OKI

INSTRUCTION MANUAL

nX-4/250/300 Core

CMOS 4-BIT MICROCONTROLLER

FIRST EDITION
ISSUE DATE: Jun., 1997

Table of Contents

Introduction

Chapter 1 - Architecture

1. OVERVIEW 1-1
2. CPU RESOURCES AND PROGRAMMING MODEL 1-1
2.1 Registers 1-3
2.1.1 Accumulator (A) 1-4
2.1.2 Flag register 1-4
2.1.2.1 Carry flag (C) 1-4
2.1.2.2 Zero flag (Z) 1-4
2.1.2.3 G flag (G) 1-4
2.1.3 Master interrupt enable flag (MIE) 1-5
2.1.4 H register, L register, X register, Y register 1-5
2.1.5 Current bank register (CBR), extra bank register (EBR) 1-6
2.1.6 RA register (RA3, RA2, RA1, RA0) 1-7
2.1.7 Program counter (PC) 1-7
2.1.8 Stack pointer (SP, SPH/SPL) 1-8
2.1.9 Register stack pointer (RSP) 1-9
2.2 Memory spaces 1-10
2.2.1 Program memory space 1-10
2.2.2 Data memory space 1-13
2.2.2.1 SFR space 1-14
2.2.3 External memory space 1-15
2.3 Addressing modes 1-16
2.3.1 Register addressing modes 1-16
2.3.1.1 Register direct addressing mode 1-16
2.3.1.2 Bit direct addressing mode 1-17
2.3.1.3 Immediate addressing mode 1-17
2.3.2 Data memory addressing modes 1-18
2.3.2.1 Direct addressing mode 1-19
2.3.2.2 SFR bank internal direct addressing mode 1-19
2.3.2.3 Current bank internal direct addressing mode 1-20
2.3.2.4 HL register indirect addressing mode 1-20
2.3.2.5 XY register indirect addressing mode 1-21
2.3.2.6 Extra bank HL indirect addressing mode 1-21
2.3.2.7 Extra band XY indirect addressing mode 1-22
2.3.2.8 HL register indirect addressing mode with post increment 1-22
2.3.2.9 XY register indirect addressing mode with post-increment 1-23
2.3.2.10 Extra bank HL register indirect addressing mode with post-increment 1-23
2.3.2.11 Extra bank XY register indirect addressing mode with post-increment 1-24

2.3.2.12 Bit direct addressing mode 1-24
2.3.2.13 Bit indirect addressing mode 1-25
2.3.3 Addressing modes for program memory 1-26
2.3.3.1 64K word direct addressing mode 1-26
2.3.3.2 4K word page addressing mode 1-27
2.3.3.3 RA register indirect addressing mode 1-27
2.3.3.4 PC relative addressing mode 1-28
2.3.3.5 PC based addressing mode 1-29
2.3.4 Addressing mode for external memory 1-29
2.3.4.1 RA register indirect addressing mode 1-29
2.3.4.2 Direct addressing mode 1-30

Chapter 2 - Instruction set

1. OVERVIEW 2-1
2. OPERAND EXPRESSION 2-3
3. LIST OF INSTRUCTIONS 2-5
4. INSTRUCTION DESCRIPTIONS 2-33

Introduction

This manual describes the instruction set of the nX-4/250 core and nX-4/300 core, which is designed for use as the CPU core for the original OKI CMOS 4-bit microcontroller.
This manual is designed on the basis of the nX-4/250 core and nX-4/300 core basic architecture. The basic architecture of these constitutes the most important functional specifications of the nX-4/250 core and nX-4/300 core. Depending on the model you are using the actual supported memory capacity may be a subset of the basic architecture. Please refer to the individual user's manual for such information.
The following manuals related to the product line-up built around the nX-4/250 core and nX-4/300 core are available.
Please refer to them for additional information.

- MSM63XXX User's Manual
  Hardware description

- ASM63KN Cross-assembler User's Manual
  Description of assembler operation and language specifications

- EASE63XXX User's Manual
  Emulator hardware description

- SID63K Operation Manual
  Debugger command description

This manual consists of two chapters.

Chapter 1 discusses the nX-4/250 core and nX-4/300 core basic architecture, beginning with explanations of the major resources used by the program, such as registers and memory, and then discussing addressing modes.

Chapter 2 explains the functions of each instruction, covering instruction function, their detailed action, and the instruction codes. Instruction descriptions are arranged in alphabetical order to serve as a reference.

Chapter 1

ARCHITECTURE

This chapter describes the basic architecture of the nX-4/250 core and nX-4/300 core. The basic architecture is the most significant functional specification of the nX-4/250 core and nX-4/300. All microcontrollers using this core will have the same function as the basic architecture, or a subset thereof. This chapter covers the objectives of this document and its composition.

1. OVERVIEW

The nX-4/250 core and nX-4/300 core instruction set consist of 440/450 instructions.
The memory spaces are divided into a 16-bit width program memory space, a 4-bit width data memory space, and external 8-bit width memory.
The program counter save stack (call stack) for subroutine call or interruption and register save stack (register stack) are prepared separately from memory space.
The nX-4/250 core is a downward version of the nX-4/300 core.
The differences between them are shown below.

Table 1-1 Differences between nX-4/250 Core and nX-4/300 Core

Core	MMOV instruction	BMOV instruction	FCLR FLAG instruction	FSET FLAG instruction
nX-4/250	Not provided	Not provided	Not provided	Not provided
nX-4/300	provided	provided	provided	provided

2. CPU RESOURCES AND PROGRAMMING MODEL

This section describes the configuration of the CPU resources used in programming, such as registers and memory, along with their roles.
Figures 1-1 to 1-2 show the relationship between program memory space, external memory, data memory and registers.
Figures 1-3 show the relationship between these memory spaces and stack registers.

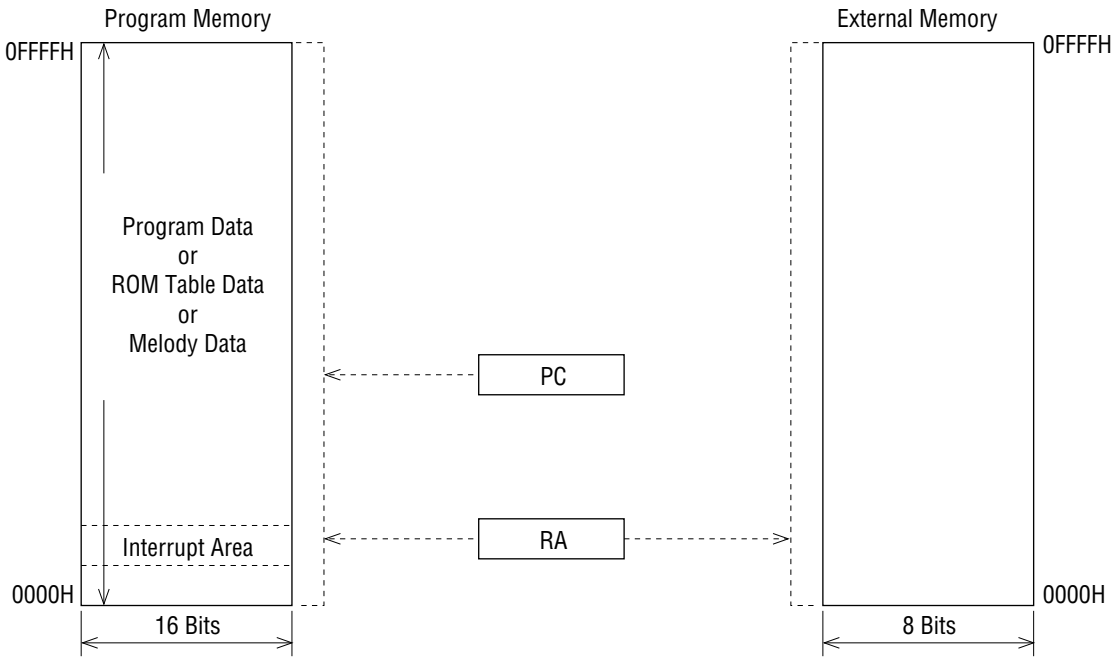


Fig. 1-1 Relationship between Program Memory, External Memory, and Registers

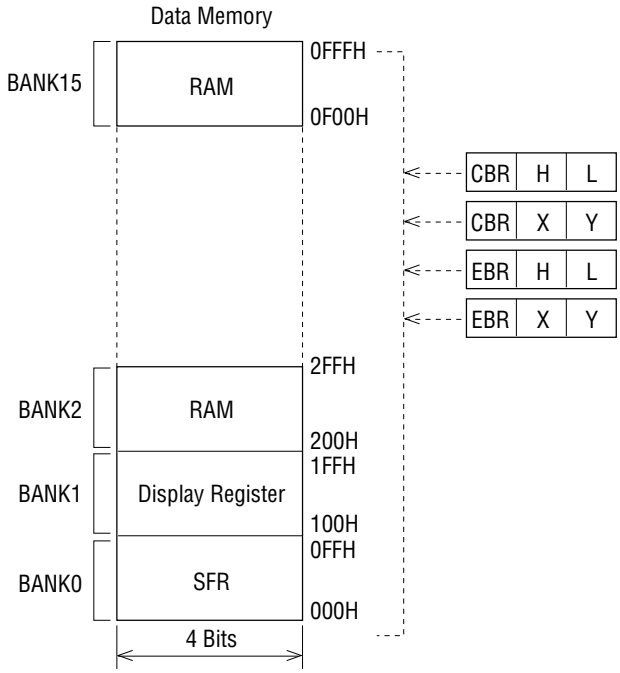


Fig. 1-2 Relationship between Data Memory and Registers

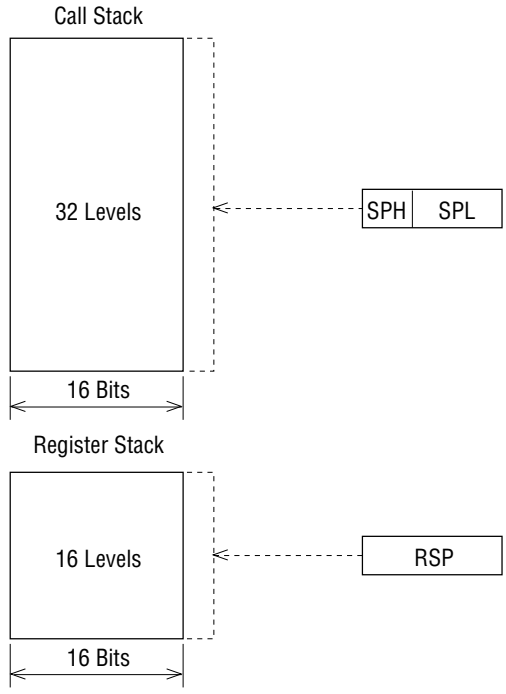


Fig. 1-3 Relationship between Stacks and Registers

2.1 Registers

The nX-4/250 core and nX-4/300 core adopt a processing method that uses primarily accumulators and registers. The register set uses the programming model, with data memory addresses stored in the HL register, XY register, current bank register (CBR), extra bank register (EBR), and external memory and program memory addresses in the RA register. In addition registers are provided to control program flow, flags and memory.
Figures 1-4 show register configurations.

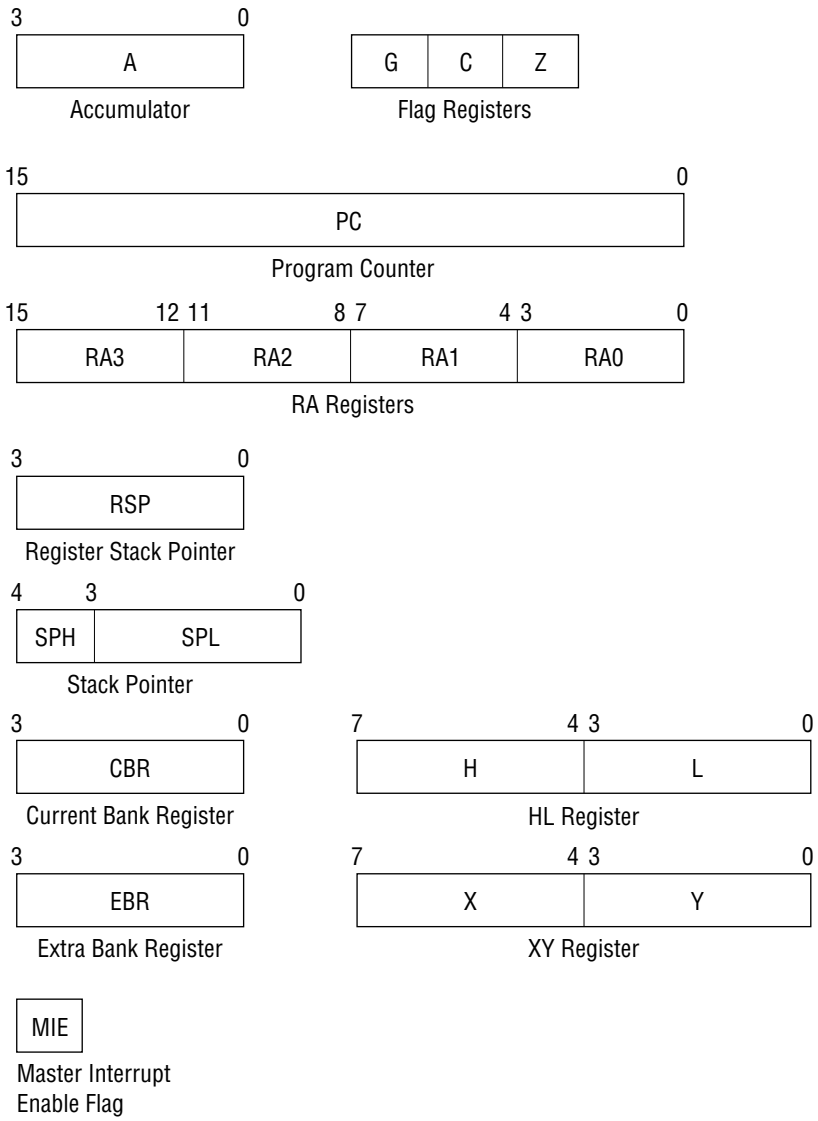


Fig. 1-4 Register Configuration

2.1.1 Accumulator (A)

Accumulator (A) is a critical register for arithmetic operations.
The accumulator is initialized to zero at reset. In the event that accumulator contents must be saved when an interrupt is generated or at other times, the PUSH HL instruction is used to save the value to the register stack. The register is restored with the POP HL instruction.

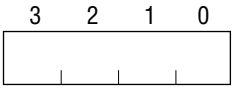


Fig. 1-5 Accumulator (A)

2.1.2 Flag registers

The flag register consists of three flags: the carry flag (C), the zero flag (Z) and the G flag (G). In the event that flag register contents must be saved when an interrupt is generated or at other times, the PUSH HL instruction is used to save the value to the register stack. The register is restored with the POP HL instruction.

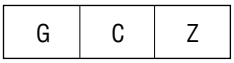


Fig. 1-6 Flag Registers

2.1.2.1 Carry flag (C)

The carry flag (C) is a 1-bit flag, and is used to load the carry for an additional instruction or the borrow for a subtraction instruction.
The carry flag is initialized to zero at reset.

2.1.2.2 Zero flag (Z)

The zero flag (Z) is a 1-bit flag, and is set to "1" when the content of the accumulator (A) is set to "0H".
It is cleared to "0" when the content of the accumulator (A) is set to any value other than "0H".
The zero flag is initialized to zero at reset.

2.1.2.3 G flag (G)

The G flag is a 1-bit flag. This flag is set to "1" when the HL register, XY register, or RA register overflows as a result of execution of an increment instruction and is cleared to "0" when an overflow does not occur.
The HL register or XY register is incremented when an indirect addressing instruction with post increment is executed or when an increment instruction is executed for the HL register or XY register. The RA register is incremented when an increment instruction is executed.

2.1.3 Master interrupt enable flag (MIE)

The master interrupt enable flag (MIE) is the flag used to control enable/disable for maskable interrupts. When set to "1" maskable interrupts are enabled, and when cleared to "0" maskable interrupts are disabled.
If a maskable interrupt is received, the MIE flag is cleared to "0", and then restored to "1" through the execution of the maskable interrupt return instruction (RTI instruction). Instruction execution in the maskable interrupt processing routine can set to the MIE flag to "1" to make possible multi-level interrupt processing.
MIE flag set and clear are implemented with the "EI" and "DI" instructions, which are only used for MIE flag operations.
The MIE flag is initialized to zero at reset. The MIE flag is allocated to address 0FFH of the special function register (SFR).

2.1.4 H register, L register, X register, Y register

The H register, L register, X register and Y register are used as working registers during program processing. The H and L registers are used as a register pair in the data memory indirect addressing mode, as are the X and Y registers.
All four registers are initialized to zero at reset. In the event that register contents must be saved when an interrupt is generated or at other times, the PUSH HL or PUSH XY instruction is used to save the value to the register stack. The register is restored with the POP HL or POP XY instruction.
The H, L, X and Y registers are allocated to addresses 0F9H through 0FCH of the special function register (SFR).

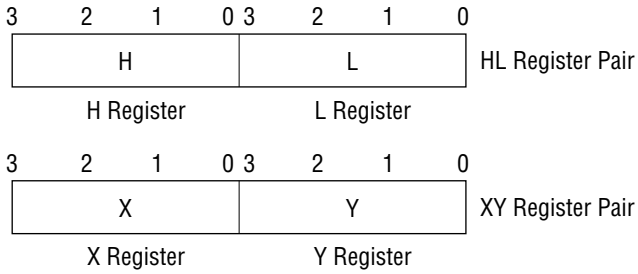


Fig. 1-7 H, L, X, and Y Registers

2.1.5 Current bank register (CBR), extra bank register (EBR)

The current bank register (CBR) and extra bank register (EBR) are used for data memory space bank specification.
CBR and EBR are initialized to zero at reset. In the event that register contents must be saved when an interrupt is generated or at other times, the PUSH XY instruction is used to save the value to the register stack. The register is restored with the POP XY instruction. The CBR and EBR are allocated to addresses 0FDH through 0FEH of the special function register (SFR).

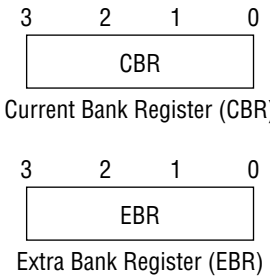


Fig. 1-8 Current Bank Register and Extra Bank Register

CBR and EBR are used in combination with the HL and XY registers for indirect addressing of data memory. CBR is used in combination with the 8-bit data in the instruction code for direct addressing within the current bank.
Fig. 1-9 indicates the register combinations used.

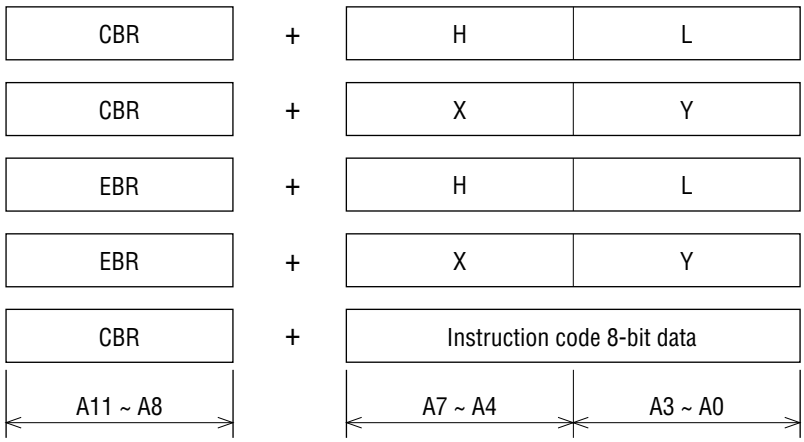


Fig. 1-9 Register Combinations

A11 to A0 indicate data memory (max. 4K nibbles) address.

2.1.6 RA register (RA3, RA2, RA1, RA0)

THe RA registers are used for program memory indirect addressing (ROM table reference instruction) and external memory indirect addressing (external memory transfer instruction).
Fig. 1-10 indicates the address configuration for the RA registers.

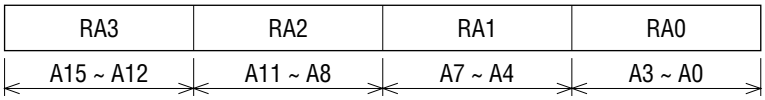


Fig. 1-10 Address Configuration for Registers RA3 through RA0

When used for ROM table reference instructions, A15 through A0 indicate a maximum of 64K bytes of program memory addresses.
When used for external memory transfer instructions, A15 through A0 indicate a maximum of 64K bytes of external memory addresses.
RA3 through RA0 are allocated to addresses 0F2H through 0F5H of the special function register (SFR).
The RA registers are initialized to zero at reset.

2.1.7 Program counter (PC)

The program counter (PC) is the counter used to store the address of the program code to be executed next. The PC bit length is 16 bits , which means it can specify an address within 64K byte program memory space.
The PC is increased immediately after the program code is fetched from program memory, and this repetition creates program flow. For a branch instruction the new program code address is set to the PC.
The PC is initialized to zero at reset. When an interrupt is generated the execution restart address is automatically saved to the call stack. This value can be restored to the PC through the RTI instruction.

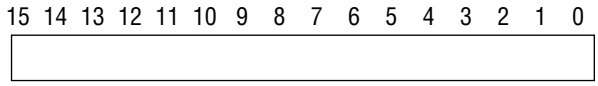


Fig. 1-11 Program Counter

2.1.8 Stack pointer (SP or SPH/SPL)

The stack pointer (SP or SPH/SPL) is a pointer indicating the head address of the call stack, which is used for saving program counters at subroutine calls and interrupt.
The stack pointer is a 5-bit up/down counter , counting up at stack save and down at stack restore. The call stack is 16 bits wide, and uses one level for PC save. It has a maximum of 32 levels.
The stack ponter is initialized to zero at reset, and points to address “00H” in the call stack. The SPH/SPL are allocated to address “0F8H” and “0F7H”, respectively, of the special function register (SFR). The stack pointer is a read-only register, and write is disabled.
Figs. 1-12 indicate the relation between the stack pointer (SP) and the call stack.

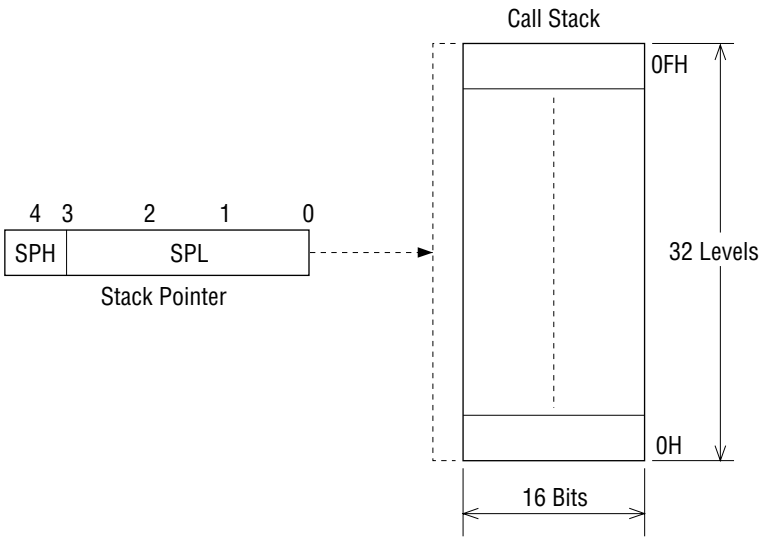


Fig. 1-12 Relation between SPH/SPL and Call Stack

2.1.9 Register stack pointer (RSP)

The register stack pointer (RSP) is a pointer indicating the address of the register stack, used for saving various registers.
The RSP is a 4-bit up/down counter, counting up at stack save (PUSH instruction) and down at a stack restore (POP instruction). The register stack is 16 bits wide, and uses one level for register save. It has a maximum of 16 levels.
The RSP is initialized to zero at reset, and points to address “0H” in the call stack. The RSP is allocated to address “0F6H” of the special function register (SFR).

Fig. 1-13 indicates the relation between the RSP and the register stack.

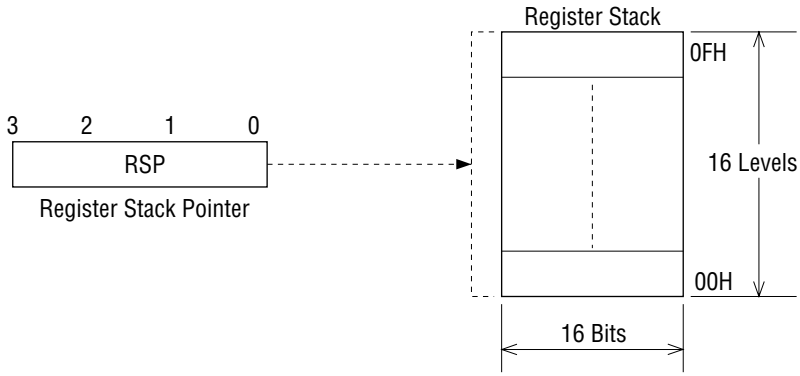


Fig. 1-13 Relation between RSP and Register Stack

The PUSH/POP instructions can be used to save various registers to the register stack, and restore them, as shown in Fig. 1-14.

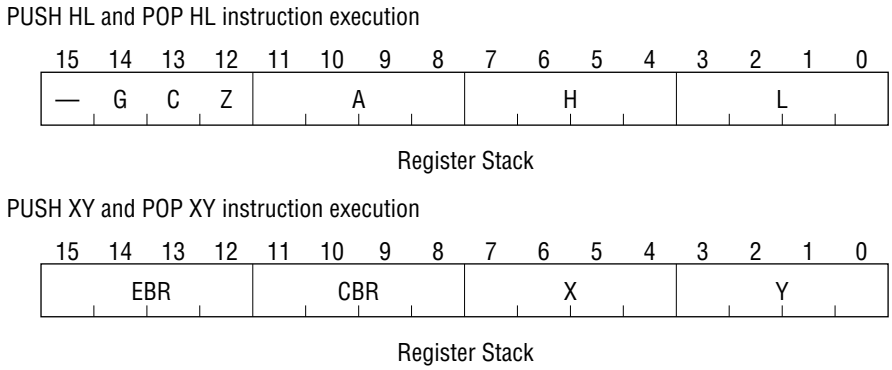


Fig. 1-14 Save/restore registers at PUSH/POP Instruction Execution

2.2 Memory spaces

The nX-4/250 core and nX-4/300 core memory spaces each consist of program memory space, data memory space and external memory space. This section discusses the structures of these memory spaces. Note that the program counter save stack (call stack) used at subroutine calls or interrupts, the address save stack (melody stack) for melody output and the register save stack (register stack) are separate from the memory space.

2.2.1 Program memory space

Program memory has a 16-bit data length with a 64 word capacity. The program memory space stores ROM data and melody data in addition to program data.
Figs. 1-15 indicate the program memory configurations.

