALLT [00B4h]	XCH A,saddr	XCH A,[HL+C]	XCH A,!addr16	XCH A,sfr	XCH A,[HL]	XCH A,[HL+byte]	XCH A,[DE]	XCH A,[DE+byte]
b	SUBC A,[HL+B]		SUBC A,[HL+C]		CALLT [0086h]	CALLT [0096h]	CALLT [00A6h]	CALLT [00B6h]	MOV ES,saddr	XCH A,[HL+B]						
c	CMP A,[HL+B]		CMP A,[HL+C]	BH $addr20	CALLT [0088h]	CALLT [0098h]	CALLT [00A8h]	CALLT [00B8h]	SKC	MOV A,[HL+B]	CALL AX	BR AX	BRK	POP PSW	MOVS [HL+byte],X	SEL RB0
d	AND A,[HL+B]		AND A,[HL+C]	BNH $addr20	CALLT [008Ah]	CALLT [009Ah]	CALLT [00AAh]	CALLT [00BAh]	SKNC	MOV [HL+B],A	CALL BC	ROR A,1	ROLC A,1	PUSH PSW	CMPS X,[HL+byte]	SEL RB1
e	OR A,[HL+B]		OR A,[HL+C]	SKH	CALLT [008Ch]	CALLT [009Ch]	CALLT [00ACh]	CALLT [00BCh]	SKZ	MOV A,[HL+C]	CALL DE	ROL A,1	RETB	HALT	ROLWC AX,1	SEL RB2
f	XOR A,[HL+B]		XOR A,[HL+C]	SKNH	CALLT [008Eh]	CALLT [009Eh]	CALLT [00AEh]	CALLT [00BEh]	SKNZ	MOV [HL+C],A	CALL HL	RORC A,1	RETI	STOP	ROLWC BC,1	SEL RB3

Table 5-9. Instruction Map (3rd MAP)

	0(low)	1(low)	2(low)	3(low)	4(low)	5(low)	6(low)	7(low)	8(low)	9(low)	a(low)	b(low)	c(low)	d(low)	e(low)	f(low)
0	SET1 !addr16.0	MOV1 saddr.0,CY	SET1 saddr.0	CLR1 saddr.0	MOV1 CY,saddr.0	AND1 CY,saddr.0	OR1 CY,saddr.0	XOR1 CY,saddr.0	CLR1 !addr16.0	MOV1 sfr.0,CY	SET1 sfr.0	CLR1 sfr.0	MOV1 CY,sfr.0	AND1 CY,sfr.0	OR1 CY,sfr.0	XOR1 CY,sfr.0
1	SET1 !addr16.1	MOV1 saddr.1,CY	SET1 saddr.1	CLR1 saddr.1	MOV1 CY,saddr.1	AND1 CY,saddr.1	OR1 CY,saddr.1	XOR1 CY,saddr.1	CLR1 !addr16.1	MOV1 sfr.1,CY	SET1 sfr.1	CLR1 sfr.1	MOV1 CY,sfr.1	AND1 CY,sfr.1	OR1 CY,sfr.1	XOR1 CY,sfr.1
2	SET1 !addr16.2	MOV1 saddr.2,CY	SET1 saddr.2	CLR1 saddr.2	MOV1 CY,saddr.2	AND1 CY,saddr.2	OR1 CY,saddr.2	XOR1 CY,saddr.2	CLR1 !addr16.2	MOV1 sfr.2,CY	SET1 sfr.2	CLR1 sfr.2	MOV1 CY,sfr.2	AND1 CY,sfr.2	OR1 CY,sfr.2	XOR1 CY,sfr.2
3	SET1 !addr16.3	MOV1 saddr.3,CY	SET1 saddr.3	CLR1 saddr.3	MOV1 CY,saddr.3	AND1 CY,saddr.3	OR1 CY,saddr.3	XOR1 CY,saddr.3	CLR1 !addr16.3	MOV1 sfr.3,CY	SET1 sfr.3	CLR1 sfr.3	MOV1 CY,sfr.3	AND1 CY,sfr.3	OR1 CY,sfr.3	XOR1 CY,sfr.3
4	SET1 !addr16.4	MOV1 saddr.4,CY	SET1 saddr.4	CLR1 saddr.4	MOV1 CY,saddr.4	AND1 CY,saddr.4	OR1 CY,saddr.4	XOR1 CY,saddr.4	CLR1 !addr16.4	MOV1 sfr.4,CY	SET1 sfr.4	CLR1 sfr.4	MOV1 CY,sfr.4	AND1 CY,sfr.4	OR1 CY,sfr.4	XOR1 CY,sfr.4
5	SET1 !addr16.5	MOV1 saddr.5,CY	SET1 saddr.5	CLR1 saddr.5	MOV1 CY,saddr.5	AND1 CY,saddr.5	OR1 CY,saddr.5	XOR1 CY,saddr.5	CLR1 !addr16.5	MOV1 sfr.5,CY	SET1 sfr.5	CLR1 sfr.5	MOV1 CY,sfr.5	AND1 CY,sfr.5	OR1 CY,sfr.5	XOR1 CY,sfr.5
6	SET1 !addr16.6	MOV1 saddr.6,CY	SET1 saddr.6	CLR1 saddr.6	MOV1 CY,saddr.6	AND1 CY,saddr.6	OR1 CY,saddr.6	XOR1 CY,saddr.6	CLR1 !addr16.6	MOV1 sfr.6,CY	SET1 sfr.6	CLR1 sfr.6	MOV1 CY,sfr.6	AND1 CY,sfr.6	OR1 CY,sfr.6	XOR1 CY,sfr.6
7	SET1 !addr16.7	MOV1 saddr.7,CY	SET1 saddr.7	CLR1 saddr.7	MOV1 CY,saddr.7	AND1 CY,saddr.7	OR1 CY,saddr.7	XOR1 CY,saddr.7	CLR1 !addr16.7	MOV1 sfr.7,CY	SET1 sfr.7	CLR1 sfr.7	MOV1 CY,sfr.7	AND1 CY,sfr.7	OR1 CY,sfr.7	XOR1 CY,sfr.7
8	SET1 CY	MOV1 [HL].0,CY	SET1 [HL].0	CLR1 [HL].0	MOV1 CY,[HL].0	AND1 CY,[HL].0	OR1 CY,[HL].0	XOR1 CY,[HL].0	CLR1 CY	MOV1 A.0,CY	SET1 A.0	CLR1 A.0	MOV1 CY,A.0	AND1 CY,A.0	OR1 CY,A.0	XOR1 CY,A.0
9		MOV1 [HL].1,CY	SET1 [HL].1	CLR1 [HL].1	MOV1 CY,[HL].1	AND1 CY,[HL].1	OR1 CY,[HL].1	XOR1 CY,[HL].1		MOV1 A.1,CY	SET1 A.1	CLR1 A.1	MOV1 CY,A.1	AND1 CY,A.1	OR1 CY,A.1	XOR1 CY,A.1
a		MOV1 [HL].2,CY	SET1 [HL].2	CLR1 [HL].2	MOV1 CY,[HL].2	AND1 CY,[HL].2	OR1 CY,[HL].2	XOR1 CY,[HL].2		MOV1 A.2,CY	SET1 A.2	CLR1 A.2	MOV1 CY,A.2	AND1 CY,A.2	OR1 CY,A.2	XOR1 CY,A.2
b		MOV1 [HL].3,CY	SET1 [HL].3	CLR1 [HL].3	MOV1 CY,[HL].3	AND1 CY,[HL].3	OR1 CY,[HL].3	XOR1 CY,[HL].3		MOV1 A.3,CY	SET1 A.3	CLR1 A.3	MOV1 CY,A.3	AND1 CY,A.3	OR1 CY,A.3	XOR1 CY,A.3
c	NOT1 CY	MOV1 [HL].4,CY	SET1 [HL].4	CLR1 [HL].4	MOV1 CY,[HL].4	AND1 CY,[HL].4	OR1 CY,[HL].4	XOR1 CY,[HL].4		MOV1 A.4,CY	SET1 A.4	CLR1 A.4	MOV1 CY,A.4	AND1 CY,A.4	OR1 CY,A.4	XOR1 CY,A.4
d		MOV1 [HL].5,CY	SET1 [HL].5	CLR1 [HL].5	MOV1 CY,[HL].5	AND1 CY,[HL].5	OR1 CY,[HL].5	XOR1 CY,[HL].5		MOV1 A.5,CY	SET1 A.5	CLR1 A.5	MOV1 CY,A.5	AND1 CY,A.5	OR1 CY,A.5	XOR1 CY,A.5
e		MOV1 [HL].6,CY	SET1 [HL].6	CLR1 [HL].6	MOV1 CY,[HL].6	AND1 CY,[HL].6	OR1 CY,[HL].6	XOR1 CY,[HL].6		MOV1 A.6,CY	SET1 A.6	CLR1 A.6	MOV1 CY,A.6	AND1 CY,A.6	OR1 CY,A.6	XOR1 CY,A.6
f		MOV1 [HL].7,CY	SET1 [HL].7	CLR1 [HL].7	MOV1 CY,[HL].7	AND1 CY,[HL].7	OR1 CY,[HL].7	XOR1 CY,[HL].7		MOV1 A.7,CY	SET1 A.7	CLR1 A.7	MOV1 CY,A.7	AND1 CY,A.7	OR1 CY,A.7	XOR1 CY,A.7

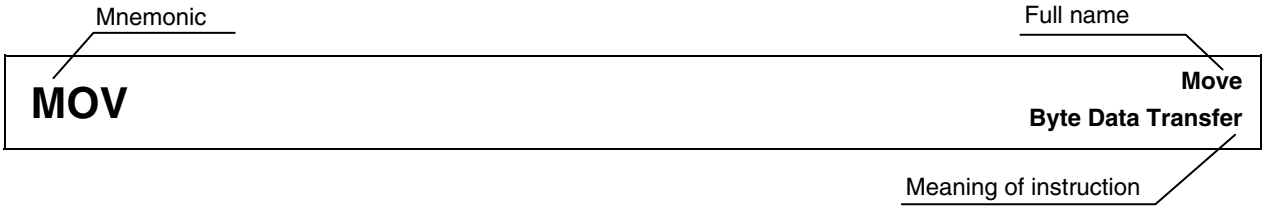
Table 5-10. Instruction Map (4th MAP)

