TLCS-870 SERIES INSTRUCTION SET

TLCS-870 CPU has 412 basic machine instruction of 129 types shown in Table 1. All of the instructions are used in any derivative of the TLCS-870 Series.
Instructions in the TLCS-870 Series are from 1 byte to a maximum of 4 bytes in length. Instructions having a high frequency of use have short object codes in order to reduce program size for good memory efficiency. For example, TLCS-870 CPU has various subroutine call instructions: CALLV, CALLP and CALL. The CALLV is single byte instruction, while the CALL is 3 bytes instruction. Both CALLV and CALL instructions can reach anywhere in the program memory space.
The TLCS-870 Series uses the memory mapped I/O format. All I/O registers are mapped to SFR and DBR areas of memory space. In the TLCS-870 Series, load instructions (LD) are used for input or output. The TLCS-870 instruction set is simple, and there are only 41 types of mnemonic shown in Table 3. 20 types of addressing modes shown in Table 2 are optimized program steps.

In addition to the basic machine instructions, expansion machine instructions are prepared in relocatable macro-assembler to upgrade coding efficiency. For more details on the expansion machine instructions, see the "TLCS-870 Development System Manual".

Transfer, Exchange	8-bit Data Transfer, Exchange		9 types	46 instructions
	16-bit Data Transfer, Exchange		6	16
	Register Bank Switching		1	1
Operation	8-bit Operation	Compare	6	30
		Increment, Decrement	4	18
		Arithmetic	20	92
		Logical	15	69
		Decimal Adjust	2	4
	16-bit Operation	Compare	2	2
		Increment, Decrement	2	2
		Arithmetic	8	8
		Logical	6	6
	Multiply, Divide		2	4
Shift, Rotate	Shift		2	4
	Rotate		2	4
Nibble Manipulation	Swap, Nibble Rotate		3	14
Bit Manipulation	Set, Clear, Complement, etc.		17	53
Flag Manipulation			3	3
Branch	Jump		5	18
	Call		5	9
	Return		3	3
Others	Push, Pop		4	4
	Software Interrupt, No Operation		2	2
Total			129 types	412 instructions

Table 1. Instruction Set

Register-Indirect	3types
Indexed	3
Direct	1
Register	1
Implied	1
Immediate	1
Relative	2
Absolute	1
Vector	1
Page	1
Direct Bit	4
Register-Indirect Bit	1
Total	20types

Table 2. Addressing Mode

1.1 EXPLANATION OF SYMBOLS

In the following explanation of instructions and addressing modes, symbols shown below and next page are used.

Symbol	Explanation	Symbol	Explanation
A	A register (8-bit accumulator)	n	4-bit or 8-bit immediate data
W	W register	mn	16-bit immediate data
B	B register	d	Signed 5-bit or 8-bit displacement
C	C register		(– 16~ + 15/ – 128~ + 127)
D	D register	b	Bit number (0~7)
E	E register	x, y	8-bit direct address (0000~00FF$_H$)
H	H register	(XX)	Contents of the memory address specified by
L	L register		effective address XX
WA	WA register pair (16-bit accumulator)	(XX+1, XX)	Contents of memory in 2 consecutive bytes
BC	BC register pair		starting from the address specified by XX
DE	DE register pair	.b	Content of bit that is specified by b
HL	HL register pair	←	Transfer
PC	Program Counter	↔	Exchange
SP	Stack Pointer	+	Add
PSW	Program Status Word	–	Subtract
RBS	Register Bank Selector	×	Multiply
JF	Jump Status Flag	÷	Divide
ZF	Zero Flag	∧	Logical AND of the corresponding bits
CF	Carry Flag (1-bit accumulator)	∨	Logical OR of the corresponding bits
HF	Half Carry Flag	∀	Logical exclusive OR of the corresponding
$\overline{CF}$	Inversion of Carry Flag Contents	null	bits
IMF	Interrupt Master Enable Flag	$	No operation
r, g	Register (See Table Ⓐ)		Header address of instruction (the contents
rr, gg	Register pair (See Table Ⓑ)		of the program counter, the instruction in
pp	Register pair (See Table Ⓒ)	(src)	which is now under execution, becomes $ + 2)
		(dst)	Source memory
			Destination memory

Object code	Register
000	A
001	W
010	C
011	B
100	E
101	D
110	L
111	H

Table Ⓐ

Object code	Register pair
00	WA
01	BC
10	DE
11	HL

Table Ⓑ

Object code	Register pair
0	DE
1	HL

Table Ⓒ

Flag Setting Condition	
*	The data specified by the operation is set.
Z	Zero detection information is set. ● Transfer When the source data is 00_H, "1" is set. When other than 00_H, "0" is set. ● Exchange When the contents of the g register or (src) before exchange are 00_H, "1" is set. When other than 00_H, "0" is set. ● Operation When the result of arithmetic/logical operations is 00_H (8-bit operation) or 0000_H (16-bit operation), "1" is set. When anything else, "0" is set. However, when the upper 8 bits of the product during multiplication, or the remainder during division, is 00_H, "1" is set. When other than 00_H, "0" is set. ● Shift and Rotation When the contents of the register after shift/rotation are 00_H, "1" is set. When other than 00_H, "0" is set. ● Other Indicates that the data which is set in ZF is also set into JF.
C	Carry information is set. ● Addition The carry from the most significant bit is set . ● Subtraction The borrow to the most significant bit is set. ● Division When the divisor is 00_H, or when the quotient is 100_H or higher, "1" is set. When anything else, "0" is set. ● Other Indicates that the data which is set in CF is also set into JF.
$\overline{C}$	The inverted value of the data to be set into CF is set.
H	Half carry information is set. ● Addition The carry from bit 3 is set. ● Subtraction The borrow to bit 3 is set.
$\overline{J}$	The inverted value of the data to be set into JF is set.
1	Always "1" is set.
0	Always "0" is set.
U	Undefined data is set.
–	Flag is not affected, the data before the instruction execution is held.

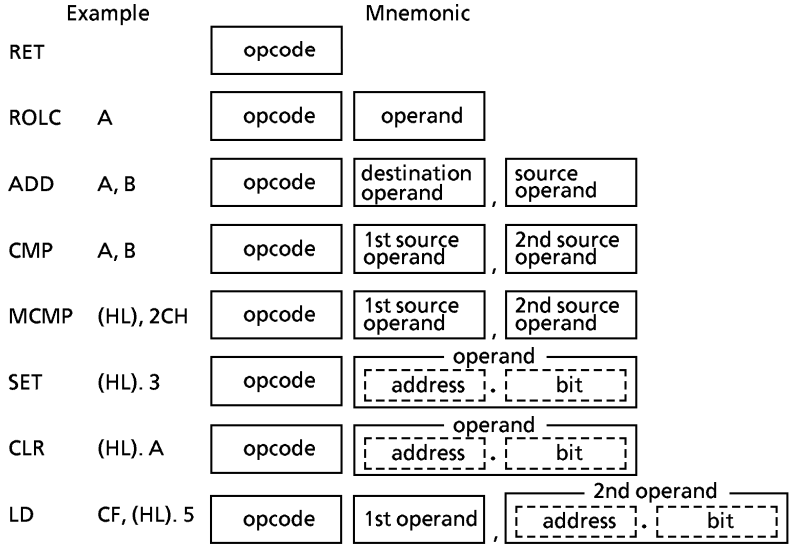
1.2 MNEMONIC

The TLCS-870's rules for instruction mnemonics are as follows:

A mnemonic usually consists of an opcode and some operands (in addition, there are also instructions without operands). At least one space is left open after the opcode and there the operand is placed.

If there are two or more operands, the operands are delimited by commas. If there are two operands, a source operand and a destination operands, always place the destination operand first. If there are multiple source operands, place the data that is to be operated on first. Note that the MCMP instruction is an exception, and the third operand A (accumulator) will be omitted.

If bit specifications are contained in an operand, delimit the address specification and the bit specification with periods.



Mnemonic	Operation	Mnemonic	Operation
ADD	Add	NOP	No Operation
ADDC	Add with Carry	OR	Logical OR
AND	Logical AND	POP	Pop up
CALL	Call	PUSH	Push down
CALLP	Page Call	RET	Return from subroutine
CALLV	Vector Call	RETI	Return from maskable Interrupt service routine
CLR	Clear Byte/Bit	RETN	Return from Non-maskable Interrupt service routine
CMP	Compare	ROLC	Rotate Left through Carry flag
CPL	1's Complement	ROLD	Rotate Left Digit
DAA	Decimal Adjust for Addition	RORC	Rotate Right through Carry flag
DAS	Decimal Adjust for Subtraction	RORD	Rotate Right Digit
DEC	Decrement	SET	Set Bit
DI*	Disable Maskable Interrupt	SHLC	Logical Shift Left with Carry flag
DIV	Divide	SHRC	Logical Shift Right with Carry flag
EI*	Enable Interrupt	SUB	Subtract
INC	Increment	SUBB	Subtract with Borrow
J*	optimized Jump	SWAP	Swap nibbles
JP	Absolute Jump	SWI	Software Interrupt
JR	Relative Jump	TEST*	Test Bit
JRS	Short Relative Jump	XCH	Exchange Word/Byte
LD	Load Word/Byte/Bit	XOR	Logical Exclusive OR
LDW	Load Word		
MCMP	Masked Compare		
MUL	Multiply		

*Note: *; Assembler expansion machine instruction*

Table 3. Mnemonic List

1.3 OBJECT CODE FORMAT

The TLCS-870 Series has 1-byte opcode instructions and 2-byte opcode instructions.

(1) Code format of 1-byte opcode instructions:

The opcode is placed in the first byte and the operands are placed in the second and subsequent bytes.
If the operand is 2-byte data, the lower byte is placed first and the upper byte is placed second. When there are two operands (the source and the destination), and if the source is immediate data, it is placed after the destination, and if the source is address information, it is placed before the destination.

Example	1st byte	2nd byte	3rd byte	4th byte
LD A, B	opcode			
LD A, n	opcode	n		
LD WA, mn	opcode	n	m	
LD (x), n	opcode	x	n	
LDW (x), mn	opcode	x	n	m
LD (x), (y)	opcode	y	x	

Notes : n, m ; *immediate data*
x ; *destination memory address*
y ; *source memory address*

(2) Code format of 2-byte opcode instructions:

The first opcode is placed in the first byte followed by the operands specified by the first opcode, then the second opcode and the operands specified by the second opcode are placed.
The first opcode is used for specifying the addressing mode, and it is also called the prefix code. The prefixes consist of the register prefix which specifies register, and the source or destination memory prefix which specifies the source or the destination memory address.

Example	1st byte	2nd byte	3rd byte	4th byte
① Register prefix				
LD B, C	1st opcode	2nd opcode		
ADD B, n	1st opcode	2nd opcode	n	
② Source memory prefix				
LD B, (HL)	1st opcode	2nd opcode		
LD B, (x)	1st opcode	x	2nd opcode	
LD B, (HL + d)	1st opcode	d	2nd opcode	
LD (HL + d), n	1st opcode	d	2nd opcode	n
③ Distination memory prefix				
LD (HL), B	1st opcode	2nd opcode		
LD (x), B	1st opcode	x	2nd opcode	
LD (HL + d), B	1st opcode	d	2nd opcode	
LD (HL + d), n	1st opcode	d	2nd opcode	n

Notes : n ; *immediate data*
x ; *source or destination memory address*
d ; *displacement*

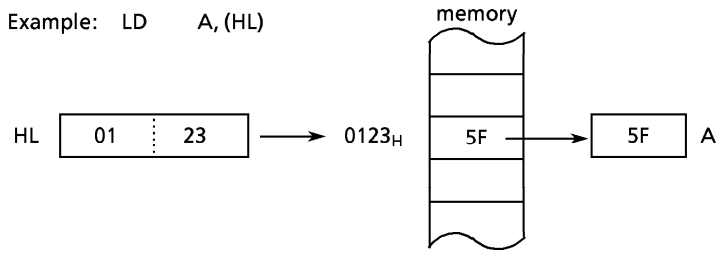
1.4 ADDRESSING MODES

There are 20 types of addressing modes (effective address specification methods) in the TLCS-870 Series. Note that multiple addressing modes may be combined depending on the instruction.

(1) Register-Indirect

① Register-Indirect **(HL)** or **(DE)**

Effective address is specified by the contents of the register pair (HL or DE) in the current register bank.

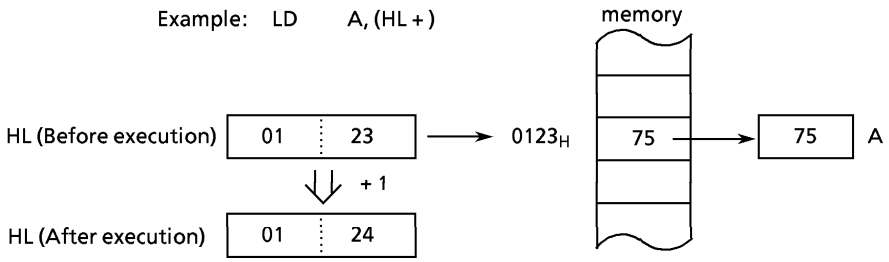


The contents ($5F_H$) of the memory address specified by the HL register pair contents (0123_H) are loaded into the accumulator.

② Register-Indirect with Auto-post-increment **(HL+)**

Effective address is specified by the contents of the HL register pair. After transfer, the HL register pair is automatically incremented by one. Note that there is no flag change due to increment of the HL register pair. Consequently, this is convenient for arithmetic and logical operations on multiple digits.

Note: Do not give the address of the H or L register in the current register bank to the effective address. For example, if the current bank is bank 0, do not make the HL contents 0046_H or 0047_H in this addressing mode.

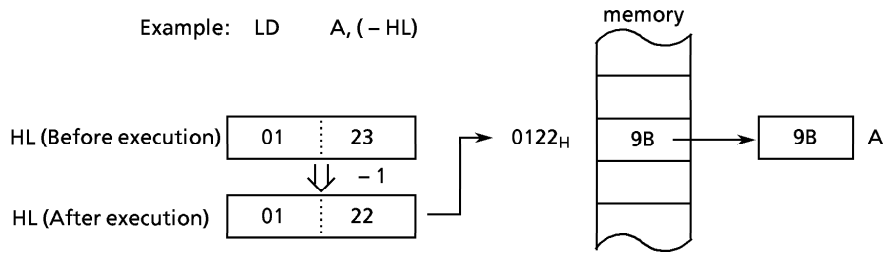


The contents (75_H) of the memory address (0123_H) specified by the HL register pair contents are loaded into the accumulator. After that, the HL register pair are incremented to become 0124_H.

③ Register-Indirect with Auto-pre-decrement **(–HL)**

First, the HL register pair is automatically decremented by one. Effective address is specified by the contents of the decremented HL register pair.

Note that there is no flag change due to decrement of the HL register pair. A software LIFO (Last-In First-Out) can be formed by combination with the addressing mode in ② above.



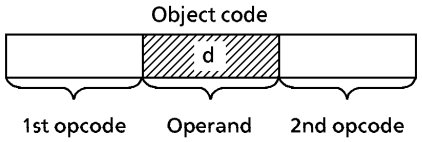
The HL register pair is decremented, and become 0122_H. The contents ($9B_H$) of the memory address (0122_H) specified by the contents of the decremented HL register pair are loaded into the accumulator.

(2) Indexed

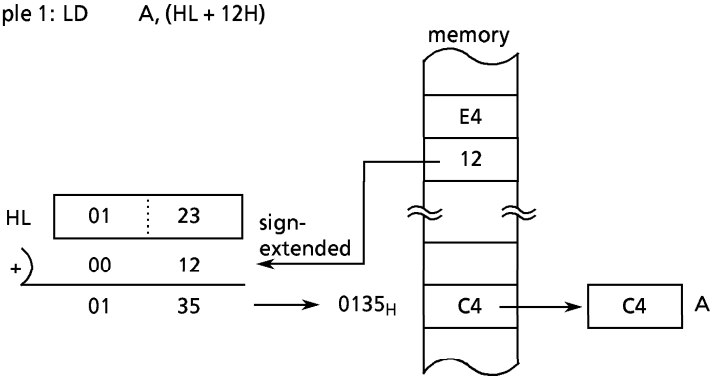
① 8-bit displacement Offset Indexed **(HL + d)**

Effective address is obtained by adding the sign-extended 8-bit displacement "d" (see table below) following the first opcode to the contents of the HL register pair. The contents of the HL register pair are not changed.

Displacement "d"	Sign-extended value
00~7F$_H$	0000~007F$_H$ (0~ + 127)
80~FF$_H$	FF80~FFFF$_H$ (– 128~ – 1)

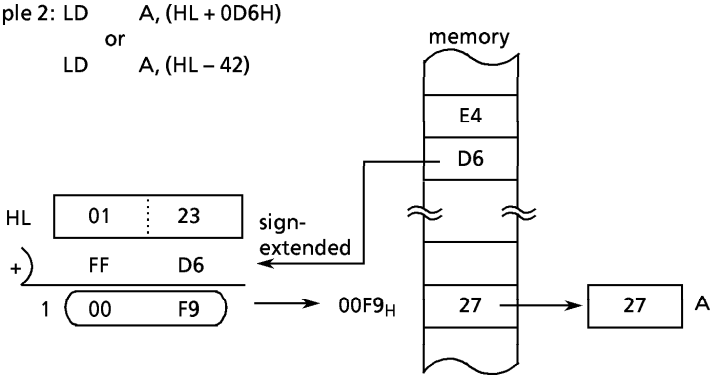


Example 1: LD A, (HL + 12H)



The contents (C4$_H$) of the memory address (0135$_H$) specified by the value obtained by adding the value (0012$_H$), which is the sign-extended displacement (12$_H$), to the contents of the HL register pair are loaded into the accumulator.