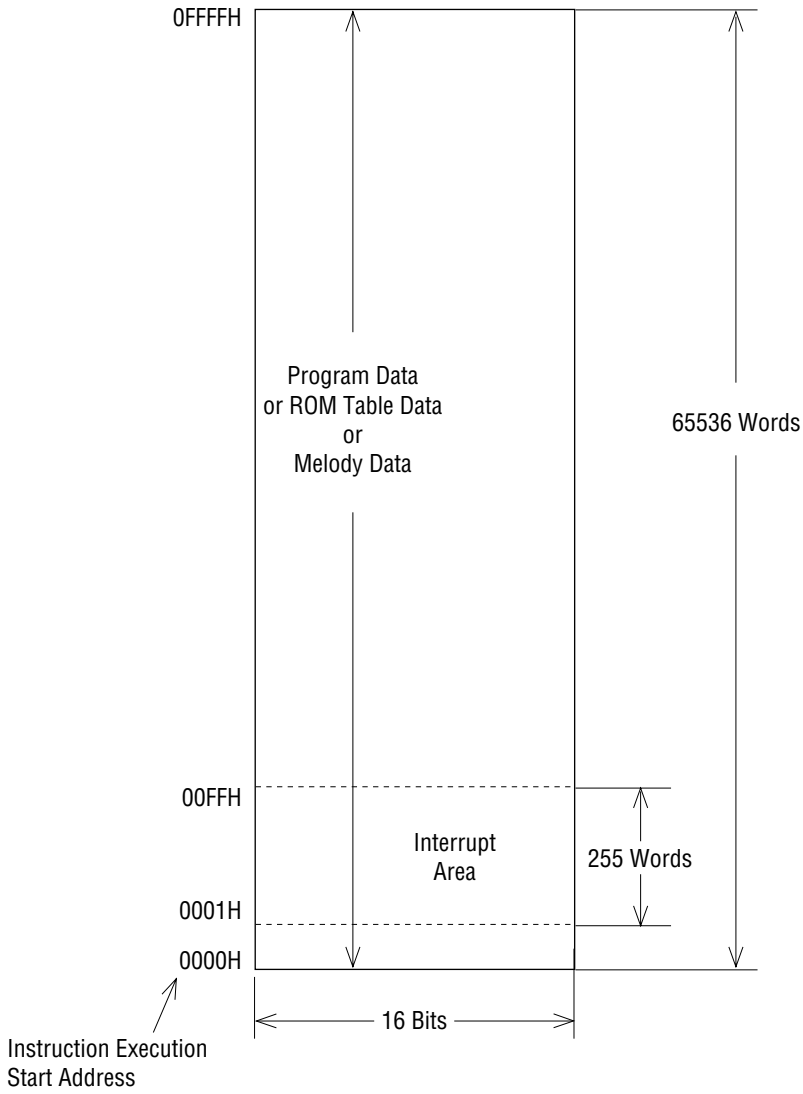


Fig. 1-15 Program Memory Configuration (nX-4/300)

Address 0000H is the instruction execution start address when the system is reset.
Allocated in 0001H through 00FFH are interrupt process routine start addresses when an interrupt occurs. See the user's manual for your device because the allocation is different depending on the type of device.
ROM table data is transfered to data memory by a ROM table reference instruction.
Melody data defines musical scale, tone length, and end tone used in a melody circuit.
Melody data is automatically transfered to a melody circuit after its start address is indicated by a MSA instruction. Ths MSA instruction cannot be used for a device in which a melody circuit is not included.

In the program memory space, 1 page consists of 4K words.
The nX-4/250 core and nX-300 core has 16 pages.
The LJMP and LCAL instructions can access the entire program memory space, but the JMP and CAL instructions can access only a page internal.
The RA register indirect addressing instruction, PC relative addressing instruction, and PC based addressing instruction can access each page irrespective of the boundaries of pages

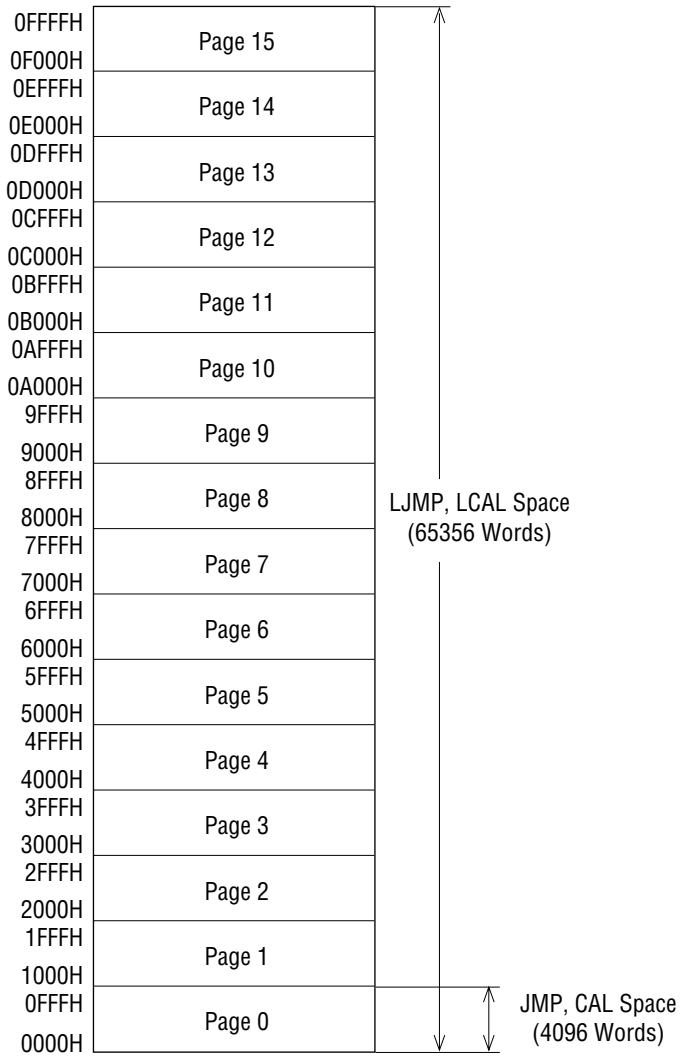


Fig. 1-16 Pages in Program Memory Space (nX-4/300)

2.2.2 Data memory space

Data memory space holds the data RAM and special function register (SFR).
As indicated in Fig. 1-17 below, the data memory consists of 16 banks, with each bank unit having 256 nibbles in size. BANK0 is assigned to SFR space, and bank1 to 15 (3480 nibbles) are assigned to data RAM.

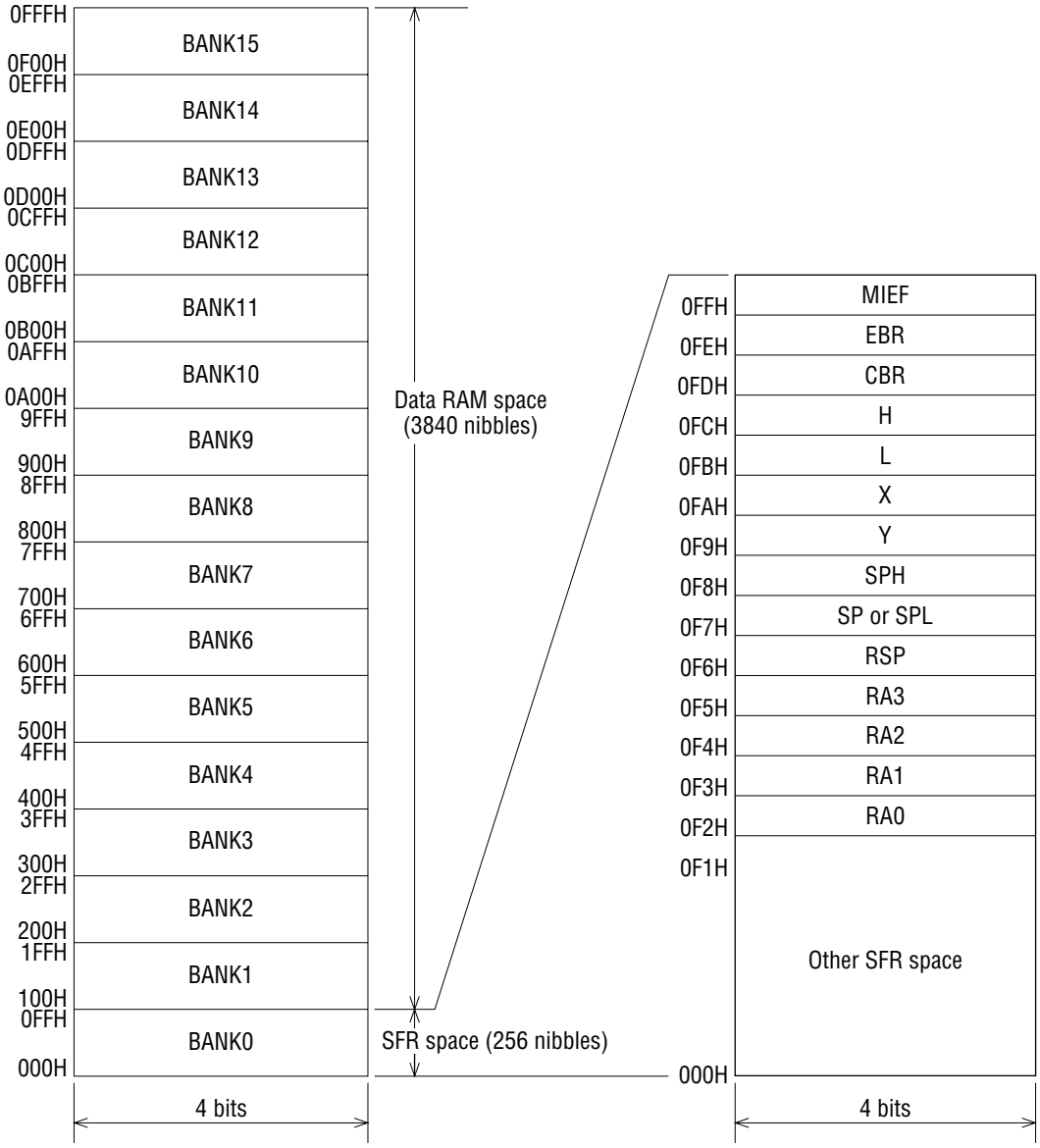


Fig. 1-17 Data Memory Space Configuration

2.2.2.1 SFR space

The 256-nibble SFR space from 000H to 0FFH contains the special function register (SFR) used to control peripheral functions connected to the CPU core. The 0F2H to 0FFH are allocated to CPU core registers as indicated in Table 1-2 below.

Table 1-2 Register SFR Allocation

Address	Register name	symbol	Content				R/W
			b3	b2	b1	b0	
0F2H	RA0 register	RA0	a3	a2	a1	a0	R/W
0F3H	RA1 register	RA1	a7	a6	a5	a4	R/W
0F4H	RA2 register	RA2	a11	a10	a9	a8	R/W
0F5H	RA3 register	RA3	a15	a14	a13	a12	R/W
0F6H	Register stack pointer	RSP	rsp3	rsp2	rsp1	rsp0	R/W
0F7H	Stack pointer L	SPL	SP3	SP2	SP1	SP0	R
0F8H	Stack pointer H	SPH	—*	—*	—*	SP4	R
0F9H	Y register	Y	y3	y2	y1	y0	R/W
0FAH	X register	X	x3	x2	x1	x0	R/W
0FBH	L register	L	l3	l2	l1	l0	R/W
0FCH	H register	H	h3	h2	h1	h0	R/W
0FDH	Current bank register	CBR	c3	c2	c1	c0	R/W
0FEH	Extra bank register	EBR	e3	e2	e1	e0	R/W
0FFH	Master interrupt enable flag	MIEF	—*	—*	—*	MIE	R

(Note) * = reserved bit : Always reads "1". Write is disabled.
"R" indicates read-only and "R/W" indicates read and write are enabled.

2.2.3 External memory space

The external memory space has an 8-bit data length data space, allocated from address 00000H to address 0FFFFH.
The external memory space is configured as indicated in Fig. 1-18.

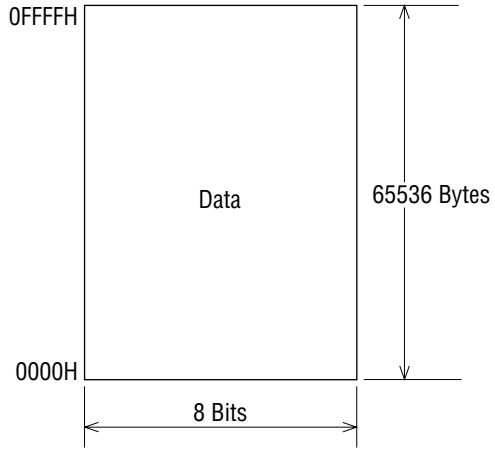


Fig. 1-18 External Memory Space Configuration

2.3 Addressing modes

Addressing modes are classified as indicated in Table 1-3.

Table 1-3 Addressing Mode Classification

Classification	Content
Register addressing modes	Register direct
	Bit direct
	Immediate
Data memory addressing modes	Direct
	SFR bank internal direct
	Current bank internal direct
	HL register indirect
	XY register indirect
	Extra bank HL register indirect
	Extra bank XY register indirect
	Post-incremented HL register indirect
	Post-incremented XY register indirect
	Post-incremented extra bank HL register indirect
	Post-incremented extra bank XY register indirect
	Bit direct
	Bit indirect
Program memory addressing modes	64 K word direct
	4K word page internal direct
	RA register indirect
	PC relative
	PC based
External memory addressing modes	RA register indirect
	Direct

2.3.1 Register addressing mode

2.3.1.1 Register direct addressing mode

Instructions can be used to directly set the accumulator (A), HL register, XY register, RA3 to RA0 registers, current flag register (CFR), extra bank register (EBR) and flag register (FLAG).

■ Operand description

- A, H, L, X, Y, CBR, EBR, FLAG
- HL, XY, RA

■ Description example

```
INCB HL              ; Increment HL register
```

2.3.1.2 Bit direct addressing mode

Instructions can be used to perform bit operations on the zero flag (Z), carry flag (C) and G flag of the flag register (FLAG).

■ Operand description

- Z, C, G

■ Description example

```
FCLR C              ; Clear carry flag
```

2.3.1.3 Immediate addressing mode

The numerical values in an instruction are directly treated as data.

■ Operand description

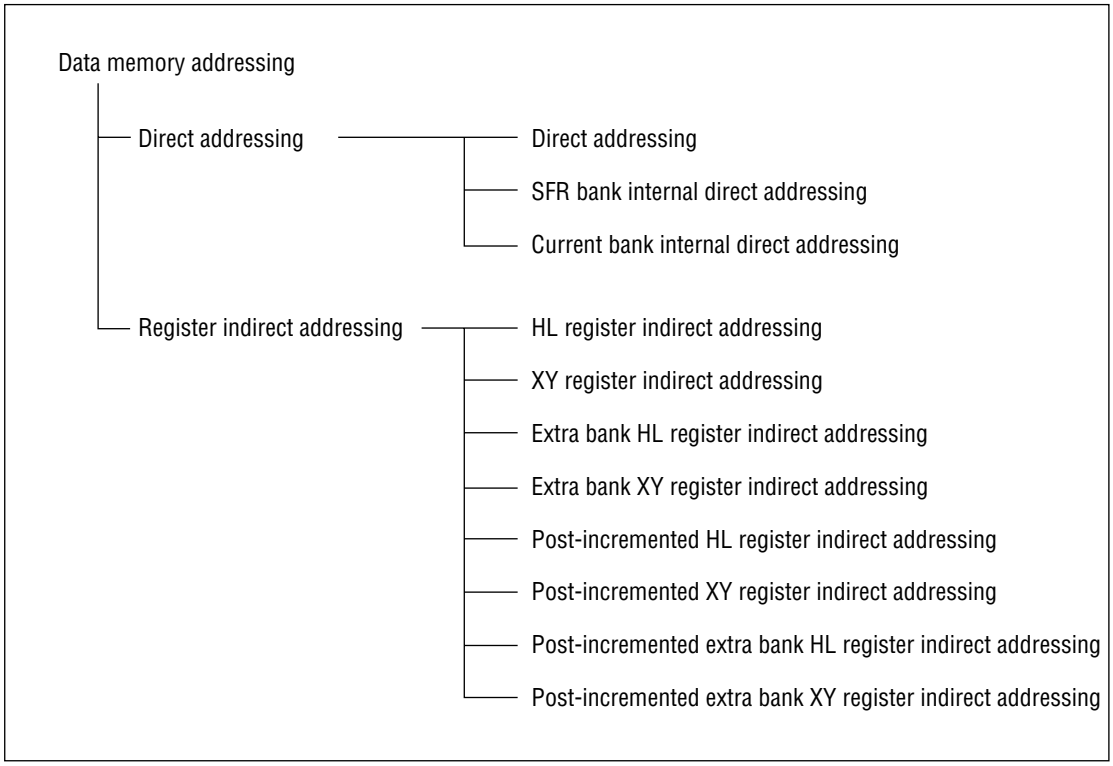
- #i

■ Description example

```
MOV CBR, #4H        ; Write immediate data 4H to CBR
```

2.3.2 Data memory addressing modes

The following data memory addressing modes are supported:



The post-incremented indirect addressing mode can add +1 or +2 to the HL or XY register after instruction execution.

- For +1

In the 4-bit unit operation mode, the HL register or XY register is incremented by one after instruction execution. If the HL or XY register overflows as a result (HL = 0 or XY = 0), the G flag is set to “1” .
If there is no overflow, the G flag is cleared to “0”.

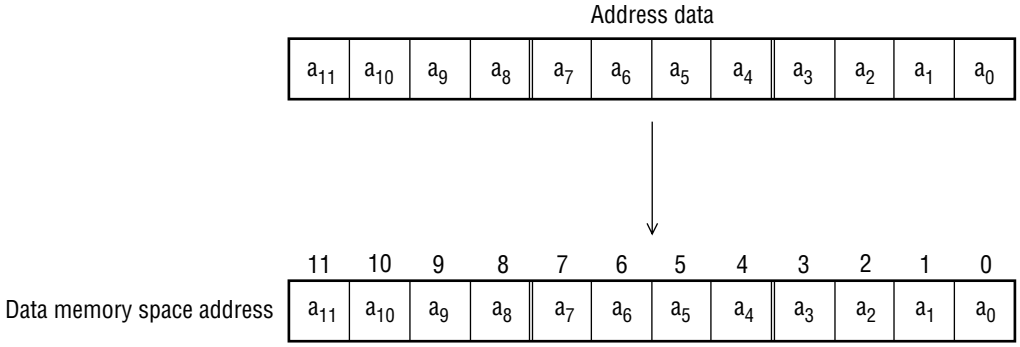
- For +2

In the 8-bit unit operation mode (ROM table reference instruction, external memory transfer instruction), the HL register or XY register is incremented by two after instruction execution. If the HL or XY register overflows as a result (HL > 0FFH or XY > 0FFH), the G flag is set to “1”. If there is no overflow, the G flag is cleared to “0”.

[Note] The post-incremented indirect addressing mode should not be used for XY, HL, CBR, and EBR registers located in 0F9H to 0FEH of the SFR space.

2.3.2.1 Direct addressing mode

The data memory space address is directly specified by the 12-bit address data in the instruction code.



■ Operand description

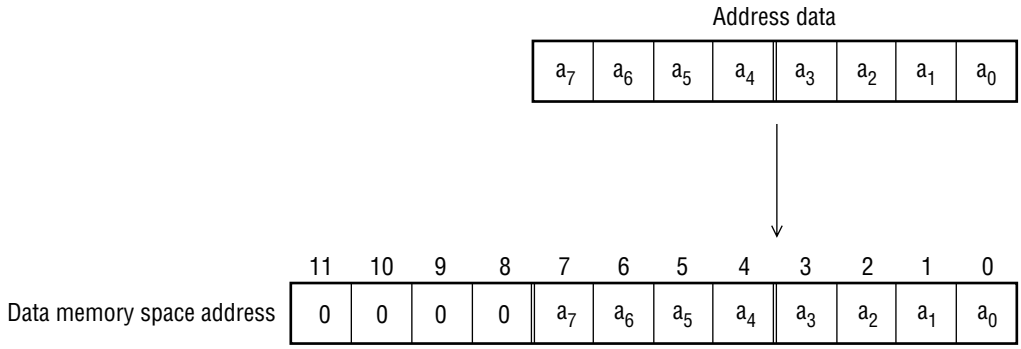
- direct

■ Description example

```
MOV 326H, A          ; Transfers accumulator content to address 26H of bank 3
```

2.3.2.2 SFR bank internal direct addressing mode

The data memory space SFR space (bank 0) is directly specified by the 8-bit address data in the instruction code.



■ Operand description

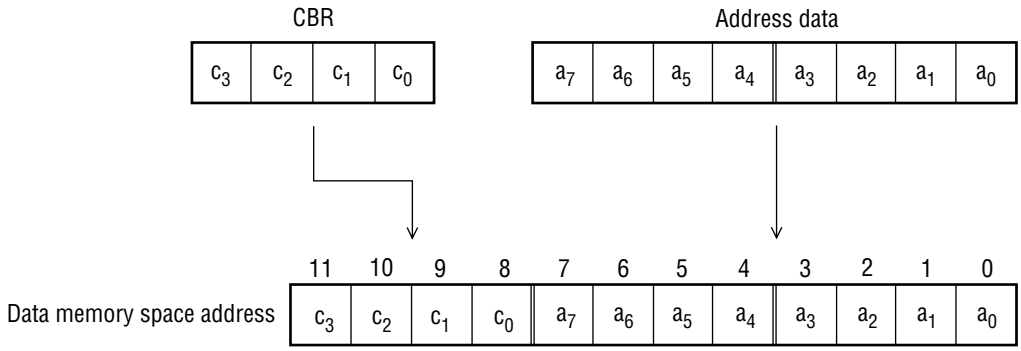
- sfr

■ Description example

```
MOV 36H, #0CH        ; Write immediate data 0CH to address 36H of the SFR (bank0)
```

2.3.2.3 Current bank internal direct addressing mode

The data memory space address is directly specified by the 8-bit address data in the current bank register (CBR).



■ Operand description

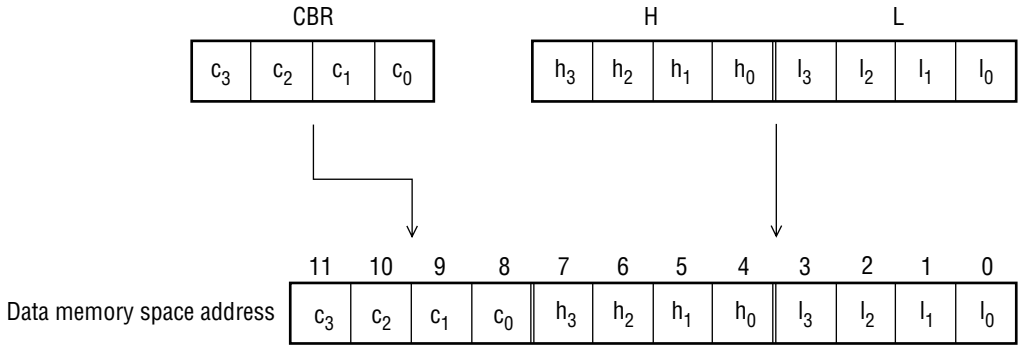
- ¥cur

■ Description example

INC ¥32H	; Increments the content of address 32H in the bank indicated by CBR, and stores the result to address 32H and accumulator

2.3.2.4 HL register indirect addressing mode

The data memory space address is indirectly specified by the current bank register (CBR) and the HL register.



■ Operand description

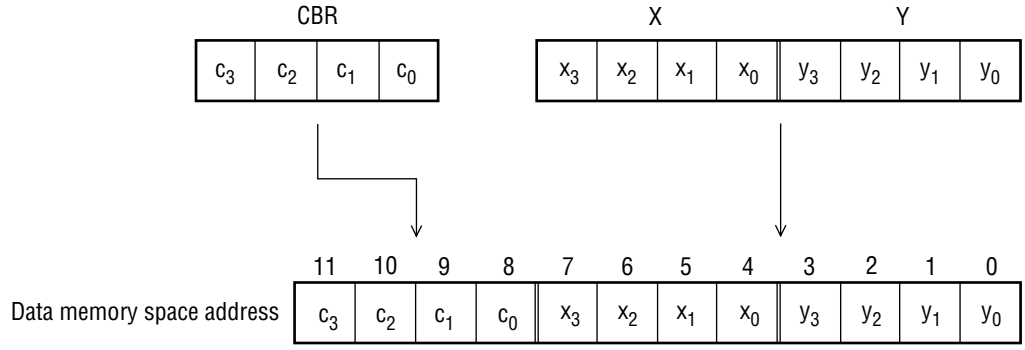
- [HL] or C:[HL]

■ Description example

MOV [HL], A	; The accumulator content is transferred to the data memory specified by the CBR and HL registers

2.3.2.5 XY register indirect addressing mode

The data memory space address is indirectly specified by the current bank register (CBR) and the XY register.



■ Operand description

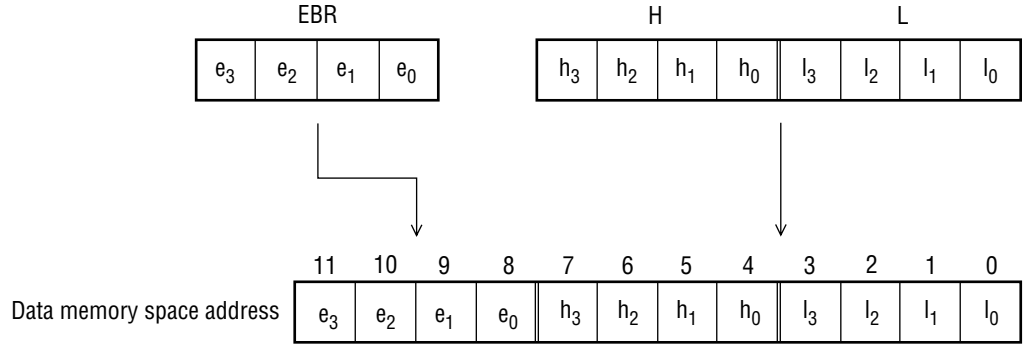
- [XY] or C:[XY]

■ Description example

DEC [XY]	; The data memory content specified by the CBR and XY registers is decremented, and the result stored to the data memory and accumulator.

2.3.2.6 Extra bank HL indirect addressing mode

The data memory space address is indirectly specified by the extra bank register (EBR) and the HL register.



■ Operand description

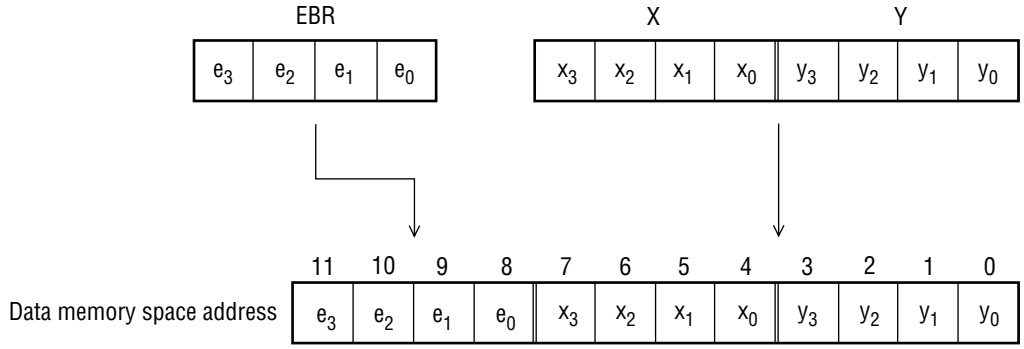
- E:[HL]

■ Description example

ROR E:[HL]	; The data memory content specified by the EBR and HL registers is rotated right, including the carry, and the result stored to the data memory and accumulator.

2.3.2.7 Extra bank XY indirect addressing mode

The data memory space address is indirectly specified by the extra bank register (EBR) and the XY register.



■ Operand description

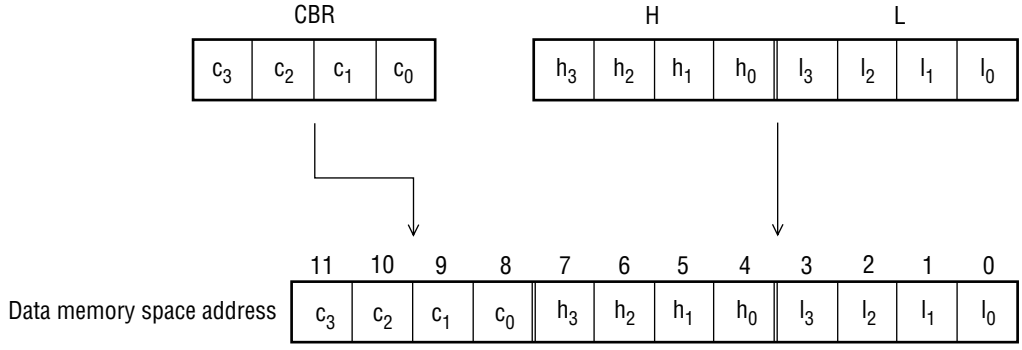
- E:[XY]

■ Description example

```
AND E:[XY], A      ; The logical product of the data memory content specified by the EBR and
                     XY registers and the accumulator is stored to the data memory and
                     accumulator.
```

2.3.2.8 HL register indirect addressing mode with post-increment

The data memory space address is indirectly specified by the current bank register (CBR) and the HL register. After execution the HL register is incremented by +1 or +2.