	0(low)	1(low)	2(low)	3(low)	4(low)	5(low)	6(low)	7(low)	8(low)	9(low)	a(low)	b(low)	c(low)	d(low)	e(low)	f(low)
0	**BTCLR saddr.0,$addr20**	**BTCLR A.0,$addr20**	**BT saddr.0,$addr20**	**BT A.0,$addr20**	**BF saddr.0,$addr20**	**BF A.0,$addr20**										
1	**BTCLR saddr.1,$addr20**	**BTCLR A.1,$addr20**	**BT saddr.1,$addr20**	**BT A.1,$addr20**	**BF saddr.1,$addr20**	**BF A.1,$addr20**		**SHL C,1**	**SHL B,1**	**SHL A,1**	**SHR A,1**	**SAR A,1**	**SHLW BC,1**	**SHLW AX,1**	**SHRW AX,1**	**SARW AX,1**
2	**BTCLR saddr.2,$addr20**	**BTCLR A.2,$addr20**	**BT saddr.2,$addr20**	**BT A.2,$addr20**	**BF saddr.2,$addr20**	**BF A.2,$addr20**		**SHL C,2**	**SHL B,2**	**SHL A,2**	**SHR A,2**	**SAR A,2**	**SHLW BC,2**	**SHLW AX,2**	**SHRW AX,2**	**SARW AX,2**
3	**BTCLR saddr.3,$addr20**	**BTCLR A.3,$addr20**	**BT saddr.3,$addr20**	**BT A.3,$addr20**	**BF saddr.3,$addr20**	**BF A.3,$addr20**		**SHL C,3**	**SHL B,3**	**SHL A,3**	**SHR A,3**	**SAR A,3**	**SHLW BC,3**	**SHLW AX,3**	**SHRW AX,3**	**SARW AX,3**
4	**BTCLR saddr.4,$addr20**	**BTCLR A.4,$addr20**	**BT saddr.4,$addr20**	**BT A.4,$addr20**	**BF saddr.4,$addr20**	**BF A.4,$addr20**		**SHL C,4**	**SHL B,4**	**SHL A,4**	**SHR A,4**	**SAR A,4**	**SHLW BC,4**	**SHLW AX,4**	**SHRW AX,4**	**SARW AX,4**
5	**BTCLR saddr.5,$addr20**	**BTCLR A.5,$addr20**	**BT saddr.5,$addr20**	**BT A.5,$addr20**	**BF saddr.5,$addr20**	**BF A.5,$addr20**		**SHL C,5**	**SHL B,5**	**SHL A,5**	**SHR A,5**	**SAR A,5**	**SHLW BC,5**	**SHLW AX,5**	**SHRW AX,5**	**SARW AX,5**
6	**BTCLR saddr.6,$addr20**	**BTCLR A.6,$addr20**	**BT saddr.6,$addr20**	**BT A.6,$addr20**	**BF saddr.6,$addr20**	**BF A.6,$addr20**		**SHL C,6**	**SHL B,6**	**SHL A,6**	**SHR A,6**	**SAR A,6**	**SHLW BC,6**	**SHLW AX,6**	**SHRW AX,6**	**SARW AX,6**
7	**BTCLR saddr.7,$addr20**	**BTCLR A.7,$addr20**	**BT saddr.7,$addr20**	**BT A.7,$addr20**	**BF saddr.7,$addr20**	**BF A.7,$addr20**		**SHL C,7**	**SHL B,7**	**SHL A,7**	**SHR A,7**	**SAR A,7**	**SHLW BC,7**	**SHLW AX,7**	**SHRW AX,7**	**SARW AX,7**
8	**BTCLR sfr.0,$addr20**	**BTCLR [HL].0,$addr20**	**BT sfr.0,$addr20**	**BT [HL].0,$addr20**	**BF sfr.0,$addr20**	**BF [HL].0,$addr20**							**SHLW BC,8**	**SHLW AX,8**	**SHRW AX,8**	**SARW AX,8**
9	**BTCLR sfr.1,$addr20**	**BTCLR [HL].1,$addr20**	**BT sfr.1,$addr20**	**BT [HL].1,$addr20**	**BF sfr.1,$addr20**	**BF [HL].1,$addr20**							**SHLW BC,9**	**SHLW AX,9**	**SHRW AX,9**	**SARW AX,9**
a	**BTCLR sfr.2,$addr20**	**BTCLR [HL].2,$addr20**	**BT sfr.2,$addr20**	**BT [HL].2,$addr20**	**BF sfr.2,$addr20**	**BF [HL].2,$addr20**							**SHLW BC,10**	**SHLW AX,10**	**SHRW AX,10**	**SARW AX,10**
b	**BTCLR sfr.3,$addr20**	**BTCLR [HL].3,$addr20**	**BT sfr.3,$addr20**	**BT [HL].3,$addr20**	**BF sfr.3,$addr20**	**BF [HL].3,$addr20**							**SHLW BC,11**	**SHLW AX,11**	**SHRW AX,11**	**SARW AX,11**
c	**BTCLR sfr.4,$addr20**	**BTCLR [HL].4,$addr20**	**BT sfr.4,$addr20**	**BT [HL].4,$addr20**	**BF sfr.4,$addr20**	**BF [HL].4,$addr20**							**SHLW BC,12**	**SHLW AX,12**	**SHRW AX,12**	**SARW AX,12**
d	**BTCLR sfr.5,$addr20**	**BTCLR [HL].5,$addr20**	**BT sfr.5,$addr20**	**BT [HL].5,$addr20**	**BF sfr.5,$addr20**	**BF [HL].5,$addr20**							**SHLW BC,13**	**SHLW AX,13**	**SHRW AX,13**	**SARW AX,13**
e	**BTCLR sfr.6,$addr20**	**BTCLR [HL].6,$addr20**	**BT sfr.6,$addr20**	**BT [HL].6,$addr20**	**BF sfr.6,$addr20**	**BF [HL].6,$addr20**							**SHLW BC,14**	**SHLW AX,14**	**SHRW AX,14**	**SARW AX,14**
f	**BTCLR sfr.7,$addr20**	**BTCLR [HL].7,$addr20**	**BT sfr.7,$addr20**	**BT [HL].7,$addr20**	**BF sfr.7,$addr20**	**BF [HL].7,$addr20**							**SHLW BC,15**	**SHLW AX,15**	**SHRW AX,15**	**SARW AX,15**

CHAPTER 6 EXPLANATION OF INSTRUCTIONS

This chapter explains the instructions of RL78 microcontrollers.

DESCRIPTION EXAMPLE



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

[Operation] **dst ← src:** Indicates instruction operation using symbols.

[Operand] Indicates operands that can be specified by this instruction. Refer to **5.2 Symbols in "Operation" Column** for the description of each operand symbol.

Mnemonic	Operand (dst, src)
MOV	r, #byte
	A, saddr
	saddr, A
	PSW, #byte

Mnemonic	Operand (dst, src)
MOV	A, PSW
	[HL], A
	A, [HL+byte]
	[HL+C], A

[Flag] Indicates the flag operation that changes by instruction execution.
Each flag operation symbol is shown in the conventions.

Z	AC	CY

Conventions

Symbol	Description
Blank	Unchanged
0	Cleared to 0
1	Set to 1
×	Set or cleared according to the result
R	Previously saved value is restored

[Description]: Describes the instruction operation in detail.

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

[Description example]

MOV A, #4DH; 4DH is transferred to the A register.

6.1 8-bit Data Transfer Instructions

The following instructions are 8-bit data transfer instructions.

MOV

Move
Byte Data Transfer

[Instruction format] MOV dst, src

[Operation] dst ← src

[Operand]

Mnemonic	Operand (dst, src)
MOV	r, #byte
	saddr, #byte
	sfr, #byte
	!addr16, #byte
	A, r Note
	r, A Note
	A, saddr
	saddr, A
	A, sfr
	sfr, A
	A, !addr16
	!addr16, A
	PSW, #byte
	A, PSW
	PSW, A
	ES, #byte
	ES, saddr
	A, ES
	ES, A
	CS, #byte
	A, CS
	CS, A
	A, [DE]
	[DE], A
	[DE+byte], #byte
	A, [DE+byte]
	[DE+byte], A
	A, [HL]

Mnemonic	Operand (dst, src)
MOV	[HL], A
	[HL+byte], #byte
	A, [HL+byte]
	[HL+byte], A
	A, [HL+B]
	[HL+B], A
	A, [HL+C]
	[HL+C], A
	word[B], #byte
	A, word[B]
	word[B], A
	word[C], #byte
	A, word[C]
	word[C], A
	word[BC], #byte
	A, word[BC]
	word[BC], A
	[SP+byte], #byte
	A, [SP+byte]
	[SP+byte], A
	B, saddr
	B, !addr16
	C, saddr
	C, !addr16
	X, saddr
	X, !addr16
	ES:!addr16, #byte
	A, ES:!addr16

Mnemonic	Operand (dst, src)
MOV	ES:!addr16, A
	A, ES:[DE]
	ES:[DE], A
	ES:[DE+byte], #byte
	A, ES:[DE+byte]
	ES:[DE+byte], A
	A, ES:[HL]
	ES:[HL], A
	ES:[HL+byte], #byte
	A, ES:[HL+byte]
	ES:[HL+byte], A
	A, ES:[HL+B]
	ES:[HL+B], A
	A, ES:[HL+C]
	ES:[HL+C], A
	ES:word[B], #byte
	A, ES:word[B]
	ES:word[B], A
	ES:word[C], #byte
	A, ES:word[C]
	ES:word[C], A
	ES:word[BC], #byte
	A, ES:word[BC]
	ES:word[BC], A
	B, ES:!addr16
	C, ES:!addr16
	X, ES:!addr16

Note Except r = A

[Flag]

PSW, #byte and PSW,
A operands

Z	AC	CY
×	×	×

All other operand combinations

Z	AC	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 are acknowledged between the MOV PSW, #byte instruction/MOV PSW, A instruction and the next instruction.

[Description example]

MOV A, #4DH; 4DH is transferred to the A register.

XCH

Exchange
Byte Data Transfer

[Instruction format] XCH dst, src

[Operation] dst ↔ src

[Operand]

Mnemonic	Operand (dst, src)
XCH	A, r Note
	A, saddr
	A, sfr
	A, !addr16
	A, [DE]
	A, [DE+byte]
	A, [HL]
	A, [HL+byte]
	A, [HL+B]

Mnemonic	Operand (dst, src)
XCH	A, [HL+C]
	A, ES:!addr16
	A, ES:[DE]
	A, ES:[DE+byte]
	A, ES:[HL]
	A, ES:[HL+byte]
	A, ES:[HL+B]
	A, ES:[HL+C]

Note Except r = A

[Flag]

Z	AC	CY

[Description]

- The 1st and 2nd operand contents are exchanged.

[Description example]

XCH A, FFEBCH; The A register contents and address FFEBCH contents are exchanged.

ONEB

One byte
Byte Data 01H Set

[Instruction format] ONEB dst

[Operation] dst ← 01H

[Operand]

Mnemonic	Operand (dst)
ONEB	A
	X
	B
	C
	saddr
	!addr16
	ES:!addr16

[Flag]

Z	AC	CY

[Description]

- 01H is transferred to the destination operand (dst) specified by the first operand.

[Description example]

ONEB A; Transfers 01H to the A register.

CLRB

Clear byte
Byte Data Clear

[Instruction format] CLRB dst

[Operation] dst ← 00H

[Operand]

Mnemonic	Operand (dst)
CLRB	A
	X
	B
	C
	saddr
	!addr16
	ES:!addr16

[Flag]

Z	AC	CY

[Description]

- 00H is transferred to the destination operand (dst) specified by the first operand.

[Description example]

CLRB A; Transfers 00H to the A register.

MOVS

Move and change PSW
Byte Data Transfer and PSW Change

[Instruction format] MOVS dst, src

[Operation] dst ← src

[Operand]

Mnemonic	Operand (dst, src)
MOVS	[HL+byte], X
	ES:[HL+byte], X

[Flag]

Z	AC	CY
×		×

[Description]

- The contents of the source operand specified by the second operand is transferred to the destination operand (dst) specified by the first operand.
- If the src value is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the register A value is 0 or if the src value is 0, the CY flag is set (1). In all other cases, the CY flag is cleared (0).

[Description example]

MOVS [HL+2H], X; When HL = FE00H, X = 55H, A = 0H

"X = 55H" is stored at address FE02H.

Z flag = 0

CY flag = 1 (since A register = 0)

6.2 16-bit Data Transfer Instructions

The following instructions are 16-bit data transfer instructions.

MOVW

Move Word
Word Data Transfer

[Instruction format] MOVW dst, src

[Operation] dst ← src

[Operand]

Mnemonic	Operand (dst, src)
MOVW	rp, #word
	saddrp, #word
	sfrp, #word
	AX, saddrp
	saddrp, AX
	AX, sfrp
	sfrp, AX
	AX, rp Note
	rp, AX Note
	AX, !addr16
	!addr16, AX
	AX, [DE]
	[DE], AX
	AX, [DE+byte]
	[DE+byte], AX
	AX, [HL]
	[HL], AX
	AX, [HL+byte]
	[HL+byte], AX
	AX, word[B]
	word[B], AX
	AX, word[C]
	word[C], AX
	AX, word[BC]
	word[BC], AX
	AX, [SP+byte]

Mnemonic	Operand (dst, src)
MOVW	[SP+byte], AX
	BC, saddrp
	BC, !addr16
	DE, saddrp
	DE, !addr16
	HL, saddrp
	HL, !addr16
	AX, ES:!addr16
	ES:!addr16, AX
	AX, ES:[DE]
	ES:[DE], AX
	AX, ES:[DE+byte]
	ES:[DE+byte], AX
	AX, ES:[HL]
	ES:[HL], AX
	AX, ES:[HL+byte]
	ES:[HL+byte], AX
	AX, ES:word[B]
	ES:word[B], AX
	AX, ES:word[C]
	ES:word[C], AX
	AX, ES:word[BC]
	ES:word[BC], AX
	BC, ES:!addr16
	DE, ES:!addr16
	HL, ES:!addr16

Note Only when rp = BC, DE or HL

[Flag]

Z	AC	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.