Example 2: LD A, (HL + 0D6H)
or
LD A, (HL – 42)



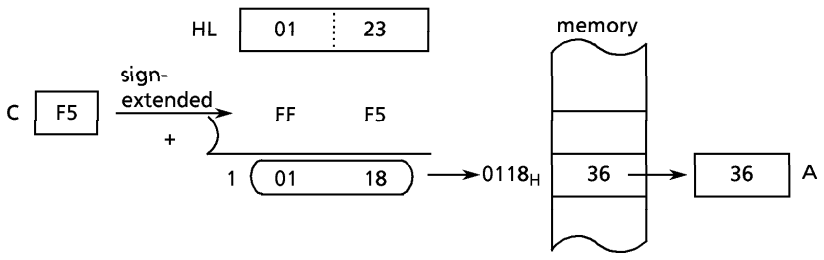
The contents (27$_H$) of the memory address (00F9$_H$) specified by the value obtained by adding the value (FFD6$_H$), which is the sign-extended displacement (D6$_H$), to the contents of the HL register pair (0123$_H$) are loaded into the accumulator.

② Register-Indexed **(HL + C)**

Effective address is obtained by adding the value (see table at right), which is the sign-extended C register contents, to the HL register pair contents. Note that the contents of the HL register pair and the C register are not changed. This addressing mode can be used only in specifications of source memory address.

C Register	Sign-extended value
00~7F$_H$	0000~007F$_H$ (0~ + 127)
80~FF$_H$	FF80~FFFF$_H$ (– 128~ – 1)

Example: LD A, (HL + C)



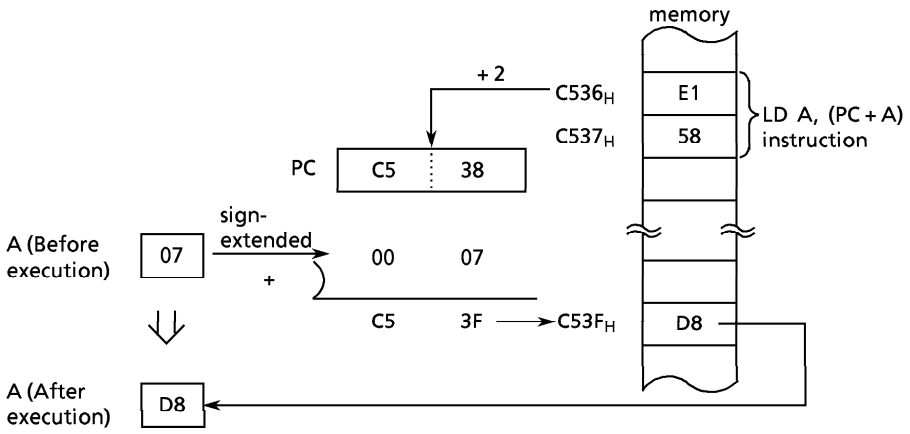
The contents (36$_H$) of the memory address (0118$_H$) specified by the value obtained by adding the value (FFF5$_H$), which is the sign-extended C register contents (F5$_H$), to the contents (0123$_H$) of the HL register pair are loaded into the accumulator.

③ Register-Offset Relative Indexed **(PC + A)**

Effective address is obtained by adding the sign-extended accumulator contents (see table at right) to the current contents (header address of execution instruction + 2) of the program counter. This addressing mode can be used only in specifications of source memory address. By using this addressing mode, code conversions such as BCD code to 7-segment code, table look-up, table search and n-ways multiple direction branching processing can be easily programmed.

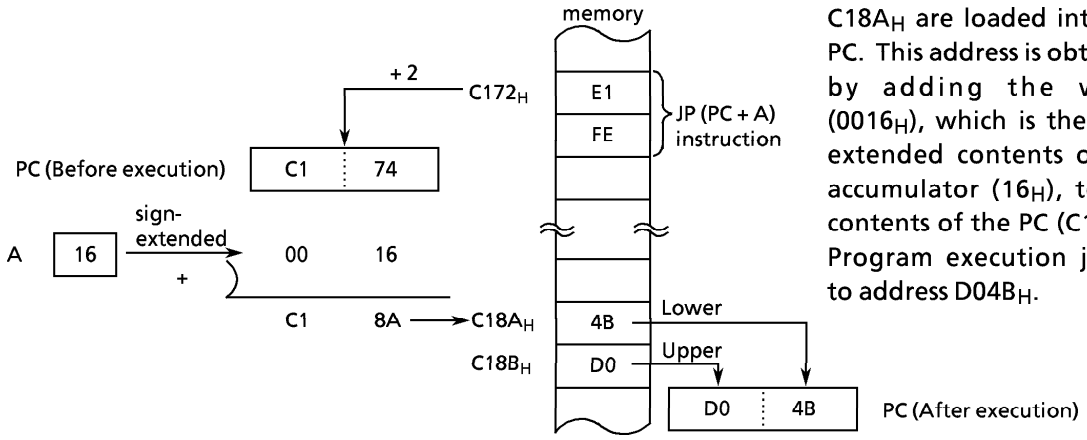
Accumulator	Sign-extended value
$00 \sim 7F_H$	$0000 \sim 007F_H$ (0 ~ + 127)
$80 \sim FF_H$	$FF80 \sim FFFF_H$ (– 128 ~ – 1)

Example 1: LD A, (PC + A)



The contents ($D8_H$) of the memory address ($C53F_H$) specified by the value obtained by add-ing the value (0007_H), which is the sign-extended contents of the accumulator (07_H), to the current contents of the PC ($C538_H$) are loaded into the accumulator.

Example 2: JP (PC + A)

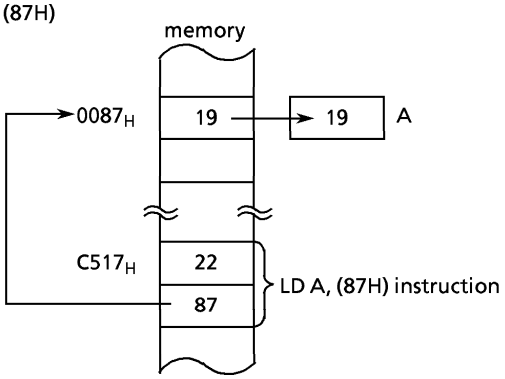


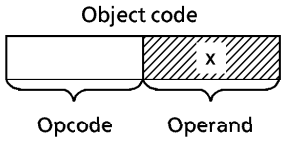
Memory contents ($D04B_H$) in the 2 consecutive bytes starting from address $C18A_H$ are loaded into the PC. This address is obtained by adding the value (0016_H), which is the sign-extended contents of the accumulator (16_H), to the contents of the PC ($C174_H$). Program execution jumps to address $D04B_H$.

(3) Direct **(x)**

Effective address is specified by an 8-bit data "x" in the object code. This addressing allows to directly address the lowest 256 bytes (0000-00FF$_H$) in memory. This memory space includes the special function registers and 192 bytes of on-chip RAM.

Example: LD A, (87H)





The contents of the memory address (0087$_H$) directly specified by the operand "x" (87$_H$) are loaded into the accumulator.

(4) Register **r or rr**

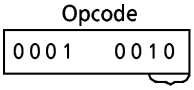
The register which is specified by the register specification field in the opcode becomes the operation object.

Example 1: LD A, B



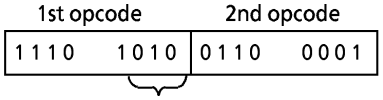
register specification field
011→B register

Example 2: INC DE



register specification field
10→DE register pair

Example 3: ADD A, C



register specification field
010→C register

(5) Implied (Inherent)

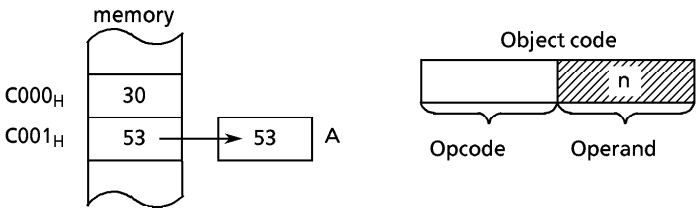
Information such as register specifications is contained in the opcode.

Example 1: SET CF Example 2: SHLC A Example 3: RET Example 4: NOP

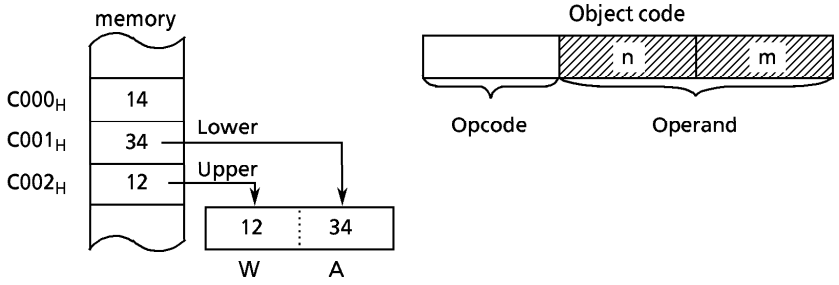
(6) Immediate **n** or **mn**

The immediate data (constants) in the object code becomes the operation object. Note that when the data is 16-bit immediate data, it is stored in memory in ascending order from the lower 8 bits to the upper 8 bits.

Example 1: LD A, 53H



Example 2: LD WA, 1234H



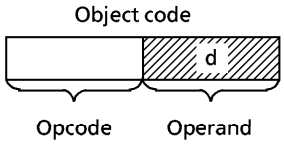
Note: Immediate data cannot be enclosed in parentheses "(" and ")" in assembler source program descriptions. If immediate data is enclosed in parentheses, it will be considered to be in the direct addressing mode.

(7) Relative

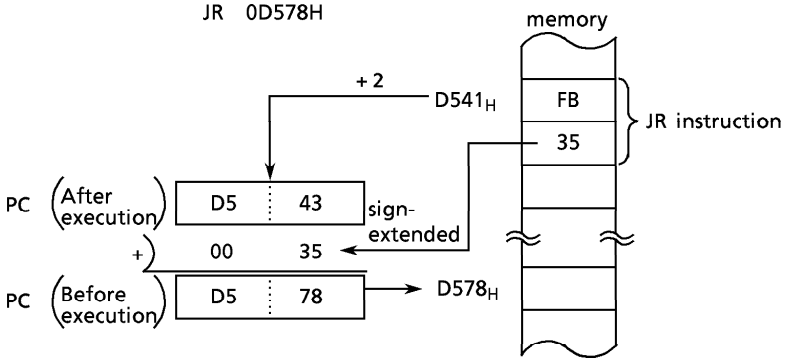
① 8-bit displacement PC-Relative

Effective address is obtained by adding the sign-extended 8-bit displacement "d" following the opcode (see table below) to the current contents of the program counter (header address of execution instruction + 2). This addressing mode is only used in the JR instructions.

Displacement "d"	Sign-extended value
00~7F$_H$	0000~007F$_H$ (0~ + 127)
80~FF$_H$	FF80~FFFF$_H$ (– 128~ – 1)



Example: JR $ + 2 + 35H
or
JR 0D578H



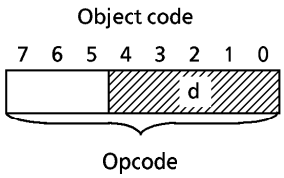
The program execution jumps to D578$_H$, the address specified by the value obtained by adding the sign-extended displacement (35$_H$) to the current contents of the PC (D543$_H$).

Note: Symbol "$" means the header address of the current execution instruction.

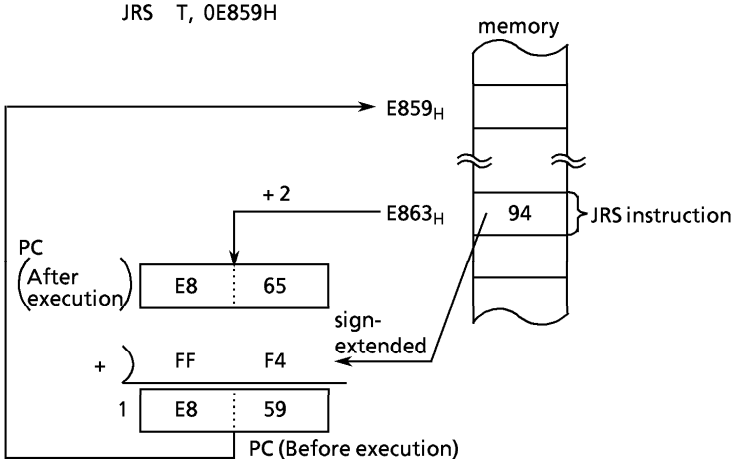
② 5-bit displacement PC-Relative

Effective address is obtained by adding the sign-extended 5-bit displacement "d" in the opcode to the current contents of the program counter (header address of execution instruction + 2). This addressing mode is only used in the JRS instructions.

Displacement "d"	Sign-extended value
00~0F$_H$	0000~000F$_H$ (0~ + 15)
10~1F$_H$	FFF0~FFFF$_H$ (– 16~ – 1)



Example: JRS T, $ + 2 + 14H
or
JRS T, 0E859H

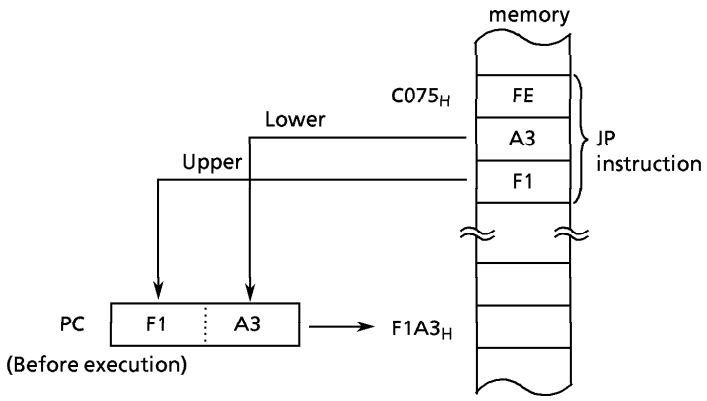


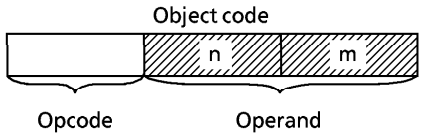
If the jump status flag is "1", a jump is made to the address E859$_H$ specified by the data obtained by adding the sign-extended displacement (14$_H$) to the current contents of the PC (E865$_H$).

(8) Absolute

Effective address is specified by a 16-bit data "mn" (stored in sequence from the lower 8 bits to the upper 8 bits) in the object code. This addressing mode is used in JP mn and CALL mn instructions. These instructions are capable of addressing 64Kbytes of momery.

Example: JP 0F1A3H

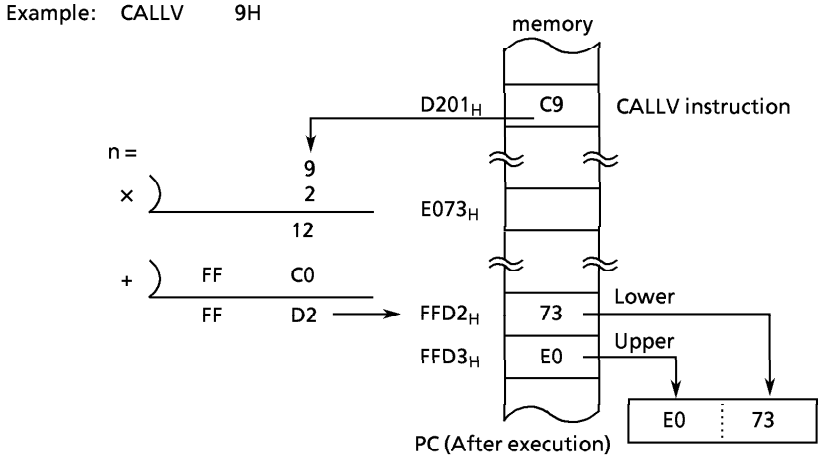


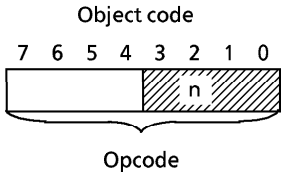


A jump is made to address F1A3$_H$ specified by the operand.

(9) Vector

Effective address is specified by 16-bit data (vector address) that is read out from program memory, the address of which is the doubled 4-bit data "n" in the opcode that is added to $FFC0_H$. This addressing mode is only used in the CALLV instruction.



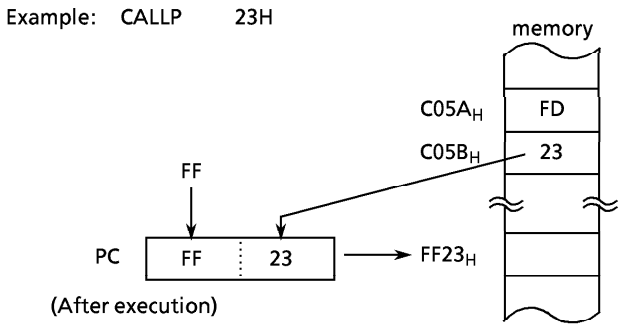


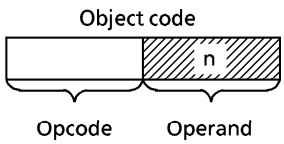
The contents ($E073_H$) in the 2 consecutive bytes starting from address $FFD2_H$, which is the data 0btained by adding $FFC0_H$ to the doubled "n" (9_H) in the opcode, are called.

Note: When a label is used in the reference address of the CALLV instruction, there are items limited by the format of the assembler source program. For more detailed information, see the "TLCS-870 development system manual".

(10) Page

Effective address ($FF00\text{-}FFFF_H$) is specified by the value obtained by adding the 8-bit data "n" in the object code to $FF00_H$. This addressing mode is only used in the CALLP instruction.





Effective address ($FF23_H$), which has its lower 8 bits as the operand "n" (23_H) and its upper 8 bits as FF_H, is called.

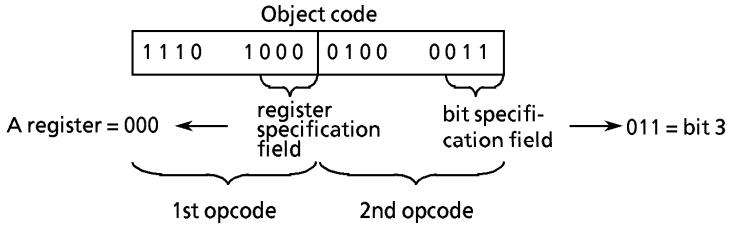
(11) Direct Bit

① Register Bit

The register bit, which is specified by the register specification field and the bit specification field in the opcode, becomes the operation object.