■ Operand description

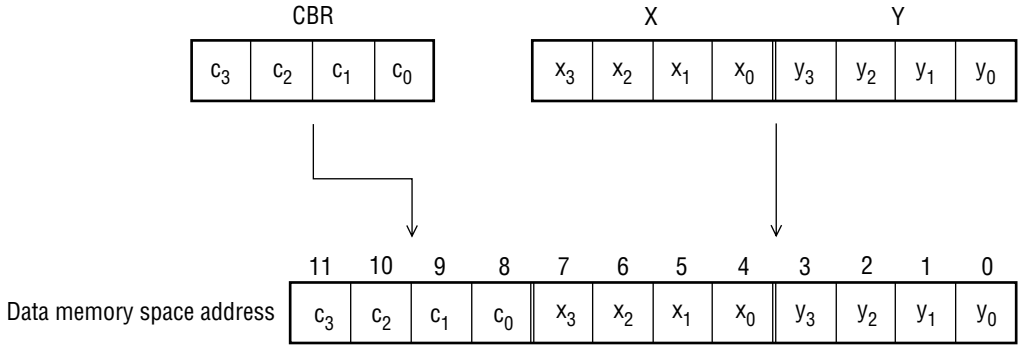
- [HL+] or C:[HL+]

■ Description example

```
MOV [HL+], A       ; The accumulator content is transferred to the data memory specified by
                     the CBR and HL registers and then the HL register is incremented by one.
```

2.3.2.9 XY register indirect addressing mode with post-increment

The data memory space address is indirectly specified by the current bank register (CBR) and the XY register. After execution the XY register is incremented by +1 or +2.



■ Operand description

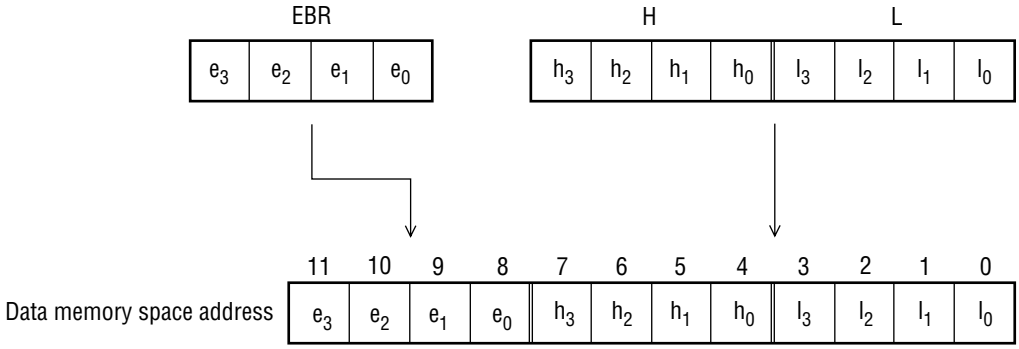
- [XY+] or C:[XY+]

■ Description example

```
XOR [XY+], A    ; The results of an exlusive-OR between accumulator and the data memory
                  content specified by the CBR and XY registers are stored to the accumu-
                  lator and data memory. Then the XY register is incremented by one.
```

2.3.2.10 Extra bank HL register indirect addressing mode with post-increment

The data memory space address is indirectly specified by the extra bank register (EBR) and the HL register. After execution the HL register is incremented by +1 or +2.



■ Operand description

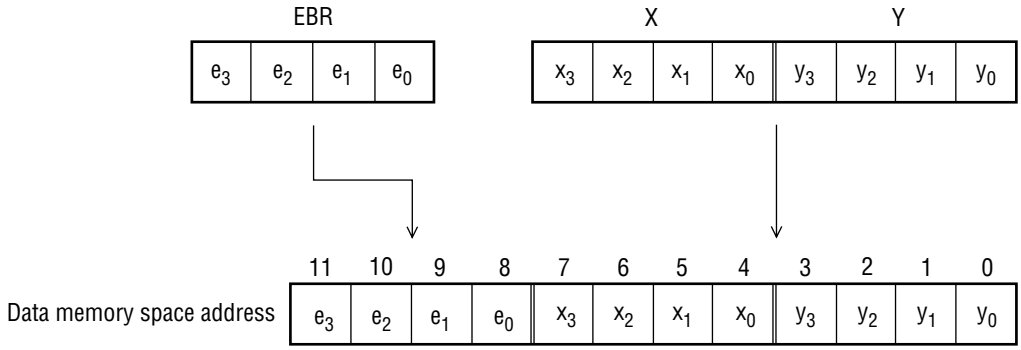
- E:[HL+]

■ Description example

```
MOV E:[HL+], #00H  ; Immediate data 00H is transferred to the accumulator and the data
                     memory specified by the EBR and HL registers, then the HL register is
                     incremented by one.
```

2.3.2.11 Extra bank XY register indirect addressing mode with post-increment

The data memory space address is indirectly specified by the extra bank register (EBR) and the XY register. After execution the XY register is incremented by +1 or +2.



■ Operand description
- E:[XY+]

■ Description example

```
MOV E:[XY+], A      ; The accumulator content is transferred to the data memory specified by
                      the EBR and XY registers, then the XY register is incremented by one.
```

2.3.2.12 Bit direct addressing mode

Executes data memory bit operations, which may be bit operations in 1-bit units (specified with ".n") or multi-bit operations (specified with "#m").

■ Operand description
- [HL].n, [XY].n, C:[HL].n, C:[XY].n,
  E:[HL].n, E:[XY].n, [HL+].n, [XY+].n,
  E:[HL+].n, E:[XY+].n, ¥cur.n
  (n = 0 ~ 3, 0:LSB, 3:MSB)

■ Description example

```
BCLR [XY].3         ; The third bit of the data memory specified by the CBR and XY register is
                      cleared, and the result stored to the data memory and accumulator.
```

■ Operand description
- #m

■ Description example

```
MCLR [XY].#3H       ; Bits 0 and 1 of the data memory specified by the CBR and XY register are
                      cleared, and the result stored to the data memory and accumulator.
```

2.3.2.13 Bit indirect addressing mode

Executes data memory bit operations. The operated bit is specified by the content of the accumulator (A).

■ Operand description

- A

■ Description example

```
MTST 35H, A      ; When A=3H, bits 0,1 of address 35H of the SFR (bank 0) are tested, and
                   if either or both is zero the Z flag is set.
```

2.3.3 Addressing modes for program memory

2.3.3.1 64K word direct addressing mode

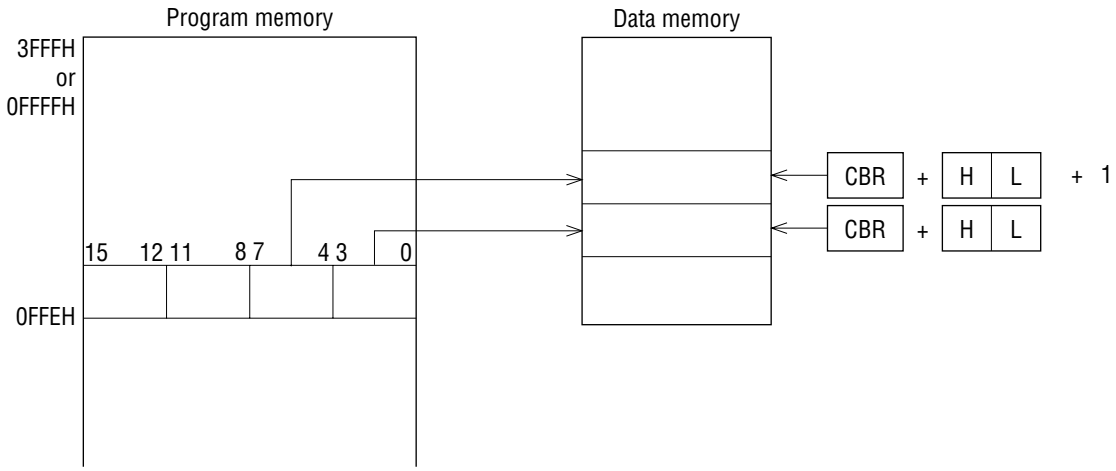
16-bit address data are used to address the entire program memory space.

■ Operand description

- cadr16

■ Description example

```
MOVLB [HL], 0FFEH ; The lower 8bits of program memory address 0FFEH are transferred to the
                    data memory specified by CBR and HL registers, and to the next address
                    in data memory.
```



■ Description example

```
LJMP 0234H          ; Branch to the program memory address specified by the 16-bit direct
                      address
```

2.3.3.2 4K word page addressing mode

12-bit address data is used to address the program memory page.

■ Operand description

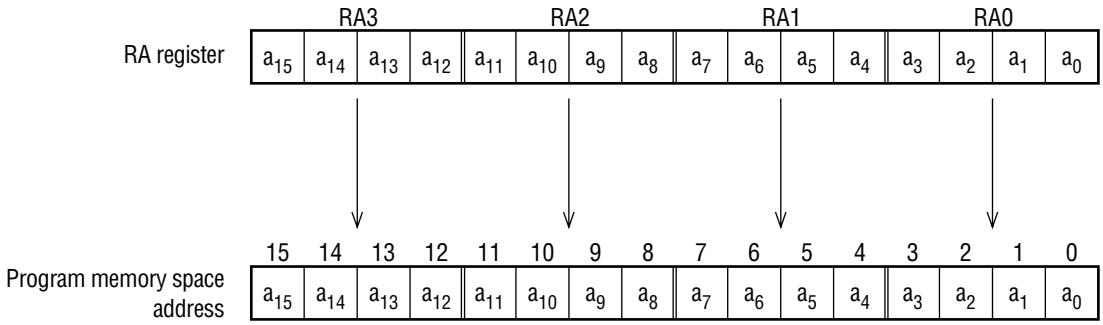
- cadr12

■ Description example

```
JMP 123H        ; Branch to the program memory address within that page, specified by the
                  12-bit address
```

2.3.3.3 RA register indirect addressing mode

The program memory space address is indirectly specified with the 16-bit address data in the RA register.

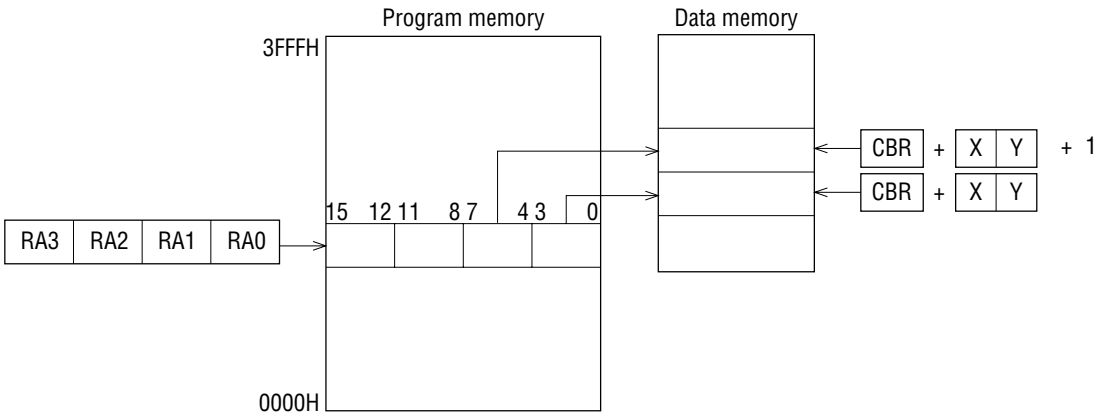


■ Operand description

- [RA]

■ Description example

```
MOVLB [XY], [RA]    ; The lower 8 bits of program memory address specified by RA are
                      transferred to the data memory specified by CBR and XY registers, and to
                      the next address in data memory
```



[Note] The RA registers from 0F2F to 0F5H of SFR space and XY, HL, CBR and EBR registers from 0F9H to 0FEH should not be addressed as a transfer destination to data memory. If they are addressed, the contents of the addressed registers are undefined.

2.3.3.4 PC relative addressing mode

The 8-bit address data in the instruction code is expanded to 16 bits, and the program branches to the address equal to that plus the address (PC value) for the next instruction. As a result, this type of addressing can specify a range of bytes from -128 to +127, centered on the next instruction.

■ Operand description

- radr8

■ Description example

```
SJMP 35H          ; Jumps to the address equal to the address for the next instuction plus 35H.
```

2.3.3.5 PC based addressing mode

The current address (PC value) is added to the content of the accumulator, and the program branches to that address plus one. Depending on the content of the accumulator, this gives 16 possible results.

■ Operand description

- PC + A

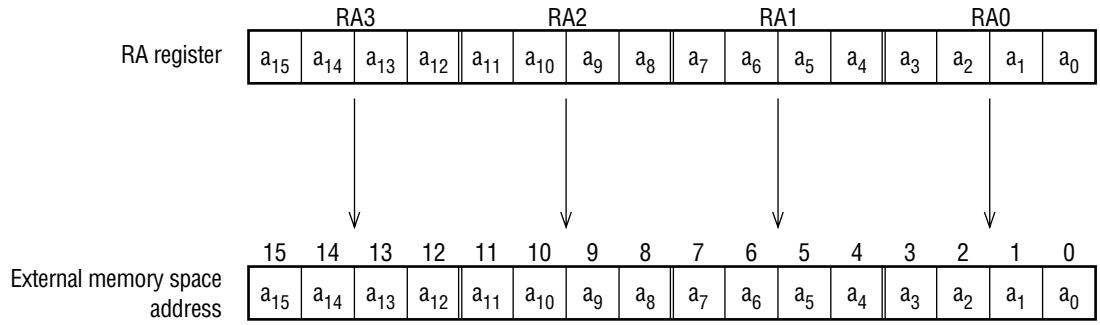
■ Description example

```
JMP PC+A        ; Jumps to the address equal to the address for the next instruction (PC +
                  1) plus the accumulator content.
```

2.3.4 Addressing mode for external memory

2.3.4.1 RA register indirect addressing mode

The external memory space address is indirectly specified by the 16-bit address data of the RA register.



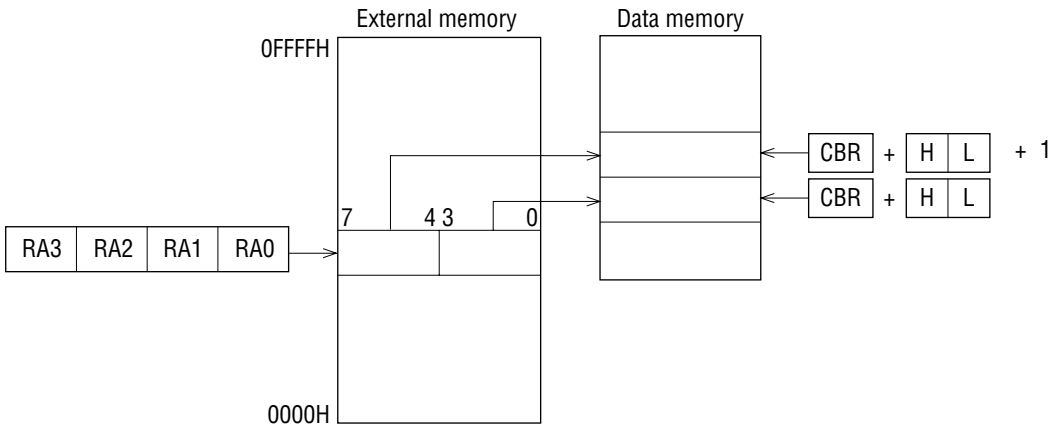
■ Operand description

- [RA]

■ Description example

```
MOVXB [HL], [RA]    ; The content of the external memory specified by RA is transferred to the
                      data memory specified by CBR and HL registers, and to the next address
                      in data memory.
```

[Note] The RA registers from 0F2F to 0F5H of SFR space and XY, HL, CBR, EBR registers from 0F9H to 0FEH should not be addressed as a transfer destination to data memory. If they are addressed, the contents of the addressed registers are undefined.



2.3.4.2 Direct addressing mode

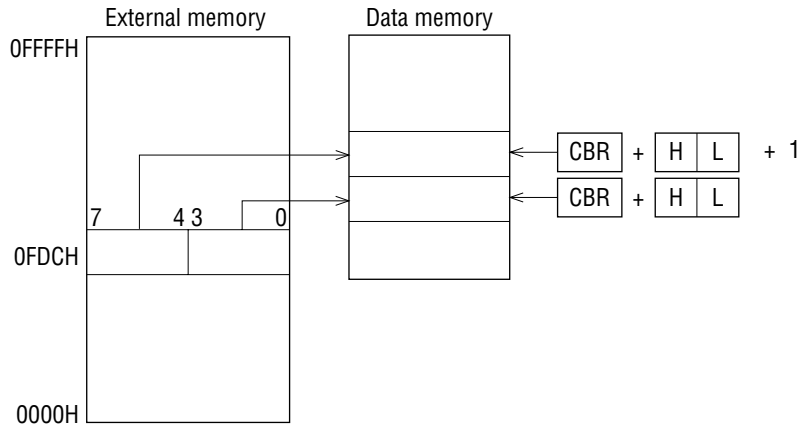
The 16-bit address data is used to address all external memory space.

■ Operand description

- xadr16

■ Description example

MOVXB [HL], 0FDCH;	The content of external memory address 0FDCH is transferred to the data memory specified by CBR and HL registers, and to the next address in data memory.



[Note] The RA registers from 0F2F to 0F5H of SFR space and XY, HL, CBR and EBR registers from 0F9H to 0FEH should not be addressed as a transfer destination to data memory. If they are addressed, the contents of the addressed registers are undefined.

Chapter 2

INSTRUCTION SET

This chapter presents details on the function of the nX-4/250 core and nX-4/300 core instructions.

1. OVERVIEW

The nX-4/250 core and nX-4/300 core instruction codes are 16-bit length, and most instructions operate at one word per cycle.
The nX/-4/250 core and nX/300 core have 440 kinds of instruction, and 450 kinds of instruction respectively.

	nX-4/250	nX-4/300
• Transfer instructions	38 types	38 types
• Rotate instructions	20 types	20 types
• Increment/decrement instructions	20 types	20 types
• Arithmetic operation instructions	96 types	96 types
• Comparison instructions	19 types	19 types
• Logical operation instructions	57 types	57 types
• Mask operation instructions	46 types	50 types
• Bit operation instructions	36 types	40 types
• ROM table reference instructions	32 types	32 types
• External memory transfer instructions	32 types	32 types
• Stack operation instructions	4 types	4 types
• Flag operation instructions	6 types	8 types
• Branch instructions	4 types	4 types
• Conditional branch instructions	7 types	7 types
• Call/return instructions	5 types	5 types
• Control instructions	18 types	18 types

These instructions can be classified by word length and machine cycle as shown below.

• 1 word/1 machine cycle	369 types	379 types
• 1 word/2 machine cycles	28 types	28 types
• 1 word/3 machine cycles	8 types	8 types
• 2 words/2 machine cycles	2 types	2 types
• 2 words/3 machine cycles	33 types	33 types

Instructions have the following functions:

(1) Transfer instructions
These instructions handle data memory transfer and immediate data transfer between accumulator and data memory.

(2) Rotate instructions
These instructions rotate data memory left and right.

(3) Increment/decrement instructions
These instructions handle data memory increment and decrement.

(4) Arithmetic operation instructions
These instructions handle add and subtract operations for data memory and accumulator, or data memory and immediate data.

(5) Comparison instructions
These instructions handle comparison between data memory and accumulator, or data memory and immediate data.

(6) Logical operation instructions
These instructions handle logical product, logical add and exclusive OR between data

memory and accumulator, or data memory and immediate data.

(7) Mask operation instructions
These instructions handle set, clear and transfer operations for non-masked data memory bits.

(8) Bit operation instructions
These instructions handle set, clear and transfer operations for special data memory bits.

(9) ROM table reference instructions
These instructions handle transfer of program memory table data to data memory.

(10) External memory transfer instructions
These instructions handle transfer of external memory data to data memory.

(11) Stack operation instructions
These instructions handle save of flag register, accumulator, HL registers, XY register, extra bank register and current bank register to the register stack, and restore operations.

(12) Flag operation instructions
These instructions handle set and clear operations for the zero flag, carry flag and G flag.

(13) Branch instructions
These instructions execute branches to program memory absolute and relative addresses.

(14) Conditional branch instructions
These instructions execute branches as per the contents of the zero flag, carry flag and G flag.

(15) Call/return instructions
These instructions execute subroutine call/return processing, and return from interrupt routines.

(16) Control instructions
These instructions set the current bank register, extra bank register, RA0 to RA3 registers, HL register, and XY register, as well as starting melody and halting the CPU.

2. OPERAND EXPRESSION

There are two types of instructions: those which have one or two operands, and those which have none.
Operands are specified in the order of <destination>, <source>. Note that for the MTST, MCLR, MSET and MNOT instructions, the mask value is given for the second operand.

■ Explanation of symbols
The following symbols are used in this chapter.

[Register names]

- A ..... Accumulator
- Z ..... Zero flag
- C ..... Carry flag
- G ..... G flag
- FLAG ..... Flag register (cumulative name for Z, C and G flags)
- CBR ..... Current bank register
- EBR ..... Extra bank register
- H ..... H register
- L ..... L register
- HL ..... HL register
- X ..... X register
- Y ..... Y register
- XY ..... XY register
- RA0 ..... RA0 register
- RA1 ..... RA1 register
- RA2 ..... RA2 register
- RA3 ..... RA3 register
- RA ..... RA register (cumulative name for RAO to RA3 registers)
- SP ..... Stack pointer (for call stack)
- RSP ..... Register stack pointer (for register stack)
- MIE ..... Master interrupt enable flag

[Addressing names]

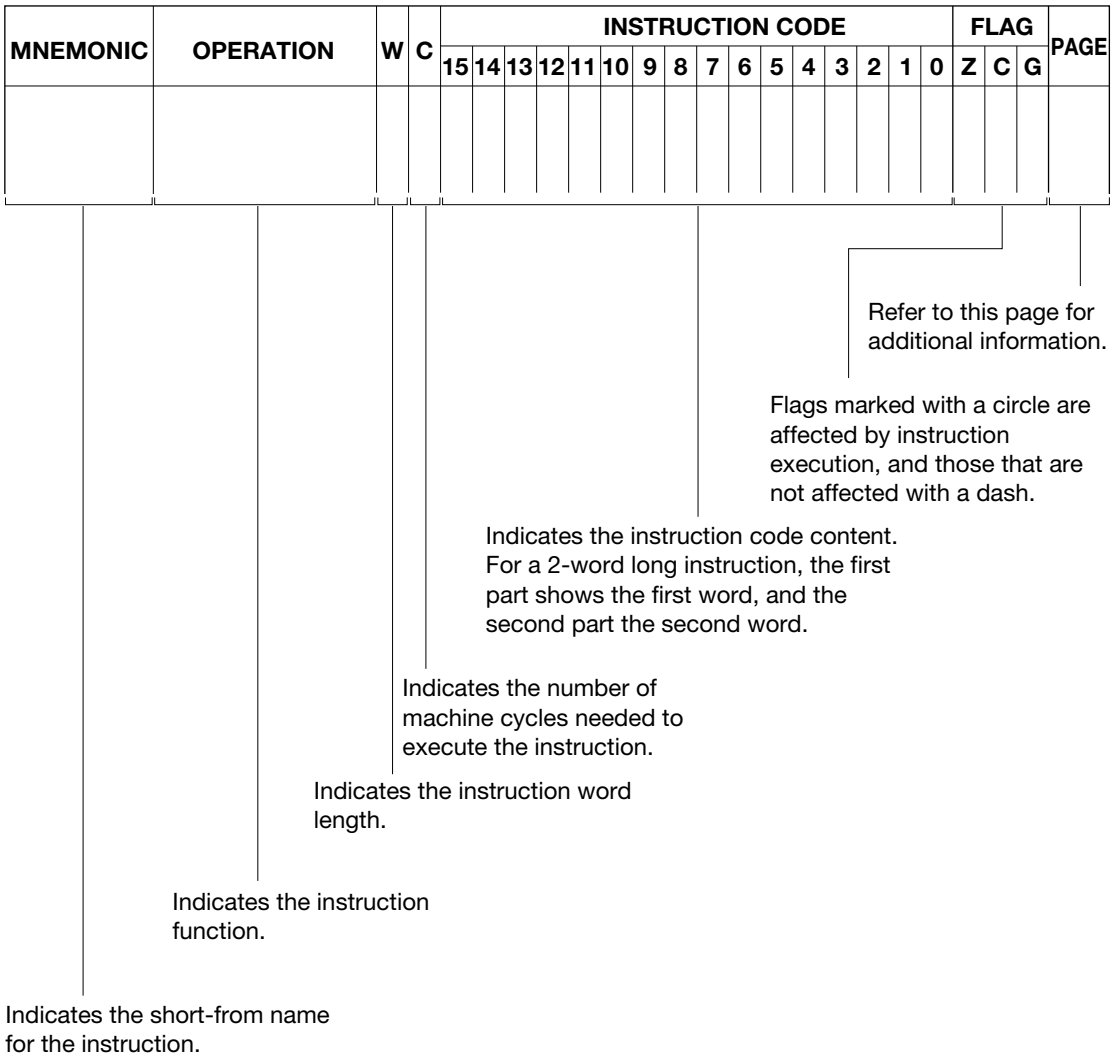
- direct ..... Direct addressing
- sfr ..... SFR bank (bank 0) internal direct addressing
- cur ..... Current bank internal direct addressing
- bank:[reg] ..... Cumulative name for register indirect addressing
- [HL] ..... HL register indirect addressing
- [XY] ..... XY register indirect addressing
- C:[HL] ..... HL register indirect addressing
- C:[XY] ..... XY register indirect addressing
- E:[HL] ..... Extra bank HL register indirect addressing
- E:[XY] ..... Extra bank XY register indirect addressing
- bank:[reg+] ..... Cumulative name for post-incremented register indirect addressing
- [HL+] ..... Post-incremented HL register indirect addressing
- [XY+] ..... Post-incremented XY register indirect addressing
- C:[HL+] ..... Post-incremented HL register indirect addressing
- C:[XY+] ..... Post-incremented XY register indirect addressing
- E:[HL+] ..... Post-incremented extra bank HL register indirect addressing
- E:[XY+] ..... Post-incremented extra bank XY register indirect addressing
- [RA] ..... RA register indirect addressing
- cadr16 ..... 64K word internal direct addressing
- cadr12 ..... 4K word page internal addressing
- radr8 ..... PC relative addressing
- PC+A ..... PC based addressing

[Other]

- ← ..... Indicates data replacement direction
- ↔ ..... Data swap
- + ... Add
- - ..... Subtract
- = ... Equal
- ≠ ..... Not equal
- ^ ..... Logical product
- ∨ ..... Logical add
- ∀ ..... Exclusive logical add
- — ..... Inverse (complement of 1)

3. LIST OF INSTRUCTIONS

The format used in the list of instructions is indicated below.



[Note] Only the nX-4/300 core has MMOV, BMOV, FCLR FLAG and FSET FLAG instructions.