[Description example]

MOVW AX, HL; The HL register contents are transferred to the AX register.

[Caution]

Only an even address can be specified. An odd address cannot be specified.

XCHW

Exchange Word
Word Data Exchange

[Instruction format] XCHW dst, src

[Operation] dst ↔ src

[Operand]

Mnemonic	Operand (dst, src)
XCHW	AX, rp Note

Note Only when rp = BC, DE or HL

[Flag]

Z	AC	CY

[Description]

- The 1st and 2nd operand contents are exchanged.

[Description example]

XCHW AX, BC; The memory contents of the AX register are exchanged with those of the BC register.

ONEW

One Word
Word Data 0001 Set

[Instruction format] ONEW dst

[Operation] dst ← 0001H

[Operand]

Mnemonic	Operand (dst)
ONEW	AX
	BC

[Flag]

Z	AC	CY

[Description]

- 0001H is transferred to the destination operand (dst) specified by the first operand.

[Description example]

ONEW AX; 0001H is transferred to the AX register.

CLRW

Clear Word
Word Data Clear

[Instruction format] CLRW dst

[Operation] dst ← 0000H

[Operand]

Mnemonic	Operand (dst)
CLRW	AX
	BC

[Flag]

Z	AC	CY

[Description]

- 0000H is transferred to the destination operand (dst) specified by the first operand.

[Description example]

CLRW AX; 0000H is transferred to the AX register.

6.3 8-bit Operation Instructions

The following instructions are 8-bit operation instructions.

ADD

Add
Byte Data Addition

[Instruction format] ADD dst, src

[Operation] dst, CY ← dst + src

[Operand]

Mnemonic	Operand (dst, src)
ADD	A, #byte
	saddr, #byte
	A, r Note
	r, A
	A, saddr
	A, !addr16
	A, [HL]
	A, [HL+byte]

Mnemonic	Operand (dst, src)
ADD	A, [HL+B]
	A, [HL+C]
	A, ES:!addr16
	A, ES:[HL]
	A, ES:[HL+byte]
	A, ES:[HL+B]
	A, ES:[HL+C]

Note Except r = A

[Flag]

Z	AC	CY
×	×	×

[Description]

- The destination operand (dst) specified by the 1st operand is added to the source operand (src) specified by the 2nd operand and the result is stored in the CY flag and the destination operand (dst).
- If the addition result shows that dst is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the addition generates a carry out of bit 7, the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- If the addition generates a carry for bit 4 out of bit 3, the AC flag is set (1). In all other cases, the AC flag is cleared (0).

[Description example]

ADD CR10, #56H; 56H is added to the CR10 register and the result is stored in the CR10 register.

ADDC

Add with Carry
Addition of Byte Data with Carry

[Instruction format] ADDC dst, src

[Operation] dst, CY ← dst + src + CY

[Operand]

Mnemonic	Operand (dst, src)
ADDC	A, #byte
	saddr, #byte
	A, r Note
	r, A
	A, saddr
	A, !addr16
	A, [HL]
	A, [HL+byte]

Mnemonic	Operand (dst, src)
ADDC	A, [HL+B]
	A, [HL+C]
	A, ES:!addr16
	A, ES:[HL]
	A, ES:[HL+byte]
	A, ES:[HL+B]
	A, ES:[HL+C]

Note Except r = A

[Flag]

Z	AC	CY
×	×	×

[Description]

- The destination operand (dst) specified by the 1st operand, the source operand (src) specified by the 2nd operand and the CY flag are added and the result is stored in the destination operand (dst) and the CY flag.
  The CY flag is added to the least significant bit. This instruction is mainly used to add two or more bytes.
- If the addition result shows that dst is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the addition generates a carry out of bit 7, the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- If the addition generates a carry for bit 4 out of bit 3, the AC flag is set (1). In all other cases, the AC flag is cleared (0).

[Description example]

ADDC A, [HL+B]; The A register contents and the contents at address (HL register + (B register)) and the CY flag are added and the result is stored in the A register.

SUB

Subtract
Byte Data Subtraction

[Instruction format] SUB dst, src

[Operation] dst, CY ← dst – src

[Operand]

Mnemonic	Operand (dst, src)
SUB	A, #byte
	saddr, #byte
	A, r Note
	r, A
	A, saddr
	A, !addr16
	A, [HL]
	A, [HL+byte]

Mnemonic	Operand (dst, src)
SUB	A, [HL+B]
	A, [HL+C]
	A, ES:!addr16
	A, ES:[HL]
	A, ES:[HL+byte]
	A, ES:[HL+B]
	A, ES:[HL+C]

Note Except r = A

[Flag]

Z	AC	CY
×	×	×

[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 can be cleared to 0 by equalizing the source operand (src) and the destination operand (dst).
- If the subtraction shows that dst is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the subtraction generates a borrow out of bit 7, the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- If the subtraction generates a borrow for bit 3 out of bit 4, the AC flag is set (1). In all other cases, the AC flag is cleared (0).

[Description example]

SUB D, A; The A register is subtracted from the D register and the result is stored in the D register.

SUBC

Subtract with Carry
Subtraction of Byte Data with Carry

[Instruction format] SUBC dst, src

[Operation] dst, CY ← dst – src – CY

[Operand]

Mnemonic	Operand (dst, src)
SUBC	A, #byte
	saddr, #byte
	A, r Note
	r, A
	A, saddr
	A, !addr16
	A, [HL]
	A, [HL+byte]

Mnemonic	Operand (dst, src)
SUBC	A, [HL+B]
	A, [HL+C]
	A, ES:!addr16
	A, ES:[HL]
	A, ES:[HL+byte]
	A, ES:[HL+B]
	A, ES:[HL+C]

Note Except r = A

[Flag]

Z	AC	CY
×	×	×

[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).
  The CY flag is subtracted from the least significant bit. This instruction is mainly used for subtraction of two or more bytes.
- If the subtraction shows that dst is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the subtraction generates a borrow out of bit 7, the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- If the subtraction generates a borrow for bit 3 out of bit 4, the AC flag is set (1). In all other cases, the AC flag is cleared (0).

[Description example]

SUBC A, [HL]; The (HL register) address contents and the CY flag are subtracted from the A register and the result is stored in the A register.

AND

And
Logical Product of Byte Data

[Instruction format] AND dst, src

[Operation] dst ← dst ∧ src

[Operand]

Mnemonic	Operand (dst, src)
AND	A, #byte
	saddr, #byte
	A, r Note
	r, A
	A, saddr
	A, !addr16
	A, [HL]
	A, [HL+byte]

Mnemonic	Operand (dst, src)
AND	A, [HL+B]
	A, [HL+C]
	A, ES:!addr16
	A, ES:[HL]
	A, ES:[HL+byte]
	A, ES:[HL+B]
	A, ES: [HL+C]

Note Except r = A

[Flag]

Z	AC	CY
×		

[Description]

- Bit-wise logical product is obtained from the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand and the result is stored in the destination operand (dst).
- If the logical product shows that all bits are 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).

[Description example]

AND FFEBAH, #11011100B; Bit-wise logical product of FFEBAH contents and 11011100B is obtained and the result is stored at FFEBAH.

OR

Or
Logical Sum of Byte Data

[Instruction format] OR dst, src

[Operation] dst ← dst ∨ src

[Operand]

Mnemonic	Operand (dst, src)
OR	A, #byte
	saddr, #byte
	A, r Note
	r, A
	A, saddr
	A, !addr16
	A, [HL]
	A, [HL+byte]

Mnemonic	Operand (dst, src)
OR	A, [HL+B]
	A, [HL+C]
	A, ES:!addr16
	A, ES:[HL]
	A, ES:[HL+byte]
	A, ES:[HL+B]
	A, ES:[HL+C]

Note Except r = A

[Flag]

Z	AC	CY
×		

[Description]

- The bit-wise logical sum is obtained from the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand and the result is stored in the destination operand (dst).
- If the logical sum shows that all bits are 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).

[Description example]

OR A, FFE98H; The bit-wise logical sum of the A register and FFE98H is obtained and the result is stored in the A register.

XOR

Exclusive Or
Exclusive Logical Sum of Byte Data

[Instruction format] XOR dst, src

[Operation] dst ← dst ⩛ src

[Operand]

Mnemonic	Operand (dst, src)
XOR	A, #byte
	saddr, #byte
	A, r Note
	r, A
	A, saddr
	A, !addr16
	A, [HL]
	A, [HL+byte]

Mnemonic	Operand (dst, src)
XOR	A, [HL+B]
	A, [HL+C]
	A, ES:!addr16
	A, ES:[HL]
	A, ES:[HL+byte]
	A, ES:[HL+B]
	A, ES:[HL+C]

Note Except r = A

[Flag]

Z	AC	CY
×		

[Description]

- The bit-wise exclusive logical sum is obtained from the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand and the result is stored in the destination operand (dst).
  Logical negation of all bits of the destination operand (dst) is possible by selecting #0FFH for the source operand (src) with this instruction.
- If the exclusive logical sum shows that all bits are 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).

[Description example]

XOR A, L; The bit-wise exclusive logical sum of the A and L registers is obtained and the result is stored in the A register.

CMP

Compare
Byte Data Comparison

[Instruction format] CMP dst, src

[Operation] dst – src

[Operand]

Mnemonic	Operand (dst, src)
CMP	A, #byte
	saddr, #byte
	A, r Note
	r, A
	A, saddr
	A, !addr16
	A, [HL]
	A, [HL+byte]
	A, [HL+B]

Mnemonic	Operand (dst, src)
CMP	A, [HL+C]
	!addr16, #byte
	A, ES:!addr16
	A, ES:[HL]
	A, ES:[HL+byte]
	A, ES:[HL+B]
	A, ES:[HL+C]
	ES:!addr16, #byte

Note Except r = A

[Flag]

Z	AC	CY
×	×	×

[Description]

- The source operand (src) specified by the 2nd operand is subtracted from the destination operand (dst) specified by the 1st operand.
  The subtraction result is not stored anywhere and only the Z, AC and CY flags are changed.
- If the subtraction result is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the subtraction generates a borrow out of bit 7, the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- If the subtraction generates a borrow for bit 3 out of bit 4, the AC flag is set (1). In all other cases, the AC flag is cleared (0).

[Description example]

CMP FFE38H, #38H; 38H is subtracted from the contents at address FFE38H and only the flags are changed (comparison of contents at address FFE38H and the immediate data).

CMP0

Compare 00H
Byte Data Zero Comparison

[Instruction format] CMP0 dst

[Operation] dst – 00H

[Operand]

Mnemonic	Operand (dst)
CMP0	A
	X
	B
	C
	saddr
	!addr16
	ES:!addr16

[Flag]

Z	AC	CY
×	×	×

[Description]

- 00H is subtracted from the destination operand (dst) specified by the first operand.
- The subtraction result is not stored anywhere and only the Z, AC and CY flags are changed.
- If the dst value is already 00H, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- The AC and CY flags are always cleared (0).

[Description example]

CMP0 A; The Z flag is set if the A register value is 0.

CMPS

Compare
Byte Data Comparison

[Instruction format] CMPS dst, src

[Operation] dst – src

[Operand]

Mnemonic	Operand (dst, src)
CMPS	X, [HL+byte]
	X, ES:[HL+byte]

[Flag]

Z	AC	CY
×	×	×

[Description]

- The source operand (src) specified by the 2nd operand is subtracted from the destination operand (dst) specified by the 1st operand.
  The subtraction result is not stored anywhere and only the Z, AC and CY flags are changed.
- If the subtraction result is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- When the calculation result is not 0 or when the value of either register A or dst is 0, then the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- If the subtraction generates a borrow out of bit 4 to bit 3, the AC flag is set (1). In all other cases, the AC flag is cleared (0).