Example: SET A.3

The content of bit 3 in the accumulator is set to "1".



② Register-Indirect Memory Bit

The bit directly specified in the bit specification field in the opcode at the memory address specified by register-indirect addressing modes (HL), (DE), (HL+) and (−HL) become the operation object.

Example: SET (HL). 1

③ Indexed Memory Bit

The bit directly specified in the bit specification field in the opcode at the memory address specified by the index addressing modes (HL + d), (HL + C) and (PC + A) become the operation object.

Example: SET (HL + 57H). 6

④ Direct Memory Bit

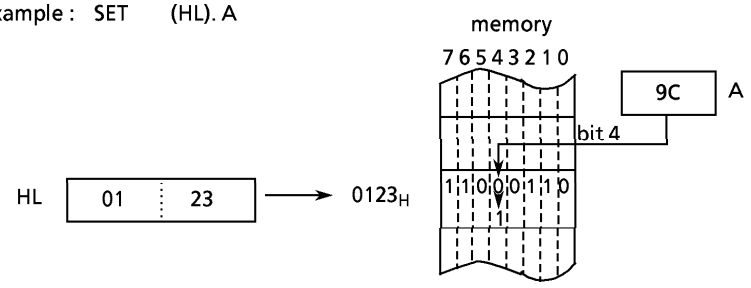
The bit directly specified in the bit specification field in the opcode at the memory address (0000-00FF$_H$) specified by the direct addressing mode (x) become the operation object.

Example: SET (0058H). 3

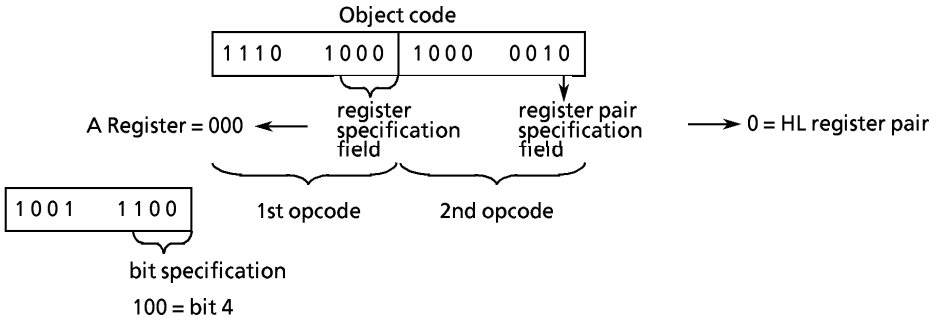
(12) Register-Indirect Bit

The bit specified by the contents of the lower 3 bits in the 8-bit register specified by the register specification field in the opcode at the memory address specified by the contents of the register pair (HL or DE) become the operation object.

Example : SET (HL). A



Bit 4, which is specified by the contents (100_B) of the lower 3 bits of the accumulator in the contents (11000110_B) at the memory address specified by the contents (0123_H) of the HL register pair, is set to "1" to become 11010110_B.)



1.5 DESCRIPTION OF INSTRUCTIONS

(1) DATA TRANSFER, EXCHANGE

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
LD A, r	0101 0rrr	5r	A←r	1	Z	–	–	1
	The contents of the register r are loaded into the accumulator. The zero flag is set to "1" if r = 00_H ; otherwise it is cleared to "0". The jump status flag is always set to "1".							
LD r, A	0101 1rrr	5 8+r	r←A	1	Z	–	–	1
	The contents of the accumulator are loaded into the register r. The zero flag is set to "1" if A = 00_H ; otherwise it is cleared to "0".							
LD r, g	1110 1ggg 0101 1rrr	E 8+g 5 8+r	r←g	1	Z	–	–	2
	The contents of the register g are loaded into the register r. The zero flag is set to "1" if g = 00_H ; otherwise it is cleared to "0".							
LD rr, gg	1110 10gg 0001 01rr	E 8+gg 1 4+rr	rr←gg	1	–	–	–	2
	The contents of the register pair gg are loaded into the register pair rr.							
LD SP, gg	1110 10gg 1111 1010	E 8+gg FA	SP←gg	1	–	–	–	3
	The contents of the register pair gg are loaded into the stack pointer (SP).							
LD gg, SP	1110 10gg 1111 1011	E 8+gg FB	gg←SP	1	–	–	–	3
	The contents of the stack pointer (SP) are loaded into the register pair gg.							
LD A, (x)	0010 0010 xxxx xxxx	22 xx	A←(x)	1	Z	–	–	3
	The contents of the memory address (0000-00FF_H) directly specified by the "x" in the object code are loaded into the accumulator. If the data to be transferred into the accumulator is 00_H, the zero flag is set to "1"; otherwise it is cleared to "0".							
LD r, (x)	1110 0000 xxxx xxxx 0101 1rrr	E0 xx 5 8+r	r←(x)	1	Z	–	–	4
	The contents of the memory address (0000-00FF_H) directly specified by the "x" in the object code are loaded into the register r. If the data to be transferred into the register r is 00_H, the zero flag is set to "1"; otherwise it is cleared to "0".							
LD A, (HL)	0010 0011	23	A←(HL)	1	Z	–	–	2
	The contents of the memory address specified by the contents of the HL register pair are loaded into the accumulator. If the data to be transferred into the accumulator is 00_H, the zero flag is set to "1".							
LD r, (pp)	1110 001p 0101 1rrr	E 2+pp 5 8+r	r←(pp)	1	Z	–	–	3
	The contents of the memory address specified by the contents of register pair pp (HL or DE) are loaded into the register r. If the data to be transferred into the register r is 00_H, the zero flag is set to "1".							
LD r, (HL+)	1110 0110 0101 1rrr	E6 5 8+r	r←(HL), HL←HL+1	1	Z	–	–	4
	The contents of the memory address specified by the contents of the HL register pair are loaded into the register r. Then, the HL register pair is incremented by one. If the data to be transferred into the register r is 00_H, the zero flag is set to "1". Note that the H or the L register cannot be specified for the register r. Example : When HL = 0123_H and the memory contents at address 0123_H are 56_H, after executing the LD A, (HL+) instruction, the A register will become 56_H, the HL register pair will become 0124_H and the zero flag will be cleared to "0".							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
LD r, (−HL)	1110 0111 0101 1rrr	E7 5 8+r	HL←HL−1, r←(HL)	1	Z	−	−	4
The HL register pair is decremented by one, and the contents of the memory address specified by the new contents of the HL register pair are loaded into the register r. If the data to be transferred into the register r is 00_H, the zero flag is set to "1".								
LD r, (HL+d)	1110 0100 dddd dddd 0101 1rrr	E4 dd 5 8+r	r←(HL+d)	1	Z	−	−	5
The contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair are loaded into the register r. If the data to be transferred into the register r is 00_H, the zero flag is set to "1".								
LD r, (HL+C)	1110 0101 0101 1rrr	E5 5 8+r	r←(HL+C)	1	Z	−	−	5
The contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair are loaded into the register r. If the data to be transferred into the register r is 00_H, the zero flag is set to "1".								
LD r, (PC+A)	1110 0001 0101 1rrr	E1 5 8+r	r←(PC+A)	1	Z	−	−	5
The contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the current contents of the program counter (PC) are loaded into the register r. If the data to be transferred into register r is 00_H, the zero flag is set to "1". This instruction is suitable for code conversion processing.								
LD rr, (x)	1110 0000 xxxx xxxx 0001 01rr	E0 xx 1 4+rr	rr←(x+1, x)	1	−	−	−	5
The contents of memory in 2 bytes from the address (0000-00FF$_H$) directly specified by the "x" in the object code are loaded into the register pair rr. Example : When the memory contents at addresses 0072_H and 0073_H are $8E_H$ and 59_H respectively, after executing the LD WA, (72H) instruction, the results will become W = 59_H and A = $8E_H$.								
LD rr, (pp)	1110 001p 0001 01rr	E 2+pp 1 4+rr	rr←(pp+1, pp)	1	−	−	−	4
The contents of memory in 2 bytes from the address specified by the contents of the register pair pp (HL or DE) are loaded into the register pair rr.								
LD rr, (HL+d)	1110 0100 dddd dddd 0001 01rr	E4 dd 1 4+rr	rr←(HL+d+1, HL+d)	1	−	−	−	6
The contents of the memory in 2 bytes from the address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair are loaded into the register pair rr.								
LD rr, (HL+C)	1110 0101 0001 01rr	E5 1 4+rr	rr←(HL+C+1, HL+C)	1	−	−	−	6
The contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair are loaded into the register pair rr.								
LD rr, (PC+A)	1110 0001 0001 01rr	E1 1 4+rr	rr←(PC+A+1, PC+A)	1	−	−	−	6
The contents of the memory in 2 bytes from the address specified by the value obtained by adding the sign-extended contents of the accumulator to the current contents of the program counter are loaded into the register pair rr. This instruction is suitable for code conversion processing. Example: When A = 43_H and the memory contents at address $C058_H$ and $C059_H$ are $6A_H$ and 18_H respectively, after executing the LD HL, (PC + A) instruction at $C013_H$, the results will become HL = $186A_H$ and JF = 1.								

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
LD (x), A	0010 1010 xxxx xxxx	2A xx	(x)←A	1	–	–	–	3
	The contents of the accumulator are stored in the address (0000-00FF$_H$) directly specified by the "x" in the object code.							
LD (x), r	1111 0000 xxxx xxxx 0101 0rrr	F0 xx 5r	(x)←r	1	–	–	–	4
	The contents of the register r are stored in the memory address (0000-00FF$_H$) directly specified by the "x" in the object code.							
LD (HL), A	0010 1011	2B	(HL)←A	1	–	–	–	2
	The contents of the accumulator are stored in the memory address specified by the contents of the HL register pair.							
LD (pp), r	1111 001p 0101 0rrr	F 2+pp 5r	(pp)←r	1	–	–	–	3
	The contents of the register r are stored in the memory address specified by the contents of the register pair pp (HL or DE).							
LD (HL+), r	1111 0110 0101 0rrr	F6 5r	(HL)←r, HL←HL+1	1	–	–	–	4
	The contents of the register r are stored in the memory addresses specified by the contents of the HL register pair. Then, the HL register pair is incremented by one. Note that the H or L register cannot be specified for register r.							
LD (–HL), r	1111 0111 0101 0rrr	F7 5r	HL←HL–1, (HL)←r	1	–	–	–	4
	The HL register pair is decremented by one. And the contents of register r are stored in the memory address specified by the new contents of the HL register pair.							
LD (HL+d), r	1111 0100 dddd dddd 0101 0rrr	F4 dd 5r	(HL+d)←r	1	–	–	–	5
	The contents of the register r are stored in the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair.							
LD (x), rr	1111 0000 xxxx xxxx 0001 00rr	F0 xx 1 rr	(x+1, x)←rr	1	–	–	–	5
	The contents of the register pair rr are stored, in the sequence lower followed by upper order, in the 2 consecutive bytes in memory starting from the address (0000-00FF$_H$) directly specified by the "x" in the object code.							
LD (pp), rr	1111 001p 0001 00rr	F 2+pp 1 rr	(pp+1, pp)←rr	1	–	–	–	4
	The contents of the register pair rr are stored, in the sequence lower followed by upper order, in the 2 consecutive bytes in memory starting from the address specified by the contents of the register pair pp (HL or DE).							
LD (HL+d), rr	1111 0100 dddd dddd 0001 00rr	F4 dd 1 rr	(HL+d+1, HL+d)←rr	1	–	–	–	6
	The contents of the register pair rr are stored, in the sequence lower followed by upper order, in the 2 consecutive bytes in memory starting from the address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
LD (x), (y)	0010 0110 yyyy yyyy xxxx xxxx	26 yy xx	(x)←(y)	1	Z	–	–	5
	The contents of the memory address (0000-00FF$_H$) directly specified by the "y" in the object code are transferred into the memory address directly specified by the "x" in the object code. If the data to be transferred is 00$_H$, the zero flag is set to "1".							
LD (x), (pp)	1110 001p 0010 0110 xxxx xxxx	E 2+pp 26 xx	(x)←(pp)	1	U	–	–	5
	The contents of the memory address specified by the contents of the register pair pp (HL or DE) are transferred into the memory address (0000-00FF$_H$) directly specified by the "x" in the object code.							
LD (x), (HL+d)	1110 0100 dddd dddd 0010 0110 xxxx xxxx	E4 dd 26 xx	(x)←(HL+d)	1	U	–	–	7
	The contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair are transferred into the memory address (0000-00FF$_H$) directly specified by the "x" in the object code.							
LD (x), (HL+C)	1110 0101 0010 0110 xxxx xxxx	E5 26 xx	(x)←(HL+C)	1	U	–	–	7
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair are transferred into the memory address (0000-000FF$_H$) directly specified by the "x" in the object code.							
LD (x), (PC+A)	1110 0001 0010 0110 xxxx xxxx	E1 26 xx	(x)←(PC+A)	1	U	–	–	7
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the current contents of the program counter are transferred into the memory address (0000-00FF$_H$) directly specified by the "x" in the object code.							
LD (HL), (x)	1110 0000 xxxx xxxx 0010 0111	E0 xx 27	(HL)←(x)	1	Z	–	–	5
	The contents of the memory address (0000-00FF$_H$) directly specified by "x" in the object code are transferred into the memory address specified by the contents of the HL register pair. When the data to be transferred is 00$_H$, the zero flag is set to "1".							
LD (HL), (pp)	1110 001p 0010 0111	E 2+pp 27	(HL)←(pp)	1	Z	–	–	4
	The contents of the memory address specified by the contents of the register pair pp (HL or DE) are transferred into the memory address specified by the contents of the HL register pair. When the data to be transferred is 00$_H$, the zero flag is set to "1". This LD (HL), (DE) instruction is convenient for block transfer processing.							
LD (HL), (HL+d)	1110 0100 dddd dddd 0010 0111	E4 dd 27	(HL)←(HL+d)	1	Z	–	–	6
	The contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair are transferred into the memory address specified by the contents of the HL register pair. When the data to be transferred is 00$_H$, the zero flag is set to "1".							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
LD (HL), (HL+C)	1110 0101 0010 0111	E5 27	(HL)←(HL+C)	1	Z	–	–	6
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair are transferred into the memory address specified by the contents of the HL register pair. When the data to be transferred is 00_H, the zero flag is set to "1".							
LD (HL), (PC+A)	1110 0001 0010 0111	E1 27	(HL)←(PC+A)	1	Z	–	–	6
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the current contents of the program counter are transferred into the memory address specified by the contents of the HL register pair. When the data to be transferred is 00_H, the zero flag is set to "1".							
LD r, n	0011 0rrr nnnn nnnn	3r nn	r←n	1	–	–	–	2
	The 8-bit immediate data "n" in the object code is loaded into the register r.							
LD rr, mn	0001 01rr nnnn nnnn mmmm mmmm	1 4+rr nn mm	rr←mn	1	–	–	–	3
	The 16-bit immediate data "mn" in the object code are loaded into the register pair rr.							
LD SP, mn	1111 1010 nnnn nnnn mmmm mmmm	FA nn mm	SP←mn	1	–	–	–	3
	The 16-bit immediate data "mn" in the object code are loaded into the stack pointer.							
LD RBS, n	0000 1111 0000 nnnn	0F 0n	RBS←n (0≦n≦15)	1	–	–	–	2
	The 4-bit immediate data "n" in the object code is loaded into the register bank selector. This is register bank switching instruction.							
CLR (x)	0010 1110 xxxx xxxx	2E xx	(x)←0	1	–	–	–	4
	00_H is stored in the memory address (0000-00FF$_H$) directly specified by the "x" in the object code. Example : After executing CLR (58H) instruction, the memory contents of address 0058_H will be cleared to 00_H.							
LD (x), n	0010 1100 xxxx xxxx nnnn nnnn	2C xx nn	(x)←n	1	–	–	–	4
	The 8-bit immediate data "n" in the object code is stored in the memory address (0000-00FF$_H$) directly specified by the "x" in the object code. If n = 00_H, the shortened form of the CLR (x) instruction can also be used.							
CLR (HL)	0010 1111	2F	(HL)←0	1	–	–	–	2
	00_H is stored in the memory address specified by the contents of the HL register pair.							
LD (HL), n	0010 1101 nnnn nnnn	2D nn	(HL)←n	1	–	–	–	3
	The 8-bit immediate data "n" in the object code is stored in the memory address specified by the contents of the HL register pair. If n = 00_H, the shortened form of the CLR (HL) instruction can also be used.							
LD (DE), n	1111 0010 0010 1100 nnnn nnnn	F2 2C nn	(DE)←n	1	–	–	–	4
	The 8-bit immediate data "n" in the object code is stored in the memory address specified by the contents of the DE register pair.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
LD (HL+), n	1111 0110 0010 1100 nnnn nnnn	F6 2C nn	(HL)←n, HL←HL+1	1	–	–	–	5
The 8-bit immediate data "n" in the object code is stored in the memory address specified by the contents of the HL register pair. Then, the HL register pair is incremented by one.								
LD (–HL), n	1111 0111 0010 1100 nnnn nnnn	F7 2C nn	HL←HL–1, (HL)←n	1	–	–	–	5
The HL register pair is decremented by one, and the 8-bit immediate data "n" in the object code is stored in the memory address specified by the new contents of the HL register pair.								
LD (HL+d), n	1111 0100 dddd dddd 0010 1100 nnnn nnnn	F4 dd 2C nn	(HL+d)←n	1	–	–	–	6
The 8-bit immediate data "n" in the object code is stored in the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair.								
LDW (x), mn	0010 0100 xxxx xxxx nnnn nnnn mmmm mmmm	24 xx nn mm	(x+1, x)←mn	1	–	–	–	6
The 16-bit immediate data "mn" in the object code are stored, in the sequence lower followed by upper order, in the 2 consecutive bytes in memory starting from the address directly specified by the "x" in the object code. Example: When LDW (73H), 1234H is executed, 34_H is written into memory address 0073_H and 12_H is written into memory address 0074_H.								
LDW (HL), mn	0010 0101 nnnn nnnn mmmm mmmm	25 nn mm	(HL+1, HL)←mn	1	–	–	–	5
The 16-bit immediate data "mn" in the object code are stored, in the sequence of lower followed by upper order, in the 2 consecutive bytes in memory starting from the address specified by the contents of the HL register pair.								
XCH r, g	1110 1ggg 1010 1rrr	E 8+g A 8+r	r↔g	1	Z	–	–	3
The contents of the register r are exchanged for the contents of the register g. When the contents of the register g are 00_H before the exchange, the zero flag is set to "1". When the contents are other than 00_H, the zero flag is cleared to "0". Example: When A = $3C_H$ and B = $5F_H$, after executing the XCH A, B instruction, the results will become A = $5F_H$, B = $3C_H$, ZF = 0, and JF = 1.								
XCH rr, gg	1110 10gg 0001 00rr	E 8+gg 1 0+rr	rr↔gg	1	–	–	–	3
The contents of the register pair rr are exchanged for the contents of the register pair gg. Example: When HL = 0123_H and DE = 9587_H, after executing the XCH HL, DE instruction, the results will become HL = 9587_H and DE = 0123_H.								
XCH r, (x)	1110 0000 xxxx xxxx 1010 1rrr	E0 xx A 8+r	r↔(x)	1	Z	–	–	5
The contents of the memory address (000-00FF_H) directly specified by the "x" in the object code are exchanged for the register r. When the data to be transferred into the register r is 00_H, the zero flag is set to "1".								