● **Transfer Instructions**

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
MOV direct,A	direct←A	1	1	1	1	0	0	r_{11}	r_{10}	r_9	r_8	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	—	—	—	
MOV [HL],A	[HL]←A	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	—	—	—	
MOV [XY],A	[XY]←A	1	1	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	—	—	—	
MOV E:[HL],A	E:[HL]←A	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	—	—	—	
MOV E:[XY],A	E:[XY]←A	1	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	—	—	—	66
MOV [HL+],A	[HL]←A,HL←HL+1	1	1	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	—	—	❍	
MOV [XY+],A	[XY]←A,XY←XY+1	1	1	0	0	0	0	0	1	0	0	0	0	1	1	1	0	0	0	—	—	❍	
MOV E:[HL+],A	E:[HL]←A,HL←HL+1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	—	—	❍	
MOV E:[XY+],A	E:[XY]←A,XY←XY+1	1	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	—	—	❍	
MOV ¥cur,#i4	cur,A←i4	1	1	0	1	0	0	i_3	i_2	i_1	i_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	
MOV [HL],#i4	[HL],A←i4	1	1	0	0	0	0	0	1	1	0	0	1	1	0	i_3	i_2	i_1	i_0	❍	—	—	
MOV [XY],#i4	[XY],A←i4	1	1	0	0	0	0	0	1	1	0	0	1	1	1	i_3	i_2	i_1	i_0	❍	—	—	
MOV E:[HL],#i4	E:[HL],A←i4	1	1	0	0	0	0	0	1	1	0	0	1	0	0	i_3	i_2	i_1	i_0	❍	—	—	
MOV E:[XY],#i4	E:[XY],A←i4	1	1	0	0	0	0	0	1	1	0	0	1	0	1	i_3	i_2	i_1	i_0	❍	—	—	
MOV [HL+],#i4	[HL],A←i4,HL←HL+1	1	1	0	0	0	0	0	1	1	1	0	1	1	0	i_3	i_2	i_1	i_0	❍	—	❍	67
MOV [XY+],#i4	[XY],A←i4,XY←XY+1	1	1	0	0	0	0	0	1	1	1	0	1	1	1	i_3	i_2	i_1	i_0	❍	—	❍	
MOV E:[HL+],#i4	E:[HL],A←i4,HL←HL+1	1	1	0	0	0	0	0	1	1	1	0	1	0	0	i_3	i_2	i_1	i_0	❍	—	❍	
MOV E:[XY+],#i4	E:[XY],A←i4,XY←XY+1	1	1	0	0	0	0	0	1	1	1	0	1	0	1	i_3	i_2	i_1	i_0	❍	—	❍	
MOV A,#i4	A←i4	1	1	0	0	0	0	0	0	0	1	1	1	0	0	i_3	i_2	i_1	i_0	❍	—	—	65
MOV A,direct	A←direct	1	1	1	1	0	1	r_{11}	r_{10}	r_9	r_8	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	
MOV A,[HL]	A←[HL]	1	1	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	❍	—	—	
MOV A,[XY]	A←[XY]	1	1	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	❍	—	—	
MOV A,E:[HL]	A←E:[HL]	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	❍	—	—	
MOV A,E:[XY]	A←E:[XY]	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	❍	—	—	64
MOV A,[HL+]	A←[HL],HL←HL+1	1	1	0	0	0	0	0	0	1	1	0	0	1	0	1	0	0	0	❍	—	❍	
MOV A,[XY+]	A←[XY],XY←XY+1	1	1	0	0	0	0	0	0	1	1	0	0	1	1	1	0	0	0	❍	—	❍	
MOV A,E:[HL+]	A←E:[HL],HL←HL+1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	❍	—	❍	
MOV A,E:[XY+]	A←E:[XY],XY←XY+1	1	1	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	❍	—	❍	

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
XCH A,sfr	A↔sfr	1	1	0	0	1	0	1	1	1	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	—	—	—	97
XCH A,¥cur	A↔cur	1	1	0	0	1	1	1	1	1	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	—	—	—	
XCH A,[HL]	A↔[HL]	1	1	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	1	—	—	—	
XCH A,[XY]	A↔[XY]	1	1	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	1	—	—	—	
XCH A,E:[HL]	A↔E:[HL]	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	—	—	—	
XCH A,E:[XY]	A↔E:[XY]	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	1	—	—	—	
XCH A,[HL+]	A↔[HL],HL←HL+1	1	1	0	0	0	0	0	0	1	1	0	0	1	0	1	0	0	1	—	—	❍	
XCH A,[XY+]	A↔[XY],XY←XY+1	1	1	0	0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	—	—	❍	
XCH A,E:[HL+]	A↔E:[HL],HL←HL+1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	1	—	—	❍	
XCH A,E:[XY+]	A↔E:[XY],XY←XY+1	1	1	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	1	—	—	❍	

● **Rotate Instructions**

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
ROL sfr	C←{3sfr0}←C,A←sfr	1	1	0	0	1	0	0	0	1	0	r7	r6	r5	r4	r3	r2	r1	r0	❍	❍	—	87
ROL ¥cur	C←{3cur0}←C,A←cur	1	1	0	0	1	1	0	0	1	0	r7	r6	r5	r4	r3	r2	r1	r0	❍	❍	—	
ROL [HL]	C←{3[HL]0}←C,A←[HL]	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0	❍	❍	—	
ROL [XY]	C←{3[XY]0}←C,A←[XY]	1	1	0	0	0	0	0	1	0	0	0	0	1	1	0	1	1	0	❍	❍	—	
ROL E:[HL]	C←{3E:[HL]0}←C,A←E:[HL]	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	❍	❍	—	
ROL E:[XY]	C←{3E:[XY]0}←C,A←E:[XY]	1	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0	❍	❍	—	
ROL [HL+]	C←{3[HL]0}←C,A←[HL], HL←HL+1	1	1	0	0	0	0	0	1	0	0	0	0	1	0	1	1	1	0	❍	❍	❍	
ROL [XY+]	C←{3[XY]0}←C,A←[XY], XY←XY+1	1	1	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	0	❍	❍	❍	
ROL E:[HL+]	C←{3E:[HL]0}←C, A←E:[HL],HL←HL+1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1	1	0	❍	❍	❍	
ROL E:[XY+]	C←{3E:[XY]0}←C, A←E:[XY],XY←XY+1	1	1	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0	❍	❍	❍	
ROR sfr	C→{3sfr0}→C,A←sfr	1	1	0	0	1	0	0	0	1	1	r7	r6	r5	r4	r3	r2	r1	r0	❍	❍	—	88
ROR ¥cur	C→{3cur0}→C,A←cur	1	1	0	0	1	1	0	0	1	1	r7	r6	r5	r4	r3	r2	r1	r0	❍	❍	—	
ROR [HL]	C→{3[HL]0}→C,A←[HL]	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	1	❍	❍	—	
ROR [XY]	C→{3[XY]0}→C,A←[XY]	1	1	0	0	0	0	0	1	0	0	0	0	1	1	0	1	1	1	❍	❍	—	
ROR E:[HL]	C→{3E:[HL]0}→C,A←E:[HL]	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	❍	❍	—	
ROR E:[XY]	C→{3E:[XY]0}→C,A←E:[XY]	1	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	1	1	❍	❍	—	
ROR [HL+]	C→{3[HL]0}→C,A←[HL], HL←HL+1	1	1	0	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	❍	❍	❍	
ROR [XY+]	C→{3[XY]0}→C,A←[XY], XY←XY+1	1	1	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	1	❍	❍	❍	
ROR E:[HL+]	C→{3E:[HL]0}→C, A←E:[HL],HL←HL+1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1	1	1	❍	❍	❍	
ROR E:[XY+]	C→{3E:[XY]0}→C, A←E:[XY],XY←XY+1	1	1	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	1	❍	❍	❍	

● **Increment/decrement Instructions**

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
INC sfr	sfr,A←sfr+1	1	1	0	0	1	0	0	0	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	57
INC ¥cur	cur,A←cur+1	1	1	0	0	1	1	0	0	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	
INC [HL]	[HL],A←[HL]+1	1	1	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	❍	❍	—	
INC [XY]	[XY],A←[XY]+1	1	1	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0	❍	❍	—	
INC E:[HL]	E:[HL],A←E:[HL]+1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	❍	❍	—	
INC E:[XY]	E:[XY],A←E:[XY]+1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	❍	❍	—	
INC [HL+]	[HL],A←[HL]+1, HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	0	1	0	1	0	0	0	❍	❍	❍	
INC [XY+]	[XY],A←[XY]+1, XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	0	1	1	1	0	0	0	❍	❍	❍	
INC E:[HL+]	E:[HL],A←E:[HL]+1, HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	❍	❍	❍	
INC E:[XY+]	E:[XY],A←E:[XY]+1, XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	1	0	0	0	❍	❍	❍	
DEC sfr	sfr,A←sfr−1	1	1	0	0	1	0	0	0	0	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	50
DEC ¥cur	cur,A←cur−1	1	1	0	0	1	1	0	0	0	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	
DEC [HL]	[HL],A←[HL]−1	1	1	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	1	❍	❍	—	
DEC [XY]	[XY],A←[XY]−1	1	1	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	1	❍	❍	—	
DEC E:[HL]	E:[HL],A←E:[HL]−1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	❍	❍	—	
DEC E:[XY]	E:[XY],A←E:[XY]−1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	1	❍	❍	—	
DEC [HL+]	[HL],A←[HL]−1, HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	0	1	0	1	0	0	1	❍	❍	❍	
DEC [XY+]	[XY],A←[XY]−1, XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	0	1	1	1	0	0	1	❍	❍	❍	
DEC E:[HL+]	E:[HL],A←E:[HL]−1, HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	❍	❍	❍	
DEC E:[XY+]	E:[XY],A←E:[XY]−1, XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	1	0	0	1	❍	❍	❍	

● **Arithmetic Operation Instructions**

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
ADD sfr,A	sfr,A←sfr+A	1	1	0	0	1	0	0	1	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	37
ADD ¥cur,A	cur,A←cur+A	1	1	0	0	1	1	0	1	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	
ADD [HL],A	[HL],A←[HL]+A	1	1	0	0	0	0	0	1	0	1	0	0	1	0	0	0	1	0	❍	❍	—	
ADD [XY],A	[XY],A←[XY]+A	1	1	0	0	0	0	0	1	0	1	0	0	1	1	0	0	1	0	❍	❍	—	
ADD E:[HL],A	E:[HL],A←E:[HL]+A	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	❍	❍	—	
ADD E:[XY],A	E:[XY],A←E:[XY]+A	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	0	❍	❍	—	
ADD [HL+],A	[HL],A←[HL]+A, HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	0	1	0	1	0	1	0	❍	❍	❍	
ADD [XY+],A	[XY],A←[XY]+A, XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	0	1	1	1	0	1	0	❍	❍	❍	
ADD E:[HL+],A	E:[HL],A←E:[HL]+A, HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0	1	0	❍	❍	❍	
ADD E:[XY+],A	E:[XY],A←E:[XY]+A, XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	1	0	1	0	❍	❍	❍	
ADD ¥cur,#i4	cur,A←cur+i4	1	1	1	0	0	0	i_3	i_2	i_1	i_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	38
ADD [HL],#i4	[HL],A←[HL]+i4	1	1	0	0	0	0	0	0	0	0	1	0	1	0	i_3	i_2	i_1	i_0	❍	❍	—	
ADD [XY],#i4	[XY],A←[XY]+i4	1	1	0	0	0	0	0	0	0	0	1	0	1	1	i_3	i_2	i_1	i_0	❍	❍	—	
ADD E:[HL],#i4	E:[HL],A←E:[HL]+i4	1	1	0	0	0	0	0	0	0	0	1	0	0	0	i_3	i_2	i_1	i_0	❍	❍	—	
ADD E:[XY],#i4	E:[XY],A←E:[XY]+i4	1	1	0	0	0	0	0	0	0	0	1	0	0	1	i_3	i_2	i_1	i_0	❍	❍	—	
ADD [HL+],#i4	[HL],A←[HL]+i4, HL←HL+1	1	1	0	0	0	0	0	0	0	1	1	0	1	0	i_3	i_2	i_1	i_0	❍	❍	❍	
ADD [XY+],#i4	[XY],A←[XY]+i4, XY←XY+1	1	1	0	0	0	0	0	0	0	1	1	0	1	1	i_3	i_2	i_1	i_0	❍	❍	❍	
ADD E:[HL+],#i4	E:[HL],A←E:[HL]+i4, HL←HL+1	1	1	0	0	0	0	0	0	0	1	1	0	0	0	i_3	i_2	i_1	i_0	❍	❍	❍	
ADD E:[XY+],#i4	E:[XY],A←E:[XY]+i4, XY←XY+1	1	1	0	0	0	0	0	0	0	1	1	0	0	1	i_3	i_2	i_1	i_0	❍	❍	❍	
ADC sfr,A	sfr,A←sfr+A+C	1	1	0	0	1	0	0	1	0	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	34
ADC ¥cur,A	cur,A←cur+A+C	1	1	0	0	1	1	0	1	0	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	
ADC [HL],A	[HL],A←[HL]+A+C	1	1	0	0	0	0	0	1	0	1	0	0	1	0	0	0	1	1	❍	❍	—	
ADC [XY],A	[XY],A←[XY]+A+C	1	1	0	0	0	0	0	1	0	1	0	0	1	1	0	0	1	1	❍	❍	—	
ADC E:[HL],A	E:[HL],A←E:[HL]+A+C	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	1	❍	❍	—	
ADC E:[XY],A	E:[XY],A←E:[XY]+A+C	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	❍	❍	—	

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
ADC [HL+],A	[HL],A←[HL]+A+C, HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	0	1	0	1	0	1	1	❍	❍	❍	34
ADC [XY+],A	[XY],A←[XY]+A+C, XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	0	1	1	1	0	1	1	❍	❍	❍	
ADC E:[HL+],A	E:[HL],A←E:[HL]+A+C, HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0	1	1	❍	❍	❍	
ADC E:[XY+],A	E:[XY],A←E:[XY]+A+C, XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	1	0	1	1	❍	❍	❍	
ADCD sfr,A	sfr,A←decimal adjustment (sfr+A+C)	1	1	0	0	1	0	0	1	1	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	35
ADCD ¥cur,A	cur,A←decimal adjustment (cur+A+C)	1	1	0	0	1	1	0	1	1	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	
ADCD [HL],A	[HL],A←decimal adjustment ([HL]+A+C)	1	1	0	0	0	0	0	1	0	1	0	0	1	0	0	1	0	0	❍	❍	—	
ADCD [XY],A	[XY],A←decimal adjustment ([XY]+A+C)	1	1	0	0	0	0	0	1	0	1	0	0	1	1	0	1	0	0	❍	❍	—	
ADCD E:[HL],A	E:[HL],A←decimal adjustment (E:[HL]+A+C)	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	❍	❍	—	
ADCD E:[XY],A	E:[XY],A←decimal adjustment (E:[XY]+A+C)	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	❍	❍	—	
ADCD [HL+],A	[HL],A←decimal adjustment ([HL]+A+C), HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	0	1	0	1	1	0	0	❍	❍	❍	
ADCD [XY+],A	[XY],A←decimal adjustment ([XY]+A+C), XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	0	1	1	1	1	0	0	❍	❍	❍	
ADCD E:[HL+],A	E:[HL],A←decimal adjustment (E:[HL]+A+C), HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	1	1	0	0	❍	❍	❍	
ADCD E:[XY+],A	E:[XY],A←decimal adjustment (E:[XY]+A+C), XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	1	1	0	0	❍	❍	❍	

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
ADCJ ¥cur,n	cur,A←base-n adjustment (cur+C)	1	1	0	0	0	1	0	n_2	n_1	n_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	
ADCJ [HL],n	[HL],A←base-n adjustment ([HL]+C)	1	1	0	0	0	0	0	1	1	0	0	0	1	0	0	n_2	n_1	n_0	❍	❍	—	
ADCJ [XY],n	[XY],A←base-n adjustment ([XY]+C)	1	1	0	0	0	0	0	1	1	0	0	0	1	1	0	n_2	n_1	n_0	❍	❍	—	
ADCJ E:[HL],n	E:[HL],A←base-n adjustment (E:[HL]+C)	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	n_2	n_1	n_0	❍	❍	—	
ADCJ E:[XY],n	E:[XY],A←base-n adjustment (E:[XY]+C)	1	1	0	0	0	0	0	1	1	0	0	0	0	1	0	n_2	n_1	n_0	❍	❍	—	36
ADCJ [HL+],n	[HL],A←base-n adjustment ([HL]+C),HL←HL+1	1	1	0	0	0	0	0	1	1	1	0	0	1	0	0	n_2	n_1	n_0	❍	❍	❍	
ADCJ [XY+],n	[XY],A←base-n adjustment ([XY]+C),XY←XY+1	1	1	0	0	0	0	0	1	1	1	0	0	1	1	0	n_2	n_1	n_0	❍	❍	❍	
ADCJ E:[HL+],n	E:[HL],A←base-n adjustment (E:[HL]+C),HL←HL+1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	n_2	n_1	n_0	❍	❍	❍	
ADCJ E:[XY+],n	E:[XY],A←base-n adjustment (E:[XY]+C),XY←XY+1	1	1	0	0	0	0	0	1	1	1	0	0	0	1	0	n_2	n_1	n_0	❍	❍	❍	
SUB sfr,A	sfr,A←sfr–A	1	1	0	0	1	0	0	1	1	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	
SUB ¥cur,A	cur,A←cur–A	1	1	0	0	1	1	0	1	1	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	
SUB [HL],A	[HL],A←[HL]–A	1	1	0	0	0	0	0	1	0	1	0	0	1	0	0	1	0	1	❍	❍	—	
SUB [XY],A	[XY],A←[XY]–A	1	1	0	0	0	0	0	1	0	1	0	0	1	1	0	1	0	1	❍	❍	—	
SUB E:[HL],A	E:[HL],A←E:[HL]–A	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1	❍	❍	—	
SUB E:[XY],A	E:[XY],A←E:[XY]–A	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	1	❍	❍	—	
SUB [HL+],A	[HL],A←[HL]–A, HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	0	1	0	1	1	0	1	❍	❍	❍	95
SUB [XY+],A	[XY],A←[XY]–A, XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	0	1	1	1	1	0	1	❍	❍	❍	
SUB E:[HL+],A	E:[HL],A←E:[HL]–A, HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	1	1	0	1	❍	❍	❍	
SUB E:[XY+],A	E:[XY],A←E:[XY]–A, XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	1	1	0	1	❍	❍	❍	

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
SUB ¥cur,#i4	cur,A←cur–i4	1	1	1	0	0	1	i_3	i_2	i_1	i_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	
SUB [HL],#i4	[HL],A←[HL]–i4	1	1	0	0	0	0	0	0	1	0	1	0	1	0	i_3	i_2	i_1	i_0	❍	❍	—	
SUB [XY],#i4	[XY],A←[XY]–i4	1	1	0	0	0	0	0	0	1	0	1	0	1	1	i_3	i_2	i_1	i_0	❍	❍	—	
SUB E:[HL],#i4	E:[HL],A←E:[HL]–i4	1	1	0	0	0	0	0	0	1	0	1	0	0	0	i_3	i_2	i_1	i_0	❍	❍	—	
SUB E:[XY],#i4	E:[XY],A←E:[XY]–i4	1	1	0	0	0	0	0	0	1	0	1	0	0	1	i_3	i_2	i_1	i_0	❍	❍	—	
SUB [HL+],#i4	[HL],A←[HL]–i4, HL←HL+1	1	1	0	0	0	0	0	0	1	1	1	0	1	0	i_3	i_2	i_1	i_0	❍	❍	❍	96
SUB [XY+],#i4	[XY],A←[XY]–i4, XY←XY+1	1	1	0	0	0	0	0	0	1	1	1	0	1	1	i_3	i_2	i_1	i_0	❍	❍	❍	
SUB E:[HL+],#i4	E:[HL],A←E:[HL]–i4, HL←HL+1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	i_3	i_2	i_1	i_0	❍	❍	❍	
SUB E:[XY+],#i4	E:[XY],A←E:[XY]–i4, XY←XY+1	1	1	0	0	0	0	0	0	1	1	1	0	0	1	i_3	i_2	i_1	i_0	❍	❍	❍	
SBC sfr,A	sfr,A←sfr–A–C	1	1	0	0	1	0	1	0	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	
SBC ¥cur,A	cur,A←cur–A–C	1	1	0	0	1	1	1	0	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	
SBC [HL],A	[HL],A←[HL]–A–C	1	1	0	0	0	0	0	1	0	1	0	0	1	0	0	1	1	0	❍	❍	—	
SBC [XY],A	[XY],A←[XY]–A–C	1	1	0	0	0	0	0	1	0	1	0	0	1	1	0	1	1	0	❍	❍	—	
SBC E:[HL],A	E:[HL],A←E:[HL]–A–C	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	0	❍	❍	—	
SBC E:[XY],A	E:[XY],A←E:[XY]–A–C	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	1	0	❍	❍	—	
SBC [HL+],A	[HL],A←[HL]–A–C, HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	0	1	0	1	1	1	0	❍	❍	❍	91
SBC [XY+],A	[XY],A←[XY]–A–C, XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	0	1	1	1	1	1	0	❍	❍	❍	
SBC E:[HL+],A	E:[HL],A←E:[HL]–A–C, HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	1	1	1	0	❍	❍	❍	
SBC E:[XY+],A	E:[XY],A←E:[XY]–A–C, XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	1	1	1	0	❍	❍	❍	
SBCD sfr,A	sfr,A←decimal adjustment (sfr–A–C)	1	1	0	0	1	0	1	0	0	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	
SBCD ¥cur,A	cur,A←decimal adjustment (cur–A–C)	1	1	0	0	1	1	1	0	0	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	92
SBCD [HL],A	[HL],A←decimal adjustment ([HL]–A–C)	1	1	0	0	0	0	0	1	0	1	0	0	1	0	0	1	1	1	❍	❍	—	

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
SBCD [XY],A	[XY],A←decimal adjustment ([XY]−A−C)	1	1	0	0	0	0	0	1	0	1	0	0	1	1	0	1	1	1	❍	❍	—	
SBCD E:[HL],A	E:[HL],A←decimal adjustment (E:[HL]−A−C)	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1	❍	❍	—	
SBCD E:[XY],A	E:[XY],A←decimal adjustment (E:[XY]−A−C)	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1	1	1	❍	❍	—	
SBCD [HL+],A	[HL],A←decimal adjustment ([HL]−A−C), HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	0	1	0	1	1	1	1	❍	❍	❍	
SBCD [XY+],A	[XY],A←decimal adjustment ([XY]−A−C), XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	0	1	1	1	1	1	1	❍	❍	❍	92
SBCD E:[HL+],A	E:[HL],A←decimal adjustment (E:[HL]−A−C), HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	1	1	1	1	❍	❍	❍	
SBCD E:[XY+],A	E:[XY],A←decimal adjustment (E:[XY]−A−C), XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	1	1	1	1	❍	❍	❍	
SBCJ ¥cur,n	cur,A←base-n adjustment (cur−C)	1	1	0	0	0	1	1	n_2	n_1	n_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	
SBCJ [HL],n	[HL],A←base-n adjustment ([HL]−C)	1	1	0	0	0	0	0	1	1	0	0	0	1	0	1	n_2	n_1	n_0	❍	❍	—	
SBCJ [XY],n	[XY],A←base-n adjustment ([XY]−C)	1	1	0	0	0	0	0	1	1	0	0	0	1	1	1	n_2	n_1	n_0	❍	❍	—	
SBCJ E:[HL],n	E:[HL],A←base-n adjustment (E:[HL]−C)	1	1	0	0	0	0	0	1	1	0	0	0	0	0	1	n_2	n_1	n_0	❍	❍	—	
SBCJ E:[XY],n	E:[XY],A←base-n adjustment (E:[XY]−C)	1	1	0	0	0	0	0	1	1	0	0	0	0	1	1	n_2	n_1	n_0	❍	❍	—	
SBCJ [HL+],n	[HL],A←base-n adjustment ([HL]−C),HL←HL+1	1	1	0	0	0	0	0	1	1	1	0	0	1	0	1	n_2	n_1	n_0	❍	❍	❍	
SBCJ [XY+],n	[XY],A←base-n adjustment ([XY]−C),XY←XY+1	1	1	0	0	0	0	0	1	1	1	0	0	1	1	1	n_2	n_1	n_0	❍	❍	❍	
SBCJ E:[HL+],n	E:[HL],A←base-n adjustment (E:[HL]−C),HL←HL+1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	1	n_2	n_1	n_0	❍	❍	❍	
SBCJ E:[XY+],n	E:[XY],A←base-n adjustment (E:[XY]−C),XY←XY+1	1	1	0	0	0	0	0	1	1	1	0	0	0	1	1	n_2	n_1	n_0	❍	❍	❍	93

● **Comparison Instructions**

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
CMP sfr,A	sfr–A	1	1	0	0	1	0	1	0	1	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	
CMP ¥cur,A	cur–A	1	1	0	0	1	1	1	0	1	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	
CMP [HL],A	[HL]–A	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	❍	❍	—	
CMP [XY],A	[XY]–A	1	1	0	0	0	0	0	1	0	0	0	0	1	1	0	1	0	0	❍	❍	—	
CMP E:[HL],A	E:[HL]–A	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	❍	❍	—	48
CMP E:[XY],A	E:[XY]–A	1	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	❍	❍	—	
CMP [HL+],A	[HL]–A,HL←HL+1	1	1	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	0	❍	❍	❍	
CMP [XY+],A	[XY]–A,XY←XY+1	1	1	0	0	0	0	0	1	0	0	0	0	1	1	1	1	0	0	❍	❍	❍	
CMP E:[HL+],A	E:[HL]–A,HL←HL+1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	❍	❍	❍	
CMP E:[XY+],A	E:[XY]–A,XY←XY+1	1	1	0	0	0	0	0	1	0	0	0	0	0	1	1	1	0	0	❍	❍	❍	
CMP ¥cur,#i4	cur–i4	1	1	1	0	1	0	i_3	i_2	i_1	i_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	❍	—	
CMP [HL],#i4	[HL]–i4	1	1	0	0	0	0	0	1	1	0	1	0	1	0	i_3	i_2	i_1	i_0	❍	❍	—	
CMP [XY],#i4	[XY]–i4	1	1	0	0	0	0	0	1	1	0	1	0	1	1	i_3	i_2	i_1	i_0	❍	❍	—	
CMP E:[HL],#i4	E:[HL]–i4	1	1	0	0	0	0	0	1	1	0	1	0	0	0	i_3	i_2	i_1	i_0	❍	❍	—	
CMP E:[XY],#i4	E:[XY]–i4	1	1	0	0	0	0	0	1	1	0	1	0	0	1	i_3	i_2	i_1	i_0	❍	❍	—	49
CMP [HL+],#i4	[HL]–i4,HL←HL+1	1	1	0	0	0	0	0	1	1	1	1	0	1	0	i_3	i_2	i_1	i_0	❍	❍	❍	
CMP [XY+],#i4	[XY]–i4,XY←XY+1	1	1	0	0	0	0	0	1	1	1	1	0	1	1	i_3	i_2	i_1	i_0	❍	❍	❍	
CMP E:[HL+],#i4	E:[HL]–i4,HL←HL+1	1	1	0	0	0	0	0	1	1	1	1	0	0	0	i_3	i_2	i_1	i_0	❍	❍	❍	
CMP E:[XY+],#i4	E:[XY]–i4,XY←XY+1	1	1	0	0	0	0	0	1	1	1	1	0	0	1	i_3	i_2	i_1	i_0	❍	❍	❍	

● **Logical Operation Instructions**

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
AND sfr,A	sfr,A←sfr ∧ A	1	1	0	0	1	0	1	0	1	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	39
AND ¥cur,A	cur,A←cur ∧ A	1	1	0	0	1	1	1	0	1	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	
AND [HL],A	[HL],A←[HL] ∧ A	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	❍	—	—	
AND [XY],A	[XY],A←[XY] ∧ A	1	1	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	❍	—	—	
AND E:[HL],A	E:[HL],A←E:[HL] ∧ A	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	❍	—	—	
AND E:[XY],A	E:[XY],A←E:[XY] ∧ A	1	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	❍	—	—	
AND [HL+],A	[HL],A←[HL] ∧ A, HL←HL+1	1	1	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	1	❍	—	❍	
AND [XY+],A	[XY],A←[XY] ∧ A, XY←XY+1	1	1	0	0	0	0	0	1	0	0	0	0	1	1	1	0	0	1	❍	—	❍	
AND E:[HL+],A	E:[HL],A←E:[HL] ∧ A, HL←HL+1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	❍	—	❍	
AND E:[XY+],A	E:[XY],A←E:[XY] ∧ A, XY←XY+1	1	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	1	❍	—	❍	
AND ¥cur,#i4	cur,A←cur ∧ i4	1	1	0	1	0	1	i_3	i_2	i_1	i_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	40
AND [HL],#i4	[HL],A←[HL] ∧ i4	1	1	0	0	0	0	0	1	0	0	0	1	1	0	i_3	i_2	i_1	i_0	❍	—	—	
AND [XY],#i4	[XY],A←[XY] ∧ i4	1	1	0	0	0	0	0	1	0	0	0	1	1	1	i_3	i_2	i_1	i_0	❍	—	—	
AND E:[HL],#i4	E:[HL],A←E:[HL] ∧ i4	1	1	0	0	0	0	0	1	0	0	0	1	0	0	i_3	i_2	i_1	i_0	❍	—	—	
AND E:[XY],#i4	E:[XY],A←E:[XY] ∧ i4	1	1	0	0	0	0	0	1	0	0	0	1	0	1	i_3	i_2	i_1	i_0	❍	—	—	
AND [HL+],#i4	[HL],A←[HL] ∧ i4, HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	1	1	0	i_3	i_2	i_1	i_0	❍	—	❍	
AND [XY+],#i4	[XY],A←[XY] ∧ i4, XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	1	1	1	i_3	i_2	i_1	i_0	❍	—	❍	
AND E:[HL+],#i4	E:[HL],A←E:[HL] ∧ i4, HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	1	0	0	i_3	i_2	i_1	i_0	❍	—	❍	
AND E:[XY+],#i4	E:[XY],A←E:[XY] ∧ i4, XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	1	0	1	i_3	i_2	i_1	i_0	❍	—	❍	