[Description example]

CMPS X, [HL+F0H]; When HL = FD12H

The value of X is compared with the contents of address FFE02H, and the Z flag is set if the two values match.
The value of X is compared with the contents of address FFE02H, and the CY flag is set if the two values do not match.
The CY flag is set when the value of register A is 0.
The CY flag is set when the value of register X is 0.
The AC flag is set by borrowing from bit 4 to bit 3, similar to the CMP instruction.

6.4 16-bit Operation Instructions

The following instructions are 16-bit operation instructions.

ADDW

Add Word
Word Data Addition

[Instruction format] ADDW dst, src

[Operation] dst, CY ← dst + src

[Operand]

Mnemonic	Operand (dst, src)
ADDW	AX, #word
	AX, AX
	AX, BC
	AX, DE
	AX, HL
	AX, saddrp
	AX, !addr16
	AX, [HL+byte]
	AX, ES:!addr16
	AX, ES:[HL+byte]

[Flag]

Z	AC	CY
×	×	×

[Description]

- The destination operand (dst) specified by the 1st operand is added to the source operand (src) specified by the 2nd operand and the result is stored in the destination operand (dst).
- If the addition result shows that dst is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the addition generates a carry out of bit 15, the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- As a result of addition, the AC flag becomes undefined.

[Description example]

ADDW AX, #ABCDH; ABCDH is added to the AX register and the result is stored in the AX register.

SUBW

Subtract Word
Word Data Subtraction

[Instruction format] SUBW dst, src

[Operation] dst, CY ← dst − src

[Operand]

Mnemonic	Operand (dst, src)
SUBW	AX, #word
	AX, BC
	AX, DE
	AX, HL
	AX, saddrp
	AX, !addr16
	AX, [HL+byte]
	AX, ES:!addr16
	AX, ES:[HL+byte]

[Flag]

Z	AC	CY
×	×	×

[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.
- If the subtraction shows that dst is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the subtraction generates a borrow out of bit 15, the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- As a result of subtraction, the AC flag becomes undefined.

[Description example]

SUBW AX, #ABCDH; ABCDH is subtracted from the AX register contents and the result is stored in the AX register.

CMPW

Compare Word
Word Data Comparison

[Instruction format] CMPW dst, src

[Operation] dst – src

[Operand]

Mnemonic	Operand (dst, src)
CMPW	AX, #word
	AX, BC
	AX, DE
	AX, HL
	AX, saddrp
	AX, !addr16
	AX, [HL+byte]
	AX, ES:!addr16
	AX, ES:[HL+byte]

[Flag]

Z	AC	CY
×	×	×

[Description]

- The source operand (src) specified by the 2nd operand is subtracted from the destination operand (dst) specified by the 1st operand.
  The subtraction result is not stored anywhere and only the Z, AC and CY flags are changed.
- If the subtraction result is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the subtraction generates a borrow out of bit 15, the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- As a result of subtraction, the AC flag becomes undefined.

[Description example]

CMPW AX, #ABCDH; ABCDH is subtracted from the AX register and only the flags are changed (comparison of the AX register and the immediate data).

6.5 Multiply/Divide/Multiply & Accumulate Instructions

The following instructions are multiply/divide/multiply & accumulate instructions.

Caution The following multiply/divide/multiply & accumulate instructions are expanded instructions and mounted or not mounted by product. For details, refer to user's manual of each product.

- **MULHU (16-bit multiplication unsigned)**
- **MULH (16-bit multiplication signed)**
- **DIVHU (16-bit division unsigned)**
- **DIVWU (32-bit division unsigned)**
- **MACHU (16-bit multiplication and accumulation unsigned (16 bits × 16 bits) + 32 bits)**
- **MACH (16-bit multiplication and accumulation signed (16 bits × 16 bits) + 32 bits)**

MULU

Multiply Unsigned
Unsigned Multiplication of Data

[Instruction format] MULU src

[Operation] AX ← A × src

[Operand]

Mnemonic	Operand (src)
MULU	X

[Flag]

Z	AC	CY

[Description]

- The A register contents and the source operand (src) data are multiplied as unsigned data and the result is stored in the AX register.

[Description example]

MULU X; The A register contents and the X register contents are multiplied and the result is stored in the AX register.

MULHU

Multiply Unsigned
Unsigned Multiplication of Data

[Instruction format] MULHU

[Operation] BCAX ← AX × BC

[Operand]

Mnemonic	Operand (src)
MULHU	

[Flag]

Z	AC	CY

[Description]

- The content of AX register and the content of BC register are multiplied as unsigned data, upper 16 bits of the result are stored in the BC register, and lower 16 bits of the result are stored in the AX register.

[Description example]

```
MOVW AX, #0C000H
MOVW BC, #1000H
MULHU
MOVW !addr16, AX
MOVW AX, BC
```

MOVW!addr16, AX; C000H and 1000H are multiplied, and the result C000000H is stored in memory indicated by !addr16.

MULH

Multiply Signed
Signed Multiplication of Data

[Instruction format] MULH

[Operation] BCAX ← AX × BC

[Operand]

Mnemonic	Operand (src)
MULH	

[Flag]

Z	AC	CY

[Description]

- The content of AX register and the content of BC register are multiplied as signed data, upper 16 bits of the result are stored in the BC register, and lower 16 bits of the result are stored in the AX register.

[Description example]

MOVW AX, #0C000H
MOVW BC, #1000H
MULH
MOVW !addr16, AX
MOVW AX, BC
MOVW!addr16, AX; C000H and 1000H are multiplied, and the result FC000000H is stored in memory indicated by !addr16.

DIVHU

16-bit Divide Unsigned
Unsigned Division of Data

[Instruction format] DIVHU

[Operation] AX (quotient), DE (remainder) ← AX ÷ DE

[Operand]

Mnemonic	Operand (src)
DIVHU	

[Flag]

Z	AC	CY

[Description]

- The content of AX register is divided by the content of DE register, the quotient is stored in AX register, and the remainder is stored in DE register. The division treats the content of AX register and DE register as unsigned data. However, when the content of DE register is 0, the content of AX register is stored in DE register and then the content of AX register becomes 0FFFFH.

[Description example]

```
MOVW AX, #8081H
MOVW DE, #0002H
DIVHU
MOVW !addr16, AX
MOVW AX, DE
MOVW !addr16, AX;
```

8081H is divided by 0002H, and the quotient in AX register (4040H) and the remainder (0001H) in DE register are stored in memory indicated by !addr16.

DIVWU

32-bit Divide Unsigned
Unsigned Division of Data

[Instruction format] DIVWU

[Operation] BCAX (quotient), HLDE (remainder) ← BCAX ÷ HLDE

[Operand]

Mnemonic	Operand (src)
DIVWU	

[Flag]

Z	AC	CY

[Description]

- The content of BCAX register is divided by the content of HLDE register, the quotient is stored in BCAX register, and the remainder is stored in HLDE register. The division treats the content of BCAX register and HLDE register as unsigned data.
  However, when the content of HLDE register is 0, the content of BCAX register is stored in HLDE register and then the content of BCAX register becomes 0FFFFFFFFH.

[Description example]

```
MOVW AX, #8081H
MOVW BC, #8080H
MOVW DE, #0002H
MOVW HL, #0000H
DIVWU
MOVW !addr16, AX
MOVW AX, BC
MOVW !addr16, AX
MOVW AX, DE
MOVW !addr16, AX
MOVW AX, HL
MOVW !addr16, AX;   80808081H is divided by 00000002H, and the quotient (40404040H) in BCAX register and the
                    remainder (00000001H) in HLDE register are stored in memory indicated by !addr16.
```

MACHU

Multiply and Accumulate Unsigned
Unsigned Multiplication and Accumulation of Data

[Instruction format] MACHU

[Operation] MACR ← MACR + AX × BC

[Operand]

Mnemonic	Operand (src)
MACHU	

[Flag]

Z	AC	CY
	×	×

[Description]

- The content of AX register and the content of BC register are multiplied; the result and the content of MACR register are accumulated and then stored in MACR register.
- As a result of accumulation, when overflow occurs, CY flag is set (1), and when not, CY flag is cleared (0).
- AC flag becomes 0.
- Before multiplication and accumulation, set an initial value in MACR register. In addition since MACR register is fixed, if more than one result of multiplication and accumulation are needed, save the content of MACR register first.

[Description example]

```
MOVW AX, #00000H
MOVW !0FFF2H, AX
MOVW !0FFF0H, AX
MOVW AX, #0C000H
MOVW BC, #01000H
MACHU
MOVW AX, !0FFF2H
MOVW !addr16, AX
MOVW AX, !0FFF0H
MOVW !addr16, AX;
```

The content of AX register and the content of BC register are multiplied, the result and the content of MACR register are accumulated and then stored in MACR register.

MACH

Multiply and Accumulate Signed
Signed Multiplication and Accumulation of Data

[Instruction format] MACH

[Operation] MACR ← MACR + AX × BC

[Operand]

Mnemonic	Operand (src)
MACH	

[Flag]

Z	AC	CY
	×	×

[Description]

- The content of AX register and the content of BC register are multiplied; the result and the content of MACR register are accumulated and then stored in MACR register.
- As a result of accumulation, if overflow occurs, CY flag is set (1), and if not, CY flag is cleared (0). The overflow means cases that an added result of a plus accumulated value and a plus multiplied value has exceeded 7FFFFFFFH and that an added result of a minus accumulated value and a minus multiplied value has exceeded 80000000H.
- As a result of operations, when MACR register has a plus value, AC flag is cleared (0), and when it has a minus value, AC flag is set (1).
- Before multiplication and accumulation, set an initial value in MACR register. In addition since MACR register is fixed, if more than one result of multiplication and accumulation are needed, save the content of MACR register first.

[Description example]

```
MOVW AX, #00000H
MOVW !0FFF0H, AX
MOVW AX, #08000H
MOVW !0FFF2H, AX
MOVW AX, #00001H
MOVW !0FFF0H, AX
MOVW AX, #07FFFH
MOVW BC, #0FFFFH
MACH
MOVW AX, !0FFF2H
MOVW !addr16, AX
MOVW AX, !0FFF0H
MOVW !addr16, AX;
```

The content of AX register and that of BC register are multiplied, the result and the content of MACR register are accumulated and then stored in MACR register.

6.6 Increment/Decrement Instructions

The following instructions are increment/decrement instructions.

INC

Increment
Byte Data Increment

[Instruction format] INC dst

[Operation] dst ← dst + 1

[Operand]

Mnemonic	Operand (dst)
INC	r
	saddr
	!addr16
	[HL+byte]
	ES:!addr16
	ES:[HL+byte]

[Flag]

Z	AC	CY
×	×	

[Description]

- The destination operand (dst) contents are incremented by only one.
- If the increment result is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the increment generates a carry for bit 4 out of bit 3, the AC flag is set (1). In all other cases, the AC flag is cleared (0).
- Because this instruction is frequently used for increment of a counter for repeated operations and an indexed addressing offset register, the CY flag contents are not changed (to hold the CY flag contents in multiple-byte operation).

[Description example]

INC B; The B register is incremented.

DEC

Decrement
Byte Data Decrement

[Instruction format] DEC dst

[Operation] dst ← dst – 1

[Operand]

Mnemonic	Operand (dst)
DEC	r
	saddr
	!addr16
	[HL+byte]
	ES:!addr16
	ES:[HL+byte]

[Flag]

Z	AC	CY
×	×	

[Description]

- The destination operand (dst) contents are decremented by only one.
- If the decrement result is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the decrement generates a carry for bit 3 out of bit 4, the AC flag is set (1). In all other cases, the AC flag is cleared (0).
- Because this instruction is frequently used for a counter for repeated operations, the CY flag contents are not changed (to hold the CY flag contents in multiple-byte operation).
- If dst is the B or C register or saddr, and it is not desired to change the AC and CY flag contents, the DBNZ instruction can be used.

[Description example]

DEC FFE92H; The contents at address FFE92H are decremented.