MNEMONIC	OBJECT CODE		OPERATION	FLAG				CYCLE
	Binary	Hex-decimal		JF	ZF	CF	HF	
XCH r, (pp)	1110 001p 1010 1rrr	E 2+pp A 8+r	r↔(pp)	1	Z	–	–	4
	The contents of the memory address specified by the contents of the register pair pp (HL or DE) are exchanged for the contents of the register r. When the data to be transferred into the register r is 00_H, the zero flag is set to "1".							
XCH r, (HL+)	1110 0110 1010 1rrr	E6 A 8+r	r↔(HL), HL←HL+1	1	Z	–	–	5
	The contents of the memory address specified by the contents of the HL register pair are exchanged for the contents of the register r. Then, the HL register pair is incremented by one. When the data to be transferred into the register r is 00_H, the zero flag is set to "1". Note that the H or L register cannot be specified for register r.							
XCH r, (–HL)	1110 0111 1010 1rrr	E7 A 8+r	HL←HL–1, r↔(HL)	1	Z	–	–	5
	The HL register pair is decremented by one, and the contents of the memory address specified by that value are exchanged for the register r. When the data to be transferred into the register r is 00_H, the zero flag is set to "1".							
XCH r, (HL+d)	1110 0100 dddd dddd 1010 1rrr	E4 dd A 8+r	r↔(HL+d)	1	Z	–	–	6
	The contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair are exchanged for the contents of the register r. When the data to be transferred into the register r is 00_H, the zero flag is set to "1".							
XCH r, (HL+C)	1110 0101 1010 1rrr	E5 A 8+r	r↔(HL+C)	1	Z	–	–	6
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair are exchanged for the contents of the register r. When the data to be transferred into the register r is 00_H, the zero flag is set to "1".							
XCH r, (PC+A)	1110 0001 1010 1rrr	E1 A 8+r	r↔(PC+A)	1	Z	–	–	6
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter are exchanged for the contents of the register r. When the data to be transferred into the register r is 00_H, the zero flag is set to "1".							

Note: During XCH r, (src) instruction, the contents of the output latch are read in if the I/O port (except the tri-state port) is accessed as the source memory.

(2) ARITHMETIC & LOGICAL OPERATION

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
CMP A , g	1110 1ggg 0110 0111	E 8+g 67	A − g	Z	Z	C	H	2
The contents of the accumulator are compared with the contents of the register g. If A<g, then the carry flag is set to "1". If A≧g, then the carry flag is cleared to "0".								
CMP g , A	1110 1ggg 0110 1111	E 8+g 6F	g − A	Z	Z	C	H	3
The contents of the register g are compared with the contents of the accumulator. If g>A, then the carry flag is set to "1". If g≧A, then the carry flag is cleared to "0".								
CMP WA, gg	1110 10gg 0011 0111	E 8+gg 37	WA − gg	Z	Z	C	U	4
The contents of the 16-bit accumulator (WA) are compared with the contents of the register pair gg. If WA<gg, then the carry flag is set to "1". If WA≧gg, then the carry flag is cleared to "0".								
CMP A, (x)	0111 1111 xxxx xxxx	7F xx	A − (x)	Z	Z	C	H	4
The contents of the accumulator are compared with the contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code.								
CMP A, (pp)	1110 001p 0111 1111	E 2+pp 7F	A − (pp)	Z	Z	C	H	3
The contents of the accumulator are compared with the contents of the memory address specified by the contents of the register pair pp (HL or DE).								
CMP A, (HL+)	1110 0110 0111 1111	E6 7F	A − (HL) , HL←HL+1	Z	Z	C	H	4
The contents of the accumulator are compared with the contents of the memory address specified by the contents of the HL register pair. Then, the HL register pair is incremented by one.								
CMP A, (−HL)	1110 0111 0111 1111	E7 7F	HL←HL−1, A − (HL)	Z	Z	C	H	4
The HL register pair is decremented by one, and the contents of the memory address specified by the new contents of the HL register pair are compared with the contents of the accumulator.								
CMP A, (HL+d)	1110 0100 dddd dddd 0111 1111	E4 dd 7F	A − (HL+d)	Z	Z	C	H	5
The contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair are compared with the contents of the accumulator.								
CMP A, (HL+C)	1110 0101 0111 1111	E5 7F	A − (HL+C)	Z	Z	C	H	5
The contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair are compared with the contents of the accumulator.								
CMP A, (PC+A)	1110 0001 0111 1111	E1 7F	A − (PC+A)	Z	Z	C	H	5
The contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter are compared with the contents of the accumulator.								
CMP (x), (HL)	1110 0000 xxxx xxxx 0110 0111	E0 xx 67	(x) − (HL)	Z	Z	C	H	6
The contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code are compared with the contents of the memory address specified by the contents of the HL register pair.								

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
CMP (pp), (HL)	1110 001p 0110 0111	E 2+pp 67	(pp) − (HL)	Z	Z	C	H	5
	The contents of the memory address specified by the contents of the register pair pp (HL or DE) are compared with the contents of the memory address specified by the contents of the HL register pair.							
CMP (HL+d), (HL)	1110 0100 dddd dddd 0110 0111	E4 dd 67	(HL+d) − (HL)	Z	Z	C	H	7
	The contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair are compared with the contents of the memory address specified by the contents of the HL register pair.							
CMP (HL+C), (HL)	1110 0101 0110 0111	E5 67	(HL+C) − (HL)	Z	Z	C	H	7
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair are compared with the contents of the memory address specified by the contents of the HL register pair.							
CMP (PC+A), (HL)	1110 0001 0110 0111	E1 67	(PC+A) − (HL)	Z	Z	C	H	7
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter are compared with the contents of the memory address specified by the contents of the HL register pair.							
CMP A, n	0111 0111 nnnn nnnn	77 nn	A − n	Z	Z	C	H	2
	The contents of the accumulator are compared with the immediate data "n" in the object code. If A<n, the carry flag is set to "1". If A≧n, it is cleared to "0". The contents of the accumulator are not changed.							
CMP g, n	1110 1ggg 0111 0111 nnnn nnnn	E 8+g 77 nn	g − n	Z	Z	C	H	3
	The contents of the register g are compared with the immediate data "n" in the object code. If g<n, the carry flag is set to "1". If g≧n, it is cleared to "0". The contents of the register g are not changed.							
CMP gg, mn	1110 10gg 0011 1111 nnnn nnnn mmmm mmmm	E 8+gg 3F nn mm	gg − mn	Z	Z	C	U	4
	The contents of the register pair gg are compared with the 16-bit immediate data "mn" in the object code. If gg<mn, the carry flag is set to "1". If gg≧mn, it is cleared to "0".							
CMP (x), n	1110 0000 xxxx xxxx 0111 0111 nnnn nnnn	E0 xx 77 nn	(x) − n	Z	Z	C	H	5
	The contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code are compared with the immediate data "n" in the object code.							
CMP (pp), n	1110 001p 0111 0111 nnnn nnnn	E 2+pp 77 nn	(pp) − n	Z	Z	C	H	4
	The contents of the memory address specified by the contents of the register pair pp (HL or DE) are compared with the immediate data "n" in the object code.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
CMP (HL+), n	1110 0110 0111 0111 nnnn nnnn	E6 77 nn	(HL)−n, HL←HL+1	Z	Z	C	H	5
	The contents of the memory address specified by the contents of the HL register pair are compared with the immediate data "n" in the object code. Then, the HL register pair is incremented by one.							
CMP (−HL), n	1110 0111 0111 0111 nnnn nnnn	E7 77 nn	HL←HL−1, (HL)−n	Z	Z	C	H	5
	The HL register pair is decremented by one, and the contents of the memory address specified by new contents of the HL register pair are compared with the immediate data "n" in the object code.							
CMP (HL+d), n	1110 0100 dddd dddd 0111 0111 nnnn nnnn	E4 dd 77 nn	(HL+d)−n	Z	Z	C	H	6
	The contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair are compared with the immediate data "n" in the object code.							
CMP (HL+C), n	1110 0101 0111 0111 nnnn nnnn	E5 77 nn	(HL+C)−n	Z	Z	C	H	6
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair are compared with the immediate data "n" in the object code.							
CMP (PC+A), n	1110 0001 0111 0111 nnnn nnnn	E1 77 nn	(PC+A)−n	Z	Z	C	H	6
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter are compared with the immediate data "n" in the object code.							
MCMP (x), n	1110 0000 xxxx xxxx 0010 1111 nnnn nnnn	E0 xx 2F nn	(x)∧n−A	Z	Z	C	H	6
	The data that is the logical AND of each bit of the contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code and the immediate data "n" in the object code is compared with the contents of the accumulator. If (x) ∧ n = A then the zero flag and the jump status flag are set to "1"; otherwise they are cleared to "0". Example: When A = 25$_H$, the memory contents at the address 0005$_H$ are ED$_H$, after executing the MCMP (05H), 37H instruction, the results will become ZF = 1 and CF = 0.							
MCMP (pp), n	1110 001p 0010 1111 nnnn nnnn	E 2+pp 2F nn	(pp)∧n−A	Z	Z	C	H	5
	The data that is the logical AND of each bit of the contents of the memory address specified by the contents of the register pair pp (HL or DE) and the immediate data "n" in the object code is compared with the contents of the accumulator.							
MCMP (HL+), n	1110 0110 0010 1111 nnnn nnnn	E6 2F nn	(HL)∧n−A, HL←HL+1	Z	Z	C	H	6
	The data that is the logical AND of each bit of the contents of the memory address specified by the contents of the HL register pair and the immediate data "n" in the object code is compared with the contents of the accumulator. Then, the HL register pair is incremented by one.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
MCMP (−HL), n	1110 0111 0010 1111 nnnn nnnn	E7 2F nn	HL←HL−1, (HL)∧n−A	Z	Z	C	H	6
	The HL register pair are decremented by one, and the data that is the logical AND of each bit of the contents of the memory address specified by the new contents of the HL register pair and the immediate data "n" in the object code is compared with the contents of the accumulator.							
MCMP (HL+d), n	1110 0100 dddd dddd 0010 1111 nnnn nnnn	E4 dd 2F nn	(HL+d)∧n−A	Z	Z	C	H	7
	The data that is the logical AND of each bit of the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair and the immediate data "n" in the object code is compared with the contents of the accumulator.							
MCMP (HL+C), n	1110 0101 0010 1111 nnnn nnnn	E5 2F nn	(HL+C)∧n−A	Z	Z	C	H	7
	The data that is the logical AND of each bit of the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair and the immediate data "n" in the object code is compared with the contents of the accumulator.							
MCMP (PC+A), n	1110 0001 0010 1111 nnnn nnnn	E1 2F nn	(PC+A)∧n−A	Z	Z	C	H	7
	The data that is the logical AND of each bit of the contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter and the immediate data "n" in the object code is compared with the contents of the accumulator.							
INC r	0110 0rrr	6r	r←r+1	C	Z	−	−	1
	The register r is incremented by one. When there is an overflow, both the jump status flag and the zero flag are set to "1". Example: When L = FF_H, after executing the INC L instruction, the results will become L = 00_H, ZF = 1, and JF = 1.							
INC rr	0001 00rr	1 rr	rr←rr+1	C	Z	−	−	2
	The register pair rr is incremented by one. An original value of $FFFF_H$ will overflow to 0000_H. When there is an overflow, both the jump status flag and the zero flag are set to "1". Example: When HL = 1234_H, after executing the INC HL instruction, the results will become HL = 1235_H, ZF = 0 and JF = 0.							
INC (x)	0010 0000 xxxx xxxx	20 xx	(x)←(x)+1	C	Z	−	−	5
	The contents of the memory address (0000-00FF_H) directly specified by the "x" in the object code are incremented by one. When there is an overflow in memory contents, both the jump status flag and the zero flag are set to "1".							
INC (HL)	0010 0001	21	(HL)←(HL)+1	C	Z	−	−	4
	The contents of the memory address specified by the contents of the HL register pair are incremented by one. When there is an overflow in memroy contents, both the jump status flag and the zero flag are set to "1".							
INC (DE)	1110 0010 0010 0000	E2 20	(DE)←(DE)+1	C	Z	−	−	4
	The contents of the memory address specified by the contents of the DE register pair are incremented by one. When there is an overflow in memory contents, both the jump status flag and the zero flag are set to "1".							

Note : Increment (INC) and Decrement (DEC) are read-modify-write instructions.