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
OR sfr,A	sfr,A←sfr ∨ A	1	1	0	0	1	0	1	1	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	
OR ¥cur,A	cur,A←cur ∨ A	1	1	0	0	1	1	1	1	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	
OR [HL],A	[HL],A←[HL] ∨ A	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	❍	—	—	
OR [XY],A	[XY],A←[XY] ∨ A	1	1	0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	0	❍	—	—	
OR E:[HL],A	E:[HL],A←E:[HL] ∨ A	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	❍	—	—	
OR E:[XY],A	E:[XY],A←E:[XY] ∨ A	1	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	❍	—	—	
OR [HL+],A	[HL],A←[HL] ∨ A, HL←HL+1	1	1	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	0	❍	—	❍	82
OR [XY+],A	[XY],A←[XY] ∨ A, XY←XY+1	1	1	0	0	0	0	0	1	0	0	0	0	1	1	1	0	1	0	❍	—	❍	
OR E:[HL+],A	E:[HL],A←E:[HL] ∨ A, HL←HL+1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	❍	—	❍	
OR E:[XY+],A	E:[XY],A←E:[XY] ∨ A, XY←XY+1	1	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	❍	—	❍	
OR ¥cur,#i4	cur,A←cur ∨ i4	1	1	0	1	1	0	i_3	i_2	i_1	i_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	
OR [HL],#i4	[HL],A←[HL] ∨ i4	1	1	0	0	0	0	0	0	1	0	0	1	1	0	i_3	i_2	i_1	i_0	❍	—	—	
OR [XY],#i4	[XY],A←[XY] ∨ i4	1	1	0	0	0	0	0	0	1	0	0	1	1	1	i_3	i_2	i_1	i_0	❍	—	—	
OR E:[HL],#i4	E:[HL],A←E:[HL] ∨ i4	1	1	0	0	0	0	0	0	1	0	0	1	0	0	i_3	i_2	i_1	i_0	❍	—	—	
OR E:[XY],#i4	E:[XY],A←E:[XY] ∨ i4	1	1	0	0	0	0	0	0	1	0	0	1	0	1	i_3	i_2	i_1	i_0	❍	—	—	
OR [HL+],#i4	[HL],A←[HL] ∨ i4, HL←HL+1	1	1	0	0	0	0	0	0	1	1	0	1	1	0	i_3	i_2	i_1	i_0	❍	—	❍	83
OR [XY+],#i4	[XY],A←[XY] ∨ i4, XY←XY+1	1	1	0	0	0	0	0	0	1	1	0	1	1	1	i_3	i_2	i_1	i_0	❍	—	❍	
OR E:[HL+],#i4	E:[HL],A←E:[HL] ∨ i4, HL←HL+1	1	1	0	0	0	0	0	0	1	1	0	1	0	0	i_3	i_2	i_1	i_0	❍	—	❍	
OR E:[XY+],#i4	E:[XY],A←E:[XY] ∨ i4, XY←XY+1	1	1	0	0	0	0	0	0	1	1	0	1	0	1	i_3	i_2	i_1	i_0	❍	—	❍	

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
XOR sfr,A	sfr,A←sfr∀A	1	1	0	0	1	0	1	1	0	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	
XOR ¥cur,A	cur,A←cur∀A	1	1	0	0	1	1	1	1	0	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	
XOR [HL],A	[HL],A←[HL]∀A	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	1	❍	—	—	
XOR [XY],A	[XY],A←[XY]∀A	1	1	0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	1	❍	—	—	
XOR E:[HL],A	E:[HL],A←E:[HL]∀A	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	❍	—	—	
XOR E:[XY],A	E:[XY],A←E:[XY]∀A	1	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	❍	—	—	
XOR [HL+],A	[HL],A←[HL]∀A,HL←HL+1	1	1	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	1	❍	—	❍	98
XOR [XY+],A	[XY],A←[XY]∀A,XY←XY+1	1	1	0	0	0	0	0	1	0	0	0	0	1	1	1	0	1	1	❍	—	❍	
XOR E:[HL+],A	E:[HL],A←E:[HL]∀A, HL←HL+1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	1	❍	—	❍	
XOR E:[XY+],A	E:[XY],A←E:[XY]∀A, XY←XY+1	1	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	1	❍	—	❍	
XOR ¥cur,#i4	cur,A←cur∀i4	1	1	0	1	1	1	i_3	i_2	i_1	i_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	
XOR [HL],#i4	[HL],A←[HL]∀i4	1	1	0	0	0	0	0	0	0	0	0	1	1	0	i_3	i_2	i_1	i_0	❍	—	—	
XOR [XY],#i4	[XY],A←[XY]∀i4	1	1	0	0	0	0	0	0	0	0	0	1	1	1	i_3	i_2	i_1	i_0	❍	—	—	
XOR E:[HL],#i4	E:[HL],A←E:[HL]∀i4	1	1	0	0	0	0	0	0	0	0	0	1	0	0	i_3	i_2	i_1	i_0	❍	—	—	
XOR E:[XY],#i4	E:[XY],A←E:[XY]∀i4	1	1	0	0	0	0	0	0	0	0	0	1	0	1	i_3	i_2	i_1	i_0	❍	—	—	
XOR [HL],#i4	[HL],A←[HL]∀i4, HL←HL+1	1	1	0	0	0	0	0	0	0	1	0	1	1	0	i_3	i_2	i_1	i_0	❍	—	❍	99
XOR [XY+],#i4	[XY],A←[XY]∀i4, XY←XY+1	1	1	0	0	0	0	0	0	0	1	0	1	1	1	i_3	i_2	i_1	i_0	❍	—	❍	
XOR E:[HL+],#i4	E:[HL],A←E:[HL]∀i4, HL←HL+1	1	1	0	0	0	0	0	0	0	1	0	1	0	0	i_3	i_2	i_1	i_0	❍	—	❍	
XOR E:[XY+],#i4	E:[XY],A←E:[XY]∀i4, XY←XY+1	1	1	0	0	0	0	0	0	0	1	0	1	0	1	i_3	i_2	i_1	i_0	❍	—	❍	

● Mask Operation Instructions

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
MMOV [HL]#m,A.n	Transfer A.n to all bits not masked by [HL]#m←A[HL]	1	1	0	0	0	0	0	n_1	n_0	0	1	1	1	0	m_3	m_2	m_1	m_0	❍	—	—	62
MMOV [XY]#m,A.n	Transfer A.n to all bits not masked by [XY]#m←A[HL]	1	1	0	0	0	0	0	n_1	n_0	0	1	1	1	1	m_3	m_2	m_1	m_0	❍	—	—	
MMOV E:[HL]#m,A.n	Transfer A.n to all bits not masked by [HL]#m←A[HL]	1	1	0	0	0	0	0	n_1	n_0	0	1	1	0	0	m_3	m_2	m_1	m_0	❍	—	—	
MMOV E:[XY]#m,A.n	Transfer A.n to all bits not masked by [XY]#m←A[HL]	1	1	0	0	0	0	0	n_1	n_0	0	1	1	0	1	m_3	m_2	m_1	m_0	❍	—	—	
MTST sfr,A	Test all bits not masked sfr A	1	1	0	0	1	0	1	1	1	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	79
MTST ¥cur,A	Test all bits not masked cur A	1	1	0	0	1	1	1	1	1	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	
MTST [HL],A	Test all bits not masked by [HL] A	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	❍	—	—	
MTST [XY],A	Test all bits not masked by [XY] A	1	1	0	0	0	0	0	1	0	0	0	0	1	1	0	1	0	1	❍	—	—	
MTST E:[HL],A	Test all bits not masked by E:[HL] A	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	❍	—	—	
MTST E:[XY],A	Test all bits not masked by E:[XY] A	1	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	1	❍	—	—	
MTST [HL+],A	Test all bits not masked by [HL] A, HL←HL+1	1	1	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	1	❍	—	❍	
MTST [XY+],A	Test all bits not masked by [XY] A, XY←XY+1	1	1	0	0	0	0	0	1	0	0	0	0	1	1	1	1	0	1	❍	—	❍	
MTST E:[HL+],A	Test all bits not masked by A E:[HL], HL←HL+1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	❍	—	❍	
MTST E:[XY+],A	Test all bits not masked by A E:[XY] , XY←XY+1	1	1	0	0	0	0	0	1	0	0	0	0	0	1	1	1	0	1	❍	—	❍	

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
MTST ¥cur,#m	Test all bits not masked by cur #m	1	1	1	0	1	1	m_3	m_2	m_1	m_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	
MTST [HL],#m	Test all bits not masked by [HL] #m	1	1	0	0	0	0	0	1	0	0	1	0	1	0	m_3	m_2	m_1	m_0	❍	—	—	
MTST [XY],#m	Test all bits not masked by [XY] #m	1	1	0	0	0	0	0	1	0	0	1	0	1	1	m_3	m_2	m_1	m_0	❍	—	—	
MTST E:[HL],#m	Test all bits not masked by E:[HL] #m	1	1	0	0	0	0	0	1	0	0	1	0	0	0	m_3	m_2	m_1	m_0	❍	—	—	
MTST E:[XY],#m	Test all bits not masked by E:[XY] #m	1	1	0	0	0	0	0	1	0	0	1	0	0	1	m_3	m_2	m_1	m_0	❍	—	—	
MTST [HL+],#m	Test all bits not masked by [HL] #m, HL←HL+1	1	1	0	0	0	0	0	1	0	1	1	0	1	0	m_3	m_2	m_1	m_0	❍	—	❍	80
MTST [XY+],#m	Test all bits not masked by [XY] #m, XY←XY+1	1	1	0	0	0	0	0	1	0	1	1	0	1	1	m_3	m_2	m_1	m_0	❍	—	❍	
MTST E:[HL+], #m	Test all bits not masked by E:[HL] #m, HL←HL+1	1	1	0	0	0	0	0	1	0	1	1	0	0	0	m_3	m_2	m_1	m_0	❍	—	❍	
MTST E:[XY+], #m	Test all bits not masked by E:[XY] #m, XY←XY+1	1	1	0	0	0	0	0	1	0	1	1	0	0	1	m_3	m_2	m_1	m_0	❍	—	❍	

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
MCLR ¥cur,#m	Clear all bit not masked by cur #m, A←cur	1	1	0	1	0	1	m_3	m_2	m_1	m_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	
MCLR [HL],#m	Clear all bit not masked by [HL] #m,A←[HL]	1	1	0	0	0	0	0	1	0	0	0	1	1	0	m_3	m_2	m_1	m_0	❍	—	—	
MCLR [XY],#m	Clear all bit not masked by [XY] #m,A←[XY]	1	1	0	0	0	0	0	1	0	0	0	1	1	1	m_3	m_2	m_1	m_0	❍	—	—	
MCLR E:[HL],#m	Clear all bit not masked by E:[HL] #m,A←E:[HL]	1	1	0	0	0	0	0	1	0	0	0	1	0	0	m_3	m_2	m_1	m_0	❍	—	—	
MCLR E:[XY],#m	Clear all bit not masked by E:[XY] #m,A←E:[XY]	1	1	0	0	0	0	0	1	0	0	0	1	0	1	m_3	m_2	m_1	m_0	❍	—	—	61
MCLR [HL+],#m	Clear all bit not masked by [HL] #m,A←[HL],HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	1	1	0	m_3	m_2	m_1	m_0	❍	—	❍	
MCLR [XY+],#m	Clear all bit not masked by [XY] #m,A←[XY],XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	1	1	1	m_3	m_2	m_1	m_0	❍	—	❍	
MCLR E:[HL+], #m	Clear all bit not masked by E:[HL] #m,A←E:[HL],HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	1	0	0	m_3	m_2	m_1	m_0	❍	—	❍	
MCLR E:[XY+], #m	Clear all bit not masked by E:[XY] #m,A←E:[XY],XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	1	0	1	m_3	m_2	m_1	m_0	❍	—	❍	

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
MSET ¥cur,#m	Set all bits not masked by cur #m, A←cur	1	1	0	1	1	0	m_3	m_2	m_1	m_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	
MSET [HL],#m	Set all bits not masked by [HL] #m, A←[HL]	1	1	0	0	0	0	0	0	1	0	0	1	1	0	m_3	m_2	m_1	m_0	❍	—	—	
MSET [XY],#m	Set all bits not masked by [XY] #m, A←[XY]	1	1	0	0	0	0	0	0	1	0	0	1	1	1	m_3	m_2	m_1	m_0	❍	—	—	
MSET E:[HL],#m	Set all bits not masked by E:[HL] #m, A←E:[HL]	1	1	0	0	0	0	0	0	1	0	0	1	0	0	m_3	m_2	m_1	m_0	❍	—	—	
MSET E:[XY],#m	Set all bits not masked by E:[XY] #m, A←E:[XY]	1	1	0	0	0	0	0	0	1	0	0	1	0	1	m_3	m_2	m_1	m_0	❍	—	—	78
MSET [HL+],#m	Set all bits not masked by [HL] #m, A←[HL],HL←HL+1	1	1	0	0	0	0	0	0	1	1	0	1	1	0	m_3	m_2	m_1	m_0	❍	—	❍	
MSET [XY+],#m	Set all bits not masked by [XY] #m, A←[XY],XY←XY+1	1	1	0	0	0	0	0	0	1	1	0	1	1	1	m_3	m_2	m_1	m_0	❍	—	❍	
MSET E:[HL+], #m	Set all bits not masked by E:[HL] #m, A←E:[HL],HL←HL+1	1	1	0	0	0	0	0	0	1	1	0	1	0	0	m_3	m_2	m_1	m_0	❍	—	❍	
MSET E:[XY+], #m	Set all bits not masked by E:[XY] #m, A←E:[XY],XY←XY+1	1	1	0	0	0	0	0	0	1	1	0	1	0	1	m_3	m_2	m_1	m_0	❍	—	❍	

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
MNOT ¥cur,#m	Invert all bits not masked by cur #m, A←cur	1	1	0	1	1	1	m_3	m_2	m_1	m_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	
MNOT [HL],#m	Invert all bits not masked by [HL] #m, A←[HL]	1	1	0	0	0	0	0	0	0	0	0	1	1	0	m_3	m_2	m_1	m_0	❍	—	—	
MNOT [XY],#m	Invert all bits not masked by [XY] #m, A←[XY]	1	1	0	0	0	0	0	0	0	0	0	1	1	1	m_3	m_2	m_1	m_0	❍	—	—	
MNOT E:[HL],#m	Invert all bits not masked by E:[HL] #m, A←E:[HL]	1	1	0	0	0	0	0	0	0	0	0	1	0	0	m_3	m_2	m_1	m_0	❍	—	—	
MNOT E:[XY],#m	Invert all bits not masked by E:[XY] #m, A←E:[XY]	1	1	0	0	0	0	0	0	0	0	0	1	0	1	m_3	m_2	m_1	m_0	❍	—	—	63
MNOT [HL+],#m	Invert all bits not masked by [HL] #m, A←[HL],HL←HL+1	1	1	0	0	0	0	0	0	0	1	0	1	1	0	m_3	m_2	m_1	m_0	❍	—	❍	
MNOT [XY+],#m	Invert all bits not masked by [XY] #m, A←[XY],XY←XY+1	1	1	0	0	0	0	0	0	0	1	0	1	1	1	m_3	m_2	m_1	m_0	❍	—	❍	
MNOT E:[HL+], #m	Invert all bits not masked by E:[HL] #m, A←E:[HL],HL←HL+1	1	1	0	0	0	0	0	0	0	1	0	1	0	0	m_3	m_2	m_1	m_0	❍	—	❍	
MNOT E:[XY+], #m	Invert all bits not masked by E:[XY] #m, A←E:[XY],XY←XY+1	1	1	0	0	0	0	0	0	0	1	0	1	0	1	m_3	m_2	m_1	m_0	❍	—	❍	

● **Bit Operation Instructions**

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
BMOV [HL].n,A.n	[HL].n←A.n,A←[HL]	1	1	0	0	0	0	0	n_1	n_0	0	1	1	1	0	r_3	r_2	r_1	r_0	❍	—	—	43
BMOV [XY].n,A.n	[XY].n←A.n,A←[XY]	1	1	0	0	0	0	0	n_1	n_0	0	1	1	1	1	r_3	r_2	r_1	r_0	❍	—	—	
BMOV E:[HL].n,A.n	E:[HL].n←A.n,A←E:[HL]	1	1	0	0	0	0	0	n_1	n_0	0	1	1	0	0	r_3	r_2	r_1	r_0	❍	—	—	
BMOV E:[XY].n,A.n	E:[XY].n←A.n,A←E:[XY]	1	1	0	0	0	0	0	n_1	n_0	0	1	1	0	1	r_3	r_2	r_1	r_0	❍	—	—	
BTST ¥cur.n	cur.n bit test	1	1	1	0	1	1	n_3	n_2	n_1	n_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	46
BTST [HL].n	[HL].n bit test	1	1	0	0	0	0	0	1	0	0	1	0	1	0	n_3	n_2	n_1	n_0	❍	—	—	
BTST [XY].n	[XY].n bit test	1	1	0	0	0	0	0	1	0	0	1	0	1	1	n_3	n_2	n_1	n_0	❍	—	—	
BTST E:[HL].n	E:[HL].n bit test	1	1	0	0	0	0	0	1	0	0	1	0	0	0	n_3	n_2	n_1	n_0	❍	—	—	
BTST E:[XY].n	E:[XY].n bit test	1	1	0	0	0	0	0	1	0	0	1	0	0	1	n_3	n_2	n_1	n_0	❍	—	—	
BTST [HL+].n	[HL].n, bit test HL←HL+1	1	1	0	0	0	0	0	1	0	1	1	0	1	0	n_3	n_2	n_1	n_0	❍	—	❍	
BTST [XY+].n	[XY].n, bit test XY←XY+1	1	1	0	0	0	0	0	1	0	1	1	0	1	1	n_3	n_2	n_1	n_0	❍	—	❍	
BTST E:[HL+].n	E:[HL].n, bit test HL←HL+1	1	1	0	0	0	0	0	1	0	1	1	0	0	0	n_3	n_2	n_1	n_0	❍	—	❍	
BTST E:[XY+].n	E:[XY].n, bit test XY←XY+1	1	1	0	0	0	0	0	1	0	1	1	0	0	1	n_3	n_2	n_1	n_0	❍	—	❍	
BCLR ¥cur.n	cur.n←0,A←cur	1	1	0	1	0	1	n_3	n_2	n_1	n_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	42
BCLR [HL].n	[HL].n←0,A←[HL]	1	1	0	0	0	0	0	1	0	0	0	1	1	0	n_3	n_2	n_1	n_0	❍	—	—	
BCLR [XY].n	[XY].n←0,A←[XY]	1	1	0	0	0	0	0	1	0	0	0	1	1	1	n_3	n_2	n_1	n_0	❍	—	—	
BCLR E:[HL].n	E:[HL].n←0,A←E:[HL]	1	1	0	0	0	0	0	1	0	0	0	1	0	0	n_3	n_2	n_1	n_0	❍	—	—	
BCLR E:[XY].n	E:[XY].n←0,A←E:[XY]	1	1	0	0	0	0	0	1	0	0	0	1	0	1	n_3	n_2	n_1	n_0	❍	—	—	
BCLR [HL+].n	[HL].n←0,A←[HL], HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	1	1	0	n_3	n_2	n_1	n_0	❍	—	❍	
BCLR [XY+].n	[XY].n←0,A←[XY], XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	1	1	1	n_3	n_2	n_1	n_0	❍	—	❍	
BCLR E:[HL+].n	E:[HL].n←0,A←E:[HL], HL←HL+1	1	1	0	0	0	0	0	1	0	1	0	1	0	0	n_3	n_2	n_1	n_0	❍	—	❍	
BCLR E:[XY+].n	E:[XY].n←0,A←E:[XY], XY←XY+1	1	1	0	0	0	0	0	1	0	1	0	1	0	1	n_3	n_2	n_1	n_0	❍	—	❍	

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
BSET ¥cur.n	cur.n←1,A←cur	1	1	0	1	1	0	n_3	n_2	n_1	n_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	—	—	—	
BSET [HL].n	[HL].n←1,A←[HL]	1	1	0	0	0	0	0	0	1	0	0	1	1	0	n_3	n_2	n_1	n_0	—	—	—	
BSET [XY].n	[XY].n←1,A←[XY]	1	1	0	0	0	0	0	0	1	0	0	1	1	1	n_3	n_2	n_1	n_0	—	—	—	
BSET E:[HL].n	E:[HL].n←1,A←E:[HL]	1	1	0	0	0	0	0	0	1	0	0	1	0	0	n_3	n_2	n_1	n_0	—	—	—	
BSET E:[XY].n	E:[XY].n←1,A←E:[XY]	1	1	0	0	0	0	0	0	1	0	0	1	0	1	n_3	n_2	n_1	n_0	—	—	—	
BSET [HL+].n	[HL].n←1,A←[HL], HL←HL+1	1	1	0	0	0	0	0	0	1	1	0	1	1	0	n_3	n_2	n_1	n_0	—	—	❍	45
BSET [XY+].n	[XY].n←1,A←[XY], XY←XY+1	1	1	0	0	0	0	0	0	1	1	0	1	1	1	n_3	n_2	n_1	n_0	—	—	❍	
BSET E:[HL+].n	E:[HL].n←1,A←E:[HL], HL←HL+1	1	1	0	0	0	0	0	0	1	1	0	1	0	0	n_3	n_2	n_1	n_0	—	—	❍	
BSET E:[XY+].n	E:[XY].n←1,A←E:[XY], XY←XY+1	1	1	0	0	0	0	0	0	1	1	0	1	0	1	n_3	n_2	n_1	n_0	—	—	❍	
BNOT ¥cur.n	cur.n←$\overline{\text{cur.n}}$,A←cur	1	1	0	1	1	1	n_3	n_2	n_1	n_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	❍	—	—	
BNOT [HL].n	[HL].n←$\overline{\text{[HL].n}}$,A←[HL]	1	1	0	0	0	0	0	0	0	0	0	1	1	0	n_3	n_2	n_1	n_0	❍	—	—	
BNOT [XY].n	[XY].n←$\overline{\text{[XY].n}}$,A←[XY]	1	1	0	0	0	0	0	0	0	0	0	1	1	1	n_3	n_2	n_1	n_0	❍	—	—	
BNOT E:[HL].n	E:[HL].n←$\overline{\text{E:[HL].n}}$,A←E:[HL]	1	1	0	0	0	0	0	0	0	0	0	1	0	0	n_3	n_2	n_1	n_0	❍	—	—	
BNOT E:[XY].n	E:[XY].n←$\overline{\text{E:[XY].n}}$,A←E:[XY]	1	1	0	0	0	0	0	0	0	0	0	1	0	1	n_3	n_2	n_1	n_0	❍	—	—	
BNOT [HL+].n	[HL].n←$\overline{\text{[HL].n}}$,A←[HL], HL←HL+1	1	1	0	0	0	0	0	0	0	1	0	1	1	0	n_3	n_2	n_1	n_0	❍	—	❍	44
BNOT [XY+].n	[XY].n←$\overline{\text{[XY].n}}$,A←[XY], XY←XY+1	1	1	0	0	0	0	0	0	0	1	0	1	1	1	n_3	n_2	n_1	n_0	❍	—	❍	
BNOT E:[HL+].n	E:[HL].n←$\overline{\text{E:[HL].n}}$,A←E:[HL],HL←HL+1	1	1	0	0	0	0	0	0	0	1	0	1	0	0	n_3	n_2	n_1	n_0	❍	—	❍	
BNOT E:[XY+].n	E:[XY].n←$\overline{\text{E:[XY].n}}$,A←E:[XY],XY←XY+1	1	1	0	0	0	0	0	0	0	1	0	1	0	1	n_3	n_2	n_1	n_0	❍	—	❍	

● **ROM Table Reference Instructions**

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
MOVHB [HL], [RA]	[HL],[HL+1]←$(RA)_{15\sim8}$	1	2	0	0	0	0	0	0	1	1	0	0	1	0	0	0	1	0	—	—	—	
MOVHB [XY], [RA]	[XY],[XY+1]←$(RA)_{15\sim8}$	1	2	0	0	0	0	0	0	1	1	0	0	1	1	0	0	1	0	—	—	—	
MOVHB E:[HL], [RA]	E:[HL],E:[HL+1]←$(RA)_{15\sim8}$	1	2	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	—	—	—	
MOVHB E:[XY], [RA]	E:[XY],E:[XY+1]←$(RA)_{15\sim8}$	1	2	0	0	0	0	0	0	1	1	0	0	0	1	0	0	1	0	—	—	—	69
MOVHB [HL+], [RA]	[HL],[HL+1]←$(RA)_{15\sim8}$, HL←HL+2	1	2	0	0	0	0	0	0	1	1	0	0	1	0	1	0	1	0	—	—	❍	
MOVHB [XY+], [RA]	[XY],[XY+1]←$(RA)_{15\sim8}$, XY←XY+2	1	2	0	0	0	0	0	0	1	1	0	0	1	1	1	0	1	0	—	—	❍	
MOVHB E:[HL+], [RA]	E:[HL],E:[HL+1]←$(RA)_{15\sim8}$,HL←HL+2	1	2	0	0	0	0	0	0	1	1	0	0	0	0	1	0	1	0	—	—	❍	
MOVHB E:[XY+], [RA]	E:[XY],E:[XY+1]←$(RA)_{15\sim8}$,XY←XY+2	1	2	0	0	0	0	0	0	1	1	0	0	0	1	1	0	1	0	—	—	❍	
MOVHB [HL], cadr16	[HL],[HL+1]←$(cadr16)_{15\sim8}$	2	3	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0	0	—	—	—	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVHB [XY], cadr16	[XY],[XY+1]←$(cadr16)_{15\sim8}$	2	3	0	0	0	0	0	0	1	1	0	0	1	1	0	1	0	0	—	—	—	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVHB E:[HL], cadr16	E:[HL],E:[HL+1]←$(cadr16)_{15\sim8}$	2	3	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	—	—	—	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVHB E:[XY], cadr16	E:[XY],E:[XY+1]←$(cadr16)_{15\sim8}$	2	3	0	0	0	0	0	0	1	1	0	0	0	1	0	1	0	0	—	—	—	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				70
MOVHB [HL+], cadr16	[HL],[HL+1]←$(cadr16)_{15\sim8}$,HL←HL+2	2	3	0	0	0	0	0	0	1	1	0	0	1	0	1	1	0	0	—	—	❍	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVHB [XY+], cadr16	[XY],[XY+1]←$(cadr16)_{15\sim8}$,XY←XY+2	2	3	0	0	0	0	0	0	1	1	0	0	1	1	1	1	0	0	—	—	❍	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVHB E:[HL+], cadr16	E:[HL],E:[HL+1]←$(cadr16)_{15\sim8}$,HL←HL+2	2	3	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	—	—	❍	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVHB E:[XY+], cadr16	E:[XY],E:[XY+1]←$(cadr16)_{15\sim8}$,XY←XY+2	2	3	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0	0	—	—	❍	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
MOVLB [HL], [RA]	$[HL],[HL+1] \leftarrow (RA)_{7\sim0}$	1	2	0	0	0	0	0	0	1	1	0	0	1	0	0	0	1	1	—	—	—	
MOVLB [XY], [RA]	$[XY],[XY+1] \leftarrow (RA)_{7\sim0}$	1	2	0	0	0	0	0	0	1	1	0	0	1	1	0	0	1	1	—	—	—	
MOVLB E:[HL], [RA]	$E:[HL],E:[HL+1] \leftarrow (RA)_{7\sim0}$	1	2	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	—	—	—	
MOVLB E:[XY], [RA]	$E:[XY],E:[XY+1] \leftarrow (RA)_{7\sim0}$	1	2	0	0	0	0	0	0	1	1	0	0	0	1	0	0	1	1	—	—	—	71
MOVLB [HL+], [RA]	$[HL],[HL+1] \leftarrow (RA)_{7\sim0}$, $HL \leftarrow HL+2$	1	2	0	0	0	0	0	0	1	1	0	0	1	0	1	0	1	1	—	—	❍	
MOVLB [XY+], [RA]	$[XY],[XY+1] \leftarrow (RA)_{7\sim0}$, $XY \leftarrow XY+2$	1	2	0	0	0	0	0	0	1	1	0	0	1	1	1	0	1	1	—	—	❍	
MOVLB E:[HL+], [RA]	$E:[HL],E:[HL+1] \leftarrow (RA)_{7\sim0}, HL \leftarrow HL+2$	1	2	0	0	0	0	0	0	1	1	0	0	0	0	1	0	1	1	—	—	❍	
MOVLB E:[XY+], [RA]	$E:[XY],E:[XY+1] \leftarrow (RA)_{7\sim0}, XY \leftarrow XY+2$	1	2	0	0	0	0	0	0	1	1	0	0	0	1	1	0	1	1	—	—	❍	
MOVLB [HL], cadr16	$[HL],[HL+1] \leftarrow (cadr16)_{7\sim0}$	2	3	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0	1	—	—	—	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVLB [XY], cadr16	$[XY],[XY+1] \leftarrow (cadr16)_{7\sim0}$	2	3	0	0	0	0	0	0	1	1	0	0	1	1	0	1	0	1	—	—	—	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVLB E:[HL], cadr16	$E:[HL],E:[HL+1] \leftarrow (cadr16)_{7\sim0}$	2	3	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	1	—	—	—	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVLB E:[XY], cadr16	$E:[XY],E:[XY+1] \leftarrow (cadr16)_{7\sim0}$	2	3	0	0	0	0	0	0	1	1	0	0	0	1	0	1	0	1	—	—	—	72
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVLB [HL+], cadr16	$[HL],[HL+1] \leftarrow (cadr16)_{7\sim0}, HL \leftarrow HL+2$	2	3	0	0	0	0	0	0	1	1	0	0	1	0	1	1	0	1	—	—	❍	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVLB [XY+], cadr16	$[XY],[XY+1] \leftarrow (cadr16)_{7\sim0}, XY \leftarrow XY+2$	2	3	0	0	0	0	0	0	1	1	0	0	1	1	1	1	0	1	—	—	❍	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVLB E:[HL+], cadr16	$E:[HL],E:[HL+1] \leftarrow (cadr16)_{7\sim0}, HL \leftarrow HL+2$	2	3	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	1	—	—	❍	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVLB E:[XY+], cadr16	$E:[XY],E:[XY+1] \leftarrow (cadr16)_{7\sim0}, XY \leftarrow XY+2$	2	3	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0	1	—	—	❍	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				