INCW

Increment Word
Word Data Increment

[Instruction format] INCW dst

[Operation] dst ← dst + 1

[Operand]

Mnemonic	Operand (dst)
INCW	rp
	saddrp
	!addr16
	[HL+byte]
	ES:!addr16
	ES:[HL+byte]

[Flag]

Z	AC	CY

[Description]

- The destination operand (dst) contents are incremented by only one.
- Because this instruction is frequently used for increment of a register (pointer) used for addressing, the Z, AC and CY flag contents are not changed.

[Description example]

INCW HL; The HL register is incremented.

DECW

Decrement Word
Word Data Decrement

[Instruction format] DECW dst

[Operation] dst ← dst – 1

[Operand]

Mnemonic	Operand (dst)
DECW	rp
	saddrp
	!addr16
	[HL+byte]
	ES:!addr16
	ES:[HL+byte]

[Flag]

Z	AC	CY

[Description]

- The destination operand (dst) contents are decremented by only one.
- Because this instruction is frequently used for decrement of a register (pointer) used for addressing, the Z, AC and CY flag contents are not changed.

[Description example]

DECW DE; The DE register is decremented.

6.7 Shift Instructions

The following instructions are shift instructions.

SHR

Shift Right
Logical Shift to the Right

[Instruction format] SHR dst, cnt

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

[Operand]

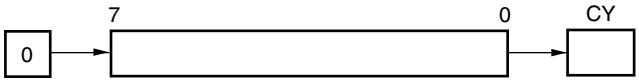
Mnemonic	Operand (dst, cnt)
SHR	A, cnt

[Flag]

Z	AC	CY
		×

[Description]

- The destination operand (dst) specified by the first operand is shifted to the right the number of times specified by cnt.
- “0” is entered to the MSB (bit 7) and the value shifted last from bit 0 is entered to CY.
- cnt can be specified as any value from 1 to 7.



[Description example]

SHR A, 3; When the A register's value is F5H, A = 1EH and CY = 1.

A = 1111_0101B CY = 0
A = 0111_1010B CY = 1 1 time
A = 0011_1101B CY = 0 2 times
A = 0001_1110B CY = 1 3 times

SHRW

Shift Right Word
Logical Shift to the Right

[Instruction format] SHRW dst, cnt

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

[Operand]

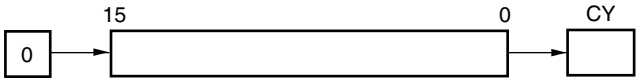
Mnemonic	Operand (dst, cnt)
SHRW	AX, cnt

[Flag]

Z	AC	CY
		×

[Description]

- The destination operand (dst) specified by the first operand is shifted to the right the number of times specified by cnt.
- “0” is entered to the MSB (bit 15) and the value shifted last from bit 0 is entered to CY.
- cnt can be specified as any value from 1 to 15.



[Description example]

SHRW AX 3; When the AX register's value is AAF5H, AX = 155EH and CY = 1.

AX = 1010_1010_1111_0101B CY = 0
AX = 0101_0101_0111_1010B CY = 1 1 time
AX = 0010_1010_1011_1101B CY = 0 2 times
AX = 0001_0101_0101_1110B CY = 1 3 times

SHL

Shift Left
Logical Shift to the Left

[Instruction format] SHL dst, cnt

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

[Operand]

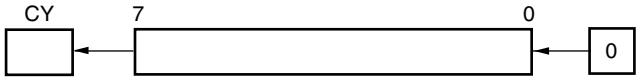
Mnemonic	Operand (dst, cnt)
SHL	A, cnt
	B, cnt
	C, cnt

[Flag]

Z	AC	CY
		×

[Description]

- The destination operand (dst) specified by the first operand is shifted to the left the number of times specified by cnt.
- “0” is entered to the LSB (bit 0) and the value shifted last from bit 7 is entered to CY.
- cnt can be specified as any value from 1 to 7.



[Description example]

SHL A, 3; When the A register's value is 5DH, A = E8H and CY = 0.

CY = 0 A = 0101_1101B
CY = 0 A = 1011_1010B 1 time
CY = 1 A = 0111_0100B 2 times
CY = 0 A = 0110_1000B 3 times

SHLW

Shift Left Word
Logical Shift to the Left

[Instruction format] SHLW dst, cnt

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

[Operand]

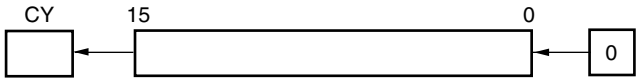
Mnemonic	Operand (dst, cnt)
SHLW	AX, cnt
	BC, cnt

[Flag]

Z	AC	CY
		×

[Description]

- The destination operand (dst) specified by the first operand is shifted to the left the number of times specified by cnt.
- “0” is entered to the LSB (bit 0) and the value shifted last from bit 15 is entered to CY.
- cnt can be specified as any value from 1 to 15.



[Description example]

SHLW BC, 3; When the BC register's value is C35DH, BC = 1AE8H and CY = 0.

CY = 0 BC = 1100_0011_0101_1101B
CY = 1 BC = 1000_0110_1011_1010B 1 time
CY = 1 BC = 0000_1101_0111_0100B 2 times
CY = 0 BC = 0001_1010_1110_1000B 3 times

SAR

Shift Arithmetic Right
Arithmetic Shift to the Right

[Instruction format] SAR dst, cnt

[Operation] (CY ← dst_0, dst_{m-1} ← dst_m, dst_7 ← dst_7) × cnt

[Operand]

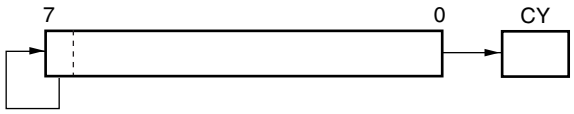
Mnemonic	Operand (dst, cnt)
SHR	A, cnt

[Flag]

Z	AC	CY
		×

[Description]

- The destination operand (dst) specified by the first operand is shifted to the right the number of times specified by cnt.
- The same value is retained in the MSB (bit 7), and the value shifted last from bit 0 is entered to CY.
- cnt can be specified as any value from 1 to 7.



[Description example]

SAR A, 4; When the A register's value is 8CH, A = F8H and CY = 1.

A = 1000_1100B CY = 0
A = 1100_0110B CY = 0 1 time
A = 1110_0011B CY = 0 2 times
A = 1111_0001B CY = 1 3 times
A = 1111_1000B CY = 1 4 times

SARW

Shift Arithmetic Right Word
Arithmetic Shift to the Right

[Instruction format] SARW dst, cnt

[Operation] (CY ← dst_0, dst_{m-1} ← dst_m, dst_{15} ← dst_{15}) × cnt

[Operand]

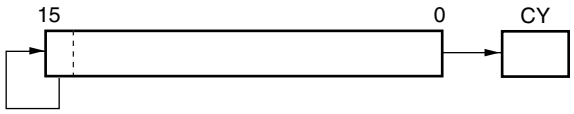
Mnemonic	Operand (dst, cnt)
SARW	AX, cnt

[Flag]

Z	AC	CY
		×

[Description]

- The destination operand (dst) specified by the first operand is shifted to the right the number of times specified by cnt.
- The same value is retained in the MSB (bit 15), and the value shifted last from bit 0 is entered to CY.
- cnt can be specified as any value from 1 to 15.



[Description example]

SAR AX, 4; When the AX register's value is A28CH, AX = FA28H and CY = 1.

AX = 1010_0010_1000_1100B CY = 0
AX = 1101_0001_0100_0110B CY = 0 1 time
AX = 1110_1000_1010_0011B CY = 0 2 times
AX = 1111_0100_0101_0001B CY = 1 3 times
AX = 1111_1010_0010_1000B CY = 1 4 times

6.8 Rotate Instructions

The following instructions are rotate instructions.

ROR

Rotate Right
Byte Data Rotation to the Right

[Instruction format] ROR dst, cnt

[Operation] (CY, $dst_7 \leftarrow dst_0$, $dst_{m-1} \leftarrow dst_m$) × one time

[Operand]

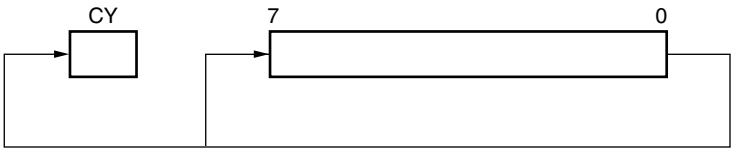
Mnemonic	Operand (dst, cnt)
ROR	A, 1

[Flag]

Z	AC	CY
		×

[Description]

- The destination operand (dst) contents specified by the 1st operand are rotated to the right just once.
- The LSB (bit 0) contents are simultaneously rotated to the MSB (bit 7) and transferred to the CY flag.



[Description example]

ROR A, 1; The A register contents are rotated to the right by one bit.

ROL

Rotate Left
Byte Data Rotation to the Left

[Instruction format] ROL dst, cnt

[Operation] (CY, $dst_0 \leftarrow dst_0$, $dst_{m+1} \leftarrow dst_m$) × one time

[Operand]

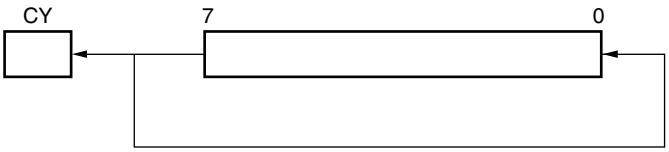
Mnemonic	Operand (dst, cnt)
ROL	A, 1

[Flag]

Z	AC	CY
		×

[Description]

- The destination operand (dst) contents specified by the 1st operand are rotated to the left just once.
- The MSB (bit 7) contents are simultaneously rotated to the LSB (bit 0) and transferred to the CY flag.



[Description example]

ROL A, 1; The A register contents are rotated to the left by one bit.

RORC

Rotate Right with Carry
Byte Data Rotation to the Right with Carry

[Instruction format] RORC dst, cnt

[Operation] (CY ← dst_0, dst_7 ← CY, dst_{m-1} ← dst_m) × one time

[Operand]

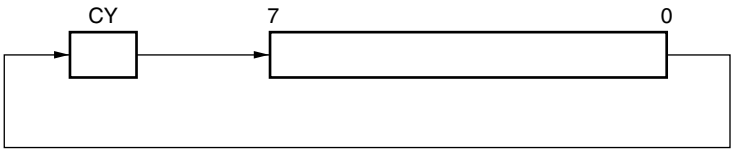
Mnemonic	Operand (dst, cnt)
RORC	A, 1

[Flag]

Z	AC	CY
		×

[Description]

- The destination operand (dst) contents specified by the 1st operand are rotated just once to the right with carry.



[Description example]

RORC A, 1; The A register contents are rotated to the right by one bit including the CY flag.

ROLC

Rotate Left with Carry
Byte Data Rotation to the Left with Carry

[Instruction format] ROLC dst, cnt

[Operation] (CY ← dst_7, dst_0 ← CY, dst_{m+1} ← dst_m) × one time

[Operand]

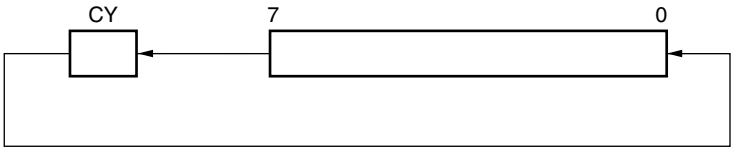
Mnemonic	Operand (dst, cnt)
ROLC	A, 1

[Flag]

Z	AC	CY
		×

[Description]

- The destination operand (dst) contents specified by the 1st operand are rotated just once to the left with carry.



[Description example]

ROLC A, 1; The A register contents are rotated to the left by one bit including the CY flag.

ROLWC

Rotate Left word with Carry
Word Data Rotation to the Left with Carry

[Instruction format] ROLWC dst, cnt

[Operation] (CY ← dst_{15}, dst_0 ← CY, dst_{m+1} ← dst_m) × one time

[Operand]

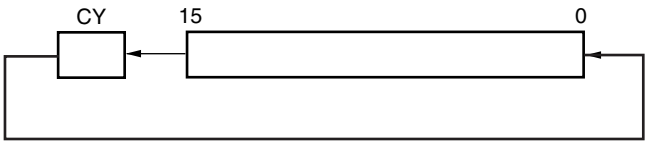
Mnemonic	Operand (dst, cnt)
ROLWC	AX, 1
	BC, 1

[Flag]

Z	AC	CY
		×

[Description]

- The destination operand (dst) contents specified by the 1st operand are rotated just once to the left with carry.