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
INC (HL+)	1110 0110 0010 0000	E6 20	(HL) ← (HL)+1, HL←HL+1	C	Z	–	–	5
	The contents of the memory address specified by the contents of the HL register pair are incremented by one. Then, the HL register pair is incremented by one. When there is an overflow in memory contents, both the jump status flag and the zero flag are set to "1".							
INC (–HL)	1110 0111 0010 0000	E7 20	HL←HL–1, (HL) ← (HL)+1	C	Z	–	–	5
	The HL register pair is decremented by one, and the contents of the memory address specified by the new contents of the HL register pair are incremented by one. When there is an overflow in memory contents, both the jump status flag and the zero flag are set to "1".							
INC (HL+d)	1110 0100 dddd dddd 0010 0000	E4 dd 20	(HL+d)←(HL+d)+1	C	Z	–	–	6
	The contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair are incremented by one. When there is an overflow in memory contents, both the jump status flag and the zero flag are set to "1".							
INC (HL+C)	1110 0101 0010 0000	E5 20	(HL+C) ← (HL+C)+1	C	Z	–	–	6
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register are incremented by one. When there is an overflow in memory contents, both the jump status flag and the zero flag are set to "1".							
INC (PC+A)	1110 0001 0010 0000	E1 20	(PC+A) ← (PC+A)+1	C	Z	–	–	6
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter are incremented by one. When there is an overflow in memory contents, both the jump status flag and the zero flag are set to "1".							
DEC r	0110 1rrr	6 8+r	r←r–1	C	Z	–	–	1
	The contents of the register r are decremented by one. When there is an underflow, the jump status flag is set to "1". Example: When L = 00_H, after executing the DEC L instruction, the results will become L = FF_H, ZF = 0 and JF = 1.							
DEC rr	0001 10rr	1 8+rr	rr←rr–1	C	Z	–	–	2
	The contents of the register pair rr are decremented by one. An original value of 0000_H will underflow to $FFFF_H$. When there is an underflow, the jump status flag is set to "1". Example: When HL = 8765_H, after executing the DEC HL instruction, the results will become HL = 8764_H, ZF = 1 and JF = 0.							
DEC (x)	0010 1000 xxxx xxxx	28 xx	(x)←(x)–1	C	Z	–	–	5
	The contents of the memory address (0000-00FF_H) directly specified by the "x" in the object code are decremented by one. When there is an underflow in memory contents, the jump status flag is set to "1".							
DEC (HL)	0010 1001	29	(HL)←(HL)–1	C	Z	–	–	4
	The contents of the memory address specified by the contents of the HL register pair are decremented by one. When there is an underflow in memory contents, the jump status flag is set to "1".							
DEC (DE)	1110 0010 0010 1000	E2 28	(DE)←(DE)–1	C	Z	–	–	4
	The contents of the memory address specified by the contents of the DE register pair are decremented by one. When there is an underflow in memory contents, the jump status flag is set to "1".							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
DEC (HL+)	1110 0110 0010 1000	E6 28	(HL) ← (HL)−1, HL←HL+1	C	Z	−	−	5
	The contents of the memory address specified by the contents of the HL register pair are decremented by one. Then, the HL register pair is incremented by one. When there is an underflow in memory contents, the jump status flag is set to "1".							
DEC (−HL)	1110 0111 0010 1000	E7 28	HL←HL−1, (HL) ← (HL)−1	C	Z	−	−	5
	The HL register pair is decremented by one, and the contents of the memory address specified by the new contents of the HL register pair are decremented by one. When there is an underflow in memory contents, the jump status flag is set to "1".							
DEC (HL+d)	1110 0100 dddd dddd 0010 1000	E4 dd 28	(HL+d)←(HL+d)−1	C	Z	−	−	6
	The contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair are decremented by one. When there is an underflow in memory contents, the jump status flag is set to "1".							
DEC (HL+C)	1110 0101 0010 1000	E5 28	(HL+C) ← (HL+C)−1	C	Z	−	−	6
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair are decremented by one. When there is an underflow in memory contents, the jump status flag is set to "1".							
DEC (PC+A)	1110 0001 0010 1000	E1 28	(PC+A) ← (PC+A)−1	C	Z	−	−	6
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter are decremented by one. When there is an underflow in memory contents, the jump status flag is set to "1".							
ADD A, g	1110 1ggg 0110 0001	E 8+g 61	A←A+g	C	Z	C	H	2
	The contents of the register g are added to the contents of the accumulator, and the result is entered into the accumulator. Example: When A = $3E_H$ and B = 49_H, after executing the ADD A, B instruction, the results will become A = 87_H, B = 49_H, JF = CF = 0, ZF = 0, and HF = 1.							
ADD g, A	1110 1ggg 0110 1001	E 8+g 69	g←g+A	C	Z	C	H	3
	The contents of the accumulator are added to the contents of the register g (g≠A), and the result is entered into the register g.							
ADD WA, gg	1110 10gg 0011 0001	E 8+gg 31	WA←WA+gg	C	Z	C	U	4
	The contents of the register pair gg are added to the contents of the 16-bit accumulator, and the result is entered into the 16-bit accumulator. If there is a carry-out from bit 15, the carry flag is set to "1"; otherwise it is cleared to "0". Example: When WA = $D546_H$, BC = 9371_H, after executing the ADD WA, BC instruction, the results will become WA = $68B7_H$, BC = 9371_H, JF = CF = 1, and ZF = 0.							
ADD A, (x)	0111 1001 xxxx xxxx	79 xx	A←A+(x)	C	Z	C	H	4
	The contents of the memory address (0000-00FF_H) directly specified by the "x" in the object code are added to the contents of the accumulator, and the result is entered into the accumulator.							
ADD A, (pp)	1110 001p 0111 1001	E 2+pp 79	A←A+(pp)	C	Z	C	H	3
	The contents of the memory address specified by the contents of the register pair pp (HL or DE) are added to the contents of the accumulator, and the result is entered into the accumulator.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
ADD A, (HL+)	1110 0110 0111 1001	E6 79	A←A+(HL), HL←HL+1	C	Z	C	H	4
	The contents of the memory address specified by the contents of the HL register pair are added to the contents of the accumulator, and the result is entered into the accumulator. Then, the HL register pair is incremented by one.							
ADD A, (−HL)	1110 0111 0111 1001	E7 79	HL←HL−1, A←A+(HL)	C	Z	C	H	4
	The HL register pair is decremented by one. Then, the contents of the memory address specified by the new contents of the HL register pair are added to the contents of the accumulator, and the result is entered into the accumulator.							
ADD A, (HL+d)	1110 0100 dddd dddd 0111 1001	E4 dd 79	A←A+(HL+d)	C	Z	C	H	5
	The contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair are added to the contents of the accumulator. The result is entered into the accumulator.							
ADD A, (HL+C)	1110 0101 0111 1001	E5 79	A←A+(HL+C)	C	Z	C	H	5
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair are added to the contents of the accumulator. The result is entered into the accumulator.							
ADD A, (PC+A)	1110 0001 0111 1001	E1 79	A←A+(PC+A)	C	Z	C	H	5
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter are added to the contents of the accumulator. The result is entered into the accumulator.							
ADD (x), (HL)	1110 0000 xxxx xxxx 0110 0001	E0 xx 61	(x)←(x)+(HL)	C	Z	C	H	7
	The contents of the memory address specified by the contents of the HL register pair are added to the contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code. The result is stored into the latter memory address (x).							
ADD (pp), (HL)	1110 001p 0110 0001	E 2+pp 61	(pp)←(pp)+(HL)	C	Z	C	H	6
	The contents of the memory address specified by the contents of the HL register pair are added to the contents of the memory address specified by the contents of the register pair pp (HL or DE). The result is stored into the latter memory address.							
ADD (HL+d), (HL)	1110 0100 dddd dddd 0110 0001	E4 dd 61	(HL+d)←(HL+d)+(HL)	C	Z	C	H	8
	The contents of the memory address specified by the contents of the HL register pair are added to the contents of the memory address specified by the value obtained by adding the coded and expanded 8-bit data "d" in the object code to the contents of the HL register pair. The result is stored into the latter memory address.							
ADD (HL+C), (HL)	1110 0101 0110 0001	E5 61	(HL+C)←(HL+C)+(HL)	C	Z	C	H	8
	The contents of the memory address specified by the contents of the HL register pair are added to the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair. The result is stored into the latter memory address.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
ADD (PC+A), (HL)	1110 0001 0110 0001	E1 61	(PC+A)←(PC+A)+(HL)	C	Z	C	H	8
	The contents of the memory address specified by the contents of the HL register pair are added to the contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter. The result is stored into the latter memory address.							
ADD A, n	0111 0001 nnnn nnnn	71 nn	A←A+n	C	Z	C	H	2
	The immediate data "n" in the object code is added to the contents of the accumulator. The result is entered into the accumulator.							
ADD g, n	1110 1ggg 0111 0001 nnnn nnnn	E 8+g 71 nn	g←g+n	C	Z	C	H	3
	The immediate data "n" in the object code is added to the contents of the register g. The result is entered into the register g.							
ADD gg, mn	1110 10gg 0011 1001 nnnn nnnn mmmm mmmm	E 8+gg 39 nn mm	gg←gg+mn	C	Z	C	U	4
	The 16-bit immediate data "mn" in the object code is added to the contents of the register pair gg. The result is entered into the register pair gg.							
ADD (x), n	1110 0000 xxxx xxxx 0111 0001 nnnn nnnn	E0 xx 71 nn	(x)←(x)+n	C	Z	C	H	6
	The immediate data "n" in the object code is added to the contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code. The result is stored into the above memory address.							
ADD (pp), n	1110 001p 0111 0001 nnnn nnnn	E 2+pp 71 nn	(pp)←(pp)+n	C	Z	C	H	5
	The immediate data "n" in the object code is added to the contents of the memory address specified by the contents of the register pair pp (HL or DE). The result is stored into the above memory address.							
ADD (HL+), n	1110 0110 0111 0001 nnnn nnnn	E6 71 nn	(HL)←(HL)+n, HL←HL+1	C	Z	C	H	6
	The immediate data "n" in the object code is added to the contents of the memory address specified by the contents of the HL register pair, and the result is stored into the above memory address. Then, the HL register pair is incremented by one.							
ADD (−HL), n	1110 0111 0111 0001 nnnn nnnn	E7 71 nn	HL←HL−1, (HL)←(HL)+n	C	Z	C	H	6
	The HL register pair is decremented by one. Then, the immediate data "n" in the object code is added to the contents of the memory address specified by the new contents of the HL register pair. The result is stored into the above memory address.							
ADD (HL+d), n	1110 0100 dddd dddd 0111 0001 nnnn nnnn	E4 dd 71 nn	(HL+d)←(HL+d)+n	C	Z	C	H	7
	The immediate data "n" in the object code is added to the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair. The result is stored into the above memory address.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
ADD (HL+C), n	1110 0101 0111 0001 nnnn nnnn	E5 71 nn	(HL+C)←(HL+C)+n	C	Z	C	H	7
	The immediate data "n" in the object code is added to the contents of the memory address specified by the value obtained by adding the sigtn-extended contents of the C register to the contents of the HL register pair. The result is stored into the above memory address.							
ADD (PC+A), n	1110 0001 0111 0001 nnnn nnnn	E1 71 nn	(PC+A)←(PC+A)+n	C	Z	C	H	7
	The immediate data "n" in the object code is added to the contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter. The result is stored into the above memory address.							
ADDC A, g	1110 1ggg 0110 0000	E 8+g 60	A←A+g+CF	C	Z	C	H	2
	The contents of the register g and the carry flag are added to the contents of the accumulator. The result is entered into the accumulator. Example: When A = 12_H, B = 34_H, CF = 1, after executing the ADDC A, B instruction, the results will become A = 47_H, B = 34_H and JF = CF = 0.							
ADDC g, A	1110 1ggg 0110 1000	E 8+g 68	g←g+A+CF	C	Z	C	H	3
	The contents of the accumulator and the carry flag are added to the contents of the register g (g≠A). The result is entered into the register g.							
ADDC WA, gg	1110 10gg 0011 0000	E 8+gg 30	WA←WA+gg+CF	C	Z	C	U	4
	The contents of the register pair gg and the carry flag are added to the contents of the 16-bit accumulator. The result is entered into the 16-bit accumulator.							
ADDC A, (x)	0111 1000 xxxx xxxx	78 xx	A←A+(x)+CF	C	Z	C	H	4
	The contents of the memory address (0000-00FF_H) directly specified by the "x" in the object code and the content of the carry flag are added to the contents of the accumulator. The result is entered into the accumulator.							
ADDC A, (pp)	1110 001p 0111 1000	E 2+pp 78	A←A+(pp)+CF	C	Z	C	H	3
	The contents of the register pair pp (HL or DE) and the content of the carry flag are added to the contents of the accumulator. The result is entered into the accumulator.							
ADDC A, (HL+)	1110 0110 0111 1000	E6 78	A←A+(HL)+CF, HL←HL+1	C	Z	C	H	4
	The contents of the memory address specified by the contents of the HL register pair and the content of the carry flag are added to the contents of the accumulator, and the result is entered into the accumulator. Then, the HL register pair is incremented by one.							
ADDC A, (−HL)	1110 0111 0111 1000	E7 78	HL←HL−1, A←A+(HL)+CF	C	Z	C	H	4
	The HL register pair is decremented by one. Then, the contents of the memory address specified by the new contents of the HL register pair and the content of the carry flag are added to the contents of the accumulator. The result is entered into the accumulator.							
ADDC A, (HL+d)	1110 0100 dddd dddd 0111 1000	E4 dd 78	A←A+(HL+d)+CF	C	Z	C	H	5
	The contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair and the content of the carry flag are added to the contents of the accumulator. The result is entered into the accumulator.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
ADDC A, (HL+C)	1110 0101 0111 1000	E5 78	A←A+(HL+C)+CF	C	Z	C	H	5
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair and the content of the carry flag are added to the contents of the accumulator. The result is entered into the accumulator.							
ADDC A, (PC+A)	1110 0001 0111 1000	E1 78	A←A+(PC+A)+CF	C	Z	C	H	5
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter and the content of the carry flag are added to the contents of the accumulator. The result is entered into the accumulator.							
ADDC (x), (HL)	1110 0000 xxxx xxxx 0110 0000	E0 xx 60	(x)←(x)+(HL)+CF	C	Z	C	H	7
	The contents of the memory address specified by the contents of the HL register pair and the content of the carry flag are added to the contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code. The result is stored into the latter memory address.							
ADDC (pp), (HL)	1110 001p 0110 0000	E 2+pp 60	(pp)←(pp)+(HL)+CF	C	Z	C	H	6
	The contents of the memory address specified by the contents of the HL register pair and the content of the carry flag are added to the contents of the memory address specified by the contents of the register pair pp (HL or DE). The result is stored into the latter memory address.							
ADDC (HL+d), (HL)	1110 0100 dddd dddd 0110 0000	E4 dd 60	(HL+d)←(HL+d)+(HL)+CF	C	Z	C	H	8
	The contents of the memory address specified by the contents of the HL register pair and the content of the carry flag are added to the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair. The result is stored into the latter memory address.							
ADDC (HL+C), (HL)	1110 0101 0110 0000	E5 60	(HL+C)←(HL+C)+(HL)+CF	C	Z	C	H	8
	The contents of the memory address specified by the contents of the HL register pair and the content of the carry flag are added to the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair. The result is stored into the latter memory address.							
ADDC (PC+A), (HL)	1110 0001 0110 0000	E1 60	(PC+A)←(PC+A)+(HL)+CF	C	Z	C	H	8
	The contents of the memory address specified by the contents of the HL register pair and the content of the carry flag are added to the contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter. The result is stored into the latter memory address.							
ADDC A, n	0111 0000 nnnn nnnn	70 nn	A←A+n+CF	C	Z	C	H	2
	The immediate data "n" in the object code and the content of the carry flag are added to the contents of the accumulator. The result is entered into the accumulator.							
ADDC g, n	1110 1ggg 0111 0000 nnnn nnnn	E 8+g 70 nn	g←g+n+CF	C	Z	C	H	3
	The immediate data "n" in the object code and the content of the carry flag are added to the contents of the register g. The result is entered into the register g.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
ADDC gg, mn	1110 10gg 0011 1000 nnnn nnnn mmmm mmmm	E 8+gg 38 nn mm	gg←gg+mn+CF	C	Z	C	U	4
	The 16-bit immediate data "mn" in the object code and the content of the carry flag are added to the contents of the register pair gg. The result is entered into the register pair gg.							
ADDC (x), n	1110 0000 xxxx xxxx 0111 0000 nnnn nnnn	E0 xx 70 nn	(x)←(x)+n+CF	C	Z	C	H	6
	The immediate data "n" in the object code and the content of the carry flag are added to the contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code. The result is stored into the above memory address.							
ADDC (pp), n	1110 001p 0111 0000 nnnn nnnn	E 2+pp 70 nn	(pp)←(pp)+n+CF	C	Z	C	H	5
	The immediate data "n" in the object code and the content of the carry flag are added to the contents of the memory address specified by the contents of the register pair pp (HL or DE). The result is stored into the above memory address.							
ADDC (HL+), n	1110 0110 0111 0000 nnnn nnnn	E6 70 nn	(HL)←(HL)+n+CF, HL←HL+1	C	Z	C	H	6
	The immediate data "n" in the object code and the content of the carry flag are added to the contents of the memory address specified by the contents of the HL register pair, and the result is stored into the above memory address. Then, the HL register pair is incremented by one.							
ADDC (−HL), n	1110 0111 0111 0000 nnnn nnnn	E7 70 nn	HL←HL−1, (HL)←(HL)+n+CF	C	Z	C	H	6
	The HL register pair is decremented by one. Then, the immediate data "n" in the object code and the content of the carry flag are added to the contents of the memory address specified by the new contents of the HL register pair. The result is stored into the above memory address.							
ADDC (HL+d), n	1110 0100 dddd dddd 0111 0000 nnnn nnnn	E4 dd 70 nn	(HL+d)←(HL+d)+n+CF	C	Z	C	H	7
	The immediate data "n" in the object code and the content of the carry flag are added to the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" to the contents of the HL register pair. The result is stored into the above memory address.							
ADDC (HL+C), n	1110 0101 0111 0000 nnnn nnnn	E5 70 nn	(HL+C)←(HL+C)+n+CF	C	Z	C	H	7
	The immediate data "n" in the object code and the content of the carry flag are added to the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair. The result is stored into the above memory address.							
ADDC (PC+A), n	1110 0001 0111 0000 nnnn nnnn	E1 70 nn	(PC+A)←(PC+A)+n+CF	C	Z	C	H	7
	The immediate data "n" in the object code and the content of the carry flag are added to the contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter. The result is stored into the above memory address.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
SUB A, g	1110 1ggg 0110 0011	E 8+g 63	A←A−g	C	Z	C	H	2
	The contents of the register g are subtracted from the contents of the accumulator, and the result is entered into the accumulator. Example: When A = 25_H and B = $3F_H$, and the SUB A, B instruction is executed, the results are A = $E6_H$, B = $3F_H$, CF = 1 and HF = 1.							
SUB g, A	1110 1ggg 0110 1011	E 8+g 6B	g←g−A	C	Z	C	H	3
	The contents of the accumulator are subtracted form the contents of the register g (g≠A), and the result is entered into the register g.							
SUB WA, gg	1110 10gg 0011 0011	E 8+gg 33	WA←WA−gg	C	Z	C	U	4
	The contents of the register pair gg are subtracted from the contents of the 16-bit accumulator, and the result is entered into the 16-bit accumulator.							
SUB A, (x)	0111 1011 xxxx xxxx	7B xx	A←A−(x)	C	Z	C	H	4
	The contents of the memory address (0000-00FF_H) directly specified by the "x" in the object code are subtracted from the contents of the accumulator, and the result is entered into the accumulator.							
SUB A, (pp)	1110 001p 0111 1011	E 2+pp 7B	A←A−(pp)	C	Z	C	H	3
	The contents of the memory address specified by the contents of the register pair pp (HL or DE) are subtracted from the contents of the accumulator, and the result is entered into the accumulator.							
SUB A, (HL+)	1110 0110 0111 1011	E6 7B	A←A−(HL), HL←HL+1	C	Z	C	H	4
	The contents of the memory address specified by the contents of the HL register pair are subtracted from the contents of the accumulator, and the result is entered into the accumulator. Then, the HL register is incremented by one.							
SUB A, (−HL)	1110 0111 0111 1011	E7 7B	HL←HL−1, A←A−(HL)	C	Z	C	H	4
	The HL register pair is decremented by one. Then, the contents of the memory address specified by the new contents of the HL register pair are subtracted from the contents of the accumulator. The result is entered into the accumulator.							
SUB A, (HL+d)	1110 0100 dddd dddd 0111 1011	E4 dd 7B	A←A−(HL+d)	C	Z	C	H	5
	The contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair are subtracted from the contents of the accumulator. The result is entered into the accumulator.							
SUB A, (HL+C)	1110 0101 0111 1011	E5 7B	A←A−(HL+C)	C	Z	C	H	5
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair are subtracted from the contents of the accumulator. The result is entered into the accumulator.							
SUB A, (PC+A)	1110 0001 0111 1011	E1 7B	A←A−(PC+A)	C	Z	C	H	5
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter are subtracted from the contents of the accumulator. The result is entered into the accumulator.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
SUB (x), (HL)	1110 0000 xxxx xxxx 0110 0011	E0 xx 63	(x)←(x)−(HL)	C	Z	C	H	7
	The contents of the memory address specified by the contents of the HL register pair are subtracted from the contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code. The result is stored into the latter memory address.							
SUB (pp), (HL)	1110 001p 0110 0011	E 2+p 63	(pp)←(pp)−(HL)	C	Z	C	H	6
	The contents of the memory address specified by the contents of the HL register pair are subtracted from the contents of the memory address specified by the contents of the register pair pp (HL or DE). The result is stored into the latter memory address.							
SUB (HL+d), (HL)	1110 0100 dddd dddd 0110 0011	E4 dd 63	(HL+d)←(HL+d)−(HL)	C	Z	C	H	8
	The contents of the memory address specified by the contents of the HL register pair are subtracted from the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair. The result is stored into the latter memory address.							
SUB (HL+C), (HL)	1110 0101 0110 0011	E5 63	(HL+C)←(HL+C)−(HL)	C	Z	C	H	8
	The contents of the memory address specified by the contents of the HL register pair are subtracted from the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair. The result is stored into the latter memory address.							
SUB (PC+A), (HL)	1110 0001 0110 0011	E1 63	(PC+A)←(PC+A)−(HL)	C	Z	C	H	8
	The contents of the memory address specified by the contents of the HL register pair are subtracted from the contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter. The result is stored into the latter memory address.							
SUB A, n	0111 0011 nnnn nnnn	73 nn	A←A−n	C	Z	C	H	2
	The immediate data "n" in the object code is subtracted from the contents of the accumulator, and the result is entered into the accumulator.							
SUB g, n	1110 1ggg 0111 0011 nnnn nnnn	E 8+g 73 nn	g←g−n	C	Z	C	H	3
	The immediate data "n" in the object code is subtracted from the contents of the register g, and the result is entered into the register g.							
SUB gg, mn	1110 10gg 0011 1011 nnnn nnnn mmmm mmmm	E 8+gg 3B nn mm	gg←gg−mn	C	Z	C	U	4
	The immediate data "mn" in the object code is subtracted from the contents of the register pair gg, and the result is entered into the register pair gg.							
SUB (x), n	1110 0000 xxxx xxxx 0111 0011 nnnn nnnn	E0 xx 73 nn	(x)←(x)−n	C	Z	C	H	6
	The immediate data "n" in the object code is subtracted from the contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code. The result is stored into the above memory address.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
SUB (pp), n	1110 001p 0111 0011 nnnn nnnn	E 2+pp 73 nn	(pp)←(pp)−n	C	Z	C	H	5
	The immediate data "n" in the object code is subtracted from the contents of the memory address specified by the contents of the register pair pp (HL or DE). The result is stored into the above memory address.							
SUB (HL+), n	1110 0110 0111 0011 nnnn nnnn	E6 73 nn	(HL)←(HL)−n, HL←HL+1	C	Z	C	H	6
	The immediate data "n" in the object code is subtracted from the contents of the memory address specified by the contents of the HL register pair, and the result is stored into the above memory address. Then, the HL register pair is incremented by one.							
SUB (−HL), n	1110 0111 0111 0011 nnnn nnnn	E7 73 nn	HL←HL−1, (HL)←(HL)−n	C	Z	C	H	6
	The HL register pair is decremented by one. Then, the immediate data "n" in the object code is subtracted from the contents of the memory address specified by the new contents of the HL register pair. The result is stored into the above memory address.							
SUB (HL+d), n	1110 0100 dddd dddd 0111 0011 nnnn nnnn	E4 dd 73 nn	(HL+d)←(HL+d)−n	C	Z	C	H	7
	The immediate data "n" in the object code is subtracted from the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair. The result is stored into the above memory address.							
SUB (HL+C), n	1110 0101 0111 0011 nnnn nnnn	E5 73 nn	(HL+C)←(HL+C)−n	C	Z	C	H	7
	The immediate data "n" in the object code is subtracted from the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair. The result is stored into the above memory address.							
SUB (PC+A), n	1110 0001 0111 0011 nnnn nnnn	E1 73 nn	(PC+A)←(PC+A)−n	C	Z	C	H	7
	The immediate data "n" in the object code is subtracted from the contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter. The result is stored into the above memory address.							
SUBB A, g	1110 1ggg 0110 0010	E 8+g 62	A←A−g−CF	C	Z	C	H	2
	The contents of the register g and the carry flag are subtracted from the contents of the accumulator. The result is entered into the accumulator. Example: When A = 87_H, B = 51_H and CF = 1, after executing the SUB A, B instruction, the results will become A = 35_H, B = 51_H, CF = 0 and HF = 0.							
SUBB g, A	1110 1ggg 0110 1010	E 8+g 6A	g←g−A−CF	C	Z	C	H	3
	The contents of the accumulator and the carry flag are subtracted from the contents of the register g (g≠A). The result is entered into the register g.							
SUBB WA, gg	1110 10gg 0011 0010	E 8+gg 32	WA←WA−gg−CF	C	Z	C	U	4
	The contents of the register pair gg and the carry flag are subtracted from the contents of the 16-bit accumulator. The result is entered into the 16-bit accumulator.							