● **External Memory Transfer Instructions**

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
MOVXB [HL], [RA]	[HL],[HL+1]←(RA)	1	2	0	0	0	0	0	0	1	1	0	0	1	0	0	1	1	0	—	—	—	
MOVXB [XY], [RA]	[XY],[XY+1]←(RA)	1	2	0	0	0	0	0	0	1	1	0	0	1	1	0	1	1	0	—	—	—	
MOVXB E:[HL], [RA]	E:[HL],E:[HL+1]←(RA)	1	2	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	—	—	—	
MOVXB E:[XY], [RA]	E:[XY],E:[XY+1]←(RA)	1	2	0	0	0	0	0	0	1	1	0	0	0	1	0	1	1	0	—	—	—	73
MOVXB [HL+], [RA]	[HL],[HL+1]←(RA), HL←HL+2	1	2	0	0	0	0	0	0	1	1	0	0	1	0	1	1	1	0	—	—	❍	
MOVXB [XY+], [RA]	[XY],[XY+1]←(RA), XY←XY+2	1	2	0	0	0	0	0	0	1	1	0	0	1	1	1	1	1	0	—	—	❍	
MOVXB E:[HL+], [RA]	E:[HL],E:[HL+1]←(RA), HL←HL+2	1	2	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	0	—	—	❍	
MOVXB E:[XY+], [RA]	E:[XY],E:[XY+1]←(RA), XY←XY+2	1	2	0	0	0	0	0	0	1	1	0	0	0	1	1	1	1	0	—	—	❍	
MOVXB [RA], [HL]	(RA)←[HL],[HL+1]	1	3	0	0	0	0	0	0	1	1	0	0	1	0	0	1	1	1	—	—	—	
MOVXB [RA], [XY]	(RA)←[XY],[XY+1]	1	3	0	0	0	0	0	0	1	1	0	0	1	1	0	1	1	1	—	—	—	
MOVXB [RA], E:[HL]	(RA)←E:[HL],E:[HL+1]	1	3	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1	—	—	—	
MOVXB [RA], E:[XY]	(RA)←E:[XY],E:[XY+1]	1	3	0	0	0	0	0	0	1	1	0	0	0	1	0	1	1	1	—	—	—	75
MOVXB [RA], [HL+]	(RA)←[HL],[HL+1], HL←HL+2	1	3	0	0	0	0	0	0	1	1	0	0	1	0	1	1	1	1	—	—	❍	
MOVXB [RA], [XY+]	(RA)←[XY],[XY+1], XY←XY+2	1	3	0	0	0	0	0	0	1	1	0	0	1	1	1	1	1	1	—	—	❍	
MOVXB [RA], E:[HL+]	(RA)←E:[HL],E:[HL+1], HL←HL+2	1	3	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	—	—	❍	
MOVXB [RA], E:[XY+]	(RA)←E:[XY],E:[XY+1], XY←XY+2	1	3	0	0	0	0	0	0	1	1	0	0	0	1	1	1	1	1	—	—	❍	

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
MOVXB [HL], xadr16	[HL],[HL+1]←(xadr16)	2	3	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	—	—	—	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVXB [XY], xadr16	[XY],[XY+1]←(xadr16)	2	3	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	—	—	—	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVXB E:[HL], xadr16	E:[HL],E:[HL+1]← (xadr16)	2	3	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	—	—	—	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVXB E:[XY], xadr16	E:[XY],E:[XY+1]← (xadr16)	2	3	0	0	0	0	0	0	1	1	1	1	0	1	0	0	0	0	—	—	—	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				74
MOVXB [HL+], xadr16	[HL],[HL+1]←(xadr16), HL←HL+2	2	3	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0	—	—	❍	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVXB [XY+], xadr16	[XY],[XY+1]←(xadr16), XY←XY+2	2	3	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	—	—	❍	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVXB E:[HL+], xadr16	E:[HL],E:[HL+1]← (xadr16),HL←HL+2	2	3	0	0	0	0	0	0	1	1	1	1	0	0	1	0	0	0	—	—	❍	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVXB E:[XY+], xadr16	E:[XY],E:[XY+1]← (xadr16),XY←XY+2	2	3	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	—	—	❍	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVXB xadr16, [HL]	(xadr16)←[HL],[HL+1]	2	3	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1	—	—	—	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVXB xadr16, [XY]	(xadr16)←[XY],[XY+1]	2	3	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	1	—	—	—	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVXB xadr16, E:[HL]	(xadr16)←E:[HL],E: [HL+1]	2	3	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1	—	—	—	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVXB xadr16, E:[XY]	(xadr16)←E:[XY],E: [XY+1]	2	3	0	0	0	0	0	0	1	1	1	1	0	1	0	0	0	1	—	—	—	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				76
MOVXB xadr16, [HL+]	(xadr16)←[HL],[HL+1], HL←HL+2	2	3	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	1	—	—	❍	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVXB xadr16, [XY+]	(xadr16)←[XY],[XY+1], XY←XY+2	2	3	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	1	—	—	❍	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVXB xadr16, E:[HL+]	(xadr16)←E:[HL],E: [HL+1],HL←HL+2	2	3	0	0	0	0	0	0	1	1	1	1	0	0	1	0	0	1	—	—	❍	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
MOVXB xadr16, E:[XY+]	(xadr16)←E:[XY],E: [XY+1],XY←XY+2	2	3	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	1	—	—	❍	
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				

● **Stack Operation Instructions**

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
PUSH HL	(RSP)←{FLAG,A,HL}, RSP←RSP+1	1	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	—	—	—	85
PUSH XY	(RSP)←{CBR,EBR,XY}, RSP←RSP+1	1	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	—	—	—	86
POP HL	RSP←RSP−1, {FLAG,A,HL}←(RSP)	1	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	❍	❍	❍	84
POP XY	RSP←RSP−1, {CBR,EBR,XY}←(RSP)	1	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	—	—	—	85

● **Flag Operation Instructions**

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
FCLR G	G←0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	—	—	❍	53
FCLR C	C←0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	—	❍	—	52
FCLR Z	Z←0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	❍	—	—	53
FCLR FLAG	Z,C,G←0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	❍	—	—	52
FSET G	G←1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	—	—	❍	55
FSET C	C←1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	—	❍	—	54
FSET Z	Z←1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	❍	—	—	55
FSET FLAG	Z,C,G←0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	❍	❍	❍	54

● **Branch Instructions**

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
LJMP cadr16	PC←cadr16	2	2	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	—	—	—	60
				0	0	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
JMP cadr12	$PC_{11\sim0}$←cadr12	1	1	1	1	1	0	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	—	—	—	59
SJMP radr8	PC←Next PC+radr8	1	1	0	0	0	0	1	0	0	a_7	1	a_6	a_5	a_4	a_3	a_2	a_1	a_0	—	—	—	94
JMP PC+A	PC←PC+A+1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	—	—	—	59

● Conditional Branch Instructions

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
BC radr8 BLT radr8	if C=1 then PC←Next PC+radr8(<)	1	1	0	0	0	0	1	0	1	a_7	0	a_6	a_5	a_4	a_3	a_2	a_1	a_0	—	—	—	41
BNC radr8 BGE radr8	if C=0 then PC←Next PC+radr8(≧)	1	1	0	0	0	0	1	0	1	a_7	1	a_6	a_5	a_4	a_3	a_2	a_1	a_0	—	—	—	
BZ radr8 BEQ radr8	if Z=1 then PC←Next PC+radr8(=)	1	1	0	0	0	0	1	1	0	a_7	0	a_6	a_5	a_4	a_3	a_2	a_1	a_0	—	—	—	
BNZ radr8 BNE radr8	if Z=0 then PC←Next PC+radr8(≠)	1	1	0	0	0	0	1	1	0	a_7	1	a_6	a_5	a_4	a_3	a_2	a_1	a_0	—	—	—	
BLE radr8	if (C=1)∨(Z=1) then PC←Next PC+radr8(≦)	1	1	0	0	0	0	1	1	1	a_7	0	a_6	a_5	a_4	a_3	a_2	a_1	a_0	—	—	—	
BGT radr8	if (C=0)∧(Z=0) then PC←Next PC+radr8(>)	1	1	0	0	0	0	1	1	1	a_7	1	a_6	a_5	a_4	a_3	a_2	a_1	a_0	—	—	—	
BNG radr8	if G=0 then PC←Next PC+radr8	1	1	0	0	0	0	1	0	0	a_7	0	a_6	a_5	a_4	a_3	a_2	a_1	a_0	—	—	—	

● Call/return Instructions

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
LCAL cadr16	(SP)←PC,PC←cadr16, SP←SP+1	2	2	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	—	—	—	60
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
CAL cadr12	(SP)←PC,$PC_{11\sim0}$← cadr12,SP←SP+1	1	1	1	1	1	1	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	—	—	—	47
RT	PC←(SP)+1,SP←SP−1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	—	—	—	89
RTI	PC←(SP)+1,SP←SP−1, MIE←1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	—	—	—	89
RTNMI	PC←(SP)+1,SP←SP−1 MIE←Pre-interrupt MIE state	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	—	—	—	90

● **Control Instructions**

MNEMONIC	OPERATION	W	C	INSTRUCTION CODE																FLAG			PAGE
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Z	C	G	
NOP	NO OPERATION	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	—	—	—	81
HALT	HALT CPU	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	—	—	—	56
EI	MIE←1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	—	—	—	51
DI	MIE←0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	—	—	—	51
INCB HL	HL←HL+1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	—	—	❍	58
INCB XY	XY←XY+1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	—	—	❍	58
INCW RA	RA←RA+1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	—	—	❍	58
MOV CBR,#i4	CBR←i4	1	1	0	0	0	0	0	0	0	0	0	0	1	1	i_3	i_2	i_1	i_0	—	—	—	68
MOV EBR,#i4	EBR←i4	1	1	0	0	0	0	0	0	0	0	0	0	1	0	i_3	i_2	i_1	i_0	—	—	—	68
MOV RA0,#i4	RA0←i4	1	1	0	0	0	0	0	0	1	0	0	0	0	0	i_3	i_2	i_1	i_0	—	—	—	68
MOV RA1,#i4	RA1←i4	1	1	0	0	0	0	0	0	1	0	0	0	0	1	i_3	i_2	i_1	i_0	—	—	—	68
MOV RA2,#i4	RA2←i4	1	1	0	0	0	0	0	0	1	0	0	0	1	0	i_3	i_2	i_1	i_0	—	—	—	68
MOV RA3,#i4	RA3←i4	1	1	0	0	0	0	0	0	1	0	0	0	1	1	i_3	i_2	i_1	i_0	—	—	—	68
MOV H,#i4	H←i4	1	1	0	0	0	0	0	0	0	1	0	0	1	1	i_3	i_2	i_1	i_0	—	—	—	68
MOV L,#i4	L←i4	1	1	0	0	0	0	0	0	0	1	0	0	1	0	i_3	i_2	i_1	i_0	—	—	—	68
MOV X,#i4	X←i4	1	1	0	0	0	0	0	0	0	1	0	0	0	1	i_3	i_2	i_1	i_0	—	—	—	68
MOV Y,#i4	Y←i4	1	1	0	0	0	0	0	0	0	1	0	0	0	0	i_3	i_2	i_1	i_0	—	—	—	68
MSA cadr16	Melody output starts	2	3	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	—	—	—	77
				a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				

4. INSTRUCTION DESCRIPTIONS

Instructions are detailed below. Refer to the following guides for information.

obj

Indicates items that can be described in the instruction operand

Function

Indicates the function using symbols

Description

Describes the instruction and gives precautions in use

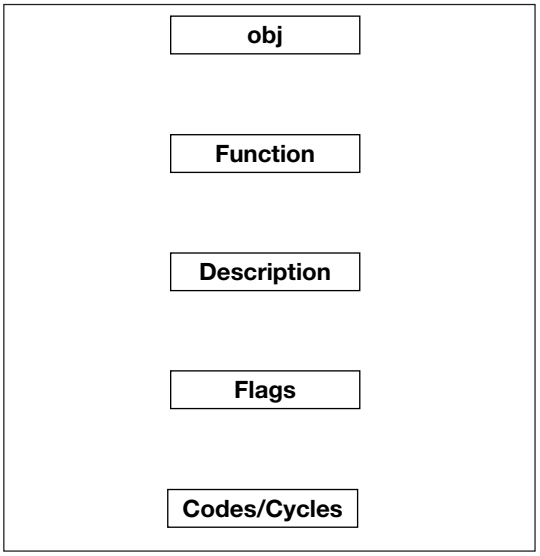
Flags

Indicates flags affected by execution of the instruction

Codes/Cycles

Indicates the instruction operand, instruction code and machine cycle

<Item name>



ADC obj, A

(add data memory and accumulator with carry)

obj

For no post-increment: sfr, ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← obj + A + C
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The data memory specified by obj is added to the accumulator and the carry flag, and the results stored to data memory and the accumulator. For post-incremented operations, the HL or XY register is then incremented.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj + A + C = 0
then Z ← 1
else Z ← 0
if obj + A + C > 0FH
then C ← 1
else C ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
sfr, A	0	0	1	0	0	1	0	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
¥cur, A	0	0	1	1	0	1	0	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], A	0	0	0	0	0	1	0	1	0	0	1	0	0	0	1	1	1
[XY], A	0	0	0	0	0	1	0	1	0	0	1	1	0	0	1	1	1
E:[HL], A	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1
E:[XY], A	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	1
[HL+], A	0	0	0	0	0	1	0	1	0	0	1	0	1	0	1	1	1
[XY+], A	0	0	0	0	0	1	0	1	0	0	1	1	1	0	1	1	1
E:[HL+], A	0	0	0	0	0	1	0	1	0	0	0	0	1	0	1	1	1
E:[XY+], A	0	0	0	0	0	1	0	1	0	0	0	1	1	0	1	1	1

ADCD obj, A

(add data memory and accumulator decimal adjustment, with carry)

obj

For no post-increment: sfr, ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← decimal adjust {obj + A + C}
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The data memory specified by obj is added to the accumulator and the carry flag, and the results stored to data memory and the accumulator if 9 or less. If the results are greater than 9, the decimal adjustment 6H is added and the results stored to data memory and the accumulator. For post-incremented operations, the HL or XY register is then incremented.

Flags

Flags affected by execution of this instruction
Flag change conditions
if decimal adjustment = 0
then Z ← 1
else Z ← 0
if overflow results from operation
C ← 1 else C ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
sfr, A	0	0	1	0	0	1	1	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
¥cur, A	0	0	1	1	0	1	1	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], A	0	0	0	0	0	1	0	1	0	0	1	0	0	1	0	0	1
[XY], A	0	0	0	0	0	1	0	1	0	0	1	1	0	1	0	0	1
E:[HL], A	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	1
E:[XY], A	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1
[HL+], A	0	0	0	0	0	1	0	1	0	0	1	0	1	1	0	0	1
[XY+], A	0	0	0	0	0	1	0	1	0	0	1	1	1	1	0	0	1
E:[HL+], A	0	0	0	0	0	1	0	1	0	0	0	0	1	1	0	0	1
E:[XY+], A	0	0	0	0	0	1	0	1	0	0	0	1	1	1	0	0	1

ADCJ obj, n

(data memory base-n adjustment addition, with carry)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A← n adjust {obj + C} (n is even number from 2 to 16)
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The data memory specified by obj is added to the carry flag, the base-n results adjusted, and the results stored to data memory and the accumulator. If the results of base-n adjustment are greater than n, the adjustment (two's complement of n) is added. n is an even number from 2 to 16. For post-incremented operations, the HL or XY register is then incremented.

Flags

Flags affected by execution of this instruction
Flag change conditions
if base-n adjustment = 0
then Z ← 1
else Z ← 0
if overflow results from adjustment, or if n = 16
C ← 1 else C ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur, n	0	0	0	1	0	n_2	n_1	n_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], n	0	0	0	0	0	1	1	0	0	0	1	0	0	n_2	n_1	n_0	1
[XY], n	0	0	0	0	0	1	1	0	0	0	1	1	0	n_2	n_1	n_0	1
E:[HL], n	0	0	0	0	0	1	1	0	0	0	0	0	0	n_2	n_1	n_0	1
E:[XY], n	0	0	0	0	0	1	1	0	0	0	0	1	0	n_2	n_1	n_0	1
[HL+], n	0	0	0	0	0	1	1	1	0	0	1	0	0	n_2	n_1	n_0	1
[XY+], n	0	0	0	0	0	1	1	1	0	0	1	1	0	n_2	n_1	n_0	1
E:[HL+], n	0	0	0	0	0	1	1	1	0	0	0	0	0	n_2	n_1	n_0	1
E:[XY+], n	0	0	0	0	0	1	1	1	0	0	0	1	0	n_2	n_1	n_0	1

[Note] The adjusted value n and the values of n2 to n0 in the instruction code are as indicated below:

Adjusted value n	2	4	6	8	10	12	14	16
Values of n_2 to n_0 in instruction code	1H	2H	3H	4H	5H	6H	7H	0H

ADD obj, A

(add data memory and accumulator)

obj

For no post-increment: sfr, ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← obj + A
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The data memory specified by obj is added to the accumulator, and the results stored to data memory and the accumulator. For post-incremented operations, the HL or XY register is then incremented.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj + A = 0
then Z ← 1
else Z ← 0
if obj + A > 0FH
then C ← 1
else C ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
sfr, A	0	0	1	0	0	1	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
¥cur, A	0	0	1	1	0	1	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], A	0	0	0	0	0	1	0	1	0	0	1	0	0	0	1	0	1
[XY], A	0	0	0	0	0	1	0	1	0	0	1	1	0	0	1	0	1
E:[HL], A	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	1
E:[XY], A	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	0	1
[HL+], A	0	0	0	0	0	1	0	1	0	0	1	0	1	0	1	0	1
[XY+], A	0	0	0	0	0	1	0	1	0	0	1	1	1	0	1	0	1
E:[HL+], A	0	0	0	0	0	1	0	1	0	0	0	0	1	0	1	0	1
E:[XY+], A	0	0	0	0	0	1	0	1	0	0	0	1	1	0	1	0	1

ADD obj, #i4

(add data memory and immediate data)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← obj + i4
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The data memory specified by obj is added to immediate data i4, and the results stored to data memory and the accumulator. For post-incremented operations, the HL or XY register is then incremented.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj + i4 = 0
then Z ← 1
else Z ← 0
if obj + i4 > 0FH
then C ← 1
else C ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur, #i4	1	0	0	0	i_3	i_2	i_1	i_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], #i4	0	0	0	0	0	0	0	0	1	0	1	0	i_3	i_2	i_1	i_0	1
[XY], #i4	0	0	0	0	0	0	0	0	1	0	1	1	i_3	i_2	i_1	i_0	1
E:[HL], #i4	0	0	0	0	0	0	0	0	1	0	0	0	i_3	i_2	i_1	i_0	1
E:[XY], #i4	0	0	0	0	0	0	0	0	1	0	0	1	i_3	i_2	i_1	i_0	1
[HL+], #i4	0	0	0	0	0	0	0	1	1	0	1	0	i_3	i_2	i_1	i_0	1
[XY+], #i4	0	0	0	0	0	0	0	1	1	0	1	1	i_3	i_2	i_1	i_0	1
E:[HL+], #i4	0	0	0	0	0	0	0	1	1	0	0	0	i_3	i_2	i_1	i_0	1
E:[XY+], #i4	0	0	0	0	0	0	0	1	1	0	0	1	i_3	i_2	i_1	i_0	1

AND obj, A

(logical product of data memory and accumulator)

obj

For no post-increment: sfr, ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← obj ^ A
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The logical product of the data memory specified by obj and the accumulator is taken, and the results stored to data memory and the accumulator. For post-incremented operations, the HL or XY register is then incremented.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj ^ A = 0
then Z ← 1
else Z ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
sfr, A	0	0	1	0	1	0	1	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
¥cur, A	0	0	1	1	1	0	1	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], A	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	1
[XY], A	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1
E:[HL], A	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1
E:[XY], A	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	1
[HL+], A	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	1	1
[XY+], A	0	0	0	0	0	1	0	0	0	0	1	1	1	0	0	1	1
E:[HL+], A	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	1
E:[XY+], A	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	1	1

AND obj, #i4

(logical product of data memory and immediate data)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← obj ^ i4
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The logical product of the data memory specified by obj and the immediate data is taken, and the results stored to data memory and the accumulator. For post-incremented operations, the HL or XY register is then incremented.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj ^ i4 = 0
then Z ← 1
else Z ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur, #i4	0	1	0	1	i_3	i_2	i_1	i_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], #i4	0	0	0	0	0	1	0	0	0	1	1	0	i_3	i_2	i_1	i_0	1
[XY], #i4	0	0	0	0	0	1	0	0	0	1	1	1	i_3	i_2	i_1	i_0	1
E:[HL], #i4	0	0	0	0	0	1	0	0	0	1	0	0	i_3	i_2	i_1	i_0	1
E:[XY], #i4	0	0	0	0	0	1	0	0	0	1	0	1	i_3	i_2	i_1	i_0	1
[HL+], #i4	0	0	0	0	0	1	0	1	0	1	1	0	i_3	i_2	i_1	i_0	1
[XY+], #i4	0	0	0	0	0	1	0	1	0	1	1	1	i_3	i_2	i_1	i_0	1
E:[HL+], #i4	0	0	0	0	0	1	0	1	0	1	0	0	i_3	i_2	i_1	i_0	1
E:[XY+], #i4	0	0	0	0	0	1	0	1	0	1	0	1	i_3	i_2	i_1	i_0	1

Bcond radr8

(conditional branch)

Function

if cond is true,
then PC ← Next PC + radr8
else PC ← PC + 1
(however,-128 ≤ radr ≤ +127
Next PC indicates the address following this instruction (PC+1))

Instruction description	Condition
BC or BLT	C = 1
BNC or BGE	C = 0
BLE	C = 1 ∨ Z = 1
BGT	C = 0 ∧ Z = 0
BNG	G = 0
BZ or BEQ	Z = 1
BNZ or BNE	Z = 0

Description

If the condition is true, the content of radr8 is added to Next PC and program execution branches. If the condition is false the content of PC is incremented by one, and the next instruction is executed.
The branch destination address is a range of -128 to +127 from the Next PC address. The 8 bits in the instruction code (a_7 to a_0) correspond to radr8, with the 8th bit (a_7) used as a sign. It indicates the relative displacement from the address immediately after the instruction.
Branches across program memory space page boundaries are possible.
In Assembler it is possible to directly specify an address within the branch range (label) instead of radr8.

Flags

Flags affected by execution of this instruction

Z	C	G
—	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
BC or BLT	0	0	0	0	1	0	1	a_7	0	a_6	a_5	a_4	a_3	a_2	a_1	a_0	1
BNC or BGE	0	0	0	0	1	0	1	a_7	1	a_6	a_5	a_4	a_3	a_2	a_1	a_0	1
BZ or BEQ	0	0	0	0	1	1	0	a_7	0	a_6	a_5	a_4	a_3	a_2	a_1	a_0	1
BNZ or BNE	0	0	0	0	1	1	0	a_7	1	a_6	a_5	a_4	a_3	a_2	a_1	a_0	1
BLE	0	0	0	0	1	1	1	a_7	0	a_6	a_5	a_4	a_3	a_2	a_1	a_0	1
BGT	0	0	0	0	1	1	1	a_7	1	a_6	a_5	a_4	a_3	a_2	a_1	a_0	1
BNG	0	0	0	0	1	0	0	a_7	0	a_6	a_5	a_4	a_3	a_2	a_1	a_0	1

BCLR obj. n

(data memory bit clear)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj. n ← 0, A ← obj (n = 0 ~ 3)
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

n bits of the data memory specified by obj are cleared. After the clear operation, the content of obj is stored to the accumulator. The value of n is 0 to 3, indicating the positions of the bits to be cleared. For post-incremented operations, the HL or XY register is then incremented.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj. n ← 0 then obj = 0
then Z ← 1
else Z ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur, A	0	1	0	1	n_3	n_2	n_1	n_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], A	0	0	0	0	0	1	0	0	0	1	1	0	n_3	n_2	n_1	n_0	1
[XY], A	0	0	0	0	0	1	0	0	0	1	1	1	n_3	n_2	n_1	n_0	1
E:[HL], A	0	0	0	0	0	1	0	0	0	1	0	0	n_3	n_2	n_1	n_0	1
E:[XY], A	0	0	0	0	0	1	0	0	0	1	0	1	n_3	n_2	n_1	n_0	1
[HL+], A	0	0	0	0	0	1	0	1	0	1	1	0	n_3	n_2	n_1	n_0	1
[XY+], A	0	0	0	0	0	1	0	1	0	1	1	1	n_3	n_2	n_1	n_0	1
E:[HL+], A	0	0	0	0	0	1	0	1	0	1	0	0	n_3	n_2	n_1	n_0	1
E:[XY+], A	0	0	0	0	0	1	0	1	0	1	0	1	n_3	n_2	n_1	n_0	1

[Note] The relation between n(0-3) in the operand and n_3-n_0 in the instruction code is shown below.

Operand value for n	0	1	2	3
n_3-n_0 values in instruction code	0EH	0DH	0BH	7H

BMOV obj. n, A. n

(transfer bit to data memory)

obj

[HL], [XY], E:[HL], E:[XY]

Function

obj.n ← A.n, A ← obj (n = 0 ~ 3)

Description

n bits of the accumulator are transferred to bit n of the data memory specified by obj. After the operation, the content of data memory is stored to the accumulator. The value of n is 0 to 3, indicating the positions of the bits to be cleared.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj.n ← A.n then obj = 0
then Z ← 1
else Z ← 0

Z	C	G
❍	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
[HL]. n, A. n	0	0	0	0	0	n_1	n_0	0	1	1	1	0	n_3	n_2	n_1	n_0	1
[XY]. n, A. n	0	0	0	0	0	n_1	n_0	0	1	1	1	1	n_3	n_2	n_1	n_0	1
E:[HL]. n, A. n	0	0	0	0	0	n_1	n_0	0	1	1	0	0	n_3	n_2	n_1	n_0	1
E:[XY]. n, A. n	0	0	0	0	0	n_1	n_0	0	1	1	0	1	n_3	n_2	n_1	n_0	1

[Note] The value n used to specify the accumulator bits is represented by n_1 and n_0 in bits 10 and 9 of the instruction code, as shown below.

Operand value for n	0	1	2	3
n_1, n_0 values in instruction code	0H	1H	2H	3H

The relation between the data memory bit specification n and n_3-n_0 in the instruction code is shown below.

Data memory value for n	0	1	2	3
n_3-n_0 values in instruction code	1H	2H	4H	8H

[Note] Only the nX-4/300 core has BMOV instruction.