[Description example]

ROLWC BC, 1; The BC register contents are rotated to the left by one bit including the CY flag.

6.9 Bit Manipulation Instructions

The following instructions are bit manipulation instructions.

MOV1

Move Single Bit
1 Bit Data Transfer

[Instruction format] MOV1 dst, src

[Operation] dst ← src

[Operand]

Mnemonic	Operand (dst, src)
MOV1	CY, saddr.bit
	CY, sfr.bit
	CY, A.bit
	CY, PSW.bit
	CY, [HL].bit
	saddr.bit, CY

Mnemonic	Operand (dst, src)
MOV1	sfr.bit, CY
	A.bit, CY
	PSW.bit, CY
	[HL].bit, CY
	CY, ES:[HL].bit
	ES:[HL].bit, CY

[Flag]

dst = CY

Z	AC	CY
		×

dst = PSW.bit

Z	AC	CY
×	×	

In all other cases

Z	AC	CY

[Description]

- Bit data of the source operand (src) specified by the 2nd operand is transferred to the destination operand (dst) specified by the 1st operand.
- When the destination operand (dst) is CY or PSW.bit, only the corresponding flag is changed.
- All interrupt requests are not acknowledged between the MOV1 PSW.bit, CY instruction and the next instruction.

[Description example]

MOV1 P3.4, CY; The CY flag contents are transferred to bit 4 of port 3.

AND1

And Single Bit 1 Bit Data Logical Product

[Instruction format] AND1 dst, src

[Operation] dst ← dst ∧ src

[Operand]

Mnemonic	Operand (dst, src)
AND1	CY, saddr.bit
	CY, sfr.bit
	CY, A.bit
	CY, PSW.bit
	CY, [HL].bit
	CY, ES:[HL].bit

[Flag]

Z	AC	CY
		×

[Description]

- Logical product of bit data of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand is obtained and the result is stored in the destination operand (dst).
- The operation result is stored in the CY flag (because of the destination operand (dst)).

[Description example]

AND1 CY, FFE7FH.3; Logical product of FFE7FH bit 3 and the CY flag is obtained and the result is stored in the CY flag.

OR1

Or Single Bit
1 Bit Data Logical Sum

[Instruction format] OR1 dst, src

[Operation] dst ← dst ∨ src

[Operand]

Mnemonic	Operand (dst, src)
OR1	CY, saddr.bit
	CY, sfr.bit
	CY, A.bit
	CY, PSW.bit
	CY, [HL].bit
	CY, ES:[HL].bit

[Flag]

Z	AC	CY
		×

[Description]

- The logical sum of bit data of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand is obtained and the result is stored in the destination operand (dst).
- The operation result is stored in the CY flag (because of the destination operand (dst)).

[Description example]

OR1 CY, P2.5; The logical sum of port 2 bit 5 and the CY flag is obtained and the result is stored in the CY flag.

XOR1

Exclusive Or Single Bit
1 Bit Data Exclusive Logical Sum

[Instruction format] XOR1 dst, src

[Operation] dst ← dst ⩛ src

[Operand]

Mnemonic	Operand (dst, src)
XOR1	CY, saddr.bit
	CY, sfr.bit
	CY, A.bit
	CY, PSW.bit
	CY, [HL].bit
	CY, ES:[HL].bit

[Flag]

Z	AC	CY
		×

[Description]

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

[Description example]

XOR1 CY, A.7; The exclusive logical sum of the A register bit 7 and the CY flag is obtained and the result is stored in the CY flag.

SET1

Set Single Bit (Carry Flag)
1 Bit Data Set

[Instruction format] SET1 dst

[Operation] dst ← 1

[Operand]

Mnemonic	Operand (dst)
SET1	saddr.bit
	sfr.bit
	A.bit
	!addr16.bit
	PSW.bit
	[HL].bit
	ES:!addr16.bit
	ES:[HL].bit
	CY

[Flag]

dst = PSW.bit

Z	AC	CY
×	×	×

dst = CY

Z	AC	CY
		1

In all other cases

Z	AC	CY

[Description]

- The destination operand (dst) is set (1).
- When the destination operand (dst) is CY or PSW.bit, only the corresponding flag is set (1).
- All interrupt requests are not acknowledged between the SET1 PSW.bit instruction and the next instruction.

[Description example]

SET1 FFE55H.1; Bit 1 of FFE55H is set (1).

CLR1

Clear Single Bit (Carry Flag)
1 Bit Data Clear

[Instruction format] CLR1 dst

[Operation] dst ← 0

[Operand]

Mnemonic	Operand (dst)
CLR1	saddr.bit
	sfr.bit
	A.bit
	!addr16.bit
	PSW.bit
	[HL].bit
	ES:!addr16.bit
	ES:[HL].bit
	CY

[Flag]

dst = PSW.bit

Z	AC	CY
×	×	×

dst = CY

Z	AC	CY
		0

In all other cases

Z	AC	CY

[Description]

- The destination operand (dst) is cleared (0).
- When the destination operand (dst) is CY or PSW.bit, only the corresponding flag is cleared (0).
- All interrupt requests are not acknowledged between the CLR1 PSW.bit instruction and the next instruction.

[Description example]

CLR1 P3.7; Bit 7 of port 3 is cleared (0).

NOT1

Not Single Bit (Carry Flag)
1 Bit Data Logical Negation

[Instruction format] NOT1 dst

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

[Operand]

Mnemonic	Operand (dst)
NOT1	CY

[Flag]

Z	AC	CY
		×

[Description]

- The CY flag is inverted.

[Description example]

NOT1 CY; The CY flag is inverted.

6.10 Call Return Instructions

The following instructions are call return instructions.

CALL

Call
Subroutine Call

[Instruction format] CALL target

[Operation]

(SP–2) ← (PC+n)S,
(SP–3) ← (PC+n)H,
(SP–4) ← (PC+n)L,
SP ← SP–4
PC ← target

Remark n is 4 when using !!addr20, 3 when using !addr16 or $!addr20, and 2 when using AX, BC, DE, or HL.

[Operand]

Mnemonic	Operand (target)
CALL	AX
	BC
	DE
	HL
	$!addr20
	!addr16
	!!addr20

[Flag]

Z	AC	CY

[Description]

- This is a subroutine call with a 20/16-bit absolute address or a register indirect address.
- The start address (PC+n) of the next instruction is saved in the stack and is branched to the address specified by the target operand (target).

[Description example]

CALL !!3E000H; Subroutine call to 3E000H

CALLT

Call Table
Subroutine Call (Refer to the Call Table)

[Instruction format] CALLT [addr5]

[Operation]

(SP−2) ← (PC+2)S,
(SP−3) ← (PC+2)H,
(SP−4) ← (PC+2)L,
PCS ← 0000,
PCH ← (0000, addr5+1),
PCL ← (0000, addr5)
SP ← SP−4

[Operand]

Mnemonic	Operand ([addr5])
CALLT	[addr5]

[Flag]

Z	AC	CY

[Description]

- This is a subroutine call for call table reference.
- The start address (PC+2) of the next instruction is saved in the stack and is branched to the address indicated with the word data of a call table (specify the even addresses of 00080H to 000BFH, with the higher 4 bits of the address fixed to 0000B, and the lower 16 bits indicated with addr5).

[Description example]

CALLT [80H]; Subroutine call to the word data addresses 00080H and 00081H.

[Remark]

Only even-numbered addresses can be specified (odd-numbered addresses cannot be specified).
addr5: Immediate data or label from 0080H to 00BFH (even-numbered addresses only)
(16-bit even addresses of 0080H to 00BFH, with bits 15 to 6 fixed to 0000000010B, bit 0 fixed to 0B, and the five bits of bits 5 to 1 varied)

BRK

Break
Software Vectored Interrupt

[Instruction format] BRK

[Operation]

(SP−1) ← PSW,
(SP−2) ← (PC+2)S,
(SP−3) ← (PC+2)H,
(SP−4) ← (PC+2)L,
PCS ← 0000,
PCH ← (0007FH),
PCL ← (0007FH),
SP ← SP−4,
IE ← 0

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- This is a software interrupt instruction.
- PSW and the next instruction address (PC+2) are saved to the stack. After that, the IE flag is cleared (0) and the saved data is branched to the address indicated with the word data at the vector address (0007EH, 0007FH). Because the IE flag is cleared (0), the subsequent maskable vectored interrupts are disabled.
- The RETB instruction is used to return from the software vectored interrupt generated with this instruction.

RET

Return
Return from Subroutine

[Instruction format] RET

[Operation]

$PC_L \leftarrow (SP)$,
$PC_H \leftarrow (SP+1)$,
$PC_S \leftarrow (SP+2)$,
$SP \leftarrow SP+4$

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- This is a return instruction from the subroutine call made with the CALL and CALLT instructions.
- The word data saved to the stack returns to the PC, and the program returns from the subroutine.

RETI

Return from Interrupt
Return from Hardware Vectored Interrupt

[Instruction format] RETI

[Operation]

$PC_L \leftarrow (SP)$,
$PC_H \leftarrow (SP+1)$,
$PC_S \leftarrow (SP+2)$,
$PSW \leftarrow (SP+3)$,
$SP \leftarrow SP+4$,

[Operand]

None

[Flag]

Z	AC	CY
R	R	R

[Description]

- This is a return instruction from the vectored interrupt.
- The data saved to the stack returns to the PC and the PSW, and the program returns from the interrupt servicing routine.
- This instruction cannot be used for return from the software interrupt with the BRK instruction.
- None of interrupts are acknowledged between this instruction and the next instruction to be executed.

[Caution]

Be sure to use the RETI instruction for restoring from the non-maskable interrupt.

RETB

Return from Break
Return from Software Vectored Interrupt

[Instruction format] RETB

[Operation]

$PC_L \leftarrow (SP)$,
$PC_H \leftarrow (SP+1)$,
$PC_S \leftarrow (SP+2)$,
$PSW \leftarrow (SP+3)$,
$SP \leftarrow SP+4$

[Operand]

None

[Flag]

Z	AC	CY
R	R	R

[Description]

- This is a return instruction from the software interrupt generated with the BRK instruction.
- The data saved in the stack returns to the PC and the PSW, and the program returns from the interrupt servicing routine.
- None of interrupts are acknowledged between this instruction and the next instruction to be executed.

6.11 Stack Manipulation Instructions

The following instructions are stack manipulation instructions.

PUSH

Push
Push

[Instruction format] PUSH src

[Operation]

When src = rp

(SP−1) ← rp_H,
(SP−2) ← rp_L,
SP ← SP−2

When src = PSW

(SP−1) ← PSW
(SP−2) ← 00H
SP ← SP−2

[Operand]

Mnemonic	Operand (src)
PUSH	PSW
	rp

[Flag]

Z	AC	CY

[Description]

- The data of the register specified by the source operand (src) is saved to the stack.

[Description example]

PUSH AX; AX register contents are saved to the stack.

POP

Pop
Pop

[Instruction format] POP dst

[Operation]

When dst = rp

$rp_L \leftarrow (SP)$,
$rp_H \leftarrow (SP+1)$,
$SP \leftarrow SP+2$

When dst = PSW

$PSW \leftarrow (SP+1)$
$SP \leftarrow SP+2$

[Operand]

Mnemonic	Operand (dst)
POP	PSW
	rp

[Flag]

dst = rp

Z	AC	CY

dst = PSW

Z	AC	CY
R	R	R

[Description]

- Data is returned from the stack to the register specified by the destination operand (dst).
- When the operand is PSW, each flag is replaced with stack data.
- None of interrupts are acknowledged between the POP PSW instruction and the subsequent instruction.

[Description example]

POP AX; The stack data is returned to the AX register.

MOVW SP, src
MOVW AX, SP

Move Word
Word Data Transfer with Stack Pointer

[Instruction format] MOVW dst, src

[Operation] dst ← src

[Operand]

Mnemonic	Operand (dst, src)
MOVW	SP, #word
	SP, AX
	AX, SP
	HL, SP
	BC, SP
	DE, SP

[Flag]

Z	AC	CY

[Description]

- This is an instruction to manipulate the stack pointer contents.
- The source operand (src) specified by the 2nd operand is stored in the destination operand (dst) specified by the 1st operand.