MNEMONIC	OBJECT CODE		OPERATION	FLAG				CYCLE
	Binary	Hex-decimal		JF	ZF	CF	HF	
SUBB A, (x)	0111 1010 xxxx xxxx	7A xx	A←A−(x)−CF	C	Z	C	H	4
	The contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code and the content of the carry flag are subtracted from the contents of the accumulator. The result is entered into the accumulator.							
SUBB A, (pp)	1110 001p 0111 1010	E 2+pp 7A	A←A−(pp)−CF	C	Z	C	H	3
	The contents of the memory address specified by the contents of the register pair pp (HL or DE) and the content of the carry flag are subtracted from the contents of the accumulator. The result is entered into the accumulator.							
SUBB A, (HL+)	1110 0110 0111 1010	E6 7A	A←A−(HL)−CF, HL←HL+1	C	Z	C	H	4
	The contents of the memory address specified by the contents of the HL register pair and the content of the carry flag are subtracted from the contents of the accumulator. The result is entered into the accumulator. Then, the HL register pair is incremented by one.							
SUBB A, (−HL)	1110 0111 0111 1010	E7 7A	HL←HL−1, A←A−(HL)−CF	C	Z	C	H	4
	The contents of the HL register pair is decremented by one. Then the contents of the memory address specified by the new contents of the HL register pair and the content of the carry flag are subtracted from the accumulator. The result is entered into the accumulator.							
SUBB A, (HL+d)	1110 0100 dddd dddd 0111 1010	E4 dd 7A	A←A−(HL+d)−CF	C	Z	C	H	5
	The contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair and the content of the carry flag are subtracted from the contents of the accumulator. The result is entered into the accumulator.							
SUBB A, (HL+C)	1110 0101 0111 1010	E5 7A	A←A−(HL+C)−CF	C	Z	C	H	5
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair and the content of the carry flag are subtracted from the contents of the accumulator. The result is entered into the accumulator.							
SUBB A, (PC+A)	1110 0001 0111 1010	E1 7A	A←A−(PC+A)−CF	C	Z	C	H	5
	The contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter and the content of the carry flag are subtracted from the contents of the accumulator. The result is entered into the accumulator.							
SUBB (x), (HL)	1110 0000 xxxx xxxx 0110 0010	E0 xx 62	(x)←(x)−(HL)−CF	C	Z	C	H	7
	The contents of the memory address specified by the contents of the HL register pair and the content of the carry flag are subtracted from the contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code. The result is stored into the latter memory address.							
SUBB (pp), (HL)	1110 001p 0110 0010	E 2+pp 62	(pp)←(pp)−(HL)−CF	C	Z	C	H	6
	The contents of the memory address specified by the contents of the HL register pair and the content of the carry flag are subtracted from the contents of the memory address specified by the contents of the register pair pp (HL or DE). The result is stored into the latter memory address.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
SUBB (HL+d), (HL)	1110 0100 dddd dddd 0110 0010	E4 dd 62	(HL+d)←(HL+d)−(HL)−CF	C	Z	C	H	8
	The contents of the memory address specified by the HL register pair and the content of the carry flag are subtracted from the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair. The result is stored into the latter memory address.							
SUBB (HL+C), (HL)	1110 0101 0110 0010	E5 62	(HL+C)←(HL+C)−(HL)−CF	C	Z	C	H	8
	The contents of the memory address specified by the contents of the HL register pair and the content of the carry flag are subtracted from the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair. The result is stored into the latter memory address.							
SUBB (PC+A), (HL)	1110 0001 0110 0010	E1 62	(PC+A)←(PC+A)−(HL)−CF	C	Z	C	H	8
	The contents of the memory address specified by the contents of the HL register pair and the content of the carry flag are subtracted from the contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter. The result is stored into the latter memory address.							
SUBB A, n	0111 0010 nnnn nnnn	72 nn	A←A−n−CF	C	Z	C	H	2
	The immediate data "n" in the object code and the content of the carry flag are subtracted from the contents of the accumulator. The result is entered into the accumulator.							
SUBB g, n	1110 1ggg 0111 0010 nnnn nnnn	E 8+g 72 nn	g←g−n−CF	C	Z	C	H	3
	The immediate data "n" in the object code and the content of the carry flag are subtracted from the contents of the register g. The result is entered into the register g.							
SUBB gg, mn	1110 10gg 0011 1010 nnnn nnnn mmmm mmmm	E 8+gg 3A nn mm	gg←gg−mn−CF	C	Z	C	U	4
	The 16-bit immediate data "mn" in the object code and the content of the carry flag are subtracted from the contents of the register pair gg. The result is entered into the register pair gg.							
SUBB (x), n	1110 0000 xxxx xxxx 0111 0010 nnnn nnnn	E0 xx 72 nn	(x)←(x)−n−CF	C	Z	C	H	6
	The immediate data "n" in the object code and the content of the carry flag are subtracted from the contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code. The result is stored into the above memory address.							
SUBB (pp), n	1110 001p 0111 0010 nnnn nnnn	E 2+gg 72 nn	(pp)←(pp)−n−CF	C	Z	C	H	5
	The immediate data "n" in the object code and the content of the carry flag are subtracted from the contents of the memory address specified by the contents of the register pair pp (HL or DE). The result is stored into the above memory address.							
SUBB (HL+), n	1110 0110 0111 0010 nnnn nnnn	E6 72 nn	(HL)←(HL)−n−CF, HL←HL+1	C	Z	C	H	6
	The immediate data "n" in the object code and the content of the carry flag are subtracted from the contents of the memory address specified by the contents of the HL register pair. The result is stored into the above memory address. Then, the HL register pair is incremented by one.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
SUBB (−HL), n	1110 0111 0111 0010 nnnn nnnn	E7 72 nn	HL←HL−1, (HL)←(HL)−n−CF	C	Z	C	H	6
	The HL register pair is decremented by one. Then the immediate data "n" in the object code and the content of the carry flag are subtracted from the contents of the memory address specified by the new contents of the HL register pair. The result is stored into the above memory address.							
SUBB (HL+d), n	1110 0100 dddd dddd 0111 0010 nnnn nnnn	E4 dd 72 nn	(HL+d)←(HL+d)−n−CF	C	Z	C	H	7
	The immediate data "n" in the object code and the content of the carry flag are subtracted from the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair. The result is stored into the above memory address.							
SUBB (HL+C), n	1110 0101 0111 0010 nnnn nnnn	E5 72 nn	(HL+C)←(HL+C)−n−CF	C	Z	C	H	7
	The immediate data "n" in the object code and the content of the carry flag are subtracted from the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair. The result is stored into the above memory address.							
SUBB (PC+A), n	1110 0001 0111 0010 nnnn nnnn	E1 72 nn	(PC+A)←(PC+A)−n−CF	C	Z	C	H	7
	The immediate data "n" in the object code and the content of the carry flag are subtracted from the contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter. The result is stored into the above memory address.							
AND A, g	1110 1ggg 0110 0100	E 8+g 64	A←A∧g	Z	Z	−	−	2
	The data that is the logical AND of each bit of the contents of the accumulator and the contents of the register g is derived. The result is entered into the accumulator. Example: When A = $D5_H$ and H = $B3_H$, after executing the AND A, H instruction, the results will become A = 91_H, H = $B3_H$ and ZF = 0.							
AND g, A	1110 1ggg 0110 1100	E 8+g 6C	g←g∧A	Z	Z	−	−	3
	The data that is the logical AND of each bit of the contents of the register g (g≠A) and the contents of the accumulator is derived. The results are entered into the register g.							
AND WA, gg	1110 10gg 0011 0100	E 8+gg 34	WA←WA∧gg	Z	Z	−	−	4
	The data that is the logical AND of each bit of the contents of the 16-bit accumulator and the contents of the register pair gg is derived. The result is entered into the 16-bit accumulator.							
AND A, (x)	0111 1100 xxxx xxxx	7C xx	A←A∧(x)	Z	Z	−	−	4
	The data that is the logical AND of each bit of the contents of the memory address (0000-00FF_H) directly specified by the "x" in the object code and the contents of the accumulator is derived. The result is entered into the accumulator.							
AND A, (pp)	1110 001p 0111 1100	E 2+pp 7C	A←A∧(pp)	Z	Z	−	−	3
	The data that is the logical AND of each bit of the contents of the accumulator and the contents of the memory address specified by the contents of the register pair pp (HL or DE) is derived. The result is entered into the accumulator.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
AND A, (HL+)	1110 0110 0111 1100	E6 7C	A←A∧(HL), HL←HL+1	Z	Z	–	–	4
	The data that is the logical AND of each bit of the contents of the accumulator and the contents of the memory address specified by the contents of the HL register pair is derived. The result is entered into the accumulator. Then, the HL register pair is incremented by one.							
AND A, (–HL)	1110 0111 0111 1100	E7 7C	HL←HL–1, A←A∧(HL)	Z	Z	–	–	4
	The HL register pair is decremented by one. Then the data that is the logical AND of each bit of the contents of the memory address specified by the new contents of the HL register pair and the contents of the accumulator is derived. The result is entered into the accumulator.							
AND A, (HL+d)	1110 0100 dddd dddd 0111 1100	E4 dd 7C	A←A∧(HL+d)	Z	Z	–	–	5
	The data that is the logical AND of each bit of the contents of the accumulator and the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data d in the object code to the contents of the HL register pair is derived. The result is entered into the accumulator.							
AND A, (HL+C)	1110 0101 0111 1100	E5 7C	A←A∧(HL+C)	Z	Z	–	–	5
	The data that is the logical AND of each bit of the contents of the accumulator and the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair is derived. The result is entered into the accumulator.							
AND A, (PC+A)	1110 0001 0111 1100	E1 7C	A←A∧(PC+A)	Z	Z	–	–	5
	The data that is the logical AND of each bit of the contents of the accumulator and the contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter is derived. The result is entered into the accumulator.							
AND (x), (HL)	1110 0000 xxxx xxxx 0110 0100	E0 xx 64	(x)←(x)∧(HL)	Z	Z	–	–	7
	The data that is the logical AND of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code is derived. The result is stored into the latter memory address.							
AND (pp), (HL)	1110 001p 0110 0100	E 2+pp 64	(pp)←(pp)∧(HL)	Z	Z	–	–	6
	The data that is the logical AND of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the memory address specified by the contents of the register pair pp (HL or DE) is derived. The result is stored into the latter memory address.							
AND (HL+d), (HL)	1110 0100 dddd dddd 0110 0100	E4 dd 64	(HL+d)←(HL+d)∧(HL)	Z	Z	–	–	8
	The data that is the logical AND of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair is derived. The result is stored into the latter memory address.							
AND (HL+C), (HL)	1110 0101 0110 0100	E5 64	(HL+C)←(HL+C)∧(HL)	Z	Z	–	–	8
	The data that is the logical AND of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair is derived. The result is stored into the latter memory address.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
AND (PC+A), (HL)	1110 0001 0110 0100	E1 64	(PC+A)←(PC+A)∧(HL)	Z	Z	–	–	8
	The data that is the logical AND of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter is derived. The result is stored into the latter memory address.							
AND A, n	0111 0100 nnnn nnnn	74 nn	A←A∧n	Z	Z	–	–	2
	The data that is the logical AND of each bit of the immediate data "n" in the object code and the contents of the accumulator is derived. The result is entered into the accumulator.							
AND g, n	1110 1ggg 0111 0100 nnnn nnnn	E 8+g 74 nn	g←g∧n	Z	Z	–	–	3
	The data that is the logical AND of each bit of the immediate data "n" in the object code and the contents of the register g is derived. The result is entered into the register g.							
AND gg, mn	1110 10gg 0011 1100 nnnn nnnn mmmm mmmm	E 8+gg 3C nn mm	gg←gg∧mn	Z	Z	–	–	4
	The data that is the logical AND of each bit of the 16-bit immediate data "mn" in the object code and the contents of the register pair gg is derived. The result is entered into the register pair gg.							
AND (x), n	1110 0000 xxxx xxxx 0111 0100 nnnn nnnn	E0 xx 74 nn	(x)←(x)∧n	Z	Z	–	–	6
	The data that is the logical AND of each bit of the immediate data "n" in the object code and the contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code is derived. The result is stored into the above memory address.							
AND (pp), n	1110 001p 0111 0100 nnnn nnnn	E 2+pp 74 nn	(pp)←(pp)∧n	Z	Z	–	–	5
	The data that is the logical AND of each bit of the immediate data "n" in the object code and the contents of the memory address specified by the contents of the register pair pp (HL or DE) is derived. The result is stored into the above memory address.							
AND (HL+), n	1110 0110 0111 0100 nnnn nnnn	E6 74 nn	(HL)←(HL)∧n, HL←HL+1	Z	Z	–	–	6
	The data that is the logical AND of each bit of the immediate data "n" in the object code and the contents of the memory address specified by the contents of the HL register pair is derived. The result is stored into the above memory address. Then, the HL register pair is incremented by one.							
AND (–HL), n	1110 0111 0111 0100 nnnn nnnn	E7 74 nn	HL←HL–1, (HL)←(HL)∧n	Z	Z	–	–	6
	The HL register pair is decremented by one. Then the data that is the logical AND of each bit of the immediate data "n" in the object code and the contents of the memory address specified by the new contents of the HL register pair is derived. The reuslt is stored into the above memory address.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
AND (HL+d), n	1110 0100 dddd dddd 0111 0100 nnnn nnnn	E4 dd 74 nn	(HL+d)←(HL+d)∧n	Z	Z	–	–	7
	The data that is the logical AND of each bit of the immediate data "n" in the object code and the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair is derived. The result is stored into the above memory address.							
AND (HL+C), n	1110 0101 0111 0100 nnnn nnnn	E5 74 nn	(HL+C)←(HL+C)∧n	Z	Z	–	–	7
	The data that is the logical AND of each bit of the immediate data "n" in the object code and the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair is derived. The result is stored into the above memory address.							
AND (PC+A), n	1110 0001 0111 0100 nnnn nnnn	E1 74 nn	(PC+A)←(PC+A)∧n	Z	Z	–	–	7
	The data that is the logical AND of each bit of the immediate data "n" in the object code and the contents of the memory address specified by the data obtained by adding the sign-extended contents of the accumulator to the contents of the program counter is derived. The result is stored into the above memory address.							
OR A, g	1110 1ggg 0110 0110	E 8+g 66	A←A∨g	Z	Z	–	–	2
	The data that is the logical OR of each bit of the contents of the register g and the contents of the accumulator is derived. The result is entered into the accumulator. Example: When A = 53_H, B = 85_H, and the OR A, B instruction is executed, the results are A = $D7_H$, B = 85_H and ZF = 0.							
OR g, A	1110 1ggg 0110 1110	E 8+g 6E	g←g∨A	Z	Z	–	–	3
	The data that is the logical OR of each bit of the contents of the accumulator and the contents of the register g (g≠A) is derived. The result is entered into the register g.							
OR WA, gg	1110 10gg 0011 0110	E 8+gg 36	WA←WA∨gg	Z	Z	–	–	4
	The data that is the logical OR of each bit of the contents of the register pair gg and the contents of 16-bit accumulator is derived. The result is entered into the 16-bit accumulator.							
OR A, (x)	0111 1110 xxxx xxxx	7E xx	A←A∨(x)	Z	Z	–	–	4
	The data that is the logical OR of each bit of the contents of the memory address (0000-00FF_H) directly specified by the "x" in the object code and the contents of the accumulator is derived. The result is entered into the accumulator.							
OR A, (pp)	1110 001p 0111 1110	E 2+pp 7E	A←A∨(pp)	Z	Z	–	–	3
	The data that is the logical OR of each bit of the contents of the memory address specified by the contents of the register pair pp (HL or DE) and the contents of the accumulator is derived. The result is entered into the accumulator.							
OR A, (HL+)	1110 0110 0111 1110	E6 7E	A←A∨(HL), HL←HL+1	Z	Z	–	–	4
	The data that is the logical OR of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the accumulator is derived. The result is entered into the accumulator. Then, the HL register pair is incremented by one.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
OR A, (−HL)	1110 0111 0111 1110	E7 7E	HL←HL−1, A←A∨(HL)	Z	Z	−	−	4
	The HL register pair is decremented by one. Then the data that is the logical OR of each bit of the contents of the memory address specified by the new contents of the HL register pair and the contents of the accumulator is derived. The result is entered into the accumulator.							
OR A, (HL+d)	1110 0100 dddd dddd 0111 1110	E4 dd 7E	A←A∨(HL+d)	Z	Z	−	−	5
	The data that is the logical OR of each bit of the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair and the contents of the accumulator is derived. The result is entered into the accumulator.							
OR A, (HL+C)	1110 0101 0111 1110	E5 7E	A←A∨(HL+C)	Z	Z	−	−	5
	The data that is the logical OR of each bit of the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair and the contents of the accumulator is derived. The result is entered into the accumulator.							
OR A, (PC+A)	1110 0001 0111 1110	E1 7E	A←A∨(PC+A)	Z	Z	−	−	5
	The data that is the logical OR of each bit of the contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter and the contents of the accumulator is derived. The result is entered into the accumulator.							
OR (x), (HL)	1110 0000 xxxx xxxx 0110 0110	E0 xx 66	(x)←(x)∨(HL)	Z	Z	−	−	7
	The data that is the logical OR of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code is derived. The result is stored into the latter addresss.							
OR (pp), (HL)	1110 001p 0110 0110	E 2+pp 66	(pp)←(pp)∨(HL)	Z	Z	−	−	6
	The data that is the logical OR of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the memory address specified by the contents of the register pair pp (HL or DE) is derived. The result is stored into the latter address.							
OR (HL+d), (HL)	1110 0100 dddd dddd 0110 0110	E4 dd 66	(HL+d)←(HL+d)∨(HL)	Z	Z	−	−	8
	The data that is the logical OR of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the memory address specified by the value obtarined by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair is derived. The result is stored into the latter address.							
OR (HL+C), (HL)	1110 0101 0110 0110	E5 66	(HL+C)←(HL+C)∨(HL)	Z	Z	−	−	8
	The data that is the logical OR of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair is derived. The result is stored into the latter address.							
OR (PC+A), (HL)	1110 0001 0110 0110	E1 66	(PC+A)←(PC+A)∨(HL)	Z	Z	−	−	8
	The data that is the logical OR of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter is derived. The result is stored into the latter address.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
OR A, n	0111 0110 nnnn nnnn	76 nn	A←A∨n	Z	Z	–	–	2
	The data that is the logical OR of each bit of the immediate data "n" in the object code and the contents of the accumulator is derived. The result is entered into the accumulator.							
OR g, n	1110 1ggg 0111 0110 nnnn nnnn	E 8+g 76 nn	g←g∨n	Z	Z	–	–	3
	The data that is the logical OR of each bit of the immediate data "n" in the object code and the contents of the register g is derived. The result is entered into the register g.							
OR gg, mn	1110 10gg 0011 1110 nnnn nnnn mmmm mmmm	E 8+gg 3E nn mm	gg←gg∨mn	Z	Z	–	–	4
	The data that is the logical OR of each bit of the 16-bit immediate data "mn" in the object code and the contents of the register pair gg is derived. The result is entered into the register pair gg.							
OR (x), n	1110 0000 xxxx xxxx 0111 0110 nnnn nnnn	E0 xx 76 nn	(x)←(x)∨n	Z	Z	–	–	6
	The data that is the logical OR of each bit of the immediate data "n" in the object code and the contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code is derived. the result is stored into the above memory address.							
OR (pp), n	1110 001p 0111 0110 nnnn nnnn	E 2+pp 76 nn	(pp)←(pp)∨n	Z	Z	–	–	5
	The data that is the logical OR of each bit of the immediate data "n" in the object code and the contents of the memory address specified by the contents of the register pair pp (HL or DE) is derived. The result is stored into the above memory address.							
OR (HL+), n	1110 0110 0111 0110 nnnn nnnn	E6 76 nn	(HL)←(HL)∨n, HL←HL+1	Z	Z	–	–	6
	The data that is the logical OR of each bit of the immediate data "n" in the object code and the contents of the memory address specified by the contents of the HL register pair is derived. The result is stored into the above memory address. Then, the HL register pair is incremented by one.							
OR (–HL), n	1110 0111 0111 0110 nnnn nnnn	E7 76 nn	HL←HL–1, (HL)←(HL)∨n	Z	Z	–	–	6
	The HL register pair is decremented by one. Then the data that is the logical OR of each bit of the immediate data "n" in the object code and the contents of the memory address specified by the new contents of the HL register pair is derived. The result is stored into the above memory address.							
OR (HL+d), n	1110 0100 dddd dddd 0111 0110 nnnn nnnn	E4 dd 76 nn	(HL+d)←(HL+d)∨n	Z	Z	–	–	7
	The data that is the logical OR of each bit of the immedaite data "n" in the object code and the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair is derived. The result is stored into the above memory address.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
OR (HL+C), n	1110 0101 0111 0110 nnnn nnnn	E5 76 nn	(HL+C)←(HL+C)∨n	Z	Z	–	–	7
The data that is the logical OR of each bit of the immedaite data "n" in the object code and the contents of the memory address specified by the data obtained by adding the coded and expanded contents of the C register to the HL register pair is derived. the result is stored into the above memory address.								
OR (PC+A), n	1110 0001 0111 0110 nnnn nnnn	E1 76 nn	(PC+A)←(PC+A)∨n	Z	Z	–	–	7
The data that is the logical OR of each bit of the immediate data "n" in the object code and the contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the program counter is derived. The result is stored into the above memory address.								
XOR A, g	1110 1ggg 0110 0101	E 8+g 65	A←A∀g	Z	Z	–	–	2
The data that is the logical exclusive OR of each bit of the contents of the accumulator and the contents of the register g is derived. The result is entered into the accumulator. Example: When A = 74_H and E = $5D_H$ and then the XOR A, E instruction is executed, the results are A = 29_H, E = $5D_H$ and JF = ZF = 0.								
XOR g, A	1110 1ggg 0110 1101	E 8+g 6D	g←g∀A	Z	Z	–	–	3
The data that is the logical exclusive OR of each bit of the contents of the accumulator and the contents of the register g (g≠A) is derived. The result is entered into the register g.								
XOR WA, gg	1110 10gg 0011 0101	E 8+gg 35	WA←WA∀gg	Z	Z	–	–	4
The data that is the logical exclusive OR of each bit of the contents of the register pair gg and the contents of the 16-bit accumulator is derived. The result is entered into the 16-bit accumulator.								
XOR A, (x)	0111 1101 xxxx xxxx	7D xx	A←A∀(x)	Z	Z	–	–	4
The data that is the logical exclusive OR of each bit of the contents of the memory address (0000-00FF_H) directly specified by the "x" in the object code and the contents of the accumulator is derived. The result is entered into the accumulator.								
XOR A, (pp)	1110 001p 0111 1101	E 2+pp 7D	A←A∀(pp)	Z	Z	–	–	3
The data that is the logical exclusive OR of each bit of the contents of the memory address specified by register pair pp (HL or DE) and the contents of the accumulator is derived. The result is entered into the accumulator.								
XOR A, (HL+)	1110 0110 0111 1101	E6 7D	A←A∀(HL), HL←HL+1	Z	Z	–	–	4
The data that is the logical exclusive OR of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the accumulator. The result is entered into the accumulator. Then, the HL register pair is incremented by one.								
XOR A, (–HL)	1110 0111 0111 1101	E7 7D	HL←HL–1, A←A∀(HL)	Z	Z	–	–	4
The HL register pair is decremented by one. Then the data that is the logical exclusive OR of each bit of the contents of the memory address specified by the new contents of the HL register pair and the contents of the accumulator is derived. The result is entered into the accumulator.								