BNOT obj. n

(data memory bit reversal)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj. n ← $\overline{\text{obj. n}}$, A ← obj (n = 0 ~ 3)
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

n bits of the data memory specified by obj are reversed. After the operation, the content of obj is stored to the accumulator. The value of n is 0 to 3, indicating the positions of the bits to be reversed. For post-incremented operations, the HL or XY register is then incremented.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj. n ← $\overline{\text{obj. n}}$ then obj = 0
then Z ← 1
else Z ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur, A	0	1	1	1	n_3	n_2	n_1	n_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], A	0	0	0	0	0	0	0	0	0	1	1	0	n_3	n_2	n_1	n_0	1
[XY], A	0	0	0	0	0	0	0	0	0	1	1	1	n_3	n_2	n_1	n_0	1
E:[HL], A	0	0	0	0	0	0	0	0	0	1	0	0	n_3	n_2	n_1	n_0	1
E:[XY], A	0	0	0	0	0	0	0	0	0	1	0	1	n_3	n_2	n_1	n_0	1
[HL+], A	0	0	0	0	0	0	0	1	0	1	1	0	n_3	n_2	n_1	n_0	1
[XY+], A	0	0	0	0	0	0	0	1	0	1	1	1	n_3	n_2	n_1	n_0	1
E:[HL+], A	0	0	0	0	0	0	0	1	0	1	0	0	n_3	n_2	n_1	n_0	1
E:[XY+], A	0	0	0	0	0	0	0	1	0	1	0	1	n_3	n_2	n_1	n_0	1

[Note] The relation between n(0-3) in the operand and n_3-n_0 in the instruction code is shown below.

Operand value for n	0	1	2	3
n_3-n_0 values in instruction code	1H	2H	4H	8H

BSET obj. n

(data memory bit set)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj. n ← 1, A ← obj (n = 0 ~ 3)
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

n bits of the data memory specified by obj are set. After the operation, the content of obj is stored to the accumulator. The value of n is 0 to 3, indicating the positions of the bits to be set. For post-incremented operations, the HL or XY register is then incremented.

Flags

Flags affected by execution of this instruction
Flag change conditions
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
—	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur. n	0	1	1	0	n_3	n_2	n_1	n_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL]. n	0	0	0	0	0	0	1	0	0	1	1	0	n_3	n_2	n_1	n_0	1
[XY]. n	0	0	0	0	0	0	1	0	0	1	1	1	n_3	n_2	n_1	n_0	1
E:[HL]. n	0	0	0	0	0	0	1	0	0	1	0	0	n_3	n_2	n_1	n_0	1
E:[XY]. n	0	0	0	0	0	0	1	0	0	1	0	1	n_3	n_2	n_1	n_0	1
[HL+]. n	0	0	0	0	0	0	1	1	0	1	1	0	n_3	n_2	n_1	n_0	1
[XY+]. n	0	0	0	0	0	0	1	1	0	1	1	1	n_3	n_2	n_1	n_0	1
E:[HL+]. n	0	0	0	0	0	0	1	1	0	1	0	0	n_3	n_2	n_1	n_0	1
E:[XY+]. n	0	0	0	0	0	0	1	1	0	1	0	1	n_3	n_2	n_1	n_0	1

[Note] The relation between n(0-3) in the operand and n_3-n_0 in the instruction code is shown below.

Operand value for n	0	1	2	3
n_3-n_0 values in instruction code	1H	2H	4H	8H

BTST obj. n

(data memory bit test)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

$Z \leftarrow \overline{obj.\ n}$, (n = 0 ~ 3)
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

n bits of the data memory specified by obj are tested, and if one then Z ← 0, and if 0 then Z ← 1. The value of n is 0 to 3, indicating the positions of the bits to be tested. For post-incremented operations, the HL or XY register is then incremented.

Flags

Flags affected by execution of this instruction
See function column
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur, A	1	0	1	1	n_3	n_2	n_1	n_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], A	0	0	0	0	0	1	0	0	1	0	1	0	n_3	n_2	n_1	n_0	1
[XY], A	0	0	0	0	0	1	0	0	1	0	1	1	n_3	n_2	n_1	n_0	1
E:[HL], A	0	0	0	0	0	1	0	0	1	0	0	0	n_3	n_2	n_1	n_0	1
E:[XY], A	0	0	0	0	0	1	0	0	1	0	0	1	n_3	n_2	n_1	n_0	1
[HL+], A	0	0	0	0	0	1	0	1	1	0	1	0	n_3	n_2	n_1	n_0	1
[XY+], A	0	0	0	0	0	1	0	1	1	0	1	1	n_3	n_2	n_1	n_0	1
E:[HL+], A	0	0	0	0	0	1	0	1	1	0	0	0	n_3	n_2	n_1	n_0	1
E:[XY+], A	0	0	0	0	0	1	0	1	1	0	0	1	n_3	n_2	n_1	n_0	1

[Note] The relation between n(0-3) in the operand and n_3-n_0 in the instruction code is shown below.

Operand value for n	0	1	2	3
n_3-n_0 values in instruction code	1H	2H	4H	8H

CAL cadr12

(subroutine call)

obj

(SP) ← PC, $PC_{11\sim0}$ ← cadr12, SP ← SP + 1

Description

The contents of the program counter (PC) are saved to the call stack, and the subroutine called. The call stack pointer is then incremented. The subroutine address is a 12-bit immediate value specifying an absolute address in the program memory space page (4K word). In other words, the content does not change between the 15-bit PC value and the 12-bit specification. cadr12 specifies the called address.

Flags

Flags affected by execution of this instruction

Z	C	G
—	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
cadr12	1	1	1	1	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	1

CMP obj, A

(comparison of data memory and accumulator)

obj

For no post-increment: sfr, ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj - A
if obj = A then Z ← 1, C ← 0
if obj > A then Z ← 0, C ← 0
if obj < A then Z ← 0, C ← 1
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The content of the data memory specified by obj are compared to the contents of the accumulator, and the results reflected in a flag. For post-incremented operations, the HL or XY register is then incremented.

Flags

Flags affected by execution of this instruction
Refer to function column
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
sfr, A	0	0	1	0	1	0	1	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
¥cur, A	0	0	1	1	1	0	1	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], A	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	1
[XY], A	0	0	0	0	0	1	0	0	0	0	1	1	0	1	0	0	1
E:[HL], A	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1
E:[XY], A	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	1
[HL+], A	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	0	1
[XY+], A	0	0	0	0	0	1	0	0	0	0	1	1	1	1	0	0	1
E:[HL+], A	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	1
E:[XY+], A	0	0	0	0	0	1	0	0	0	0	0	1	1	1	0	0	1

CMP obj, #i4

(comparison of data memory and immediate data)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj - i4
if obj = i4 then Z ← 1, C ← 0
if obj > i4 then Z ← 0, C ← 0
if obj < i4 then Z ← 0, C ← 1
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The contents of the data memory specified by obj are compared to the immediate data, and the results reflected in a flag. For post-incremented operations, the HL or XY register is then incremented.

Flags

Flags affected by execution of this instruction
Refer to function column
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur, #i4	1	0	1	0	i_3	i_2	i_1	i_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], #i4	0	0	0	0	0	1	1	0	1	0	1	0	i_3	i_2	i_1	i_0	1
[XY], #i4	0	0	0	0	0	1	1	0	1	0	1	1	i_3	i_2	i_1	i_0	1
E:[HL], #i4	0	0	0	0	0	1	1	0	1	0	0	0	i_3	i_2	i_1	i_0	1
E:[XY], #i4	0	0	0	0	0	1	1	0	1	0	0	1	i_3	i_2	i_1	i_0	1
[HL+], #i4	0	0	0	0	0	1	1	1	1	0	1	0	i_3	i_2	i_1	i_0	1
[XY+], #i4	0	0	0	0	0	1	1	1	1	0	1	1	i_3	i_2	i_1	i_0	1
E:[HL+], #i4	0	0	0	0	0	1	1	1	1	0	0	0	i_3	i_2	i_1	i_0	1
E:[XY+], #i4	0	0	0	0	0	1	1	1	1	0	0	1	i_3	i_2	i_1	i_0	1

DEC obj

(decrement)

obj

For no post-increment: sfr, ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← obj - 1
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The content of the data memory specified by obj is decremented and the results stored to the accumulator. For post-incremented operations, the HL or XY register is then incremented.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj - 1 = 0FH
then C ← 1
else C ← 0
if obj - 1 = 0
then Z ← 1
else Z ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
sfr	0	0	1	0	0	0	0	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
¥cur	0	0	1	1	0	0	0	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL]	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	1	1
[XY]	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	1	1
E:[HL]	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	1
E:[XY]	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	1	1
[HL+]	0	0	0	0	0	1	0	1	0	0	1	0	1	0	0	1	1
[XY+]	0	0	0	0	0	1	0	1	0	0	1	1	1	0	0	1	1
E:[HL+]	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	1
E:[XY+]	0	0	0	0	0	1	0	1	0	0	0	1	1	0	0	1	1

DI

(MIE flag clear)

Function

MIE ← 0

Description

Clears the master interrupt enable flag.

Flags

Flags affected by execution of this instruction

Z	C	G
—	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
DI	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

EI

(MIE flag set)

Function

MIE ← 1

Description

Sets the master interrupt enable flag.

Flags

Flags affected by execution of this instruction

Z	C	G
—	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
EI	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1

FCLR C

(carry flag clear)

Function

C ← 0

Description

Clears the carry flag.

Flags

Flags affected by execution of this instruction

Z	C	G
—	❍	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

FCLR FLAG

(zero flag, csrry flag and G flag clear)

Function

Z ← 0, C ← 0, G ← 0

Description

Clears the zero flag, carry flag and G flag.

Flags

Flags affected by execution of this instruction

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
FLAG	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

[Note] Only the nX-4/300 core has FCLR FLAG instruction.

FCLR G

(G flag clear)

Function

G ← 0

Description

Clears the G flag.

Flags

Flags affected by execution of this instruction

Z	C	G
—	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

FCLR Z

(zero flag clear)

Function

Z ← 0

Description

Clears the zero flag.

Flags

Flags affected by execution of this instruction

Z	C	G
❍	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Z	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1

FSET C

(set carry flag)

Function

C ← 1

Description

Sets the carry flag.

Flags

Flags affected by execution of this instruction

Z	C	G
—	❍	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
C	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

FSET FLAG

(zero flag, carry flag and G flag set)

Function

Z ← 1, C ← 1, G ← 1

Description

Sets the zero flag, carry flag and G flag.

Flags

Flags affected by execution of this instruction

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
FLAG	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1

[Note] Only the nX-4/300 core has FSET FLAG instruction.

FSET G

(set G flag)

Function

G ← 1

Description

Sets the G flag.

Flags

Flags affected by execution of this instruction

Z	C	G
—	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
G	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1

FSET Z

(set zero flag)

Function

Z ← 1

Description

Sets the zero flag.

Flags

Flags affected by execution of this instruction

Z	C	G
❍	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Z	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1

HALT

(CPU halt mode)

Function

Halt CPU

Description

Switches the CPU to the halt mode.
[Precautions]
Always use an NOP instruction immediately after the HALT instruction.
Example

```
•
•
•
HALT
NOP
•
•
•
```

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
HALT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

INC obj

(increment)

obj

For no post-increment: sfr, ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← obj + 1
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The content of the data memory specified by obj is incremented, and the results stored to the accumulator. For post-incremented operations, the HL or XY register is then incremented.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj + 1 = 0
then C ← 1, Z ← 1
else C ← 0, Z ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
sfr	0	0	1	0	0	0	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
¥cur	0	0	1	1	0	0	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL]	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	1
[XY]	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0	1
E:[HL]	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1
E:[XY]	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	1
[HL+]	0	0	0	0	0	1	0	1	0	0	1	0	1	0	0	0	1
[XY+]	0	0	0	0	0	1	0	1	0	0	1	1	1	0	0	0	1
E:[HL+]	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	1
E:[XY+]	0	0	0	0	0	1	0	1	0	0	0	1	1	0	0	0	1

INCB obj

(HL, XY pair register increment)

obj

HL, XY

Function

obj ← obj + 1 (byte increment)

Description

Increments the HL or XY registers in byte units.

Flags

Flags affected by execution of this instruction
Flag change conditions
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
—	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
HL	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1
XY	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1

INCW RA

(RA register increment)

obj

RA ← RA + 1 (word increment)

Description

Increments the RA register in word units.

Flags

Flags affected by execution of this instruction
Flag change conditions
if RA + 1 = 0
then G ← 1
else G ← 0

Z	C	G
—	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
RA	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1

JMP cadr12

(branch)

Function

$PC_{11\sim0} \leftarrow$ cadr12

Description

12-bit immediate data is used to specify the program memory space page region (4K word), and branch to an absolute address. There is no change in data from 15-bit PC to 12-bit.
cadr12 specifies the called address.

Flags

Flags affected by execution of this instruction

Z	C	G
—	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
cadr12	1	1	1	0	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	1

JMP PC+A

(branch as per content of accumulator)

Function

PC $\leftarrow$ PC +A+ 1

Description

The content of the accumulator is added to the program counter (PC), one added, and the result stored to the PC.
Branches are possible across program memory space page boundaries.

Flags

Flags affected by execution of this instruction

Z	C	G
—	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
PC+A	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1

LCAL cadr16

(call subroutine)

Function

(SP) ← PC, PC ← cadr16, SP ← SP + 1

Description

The contents of the second word of the program counter (PC) are saved to the call stack, and the subroutine called. The call stack pointer is then incremented. The subroutine address is a 16-bit immediate value specifying an absolute address in the program memory space (64K word).
cadr16 specifies the called address.

Flags

Flags affected by execution of this instruction

Z	C	G
—	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
cadr16	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	2
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	

LJMP cadr16

(branch)

Function

PC ← cadr16

Description

16-bit immediate data specifies an absolute address in the program memory space (64K word), and the program branches to that address.
cadr16 specifies the called address.

Flags

Flags affected by execution of this instruction

Z	C	G
—	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
cadr16	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	2
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	

MCLR obj, #m

(data memory bit clear)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

Bits not masked by obj #m ← 0, A ← obj (m = 0 ~ 0FH)
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

Clears all bits not masked by immediate data m, in the data memory specified by obj. After the clear operation the content of obj is stored to the accumulator. m is a value in the range 0 to 0FH, with a "1" masking a bit and a "0" marking a bit to be cleared. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj = 0
then Z ← 1
else Z ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur. #m	0	1	0	1	m_3	m_2	m_1	m_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL]. #m	0	0	0	0	0	1	0	0	0	1	1	0	m_3	m_2	m_1	m_0	1
[XY]. #m	0	0	0	0	0	1	0	0	0	1	1	1	m_3	m_2	m_1	m_0	1
E:[HL]. #m	0	0	0	0	0	1	0	0	0	1	0	0	m_3	m_2	m_1	m_0	1
E:[XY]. #m	0	0	0	0	0	1	0	0	0	1	0	1	m_3	m_2	m_1	m_0	1
[HL+]. #m	0	0	0	0	0	1	0	1	0	1	1	0	m_3	m_2	m_1	m_0	1
[XY+]. #m	0	0	0	0	0	1	0	1	0	1	1	1	m_3	m_2	m_1	m_0	1
E:[HL+]. #m	0	0	0	0	0	1	0	1	0	1	0	0	m_3	m_2	m_1	m_0	1
E:[XY+]. #m	0	0	0	0	0	1	0	1	0	1	0	1	m_3	m_2	m_1	m_0	1

MMOV obj, #m, A.n

(bit transfer to data memory)

obj

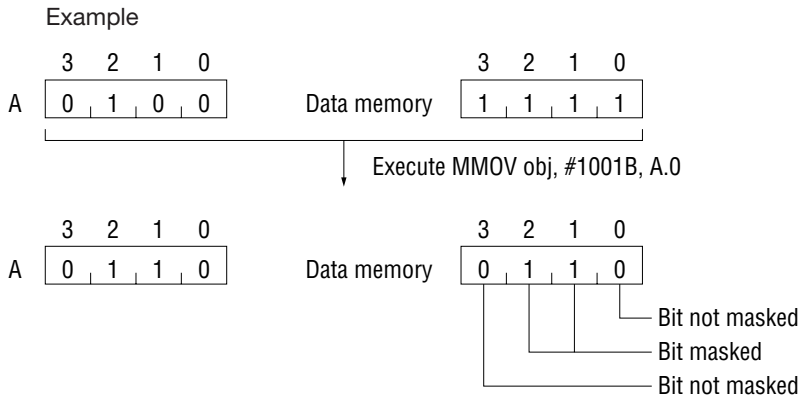
[HL], [XY], E:[HL], E:[XY]

Function

Bits not masked by obj #m ← A. n, A ← obj (m = 0 ~ 0FH, n = 0 ~ 3)

Description

Transfers bits specified by n from accumulator to bits of the data memory specified by obj which are not masked by m. After the transfer, the data memory is stored to the accumulator.
m is a value from 0 to 0FH, which a "0" masking a bit and a "1" masking a bit for transfer. n is a value 0 to 3, and marks the accumulator bit position.



Flags

Flags affected by execution of this instruction
Flag change conditions
if obj = 0
then Z ← 1
else Z ← 0

Z	C	G
❍	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
[HL], #m, A. n	0	0	0	0	0	n_1	n_0	0	1	1	1	0	m_3	m_2	m_1	m_0	1
[XY], #m, A. n	0	0	0	0	0	n_1	n_0	0	1	1	1	1	m_3	m_2	m_1	m_0	1
E:[HL], #m, A. n	0	0	0	0	0	n_1	n_0	0	1	1	0	0	m_3	m_2	m_1	m_0	1
E:[XY], #m, A. n	0	0	0	0	0	n_1	n_0	0	1	1	0	1	m_3	m_2	m_1	m_0	1

[Note] Only the nX-4/300 core has MMOV instruction.

MNOT obj, #m

(data memory bit reversal)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

Reverses bits not masked by obj #m
A ← obj (m = 0 ~ 0FH)
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

Reverses all bits not masked by m in the data memory specified by obj. After the clear operation the content of obj is stored to the accumulator. m is a value in the range 0 to 0FH, with a "0" masking a bit and a "1" marking a bit to be reversed. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj = 0
then Z ← 1
else Z ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur. #m	0	1	1	1	m_3	m_2	m_1	m_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL]. #m	0	0	0	0	0	0	0	0	0	1	1	0	m_3	m_2	m_1	m_0	1
[XY]. #m	0	0	0	0	0	0	0	0	0	1	1	1	m_3	m_2	m_1	m_0	1
E:[HL]. #m	0	0	0	0	0	0	0	0	0	1	0	0	m_3	m_2	m_1	m_0	1
E:[XY]. #m	0	0	0	0	0	0	0	0	0	1	0	1	m_3	m_2	m_1	m_0	1
[HL+]. #m	0	0	0	0	0	0	0	1	0	1	1	0	m_3	m_2	m_1	m_0	1
[XY+]. #m	0	0	0	0	0	0	0	1	0	1	1	1	m_3	m_2	m_1	m_0	1
E:[HL+]. #m	0	0	0	0	0	0	0	1	0	1	0	0	m_3	m_2	m_1	m_0	1
E:[XY+]. #m	0	0	0	0	0	0	0	1	0	1	0	1	m_3	m_2	m_1	m_0	1

MOV A, obj

(transfer data memory content to accumulator)

obj

For no post-increment: direct, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

A ← obj
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

Transfers content of data memory specified by obj to the accumulator. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
if A = 0
then Z ← 1
else Z ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
A, direct	1	1	0	1	r_{11}	r_{10}	r_9	r_8	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
A, [HL]	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	1
A, [XY]	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	1
A, E:[HL]	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1
A, E:[XY]	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1
A, [HL+]	0	0	0	0	0	0	1	1	0	0	1	0	1	0	0	0	1
A, [XY+]	0	0	0	0	0	0	1	1	0	0	1	1	1	0	0	0	1
A, E:[HL+]	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	1
A, E:[XY+]	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	1

MOV A #i4

(transfer immediate data to accumulator)

obj

A ← i4

Description

Transfers immediate data i4 to the accumulator.

Flags

Flags affected by execution of this instruction
Flag change conditions
if A = 0
then Z ← 1
else Z ← 0

Z	C	G
❍	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
A, #i4	0	0	0	0	0	0	0	1	1	1	0	0	i_3	i_2	i_1	i_0	1

MOV obj, A

(transfer accumulator to data memory)

obj

For no post-increment: direct, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj ← A
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

Transfers content of accumulator to data memory specified by obj. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
—	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
direct, A	1	1	0	0	r_{11}	r_{10}	r_9	r_8	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], A	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1
[XY], A	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	1
E:[HL], A	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
E:[XY], A	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1
[HL+], A	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	1
[XY+], A	0	0	0	0	0	1	0	0	0	0	1	1	1	0	0	0	1
E:[HL+], A	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1
E:[XY+], A	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	1

MOV obj, #i4

(transfer immediate data to accumulator and data memory)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← i4
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

Transfers immediate data i4 to accumulator and data memory specified by obj. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
if A = 0
then Z ← 1
else Z ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur, #i4	0	1	0	0	i_3	i_2	i_1	i_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], #i4	0	0	0	0	0	1	1	0	0	1	1	0	i_3	i_2	i_1	i_0	1
[XY], #i4	0	0	0	0	0	1	1	0	0	1	1	1	i_3	i_2	i_1	i_0	1
E:[HL], #i4	0	0	0	0	0	1	1	0	0	1	0	0	i_3	i_2	i_1	i_0	1
E:[XY], #i4	0	0	0	0	0	1	1	0	0	1	0	1	i_3	i_2	i_1	i_0	1
[HL+], #i4	0	0	0	0	0	1	1	1	0	1	1	0	i_3	i_2	i_1	i_0	1
[XY+], #i4	0	0	0	0	0	1	1	1	0	1	1	1	i_3	i_2	i_1	i_0	1
E:[HL+], #i4	0	0	0	0	0	1	1	1	0	1	0	0	i_3	i_2	i_1	i_0	1
E:[XY+], #i4	0	0	0	0	0	1	1	1	0	1	0	1	i_3	i_2	i_1	i_0	1

MOV obj, #i4

(transfer immediate data to registers)

obj

CBR, EBR, RA0, RA1, RA2, RA3, H, L, X, Y

Function

obj ← i4

Description

Transfer immediate data i4 to the register specified by obj.

Flags

Flags affected by execution of this instruction

Z	C	G
—	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
CBR, #i4	0	0	0	0	0	0	0	0	0	0	1	1	i_3	i_2	i_1	i_0	1
EBR, #i4	0	0	0	0	0	0	0	0	0	0	1	0	i_3	i_2	i_1	i_0	1
RA0, #i4	0	0	0	0	0	0	1	0	0	0	0	0	i_3	i_2	i_1	i_0	1
RA1, #i4	0	0	0	0	0	0	1	0	0	0	0	1	i_3	i_2	i_1	i_0	1
RA2, #i4	0	0	0	0	0	0	1	0	0	0	1	0	i_3	i_2	i_1	i_0	1
RA3, #i4	0	0	0	0	0	0	1	0	0	0	1	1	i_3	i_2	i_1	i_0	1
H, #i4	0	0	0	0	0	0	0	1	0	0	1	1	i_3	i_2	i_1	i_0	1
L, #i4	0	0	0	0	0	0	0	1	0	0	1	0	i_3	i_2	i_1	i_0	1
X, #i4	0	0	0	0	0	0	0	1	0	0	0	1	i_3	i_2	i_1	i_0	1
Y, #i4	0	0	0	0	0	0	0	1	0	0	0	0	i_3	i_2	i_1	i_0	1

MOVHB obj, [RA]

(ROM table reference)

obj

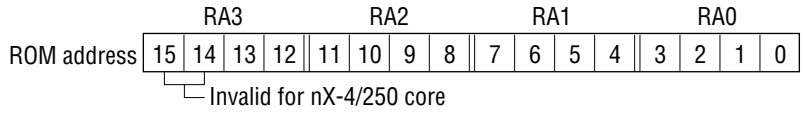
For no post-increment: [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj ← (RA) $_{11\sim8}$
obj + 1 ← (RA) $_{15\sim12}$
(for post-increment)
HL ← HL + 2 or XY ← XY + 2

Description

This instruction refers to the ROM table.
RA3 through RA0 compose the address as indicated below. The high 8-bit data at that address in program memory is transferred to the data memory specified by obj.



Bits 11 through 8 of program memory are transferred to the data memory specified by obj, and bits 15 to 12 to the data memory specified as obj+1. For post-increment operations, the HL or XY register is incremented by 2 after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
(for post-increment)
if HL + 2 > 0FFH or XY + 2 > 0FFH
then G ← 1
else G ← 0

Z	C	G
—	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
[HL], [RA]	0	0	0	0	0	0	1	1	0	0	1	0	0	0	1	0	2
[XY], [RA]	0	0	0	0	0	0	1	1	0	0	1	1	0	0	1	0	2
E:[HL], [RA]	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	2
E:[XY], [RA]	0	0	0	0	0	0	1	1	0	0	0	1	0	0	1	0	2
[HL+], [RA]	0	0	0	0	0	0	1	1	0	0	1	0	1	0	1	0	2
[XY+], [RA]	0	0	0	0	0	0	1	1	0	0	1	1	1	0	1	0	2
E:[HL+], [RA]	0	0	0	0	0	0	1	1	0	0	0	0	1	0	1	0	2
E:[XY+], [RA]	0	0	0	0	0	0	1	1	0	0	0	1	1	0	1	0	2

MOVHB obj, cadr16

(ROM table reference)

obj

For no post-increment: [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj ← (cadr16) $_{11\sim8}$
obj + 1 ← (cadr16) $_{15\sim12}$
(for post-increment)
HL ← HL + 2 or XY ← XY + 2

Description

This instruction refers to the ROM table.
The high 8-bit data in program memory specified by cadr16 is byte-transferred to the data memory specified by obj.
Bits 11 through 8 of program memory are transferred to the data memory specified by obj, and bits 15 to 12 to the data memory specified as obj+1. For post-increment operations, the HL or XY register is incremented by 2 after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
(for post-increment)
if HL + 2 > 0FFH or XY + 2 > 0FFH
then G ← 1
else G ← 0

Z	C	G
—	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
[HL], cadr16	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
[XY], cadr16	0	0	0	0	0	0	1	1	0	0	1	1	0	1	0	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
E:[HL], cadr16	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
E:[XY], cadr16	0	0	0	0	0	0	1	1	0	0	0	1	0	1	0	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
[HL+], cadr16	0	0	0	0	0	0	1	1	0	0	1	0	1	1	0	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
[XY+], cadr16	0	0	0	0	0	0	1	1	0	0	1	1	1	1	0	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
E:[HL+], cadr16	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
E:[XY+], cadr16	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	

MOVLB obj, [RA]

(ROM table reference)

obj

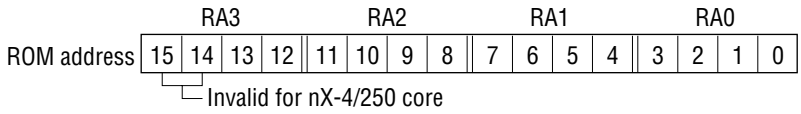
For no post-increment: [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj ← (RA) $_{3\sim0}$
obj + 1 ← (RA) $_{7\sim4}$
(for post-increment)
HL ← HL + 2 or XY ← XY + 2

Description

This instruction refers to the ROM table.
RA3 through RA0 compose the address as indicated below. The low 8-bit data at that address in program memory is transferred to the data memory specified by obj.