[Description example]

MOVW SP, #FE20H; FE20H is stored in the stack pointer.

ADDW SP, #byte

Add stack pointer
Addition of Stack Pointer

[Instruction format] ADDW SP, src

[Operation] SP ← SP+src

[Operand]

Mnemonic	Operand (src)
ADDW	SP, #byte

[Flag]

Z	AC	CY

[Description]

- The stack pointer specified by the first operand and the source operand (src) specified by the second operand are added and the result is stored in the stack pointer.

[Description example]

ADDW SP, #12H; Stack pointer and 12H are added, and the result is stored in the stack pointer.

SUBW SP, #byte

Sub stack pointer
Subtraction of Stack Pointer

[Instruction format] SUBW SP, src

[Operation] SP ← SP−src

[Operand]

Mnemonic	Operand (src)
SUBW	SP, #byte

[Flag]

Z	AC	CY

[Description]

- Source operand (src) specified by the second operand is subtracted from the stack pointer specified by the first operand, and the result is stored in the stack pointer.

[Description example]

SUBW SP, #12H; 12H is subtracted from the stack pointer, and the result is stored in the stack pointer.

6.12 Unconditional Branch Instruction

The following instruction is an unconditional branch instruction.

BR ... 169

BR

Branch
Unconditional Branch

[Instruction format] BR target

[Operation] PC ← target

[Operand]

Mnemonic	Operand (target)
BR	AX
	$addr20
	$!addr20
	!addr16
	!!addr20

[Flag]

Z	AC	CY

[Description]

- This is an instruction to branch unconditionally.
- The word data of the target address operand (target) is transferred to PC and branched.

[Description example]

BR !!12345H; Branch to address 12345H.

6.13 Conditional Branch Instructions

The following instructions are conditional branch instructions.

BC

Branch if Carry
Conditional Branch with Carry Flag (CY = 1)

[Instruction format] BC $addr20

[Operation] PC ← PC+2+jdisp8 if CY = 1

[Operand]

Mnemonic	Operand ($addr20)
BC	$addr20

[Flag]

Z	AC	CY

[Description]

- When CY = 1, data is branched to the address specified by the operand.
  When CY = 0, no processing is carried out and the subsequent instruction is executed.

[Description example]

BC $00300H; When CY = 1, data is branched to 00300H (with the start of this instruction set in the range of addresses 0027FH to 0037EH).

BNC

Branch if Not Carry
Conditional Branch with Carry Flag (CY = 0)

[Instruction format] BNC $addr20

[Operation] PC ← PC+2+jdisp8 if CY = 0

[Operand]

Mnemonic	Operand ($addr20)
BNC	$addr20

[Flag]

Z	AC	CY

[Description]

- When CY = 0, data is branched to the address specified by the operand.
  When CY = 1, no processing is carried out and the subsequent instruction is executed.

[Description example]

BNC $00300H; When CY = 0, data is branched to 00300H (with the start of this instruction set in the range of addresses 0027FH to 0037EH).

BZ

Branch if Zero
Conditional Branch with Zero Flag (Z = 1)

[Instruction format] BZ $addr20

[Operation] PC ← PC+2+jdisp8 if Z = 1

[Operand]

Mnemonic	Operand ($addr20)
BZ	$addr20

[Flag]

Z	AC	CY

[Description]

- When Z = 1, data is branched to the address specified by the operand.
  When Z = 0, no processing is carried out and the subsequent instruction is executed.

[Description example]

DEC B

BZ $003C5H; When the B register is 0, data is branched to 003C5H (with the start of this instruction set in the range of addresses 00344H to 00443H).

BNZ

Branch if Not Zero
Conditional Branch with Zero Flag (Z = 0)

[Instruction format] BNZ $addr20

[Operation] PC ← PC+2+jdisp8 if Z = 0

[Operand]

Mnemonic	Operand ($addr20)
BNZ	$addr20

[Flag]

Z	AC	CY

[Description]

- When Z = 0, data is branched to the address specified by the operand.
  When Z = 1, no processing is carried out and the subsequent instruction is executed.

[Description example]

CMP A, #55H

BNZ $00A39H; If the A register is not 55H, data is branched to 00A39H (with the start of this instruction set in the range of addresses 009B8H to 00AB7H).

BH

Branch if Higher than
Conditional branch by numeric value comparison ((Z ∨ CY) = 0)

[Instruction format] BH $addr20

[Operation] PC ← PC+3+jdisp8 if (Z ∨ CY) = 0

[Operand]

Mnemonic	Operand ($addr20)
BH	$addr20

[Flag]

Z	AC	CY

[Description]

- When (Z ∨ CY) = 0, data is branched to the address specified by the operand.
  When (Z ∨ CY) = 1, no processing is carried out and the subsequent instruction is executed.
- This instruction is used to judge which of the unsigned data values is higher. It is detected whether the first operand is higher than the second operand in the CMP instruction immediately before this instruction.

[Description example]

CMP A, C

BH $00356H; Branch to address 00356H when the A register contents are greater than the C register contents (start of the BH instruction, however, is in addresses 002D4H to 003D3H).

BNH

Branch if Not Higher than
Conditional branch by numeric value comparison ((Z ∨ CY) = 1)

[Instruction format] BNH $addr20

[Operation] PC ← PC+3+jdisp8 if (Z ∨ CY) = 1

[Operand]

Mnemonic	Operand ($addr20)
BNH	$addr20

[Flag]

Z	AC	CY

[Description]

- When (Z ∨ CY) = 1, data is branched to the address specified by the operand.
  When (Z ∨ CY) = 0, no processing is carried out and the subsequent instruction is executed.
- This instruction is used to judge which of the unsigned data values is higher. It is detected whether the first operand is not higher than the second operand (the first operand is equal to or lower than the second operand) in the CMP instruction immediately before this instruction.

[Description example]

CMP A, C

BNH $00356H; Branch to address 00356H when the A register contents are equal to or lower than the C register contents (start of the BNH instruction, however, is in addresses 002D4H to 003D3H).

BT

Branch if True
Conditional Branch by Bit Test (Byte Data Bit = 1)

[Instruction format] BT bit, $addr20

[Operation] PC ← PC+b+jdisp8 if bit = 1

[Operand]

Mnemonic	Operand (bit, $addr20)	b (Number of bytes)
BT	saddr.bit, $addr20	4
	sfr.bit, $addr20	4
	A.bit, $addr20	3
	PSW.bit, $addr20	4
	[HL].bit, $addr20	3
	ES:[HL].bit, $addr20	4

[Flag]

Z	AC	CY

[Description]

- If the 1st operand (bit) contents have been set (1), data is branched to the address specified by the 2nd operand ($addr20).
  If the 1st operand (bit) contents have not been set (1), no processing is carried out and the subsequent instruction is executed.

[Description example]

BT FFE47H.3, $0055CH; When bit 3 at address FFE47H is 1, data is branched to 0055CH (with the start of this instruction set in the range of addresses 004DAH to 005D9H).

BF

Branch if False
Conditional Branch by Bit Test (Byte Data Bit = 0)

[Instruction format] BF bit, $addr20

[Operation] PC ← PC+b+jdisp8 if bit = 0

[Operand]

Mnemonic	Operand (bit, $addr20)	b (Number of bytes)
BF	saddr.bit, $addr20	4
	sfr.bit, $addr20	4
	A.bit, $addr20	3
	PSW.bit, $addr20	4
	[HL].bit, $addr20	3
	ES:[HL].bit, $addr20	4

[Flag]

Z	AC	CY

[Description]

- If the 1st operand (bit) contents have been cleared (0), data is branched to the address specified by the 2nd operand ($addr20).
  If the 1st operand (bit) contents have not been cleared (0), no processing is carried out and the subsequent instruction is executed.

[Description example]

BF P2.2, $01549H; When bit 2 of port 2 is 0, data is branched to address 01549H (with the start of this instruction set in the range of addresses 014C6H to 015C5H).

BTCLR

Branch if True and Clear
Conditional Branch and Clear by Bit Test (Byte Data Bit =1)

[Instruction format] BTCLR bit, $addr20

[Operation] PC ← PC+b+jdisp8 if bit = 1, then bit ← 0

[Operand]

Mnemonic	Operand (bit, $addr20)	b (Number of bytes)
BTCLR	saddr.bit, $addr20	4
	sfr.bit, $addr20	4
	A.bit, $addr20	3
	PSW.bit, $addr20	4
	[HL].bit, $addr20	3
	ES:[HL].bit, $addr20	4

[Flag]

bit = PSW.bit

Z	AC	CY
×	×	×

In all other cases

Z	AC	CY

[Description]

- If the 1st operand (bit) contents have been set (1), they are cleared (0) and branched to the address specified by the 2nd operand.
  If the 1st operand (bit) contents have not been set (1), no processing is carried out and the subsequent instruction is executed.
- When the 1st operand (bit) is PSW.bit, the corresponding flag contents are cleared (0).
- All interrupt requests are not acknowledged between the BTCLR PSW.bit, $addr20 instruction and the next instruction.

[Description example]

BTCLR PSW.0, $00356H; When bit 0 (CY flag) of PSW is 1, the CY flag is cleared to 0 and branched to address 00356H (with the start of this instruction set in the range of addresses 002D4H to 003D3H).

6.14 Conditional Skip Instructions

The following instructions are conditional skip instructions.

SKC

Skip if CY
Skip with Carry Flag (CY = 1)

[Instruction format] SKC

[Operation] Next instruction skip if CY = 1

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- When CY = 1, the next instruction is skipped. The subsequent instruction is a NOP and one clock of execution time is consumed. However, if the next instruction is a PREFIX instruction (indicated by "ES:"), two clocks of execution time are consumed.
- When CY = 0, the next instruction is executed.
- All interrupt requests are not acknowledged between this instruction and the next instruction.

[Description example]

MOV A, #55H
SKC
ADD A, #55H; The A register's value = AAH when CY = 0, and 55H when CY = 1.

SKNC

Skip if not CY
Skip with Carry Flag (CY = 0)

[Instruction format] SKNC

[Operation] Next instruction skip if CY = 0

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- When CY = 0, the next instruction is skipped. The subsequent instruction is a NOP and one clock of execution time is consumed. However, if the next instruction is a PREFIX instruction (indicated by "ES:"), two clocks of execution time are consumed.
- When CY = 1, the next instruction is executed.
- All interrupt requests are not acknowledged between this instruction and the next instruction.

[Description example]

MOV A, #55H
SKNC
ADD A, #55H; The A register's value = AAH when CY = 1, and 55H when CY = 0.

SKZ

Skip if Z
Skip with Zero Flag (Z = 1)

[Instruction format] SKZ

[Operation] Next instruction skip if Z = 1

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- When Z = 1, the next instruction is skipped. The subsequent instruction is a NOP and one clock of execution time is consumed. However, if the next instruction is a PREFIX instruction (indicated by "ES:"), two clocks of execution time are consumed.
- When Z = 0, the next instruction is executed.
- All interrupt requests are not acknowledged between this instruction and the next instruction.

[Description example]

MOV A, #55H
SKZ
ADD A, #55H; The A register's value = AAH when Z = 0, and 55H when Z = 1.

SKNZ

Skip if not Z
Skip with Zero Flag (Z = 0)

[Instruction format] SKNZ

[Operation] Next instruction skip if Z = 0

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- When Z = 0, the next instruction is skipped. The subsequent instruction is a NOP and one clock of execution time is consumed. However, if the next instruction is a PREFIX instruction (indicated by “ES:”), two clocks of execution time are consumed.
- When Z = 1, the next instruction is executed.
- All interrupt requests are not acknowledged between this instruction and the next instruction.

[Description example]

MOV A, #55H
SKNZ
ADD A, #55H; The A register’s value = AAH when Z = 1, and 55H when Z = 0.