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
XOR A, (HL+d)	1110 0100 dddd dddd 0111 1101	E4 dd 7D	A←A∀(HL+d)	Z	Z	–	–	5
The data that is the logical exclusive OR of each bit of the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair and the contents of the accumulator is derived. The result is entered into the accumulator.								
XOR A, (HL+C)	1110 0101 0111 1101	E5 7D	A←A∀(HL+C)	Z	Z	–	–	5
The data that is the logical exclusive OR of each bit of the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair and the contents of the accumulator is derived. The result is entered into the accumulator.								
XOR A, (PC+A)	1110 0001 0111 1101	E1 7D	A←A∀(PC+A)	Z	Z	–	–	5
The data that is the logical exclusive OR of each bit of the contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter and the contents of the accumulator is derived. The result is entered into the accumulator.								
XOR (x), (HL)	1110 0000 xxxx xxxx 0110 0101	E0 xx 65	(x)←(x)∀(HL)	Z	Z	–	–	7
The data that is the logical exclusive OR of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code is derived. The result is stored into the latter memory address (x).								
XOR (pp), (HL)	1110 001p 0110 0101	E 2+pp 65	(pp)←(pp)∀(HL)	Z	Z	–	–	6
The data that is the logical exclusive OR of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the memory address specified by register pair pp (HL or DE) is derived. The result is stored into the latter memory address.								
XOR (HL+d), (HL)	1110 0100 dddd dddd 0110 0101	E4 dd 65	(HL+d)←(HL+d)∀(HL)	Z	Z	–	–	8
The data that is the logical exclusive OR of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair is derived. The result is stored into the latter memory address.								
XOR (HL+C), (HL)	1110 0101 0110 0101	E5 65	(HL+C)←(HL+C)∀(HL)	Z	Z	–	–	8
The data that is the logical exclusive OR of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair is derived. The result is stored into the latter memory address.								
XOR (PC+A), (HL)	1110 0001 0110 0101	E1 65	(PC+A)←(PC+A)∀(HL)	Z	Z	–	–	8
The data that is the logical exclusive OR of each bit of the contents of the memory address specified by the contents of the HL register pair and the contents of the memory address specified by the value obtained by adding the signed-extended contents of the accumulator to the contents of the program counter is derived. The result is stored into the latter memory address.								

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
XOR A, n	0111 0101 nnnn nnnn	75 nn	A←A∀n	Z	Z	–	–	2
	The data that is the logical exclusive OR of each bit of the immediate data "n" in the object code and the contents of the accumulator is derived. The result is entered into the accumulator. When n = FF_H, it becomes the 1's complement instruction.							
XOR g, n	1110 1ggg 0111 0101 nnnn nnnn	E 8+g 75 nn	g←g∀n	Z	Z	–	–	3
	The data that is the logical exclusive OR of each bit of the immediate data "n" in the object code and the contents of the register g is derived. The result is entered into the register g. When n = FF_H, it becomes 1's complement instruction.							
XOR gg, mn	1110 10gg 0011 1101 nnnn nnnn mmmm mmmm	E 8+gg 3D nn mm	gg←gg∀mn	Z	Z	–	–	4
	The data that is the logical exclusive OR of each bit of the 16-bit immediate data "mn" in the object code and the contents of the register pair gg is derived. The result is entered into the register pair gg. When mn = $FFFF_H$, it becomes 1's complement instruction.							
XOR (x), n	1110 0000 xxxx xxxx 0111 0101 nnnn nnnn	E0 xx 75 nn	(x)←(x)∀n	Z	Z	–	–	6
	The data that is the logical exclusive OR of each bit of the immediate data "n" in the object code and the contents of the memory address (0000-00FF_H) directly specified by the "x" in the object code is derived. The result is stored into the above memory address.							
XOR (pp), n	1110 001p 0111 0101 nnnn nnnn	E 2+pp 75 nn	(pp)←(pp)∀n	Z	Z	–	–	5
	The data that is the logical exclusive OR of each bit of the immediate data "n" in the object code and the contents of the memory address specified by the contents of the register pair pp (HL or DE) is derived. The result is stored into the above memory address.							
XOR (HL+), n	1110 0110 0111 0101 nnnn nnnn	E6 75 nn	(HL)←(HL)∀n, HL←HL+1	Z	Z	–	–	6
	The data that is the logical exclusive OR of each bit of the immediate data "n" in the object code and the contents of the memory address specified by the contents of the HL register pair is derived. The result is stored into the above memory address. Then, the HL register pair is incremented by one.							
XOR (–HL), n	1110 0111 0111 0101 nnnn nnnn	E7 75 nn	HL←HL–1, (HL)←(HL)∀n	Z	Z	–	–	6
	The HL register pair is decremented by one. Then the data that is the logical exclusive OR of each bit of the immediate data "n" in the object code and the contents of the memory address specified by the new contents of the HL register pair is derived. The result is stored into the above memory address.							
XOR (HL+d), n	1110 0100 dddd dddd 0111 0101 nnnn nnnn	E4 dd 75 nn	(HL+d)←(HL+d)∀n	Z	Z	–	–	7
	The data that is the logical exclusive OR of each bit of the immediate data "n" in the object code and the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair is derived. The result is stored into the above memory address.							

MNEMONIC	OBJECT CODE		OPERATION	FLAG				CYCLE
	Binary	Hex-decimal		JF	ZF	CF	HF	
XOR (HL+C), n	1110 0101 0111 0101 nnnn nnnn	E5 75 nn	(HL+C)←(HL+C)∀n	Z	Z	–	–	7
	The data that is the logical exclusive OR of each bit of the immediate data "n" in the object code and the contents memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair is derived. The result is stored into the above memory address.							
XOR (PC+A), n	1110 0001 0111 0101 nnnn nnnn	E1 75 nn	(PC+A)←(PC+A)∀n	Z	Z	–	–	7
	The data that is the logical exclusive OR of each bit of the immediate data "n" in the object code and the contents of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter is derived. The result is stored into the above memory address.							

Notes: 1. During ADD/ADDC/SUB/SUBB/AND/OR/XOR (src), (HL) instruction, the contents of the output latch are read in as the (src) memory data and the contents of the pin input are read in as the (HL) memory data if the I/O port (except the tri-state port) is accessed.

2. During ADD/ADDC/SUB/SUBB/AND/OR/XOR (src), n instruction, the contents of the output latch are read in if the I/O port (except the tri-state port) is accessed as the source memory.

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
DAA A	0000 1010	0A	Decimal Adjust A for Addition	C	Z	C	H	2

The contents of the accumulator are decimally adjusted after the addition instruction ADD/ADDC (except INC) has been executed to add 8-bit packed BCD data. If the result is greater than 99, the carry flag is se t to "1"; otherwise it is cleared to "0". When the multiple-digit BCD data is added, the addition and adjustment are repeated in 8-bit units starting from the lower order.

CF before adjustment	Upper 4 bits of Acc. before adjustment	HF before adjustment	Lower 4 bits of Acc. before adjustment	Adding value for adjustment	CF after adjustment
0	0~9	0	0~9	00	0
0	0~8	0	A~F	06	0
0	0~9	1	0~3	06	0
0	A~F	0	0~9	60	1
0	9~F	0	A~F	66	1
0	A~F	1	0~3	66	1
1	0~2	0	0~9	60	1
1	0~2	0	A~F	66	1
1	0~3	1	0~3	66	1

Example: When A = 26_H and B = 57_H, and the ADD A, B instruction is executed, the results will become A = $7D_H$, CF = 0 and HF = 0. When the DAA A instruction is executed, the results will become A = 83_H and CF = 0.

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
DAA g	1110 1ggg 0000 1010	E 8+g 0A	Decimal Adjust register g for Addition	C	Z	C	H	3

The contents of register g are decimally adjusted after the addition instruction ADD/ADDC (except INC) has been executed to add 8-bit packed BCD data. When the multiple-digit BCD data is added, the addition and adjustment are repeated in 8-bit units starting from the lower order.

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
DAS A	0000 1011	0B	Decimal Adjust A for Subtraction	C	Z	C	H	2

The contents of the accumulator are decimally adjusted after the subtraction instruction SUB/SUBB (except DEC) has been executed to subtract 8-bit packed BCD data. If the result is less than 00, the carry flag is set to "1"; otherwise it is cleared to "0". When the multiple-digit BCD data is subtracted, the subtraction and adjustment are repeated in 8-bit units starting from the lower order.