Bits 3 through 0 of program memory are transferred to the data memory specified by obj, and bits 7 to 4 to the data memory specified as obj+1. For post-increment operations, the HL or XY register is incremented by 2 after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
(for post-increment)
if HL + 2 > 0FFH or XY + 2 > 0FFH
then G ← 1
else G ← 0

Z	C	G
—	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
[HL], [RA]	0	0	0	0	0	0	1	1	0	0	1	0	0	0	1	1	2
[XY], [RA]	0	0	0	0	0	0	1	1	0	0	1	1	0	0	1	1	2
E:[HL], [RA]	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	2
E:[XY], [RA]	0	0	0	0	0	0	1	1	0	0	0	1	0	0	1	1	2
[HL+], [RA]	0	0	0	0	0	0	1	1	0	0	1	0	1	0	1	1	2
[XY+], [RA]	0	0	0	0	0	0	1	1	0	0	1	1	1	0	1	1	2
E:[HL+], [RA]	0	0	0	0	0	0	1	1	0	0	0	0	1	0	1	1	2
E:[XY+], [RA]	0	0	0	0	0	0	1	1	0	0	0	1	1	0	1	1	2

MOVLB obj, cadr16

(ROM table reference)

obj

For no post-increment: [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj ← (cadr16) $_{3\sim0}$
obj + 1 ← (cadr16) $_{7\sim4}$
(for post-increment)
HL ← HL + 2 or XY ← XY + 2

Description

This instruction refers to the ROM table.
The low 8-bit data in program memory specified by cadr16 is byte-transferred to the data memory specified by obj.
Bits 3 through 0 of program memory are transferred to the data memory specified by obj, and bits 7 to 4 to the data memory specified as obj+1. For post-increment operations, the HL or XY register is incremented by 2 after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
(for post-increment)
if HL + 2 > 0FFH or XY + 2 > 0FFH
then G ← 1
else G ← 0

Z	C	G
—	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
[HL], cadr16	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0	1	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
[XY], cadr16	0	0	0	0	0	0	1	1	0	0	1	1	0	1	0	1	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
E:[HL], cadr16	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	1	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
E:[XY], cadr16	0	0	0	0	0	0	1	1	0	0	0	1	0	1	0	1	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
[HL+], cadr16	0	0	0	0	0	0	1	1	0	0	1	0	1	1	0	1	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
[XY+], cadr16	0	0	0	0	0	0	1	1	0	0	1	1	1	1	0	1	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
E:[HL+], cadr16	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	1	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
E:[XY+], cadr16	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0	1	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	

MOVXB obj, [RA]

(transfer external memory to data memory)

obj

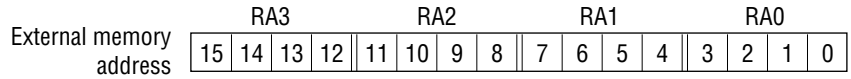
For no post-increment: [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj ← (RA) $_{3\sim0}$
obj + 1 ← (RA) $_{7\sim4}$
(for post-increment)
HL ← HL + 2 or XY ← XY + 2

Description

This instruction transfers external memory.
The address is specified by RA3 to RA0 as indicated below. 8-bit data in external memory specified is byte-transferred to the data memory specified by obj.



Bits 3 through 0 of external memory are transferred to the data memory specified by obj, and bits 7 to 4 to the data memory specified as obj+1. For post-increment operations, the HL or XY register is incremented by 2 after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
(for post-increment)
if HL + 2 > 0FFH or XY + 2 > 0FFH
then G ← 1
else G ← 0

Z	C	G
—	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
[HL], [RA]	0	0	0	0	0	0	1	1	0	0	1	0	0	1	1	0	2
[XY], [RA]	0	0	0	0	0	0	1	1	0	0	1	1	0	1	1	0	2
E:[HL], [RA]	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	2
E:[XY], [RA]	0	0	0	0	0	0	1	1	0	0	0	1	0	1	1	0	2
[HL+], [RA]	0	0	0	0	0	0	1	1	0	0	1	0	1	1	1	0	2
[XY+], [RA]	0	0	0	0	0	0	1	1	0	0	1	1	1	1	1	0	2
E:[HL+], [RA]	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	0	2
E:[XY+], [RA]	0	0	0	0	0	0	1	1	0	0	0	1	1	1	1	0	2

MOVXB obj, xadr16

(transfer external memory to data memory)

obj

For no post-increment: [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj ← (xadr16) $_{3\sim0}$
obj + 1 ← (xadr16) $_{7\sim4}$
(for post-increment)
HL ← HL + 2 or XY ← XY + 2

Description

This instruction transfers external memory.
The address is specified by xadr16. 8-bit data in external memory specified is byte-transferred to the data memory specified by obj.
Bits 3 through 0 of external memory are transferred to the data memory specified by obj, and bits 7 to 4 to the data memory specified as obj+1. For post-increment operations, the HL or XY register is incremented by 2 after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
(for post-increment)
if HL + 2 > 0FFH or XY + 2 > 0FFH
then G ← 1
else G ← 0

Z	C	G
—	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
[HL], xadr16	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
[XY], xadr16	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
E:[HL], xadr16	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
E:[XY], xadr16	0	0	0	0	0	0	1	1	1	1	0	1	0	0	0	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
[HL+], xadr16	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
[XY+], xadr16	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
E:[HL+], xadr16	0	0	0	0	0	0	1	1	1	1	0	0	1	0	0	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
E:[XY+], xadr16	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	

MOVXB [RA], obj

(transfer data memory to external memory)

obj

For no post-increment: [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

(RA) $_{3\sim0}$ ← obj
(RA) $_{7-4}$ ← obj
(for post-increment)
HL ← HL + 2 or XY ← XY + 2

Description

This instruction transfers external memory.
The address is specified by RA3 to RA0 as indicated below. Data in the data memory specified by obj is byte-transferred to the specified external memory.

	RA3				RA2				RA1				RA0			
External memory address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Data memory specified by obj is transferred to bits 3 through 0 of external memory, and data memory specified by obj+1 is transferred to external memory bits 7 to 4.
For post-increment operations, the HL or XY register is incremented by 2 after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
(for post-increment)
if HL + 2 > 0FFH or XY + 2 > 0FFH
then G ← 1
else G ← 0

Z	C	G
—	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
[RA], [HL]	0	0	0	0	0	0	1	1	0	0	1	0	0	1	1	1	3
[RA], [XY]	0	0	0	0	0	0	1	1	0	0	1	1	0	1	1	1	3
[RA], E:[HL]	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1	3
[RA], E:[XY]	0	0	0	0	0	0	1	1	0	0	0	1	0	1	1	1	3
[RA], [HL+]	0	0	0	0	0	0	1	1	0	0	1	0	1	1	1	1	3
[RA], [XY+]	0	0	0	0	0	0	1	1	0	0	1	1	1	1	1	1	3
[RA], E:[HL+]	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	3
[RA], E:[XY+]	0	0	0	0	0	0	1	1	0	0	0	1	1	1	1	1	3

MOVXB xadr16, obj

(transfer data memory to external memory)

obj

For no post-increment: [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

(cadr16) $_{3\sim0} \leftarrow$ obj
(cadr16) $_{7\sim4} \leftarrow$ obj
(for post-increment)
HL $\leftarrow$ HL + 2 or XY $\leftarrow$ XY + 2

Description

This instruction transfers external memory.
Data in the data memory specified by obj is byte-transferred to external memory specified by xadr16.
Data memory specified by obj is transferred to bits 3 through 0 of external memory, and data memory specified by obj+1 is transferred to external memory bits 7 to 4.
For post-increment operations, the HL or XY register is incremented by 2 after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
(for post-increment)
if HL + 2 > 0FFH or XY + 2 > 0FFH
then G $\leftarrow$ 1
else G $\leftarrow$ 0

Z	C	G
—	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
xadr16, [HL]	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
xadr16, [XY]	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	1	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
xadr16, E:[HL]	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
xadr16, E:[XY]	0	0	0	0	0	0	1	1	1	1	0	1	0	0	0	1	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
xadr16, [HL+]	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	1	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
xadr16, [XY+]	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	1	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
xadr16, E:[HL+]	0	0	0	0	0	0	1	1	1	1	0	0	1	0	0	1	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
xadr16, E:[XY+]	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	1	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	

MSA cadr16

(start melody output)

Function

Melody Start

Description

Starts melody output from the head address of melody data specified by a_{15} to a_0.

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
cadr16	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	3
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	

MSET obj, #m

(data memory bit set)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

Bits not masked by obj ← 1, A ← obj (m = 0 ~ 0FH)
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

Sets all bits not masked by immediate data m, in the data memory specified by obj. After the clear operation the content of obj is stored to the accumulator. m is a value in the range 0 to 0FH, with a "0" masking a bit and a "1" marking a bit to be set. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur. #m	0	1	1	0	m_3	m_2	m_1	m_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL]. #m	0	0	0	0	0	0	1	0	0	1	1	0	m_3	m_2	m_1	m_0	1
[XY]. #m	0	0	0	0	0	0	1	0	0	1	1	1	m_3	m_2	m_1	m_0	1
E:[HL]. #m	0	0	0	0	0	0	1	0	0	1	0	0	m_3	m_2	m_1	m_0	1
E:[XY]. #m	0	0	0	0	0	0	1	0	0	1	0	1	m_3	m_2	m_1	m_0	1
[HL+]. #m	0	0	0	0	0	0	1	1	0	1	1	0	m_3	m_2	m_1	m_0	1
[XY+]. #m	0	0	0	0	0	0	1	1	0	1	1	1	m_3	m_2	m_1	m_0	1
E:[HL+]. #m	0	0	0	0	0	0	1	1	0	1	0	0	m_3	m_2	m_1	m_0	1
E:[XY+]. #m	0	0	0	0	0	0	1	1	0	1	0	1	m_3	m_2	m_1	m_0	1

MTST obj, A

(data memory bit test)

obj

For no post-increment: sfr, ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

If any of the bits not masked by A is "0",
then Z ← 1
else Z ← 0
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

Tests all bits not masked by accumulator in the data memory specified by obj. If even one bit is "0", sets the Z flag. In the accumulator a "0" masks a bit and a "1" marks a bit to be tested. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
See function column
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
sfr, A	0	0	1	0	1	1	1	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
¥cur, A	0	0	1	1	1	1	1	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], A	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	1
[XY], A	0	0	0	0	0	1	0	0	0	0	1	1	0	1	0	1	1
E:[HL], A	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	1
E:[XY], A	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	1	1
[HL+], A	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	1	1
[XY+], A	0	0	0	0	0	1	0	0	0	0	1	1	1	1	0	1	1
E:[HL+], A	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	1
E:[XY+], A	0	0	0	0	0	1	0	0	0	0	0	1	1	1	0	1	1

MTST obj, #m

(data memory bit test)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

If any of the bits not masked by m is "0",
then Z ← 1
else Z ← 0
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

Tests all bits not masked by immediate data m in the data memory specified by obj. If even one bit is "0", sets the Z flag. m is a value from 0 to 0FH, with a "0" masking a bit and a "1" marking a bit to be tested. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
See function column
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur. #m	1	0	1	1	m_3	m_2	m_1	m_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL]. #m	0	0	0	0	0	1	0	0	1	0	1	0	m_3	m_2	m_1	m_0	1
[XY]. #m	0	0	0	0	0	1	0	0	1	0	1	1	m_3	m_2	m_1	m_0	1
E:[HL]. #m	0	0	0	0	0	1	0	0	1	0	0	0	m_3	m_2	m_1	m_0	1
E:[XY]. #m	0	0	0	0	0	1	0	0	1	0	0	1	m_3	m_2	m_1	m_0	1
[HL+]. #m	0	0	0	0	0	1	0	1	1	0	1	0	m_3	m_2	m_1	m_0	1
[XY+]. #m	0	0	0	0	0	1	0	1	1	0	1	1	m_3	m_2	m_1	m_0	1
E:[HL+]. #m	0	0	0	0	0	1	0	1	1	0	0	0	m_3	m_2	m_1	m_0	1
E:[XY+]. #m	0	0	0	0	0	1	0	1	1	0	0	1	m_3	m_2	m_1	m_0	1

NOP

(no operation)

Function

No Operation

Description

Do nothing for one machine cycle.

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
NOP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

OR obj, A

(logical add of data memory and accumulator)

obj

For no post-increment: sfr, ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← obj ∨ A
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

Takes the logical add for the accumulator and the data memory specified by obj. Results are written to data memory and the accumulator. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj ∨ A = 0
then Z ← 1
else Z ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
sfr, A	0	0	1	0	1	1	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
¥cur, A	0	0	1	1	1	1	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], A	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	1
[XY], A	0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	0	1
E:[HL], A	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1
E:[XY], A	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	1
[HL+], A	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	0	1
[XY+], A	0	0	0	0	0	1	0	0	0	0	1	1	1	0	1	0	1
E:[HL+], A	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	1
E:[XY+], A	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	1

OR obj, #i4

(logical add of data memory and immediate data)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← obj ∨ i4
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

Takes the logical add for the immediate data and the data memory specified by obj. Results are written to data memory and the accumulator. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj ∨ i4 = 0
then Z ← 1
else Z ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur, #i4	0	1	1	0	i_3	i_2	i_1	i_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], #i4	0	0	0	0	0	0	1	0	0	1	1	0	i_3	i_2	i_1	i_0	1
[XY], #i4	0	0	0	0	0	0	1	0	0	1	1	1	i_3	i_2	i_1	i_0	1
E:[HL], #i4	0	0	0	0	0	0	1	0	0	1	0	0	i_3	i_2	i_1	i_0	1
E:[XY], #i4	0	0	0	0	0	0	1	0	0	1	0	1	i_3	i_2	i_1	i_0	1
[HL+], #i4	0	0	0	0	0	0	1	1	0	1	1	0	i_3	i_2	i_1	i_0	1
[XY+], #i4	0	0	0	0	0	0	1	1	0	1	1	1	i_3	i_2	i_1	i_0	1
E:[HL+], #i4	0	0	0	0	0	0	1	1	0	1	0	0	i_3	i_2	i_1	i_0	1
E:[XY+], #i4	0	0	0	0	0	0	1	1	0	1	0	1	i_3	i_2	i_1	i_0	1

POP HL

(restores flag registers, accumulator and HL register)

Function

RSP ← RSP - 1, {FLAG, A, HL} ← (RSP)

Description

Decrements the register stack pointer, then restores the register stack content to the flag registers (G, C, Z), accumulator and HL register.

Flags

Flags affected by execution of this instruction
Flag change conditions

if Z = 1 when PUSH HL executed,
then Z ← 1
else Z ← 0
if C = 1 when PUSH HL executed
then C ← 1
else C ← 0
if G = 1 when PUSH HL executed
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
HL	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	2

POP XY

(restores extra bank register, current bank register and XY register)

Function

RSP ← RSP-1, {EBR, CBR, XY} ← (RSP)

Description

Decrements the register stack pointer, then restores the register bank stack content to the extra bank register, current bank register and XY register.

Flags

Flags affected by execution of this instruction

Z	C	G
—	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
XY	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	2

PUSH HL

(saves flag registers, accumulator and HL register)

Function

(RSP) ← (FLAG, A, HL), RSP ← RSP + 1

Description

Saves the flag registers (G, C, Z), accumulator and HL register to the register stack, then increments the register stack pointer.

Flags

Flags affected by execution of this instruction

Z	C	G
—	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
HL	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	2

PUSH XY

(saves extra bank register, current bank register and XY register)

Function

(RSP) ← {EBR, CBR, XY}, RSP ← RSP + 1

Description

Saves the extra bank register, current bank register and XY register to the register stack, then increments the register stack pointer.

Flags

Flags affected by execution of this instruction

Z	C	G
—	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
XY	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	2

ROL obj

(rotate left)

obj

For no post-increment: sfr, ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

$\boxed{\text{obj}_3 \leftarrow \text{obj}_2 \leftarrow \text{obj}_1 \leftarrow \text{obj}_0 \leftarrow C \leftarrow}$, $A \leftarrow \text{obj}$

(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

Rotates the data memory content specified by obj one bit left. The carry flag is stored to data memory LSB, and the MSB to the carry flag. After rotation the data memory content is stored to the accumulator. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj_3 = 1 (obj bit 3 = 1 before instruction execution)
then C ← 1
else C ← 0
if A = 0
then Z ← 1
else Z ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
sfr	0	0	1	0	0	0	1	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
¥cur	0	0	1	1	0	0	1	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL]	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0	1
[XY]	0	0	0	0	0	1	0	0	0	0	1	1	0	1	1	0	1
E:[HL]	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	1
E:[XY]	0	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0	1
[HL+]	0	0	0	0	0	1	0	0	0	0	1	0	1	1	1	0	1
[XY+]	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	0	1
E:[HL+]	0	0	0	0	0	1	0	0	0	0	0	0	1	1	1	0	1
E:[XY+]	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0	1

ROR obj

(rotate right)

obj

For no post-increment: sfr, ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

$\rightarrow obj_3 \rightarrow obj_2 \rightarrow obj_1 \rightarrow obj_0 \rightarrow C \rightarrow$,A ← obj
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

Rotates the data memory content specified by obj one bit right. The carry flag is stored to data memory MSB, and the LSB to the carry flag. After rotation the data memory content is stored to the accumulator. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj_0 = 1 (obj bit 0 = 1 before instruction execution)
then C ← 1
else C ← 0
if A = 0
then Z ← 1
else Z ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
sfr	0	0	1	0	0	0	1	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
¥cur	0	0	1	1	0	0	1	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL]	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	1	1
[XY]	0	0	0	0	0	1	0	0	0	0	1	1	0	1	1	1	1
E:[HL]	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	1
E:[XY]	0	0	0	0	0	1	0	0	0	0	0	1	0	1	1	1	1
[HL+]	0	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	1
[XY+]	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	1	1
E:[HL+]	0	0	0	0	0	1	0	0	0	0	0	0	1	1	1	1	1
E:[XY+]	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	1	1

RT

(return from subroutine)

Function

PC ← (SP) + 1, SP ← SP-1

Description

The PC (program counter) value saved to the call stack by the CAL or LCAL instruction is incremented by one then set to the PC. The call stack pointer is then decremented.

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
RT	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1

RTI

(return from maskable interrupt processing routine)

Function

PC ← (SP) + 1, SP ← SP-1, MIE ← 1

Description

The PC (program counter) value saved to the call stack when a maskable interrupt was generated is incremented by one then set to the PC. The call stack pointer is then decremented. MIE is set to 1 to enable maskable interrupts again. The RTI instruction is used as the instruction to return from a maskable interrupt routine.

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
RTI	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1

RTNMI

(return from non-maskable interrupt processing routine)

Function

PC ← (SP) + 1, SP ← SP-1, MIE ← Pre-interrupt MIE state

Description

The PC (program counter) value saved to the call stack when a non-maskable interrupt was generated is incremented by one then set to the PC. The call stack pointer is then decremented. The pre-interrupt state is reset in MIE. The RTNMI instruction is used as the instruction to return from a non-maskable interrupt routine.

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
RTNMI	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1

SBC obj, A

(data memory and accumulator subtraction with carry)

obj

For no post-increment: sfr, ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj A ← obj - A - C
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The accumulator and carry flag are subtracted from the data memory specified by obj, and the results stored to data memory and the accumulator. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj - A - C = 0
then Z ← 1
else Z ← 0
if obj - A - C < 0
then C ← 1
else C ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
sfr, A	0	0	1	0	1	0	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
¥cur, A	0	0	1	1	1	0	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], A	0	0	0	0	0	1	0	1	0	0	1	0	0	1	1	0	1
[XY], A	0	0	0	0	0	1	0	1	0	0	1	1	0	1	1	0	1
E:[HL], A	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	0	1
E:[XY], A	0	0	0	0	0	1	0	1	0	0	0	1	0	1	1	0	1
[HL+], A	0	0	0	0	0	1	0	1	0	0	1	0	1	1	1	0	1
[XY+], A	0	0	0	0	0	1	0	1	0	0	1	1	1	1	1	0	1
E:[HL+], A	0	0	0	0	0	1	0	1	0	0	0	0	1	1	1	0	1
E:[XY+], A	0	0	0	0	0	1	0	1	0	0	0	1	1	1	1	0	1

SBCD obj, A

(data memory and accumulator subtraction with carry with decimal adjust)

obj

For no post-increment: sfr, ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← decimal adjust {obj - A - C}
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The accumulator and carry flag are subtracted from the data memory specified by obj, and the results stored to data memory and the accumulator if there is no resulting borrow. If there is a borrow, the decimal adjustment AH is added and the results stored to data memory and the accumulator. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
if the decimal adjustment = 0
then Z ← 1
else Z ← 0
if a borrow results, C ← 1, else C ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
sfr, A	0	0	1	0	1	0	0	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
¥cur, A	0	0	1	1	1	0	0	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], A	0	0	0	0	0	1	0	1	0	0	1	0	0	1	1	1	1
[XY], A	0	0	0	0	0	1	0	1	0	0	1	1	0	1	1	1	1
E:[HL], A	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1	1
E:[XY], A	0	0	0	0	0	1	0	1	0	0	0	1	0	1	1	1	1
[HL+], A	0	0	0	0	0	1	0	1	0	0	1	0	1	1	1	1	1
[XY+], A	0	0	0	0	0	1	0	1	0	0	1	1	1	1	1	1	1
E:[HL+], A	0	0	0	0	0	1	0	1	0	0	0	0	1	1	1	1	1
E:[XY+], A	0	0	0	0	0	1	0	1	0	0	0	1	1	1	1	1	1

SBCJ obj, n

(data memory subtraction with carry with base-n adjust)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← n adjust {obj-C} (Even number from 2 to 16)
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The accumulator and carry flag are subtracted from the data memory specified by obj, and base-n adjustment completed. The results are stored to data memory and the accumulator. If the result of base-n adjustment is 0FH, the adjustment n is added. n is an even value from 2 to 16. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
if the base-n adjustment = 0
then Z ← 1
else Z ← 0
if an underflow results, C ← 1, else C ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur. n	0	0	0	1	1	n_2	n_1	n_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL]. n	0	0	0	0	0	1	1	0	0	0	1	0	1	n_2	n_1	n_0	1
[XY]. n	0	0	0	0	0	1	1	0	0	0	1	1	1	n_2	n_1	n_0	1
E:[HL]. n	0	0	0	0	0	1	1	0	0	0	0	0	1	n_2	n_1	n_0	1
E:[XY]. n	0	0	0	0	0	1	1	0	0	0	0	1	1	n_2	n_1	n_0	1
[HL+]. n	0	0	0	0	0	1	1	1	0	0	1	0	1	n_2	n_1	n_0	1
[XY+]. n	0	0	0	0	0	1	1	1	0	0	1	1	1	n_2	n_1	n_0	1
E:[HL+]. n	0	0	0	0	0	1	1	1	0	0	0	0	1	n_2	n_1	n_0	1
E:[XY+]. n	0	0	0	0	0	1	1	1	0	0	0	1	1	n_2	n_1	n_0	1

[Note] The relation between adjustment n and n_2 to n_0 in the instruction code is shown below.

Adjustment n	2	4	6	8	10	12	14	16
n_2 to n_0 in instruction code	1H	2H	3H	4H	5H	6H	7H	0H

SJMP radr8

(PC relative branch)

Function

PC ← NextPC + radr8
(However, -128 ≤ radr8 ≤ +127
Next PC is the address (PC+1) after the instruction)

Description

The content of radr8 is added to NextPC and the program branched.
The destination is a range from -128 to +127 from the NextPC address. The 8 bits in the instruction code (a_7 to a_0) correspond to radr8, and a_7 (8th bit) is the sign indicating the direction of displacement from the next address.
Branches across program memory page boundaries are possible.
The Assembler can directly specify addresses within branch range (labels) instead of radr8.

Flags

Flags affected by execution of this instruction

Z	C	G
—	—	—

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
radr8	0	0	0	0	1	0	0	a_7	1	a_6	a_5	a_4	a_3	a_2	a_1	a_0	1

SUB obj, A

(data memory and accumulator subtraction)

obj

For no post-increment: sfr, ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← obj - A
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The accumulator is subtracted from the data memory specified by obj, and the results stored to data memory and the accumulator. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj - A = 0
then Z ← 1
else Z ← 0
if obj - A < 0
then C ← 1
else C ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
sfr, A	0	0	1	0	0	1	1	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
¥cur, A	0	0	1	1	0	1	1	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], A	0	0	0	0	0	1	0	1	0	0	1	0	0	1	0	1	1
[XY], A	0	0	0	0	0	1	0	1	0	0	1	1	0	1	0	1	1
E:[HL], A	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1	1
E:[XY], A	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	1	1
[HL+], A	0	0	0	0	0	1	0	1	0	0	1	0	1	1	0	1	1
[XY+], A	0	0	0	0	0	1	0	1	0	0	1	1	1	1	0	1	1
E:[HL+], A	0	0	0	0	0	1	0	1	0	0	0	0	1	1	0	1	1
E:[XY+], A	0	0	0	0	0	1	0	1	0	0	0	1	1	1	0	1	1

SUB obj, #i4

(data memory and immediate data subtraction)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← obj - i4
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The immediate data i4 is subtracted from the data memory specified by obj, and the results stored to data memory and the accumulator. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj - i4 = 0
then Z ← 1
else Z ← 0
if obj - i4 < 0
then C ← 1
else C ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	❍	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur, #i4	1	0	0	1	i_3	i_2	i_1	i_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], #i4	0	0	0	0	0	0	1	0	1	0	1	0	i_3	i_2	i_1	i_0	1
[XY], #i4	0	0	0	0	0	0	1	0	1	0	1	1	i_3	i_2	i_1	i_0	1
E:[HL], #i4	0	0	0	0	0	0	1	0	1	0	0	0	i_3	i_2	i_1	i_0	1
E:[XY], #i4	0	0	0	0	0	0	1	0	1	0	0	1	i_3	i_2	i_1	i_0	1
[HL+], #i4	0	0	0	0	0	0	1	1	1	0	1	0	i_3	i_2	i_1	i_0	1
[XY+], #i4	0	0	0	0	0	0	1	1	1	0	1	1	i_3	i_2	i_1	i_0	1
E:[HL+], #i4	0	0	0	0	0	0	1	1	1	0	0	0	i_3	i_2	i_1	i_0	1
E:[XY+], #i4	0	0	0	0	0	0	1	1	1	0	0	1	i_3	i_2	i_1	i_0	1

XCH A, obj

(swap data memory and accumulator contents)

obj

For no post-increment: sfr, ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

A ↔ obj
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

The accumulator is swapped with the data memory specified by obj. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
—	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
A, sfr	0	0	1	0	1	1	1	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
A, ¥cur	0	0	1	1	1	1	1	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
A, [HL]	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	1	1
A, [XY]	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	1	1
A, E:[HL]	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1
A, E:[XY]	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	1	1
A, [HL+]	0	0	0	0	0	0	1	1	0	0	1	0	1	0	0	1	1
A, [XY+]	0	0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	1
A, E:[HL+]	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	1	1
A, E:[XY+]	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	1	1

XOR obj, A

(exclusive OR of data memory and accumulator)

obj

For no post-increment: sfr, ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← obj ∀ A
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

A exclusive OR is executed for the accumulator and the data memory specified by obj, and the results written to data memory and the accumulator. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj ∀ A = 0
then Z ← 1
else Z ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
sfr, A	0	0	1	0	1	1	0	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
¥cur, A	0	0	1	1	1	1	0	1	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], A	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	1	1
[XY], A	0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	1	1
E:[HL], A	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1
E:[XY], A	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	1
[HL+], A	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	1	1
[XY+], A	0	0	0	0	0	1	0	0	0	0	1	1	1	0	1	1	1
E:[HL+], A	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	1	1
E:[XY+], A	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	1	1

XOR obj, #i4

(exclusive OR of data memory and immediate data)

obj

For no post-increment: ¥cur, [HL], [XY], E:[HL], E:[XY]
For post-increment: [HL+], [XY+], E:[HL+], E:[XY+]

Function

obj, A ← obj ∀ i4
(for post-increment)
HL ← HL + 1 or XY ← XY + 1

Description

A exclusive OR is executed for the immediate data and the data memory specified by obj, and the results written to data memory and the accumulator. For post-increment operations, the HL or XY register is incremented after execution.

Flags

Flags affected by execution of this instruction
Flag change conditions
if obj ∀ i4 = 0
then Z ← 1
else Z ← 0
(for post-increment)
if HL + 1 = 0 or XY + 1 = 0
then G ← 1
else G ← 0

Z	C	G
❍	—	❍

Codes/Cycles

Operand	Instruction Code																Machine Cycle
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
¥cur, #i4	0	1	1	1	i_3	i_2	i_1	i_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	1
[HL], #i4	0	0	0	0	0	0	0	0	0	1	1	0	i_3	i_2	i_1	i_0	1
[XY], #i4	0	0	0	0	0	0	0	0	0	1	1	1	i_3	i_2	i_1	i_0	1
E:[HL], #i4	0	0	0	0	0	0	0	0	0	1	0	0	i_3	i_2	i_1	i_0	1
E:[XY], #i4	0	0	0	0	0	0	0	0	0	1	0	1	i_3	i_2	i_1	i_0	1
[HL+], #i4	0	0	0	0	0	0	0	1	0	1	1	0	i_3	i_2	i_1	i_0	1
[XY+], #i4	0	0	0	0	0	0	0	1	0	1	1	1	i_3	i_2	i_1	i_0	1
E:[HL+], #i4	0	0	0	0	0	0	0	1	0	1	0	0	i_3	i_2	i_1	i_0	1
E:[XY+], #i4	0	0	0	0	0	0	0	1	0	1	0	1	i_3	i_2	i_1	i_0	1