SKH

Skip if Higher than
Skip with numeric value comparison ((Z ∨ CY) = 0)

[Instruction format] SKH

[Operation] Next instruction skip if (Z ∨ CY) = 0

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- When (Z ∨ CY) = 0, the next instruction is skipped. The subsequent instruction is a NOP and one clock of execution time is consumed. However, if the next instruction is a PREFIX instruction (indicated by "ES:"), two clocks of execution time are consumed.
- When (Z ∨ CY) = 1, the next instruction is executed.
- All interrupt requests are not acknowledged between this instruction and the next instruction.

[Description example]

```
CMP A, #80H
SKH
CALL !!TARGET;
```

When the A register contents are higher than 80H, the CALL instruction is skipped and the next instruction is executed.
When the A register contents are 80H or lower, the next CALL instruction is executed and execution is branched to the target address.

SKNH

Skip if not Higher than
Skip with numeric value comparison ((Z ∨ CY) = 1)

[Instruction format] SKNH

[Operation] Next instruction skip if (Z ∨ CY) = 1

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- When (Z ∨ CY) = 1, the next instruction is skipped. The subsequent instruction is a NOP and one clock of execution time is consumed. However, if the next instruction is a PREFIX instruction (indicated by “ES:”), two clocks of execution time are consumed.
- When (Z ∨ CY) = 0, the next instruction is executed.
- All interrupt requests are not acknowledged between this instruction and the next instruction.

[Description example]

```
CMP A, #80H
SKNH
CALL !!TARGET;
```

When the A register contents are 80H or lower, the CALL instruction is skipped and the next instruction is executed.
When the A register contents are higher than 80H, the next CALL instruction is executed and execution is branched to the target address.

6.15 CPU Control Instructions

The following instructions are CPU control instructions.

SEL RBn

Select Register Bank
Register Bank Selection

[Instruction format] SEL RBn

[Operation] RBS0, RBS1 ← n; (n = 0 to 3)

[Operand]

Mnemonic	Operand (RBn)
SEL	RBn

[Flag]

Z	AC	CY

[Description]

- The register bank specified by the operand (RBn) is made a register bank for use by the next and subsequent instructions.
- RBn ranges from RB0 to RB3.

[Description example]

SEL RB2; Register bank 2 is selected as the register bank for use by the next and subsequent instructions.

NOP

No Operation
No Operation

[Instruction format] NOP

[Operation] no operation

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- Only the time is consumed without processing.

EI

Enable Interrupt
Interrupt Enabled

[Instruction format] EI

[Operation] IE ← 1

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- The maskable interrupt acknowledgeable status is set (by setting the interrupt enable flag (IE) to (1)).
- No interrupts are acknowledged between this instruction and the next instruction.
- If this instruction is executed, vectored interrupt acknowledgment from another source can be disabled. For details, refer to the description of interrupt functions in the user's manual for each product.

DI

Disable Interrupt
Interrupt Disabled

[Instruction format] DI

[Operation] IE ← 0

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- Maskable interrupt acknowledgment by vectored interrupt is disabled (with the interrupt enable flag (IE) cleared (0)).
- No interrupts are acknowledged between this instruction and the next instruction.
- For details of interrupt servicing, refer to the description of interrupt functions in the user's manual for each product.

HALT

Halt
HALT Mode Set

[Instruction format] HALT

[Operation] Set HALT Mode

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- This instruction is used to set the HALT mode to stop the CPU operation clock. The total power consumption of the system can be decreased with intermittent operation by combining this mode with the normal operation mode.

STOP

Stop
Stop Mode Set

[Instruction format] STOP

[Operation] Set STOP Mode

[Operand]

None

[Flag]

Z	AC	CY

[Description]

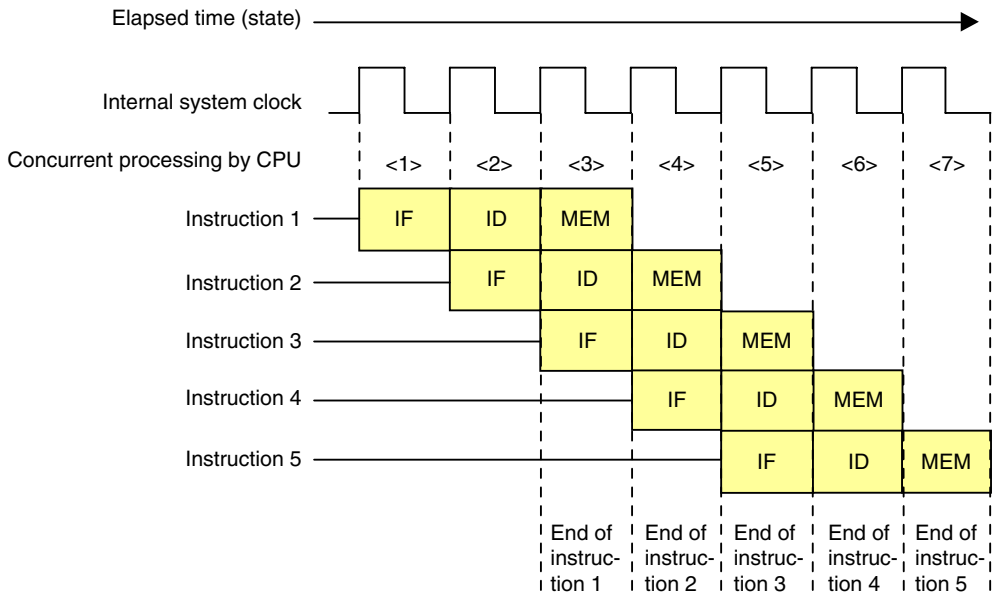
- This instruction is used to set the STOP mode to stop the main system clock oscillator and to stop the whole system. Power consumption can be minimized to only leakage current.

CHAPTER 7 PIPELINE

7.1 Features

The RL78 microcontroller uses three-stage pipeline control to enable single-cycle execution of almost all instructions. Instructions are executed in three stages: instruction fetch (IF), instruction decode (ID), and memory access (MEM).

Figure 7-1. Pipeline Execution of Five Typical Instructions (Example)



- IF (instruction fetch): Instruction is fetched and fetch pointer is incremented.
- ID (instruction decode): Instruction is decoded and address is calculated.
- MEM (memory access): Decoded instruction is executed and memory at target address is accessed.

7.2 Number of Operation Clocks

Although a problem in which the count clocks cannot be counted occurs in some other pipeline microcontrollers, the RL78 microcontroller solves this problem by maintaining operation at the same number of clocks, and thus stable programs can be provided.

These numbers of clocks are listed in **5.5 Operation List**.

7.2.1 Access to flash memory contents as data

When the content of the flash memory is accessed as data, the pipeline operation is stopped at the MEM stage. Therefore, the number of operation clocks is increased from the listed number of clocks. For details, refer to **5.5 Operation List**.

7.2.2 Access to external memory contents as data

When the content of the external memory is accessed as data, the CPU is set to wait mode. Therefore, the number of operation clocks is increased from the listed number of clocks.

For the number of increased clocks, refer to **Table 7-1** below.

Table 7-1. CPU Wait During Read/Write from/to External Memory

Clock for Selecting External Extension Clock Output (CLKOUT)	Wait Cycles
f_{CLK}	3 clocks
$f_{CLK}/2$	5 or 6 clocks
$f_{CLK}/3$	7 to 9 clocks
$f_{CLK}/4$	9 to 12 clocks

Remark 1 clock: $1/f_{CLK}$ (f_{CLK}: CPU clock)

7.2.3 Instruction fetch from RAM

When data is fetched from RAM, the instruction queue becomes empty because reading from RAM is late. So the CPU waits until the data is set to the instruction queue. During fetch from RAM, the CPU also waits if there is RAM access.

The number of clocks when instructions are fetched from the internal RAM area is twice the number of clocks plus 3, maximum (except when branching to the external memory area) when fetching an instruction from the internal ROM (flash memory) area.

7.2.4 Instruction fetch from external memory

When data is fetched from the external memory, the instruction queue becomes empty because reading from the external memory is late. So the CPU waits until the data is set to the instruction queue. During fetch from the external memory, the CPU also waits if there is external memory access.

The minimum and maximum numbers of execution clocks of each instruction when fetching instructions from the external memory are as follows, for the number of clocks when instructions are fetched from the flash memory area.

No. of Instruction Execution Clocks When Fetching Instructions from flash memory Area[Note]	When Fetching Instructions from External Memory	
	Minimum No. of Execution Clocks	Maximum No. of Execution Clocks
1	2 + 2 × Wait	5 + 3 × Wait
2	6 + 2 × Wait	7 + 6 × Wait
3	4 + 2 × Wait	8 + 8 × Wait
4	8 + 2 × Wait	10 + 10 × Wait
5	6 + 2 × Wait	12 + 9 × Wait
6	10 + 5 × Wait	14 + 11 × Wait

Note Number of clocks when the internal RAM area, SFR area, or expanded SFR area has been accessed, or when an instruction that does not access data is executed

Furthermore, the number of waits is as follows, depending on the clock selected for the CLKOUT pin.

Table 7-2. CPU Wait When Fetching Data from External Memory

Clock for Selecting External Extension Clock Output (CLKOUT)	Wait Cycles
f_{CLK}	3 clocks
$f_{CLK}/2$	5 or 6 clocks
$f_{CLK}/3$	7 to 9 clocks
$f_{CLK}/4$	9 to 12 clocks

Caution The flash memory and external memory are located in consecutive spaces, but start fetching in the external memory space by using a branch instruction (CALL or BR excluding relative addressing) in the flash memory or RAM memory.

Remark 1 clock: $1/f_{CLK}$ (f_{CLK}: CPU clock)

7.2.5 Hazards related to combined instructions

If the data of the register contents is indirectly accessed immediately after the writing to the register that is to be used for the indirect access, a one-clock wait is inserted.

Register Name	Previous Instruction	Next Instruction Operand (or Instruction)
DE	Write instruction to D register[Note] Write instruction to E register[Note] Write instruction to DE register[Note] SEL RBn	[DE], [DE+byte]
HL	Write instruction to H register[Note] Write instruction to L register[Note] Write instruction to HL register[Note] SEL RBn	[HL], [HL+byte], [HL+B], [HL+C], [HL].bit
B	Write instruction to B register[Note] SEL RBn	Word[B], [HL+B]
C	Write instruction to C register[Note] SEL RBn	Word[C], [HL+C]
BC	Write instruction to B register[Note] Write instruction to C register[Note] Write instruction to BC register[Note] SEL RBn	Word[BC], [HL+B], [HL+C]

SP	MOVW SP, #word
	MOVW SP, AX
	ADDW SP, #byte
	SUBW SP, #byte
		[SP+byte]
		CALL instruction, CALLT instruction, BRK instruction, SOFT instruction, RET instruction, RETI instruction, RETB instruction, interrupt, PUSH instruction, POP instruction

CS	MOV CS, #byte MOV CS, A	CALL rp BR AX
AX	Write instruction to A register[Note] Write instruction to X register[Note] Write instruction to AX register[Note] SEL RBn	BR AX
AX BC DE HL	Write instruction to A register[Note] Write instruction to X register[Note] Write instruction to B register[Note] Write instruction to C register[Note] Write instruction to D register[Note] Write instruction to E register[Note] Write instruction to H register[Note] Write instruction to L register[Note] Write instruction to AX register[Note] Write instruction to BC register[Note] Write instruction to DE register[Note] Write instruction to HL register[Note] SEL RBn	CALL rp

Note Register write instructions also require wait insertions when overwriting the target register values during direct addressing, short direct addressing, register indirect addressing, based addressing, or based indexed addressing.

APPENDIX A INSTRUCTION INDEX (MNEMONIC: BY FUNCTION)

[Stack manipulation instructions]

[Unconditional branch instruction]

[Conditional branch instructions]

[Conditional skip instructions]

[CPU control instructions]

APPENDIX B INSTRUCTION INDEX (MNEMONIC: IN ALPHABETICAL ORDER)

RL78 family User's Manual: Software

Publication Date: Rev.1.00 Jan 31, 2011

Published by: Renesas Electronics Corporation

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.0

RL78 family

Renesas Electronics Corporation

R01US0015EJ0100