CF before adjustment	Upper 4 bits of Acc. before adjustment	HF before adjustment	Lower 4 bits of Acc. before adjustment	Adding value for adjustment	CF after adjustment
0	0~9	0	0~9	00	0
0	0~8	1	6~F	FA	0
1	7~F	0	0~9	A0	1
1	6~F	1	6~F	9A	1

Example: When A = 87_H and B = 39_H, and the SUB A B instruction is executed, the results will become A = $4E_H$, CF = 0 and HF = 1. When the DAS A instruction is executed, the results will become A = 48_H and CF = 0.

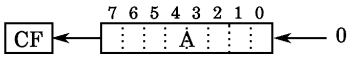
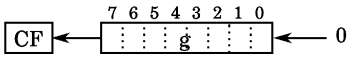
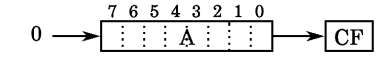
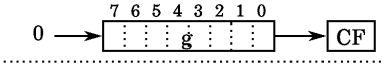
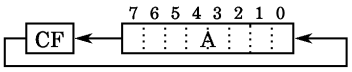
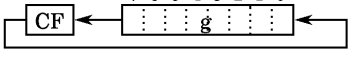
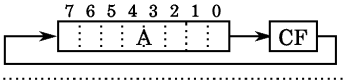
MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
DAS g	1110 1ggg 0000 1011	E 8+g 0B	Decimal Adjust register g for Subtraction	C	Z	C	H	3

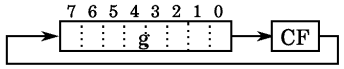
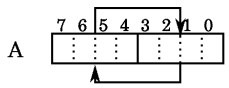
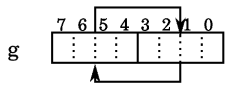
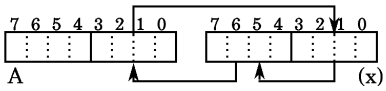
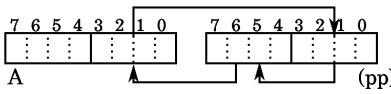
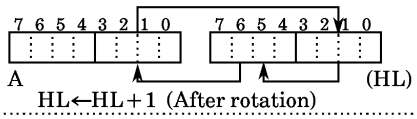
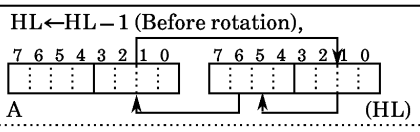
The contents of register g are decimally adjusted after the subtraction instruction SUB/SUBB (except DEC) has been executed to subtract 8-bit packed BCD data. When the multiple-digit BCD data is subtracted, the subtraction and adjustment are repeated in 8-bit units starting from the lower order.

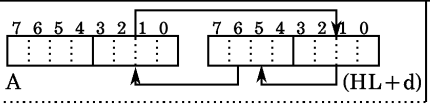
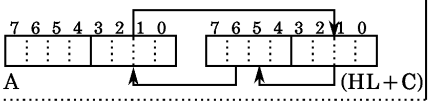
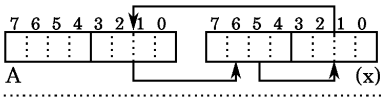
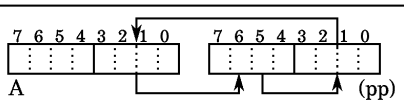
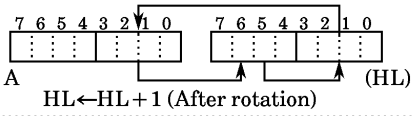
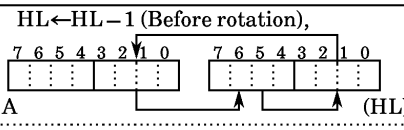
Note: The decimal adjustment instructions DAA/DAS are not the instruction for directly converting binary to BCD (binary coded decimal).

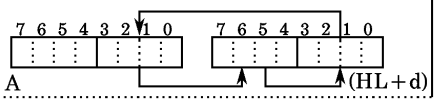
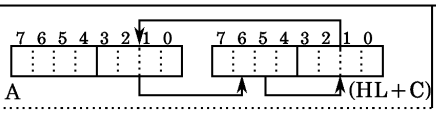
MNEMONIC	OBJECT CODE		OPERATION	FLAG				CYCLE
	Binary	Hex-decimal		JF	ZF	CF	HF	
MUL W, A	0000 0010	02	WA←W×A	Z	Z	–	–	7
	The contents of the W register (unsigned integer) are multiplied by the contents of the accumulator (unsigned integer) and the result (16-bit product) is entered into the 16-bit accumulator. When the product's upper 8 bits (data to be stored in the W register) are 00_H, the zero flag is set to "1". When other than 00_H, the zero flag is cleared to "0". Example 1: When $W = 87_H$ and $A = F2_H$, after executing this instruction, the product will become $WA = 7F9E_H$ and JF = ZF = 0. Example 2: When $W = 16_H$ and $A = 05_H$, after executing this instruction, the product will become $WA = 006E_H$ and JF = ZF = 1.							
MUL B, C	1110 1001 0000 0010	E9 02	BC←B×C	Z	Z	–	–	8
	The contents of the B register (unsigned integer) are multiplied by the contents of the C register (unsigned integer) and the result is entered into the BC register pair. When the product's upper 8 bits (data to be stored in the B register) are 00_H, the zero flag is set to "1".							
MUL D, E	1110 1010 0000 0010	EA 02	DE←D×E	Z	Z	–	–	8
	The contents of the D register (unsigned integer) are multiplied by the contents of the E register (unsigned integer) and the result is entered into the DE register pair. When the product's uppper 8 bits (data to be stored in the D register) are 00_H, the zero flag is set to "1".							
MUL H, L	1110 1011 0000 0010	EB 02	HL←H×L	Z	Z	–	–	8
	The contents of the H register (unsigned integer) are multiplied by the contents of the L register (unsigned integer) and the results are entered into the HL register. When the product's upper 8 bits (data to be stored in the H register) are 00_H, the zero flag is set to "1".							
DIV WA, C	0000 0011	03	A←WA÷C, W←Remainder	Z	Z	C	–	7
	The contents of the 16-bit accumulator (unsigned integer) are devided by the contents of the C register (unsigned integer), the quotient is entered into the accumulator and the remainder is entered into the W register. When the divisor (the contents of the C register) is 00_H or the quotient is 100_H or more, the carry flag is set to "1". When the divisor and the quotient are anything else, the carry flag is cleared to "0". (When the quotient is 100_H or more, undefined value is stored in the WA register pair.) Also, when the remainder is 00_H, the zero flag is set to "1", and when it is anything other than 00_H, it is cleared to "0". Note that the contents of the C register do not change. This instruction is suitable for radix conversion processing such as converting binary to BCD. Example 1: When $WA = 1234_H$ and $C = 56_H$, after executing the DIV WA, C instruction, the results will become $A = 36_H$, $W = 10_H$, CF = 0 and ZF = 0. Example 2: When $WA = 4830_H$ and $C = 9A_H$, after executing this instruction, the results will become $A = 78_H$, $W = 00_H$, CF = 0 and ZF = 1. Example 3: When $WA = 3210_H$ and $C = 27_H$, after executing this instruction, the results will become $A = 48_H$, $W = 18_H$, CF = 1 and ZF = 0.							
DIV DE, C	1110 1010 0000 0011	EA 03	E←DE÷C, D←Remainder	Z	Z	C	–	8
	The contents of the DE register pair (unsigned integer) are divided by the contents of the C register (unsigned integer), the quotient is entered into the E register and the remainder is entered into the D register. When the divisor (the contents of the C register) is 00_H or the quotient is 100_H or more, the carry flag is set to "1". (When the quotient is 100_H or more, undefined value is stored in the DE register pair.) Also, when the remainder is 00_H, the zero flag is set to "1". Note that the contents of the C register do not change.							
DIV HL, C	1110 1011 0000 0011	EB 03	L←HL÷C, H←Remainder	Z	Z	C	–	8
	The contents of the HL register pair (unsigned integer) are divided by the contents of the C register (unsigned integer), the quotient is entered into the L register and the remainder is entered into the H register. When the divisor (the contents of the C register) is 00_H or the quotient is 100_H or more, the carry flag is set to "1". (When the quotient is 100_H or more, undefined value is stored in the HL register pair.) Also, when the remainder is 00_H, the zero flag is set to "1". Note that the contents of the C register do not change.							

(3) SHIFT, ROTATE and NIBBLE MANIPULATION

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
SHLC A	0001 1100	1C		C	Z	*	–	1
	The contents of the accumulator are logically shifted 1 bit at a time to the left. ("0" is entered into the least significant bit of the accumulator. The original content of the accumulater's most significant bit is entered into the carry flag.) If the contents of the accumulator after the shift are 00_H, the zero flag is set to "1". When the contents of the accumulator are unsigned integer, the data is doubled by executing this instruction. Example: When A = 00111011_B and CF = 1, after executing the SHLC A instruction, the results will become A = 01110110_B, JF = 0, CF = 0 and ZF = 0.							
SHLC g	1110 1ggg 0001 1100	E 8+g 1C		C	Z	*	–	2
	The contents of register g are logically shifted 1 bit at a time to the left. ("0" is entered into the least significant bit of register g. The original content of the register g's most significant bit is entered into the carry flag.) If the contents of register g after the shift are 00_H, the zero flag is set to "1".							
SHRC A	0001 1101	1D		C	Z	*	–	1
	The contents of the accumulator are logically shifted 1 bit at a time to the right. ("0" is entered into the most significant bit of the accumulator. The original content of accumulator's least significant bit is entered into the carry flag.) If the contents of the accumulator after the shift are 00_H, the zero flag is set to "1". When the contents of the accumulator are unsigned integer, the data is doubled by executing this instruction. Example: When A = 01011101_B and CF = 0, after executing the SHRC A instruction, the results will become A = 00101110_B, JF = 1, CF = 1 and ZF = 0.							
SHRC g	1110 1ggg 0001 1101	E 8+g 1D		C	Z	*	–	2
	The contents of register g are logically shifted 1 bit at a time to the right. ("0" is entered into the most significant bit of the register g. The original content of the register g's least significant bit is entered into the carry flag.) If the contents of register g after the shift are 00_H, the zero flag is set to "1".							
ROLC A	0001 1110	1E		C	Z	*	–	1
	The contents of the accumulator and the carry flag are rotated 1 bit at a time to the left. (The most significant bit of the accumulator is entered into the carry flag; the original content of the carry flag is entered into the least significant bit of the accumulator.) If the contents of the accumulator after rotation are 00_H, the zero flag is set to "1"; otherwise it is cleared to "0". Example 1: When A = 10010110_B and CF = 0, after executing the ROLC A instruction, the results will become A = 00101100_B, JF = 1, CF = 1 and ZF = 0. Example 2: When A = 100000000_B and CF = 0, after executing the ROLC A instruction, the results will become A = 00000000_B, JF = 1, CF = 1 and ZF = 1.							
ROLC g	1110 1ggg 0001 1110	E 8+g 1E		C	Z	*	–	2
	The contents of register g and the carry flag are rotated 1 bit at a time to the left. (The most significant bit of the register g is entered into the carry flag.) If the contents of register g after rotation are 00_H, the zero flag is set to "1"; otherwise it is cleared to "0".							
RORC A	0001 1111	1F		C	Z	*	–	1
	The contents of the accumulator and the carry flag are rotated 1 bit at a time to the right. (The least significant bit of the accumulator is entered into the carry flag; the original content of the carry flag is entered into the most significant bit of the accumulator.) If the contents of the accumulator are 00_H after rotation, the zero flag is set to "1"; otherwise it is cleared to "0". Example: When A = 01101101_B and CF = 1, after executing the RORC A instruction, the reuslts will become A = 101101110_B, JF = 1, CF = 1 and ZF = 0.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
RORC g	1110 1ggg 0001 1111	E 8+g 1F		C	Z	*	–	2
	The contents of register g and the carry flag are rotated 1 bit at a time to the right. If the contents of register g are 00_H after rotation, the zero flag is set to "1"; otherwise it is cleared to "0".							
SWAP A	0000 0001	01		1	–	–	–	3
	The accumulator's upper 4-bit contents are swapped for its lower 4-bit contents. Example: When A = 25_H, after executing the instruction, the results will become A = 52_H and JF = 1.							
SWAP g	1110 1ggg 0000 0001	E 8+g 01		1	–	–	–	4
	The register g's upper 4-bit contents are swapped for its lower 4-bit contents.							
ROLD A, (x)	1110 0000 XXXX XXXX 0000 1000	E0 XX 08		1	–	–	–	8
	The 12-bit data obtained by linking the accumulator's lower 4-bit contents to the contents of the memory address (0000-00FF_H) directly specified by the "x" in the object code is rotated in 4-bit units to the left. Note that there is no change in the accumulator's upper 4-bit contents. Example: When A = 12_H and the memory contents at the address 0087_H are 56_H, after executing the ROLD A, (87H) instruction, the A will become 15_H and the memory contents at the address 0087_H will become 62_H.							
ROLD A, (pp)	1110 001p 0000 1000	E 2+pp 08		1	–	–	–	7
	The 12-bit data obtained by linking the accumulator's lower 4-bit contents to the contents of the memory address specified by the contents of the register pair pp (HL or DE) is rotated in 4-bit units to the left. Note that there is no change in the accumulator's upper 4-bit contents.							
ROLD A, (HL+)	1110 0110 0000 1000	E6 08		1	–	–	–	8
	The 12-bit data obtained by linking the accumulator's lower 4-bit contents to the contents of the memory address specified by the contents of the HL register pair is rotated in 4-bit units to the left. Then, the HL register pair is incremented by one. Note that there is no change in the accumulator's upper 4-bit contents. This instruction is well suited to the packing and unpacking processing of multiple-digit BCD data.							
ROLD A, (–HL)	1110 0111 0000 1000	E7 08		1	–	–	–	8
	The HL register pair is decremented by one. Then the 12-bit data obtained by linking the accumulator's lower 4-bit contents to the contents of the memory address specified by the new contents of the HL register pair is rotated in 4-bit units to the left. Note that there is no change in the accumulator's upper 4-bit contents.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
ROLD A, (HL+d)	1110 0100 dddd dddd 0000 1000	E4 dd 08		1	–	–	–	9
	The 12-bit data obtained by linking the accumulator's lower 4-bit contents to the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair is rotated in 4-bit units to the left. Note that there is no change in the accumulator's upper 4-bit contents.							
ROLD A, (HL+C)	1110 0101 0000 1000	E5 08		1	–	–	–	9
	The 12-bit data obtained by linking the accumulator's lower 4-bit contents to the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair is rotated in 4-bit units to the left. Note that there is no change in the accumulator's upper 4-bit contents.							
RORD A, (x)	1110 0000 xxxx xxxx 0000 1001	E0 xx 09		1	–	–	–	8
	The 12-bit data obtained by linking the accumulator's lower 4-bit contents to the contents of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code is rotated in 4-bit units to the right. Note that there is no change in the accumulator's upper 4-bit contents. Example: When A = 12$_H$ and the memory contents at the address 0087$_H$ are 56$_H$, after executing the RORD A, (87H) instruction, the A register will become 16$_H$ and the contents at the address 0087$_H$ will become 25$_H$.							
RORD A, (pp)	1110 001p 0000 1001	E 2+pp 09		1	–	–	–	7
	The 12-bit data obtained by linking the accumulator's lower 4-bit contents to the contents of the memory address specified by the contents of the register pair pp (HL or DE) is rotated in 4-bit units to the right. Note that there is no change in the accumulator's upper 4-bit contents.							
RORD A, (HL+)	1110 0110 0000 1001	E6 09		1	–	–	–	8
	The 12-bit data obtained by linking the accumulator's lower 4-bit contents to the contents of the memory address specified by the contents of the HL register pair is rotated in 4-bit units to the right. Then, the HL register pair is incremented by one. Note that there is no change in the accumulator's upper 4-bit contents. This instruction is well suited to the packing and unpacking processing of multiple-digit BCD data.							
RORD A, (–HL)	1110 0111 0000 1001	E7 09		1	–	–	–	8
	The HL register pair is decremented by one. Then the 12-bit data obtained by linking the accumulator's lower 4-bit contents to the contents of the memory address specified by the new contents of the HL register pair is rotated in 4-bit units to the right. Note that there is no change in the accumulator's upper 4-bit contents.							

MNEMONIC	OBJECT CODE		OPERATION	FLAG				CYCLE
	Binary	Hex-decimal		JF	ZF	CF	HF	
RORD A, (HL+d)	1110 0100 dddd dddd 0000 1001	E4 dd 09		1	–	–	–	9
	The 12-bit data obtained by linking the accumulator's lower 4-bit conents to the contents of the memory address specified by the value obtained by adding the sign-extended 8-bit data d in the object code to the contents of the HL register pair is rotated in 4-bit units to the right. Note that there is no change in the accumulator's upper 4-bid contents.							
RORD A, (HL+C)	1110 0101 0000 1001	E5 09		1	–	–	–	9
	The 12-bit data obtained by linking the accumulator's lower 4-bit contents to the contents of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair is rotated in 4-bit units to the right. Note that there is no change in the accumulator's upper 4-bit contents.							

(4) BIT MANIPULATION

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
TEST g. b	1110 1ggg 1101 1bbb	E 8+g D 8+b	JF←$\overline{g.b}$	*	–	$\overline{J}$	–	2
	The inverted data of the content of the bit directly specified by the "b" in the object code of the register g is entered into the jump status flag. This instruction is similar to LD CF, g.b instruction. Example: When A = 01011100_B, after executing the TEST A.5 instruction, the results will become JF = 1 and CF = 0.							
TEST (x). b	1101 1bbb xxxx xxxx	D 8+b xx	JF←$\overline{(x).b}$	*	–	$\overline{J}$	–	4
	The inverted data of the content of the bit specified by the "b" in the object code of the memory address (0000-00FF_H) directly specified by the "x" in the object code is entered into the jump status flag.							
TEST (pp). b	1110 001p 1101 1bbb	E 2+pp D 8+b	JF←$\overline{(pp).b}$	*	–	$\overline{J}$	–	3
	The inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the contents of the register pair pp (HL or DE) is entered into the jump status flag.							
TEST (HL+). b	1110 0110 1101 1bbb	E6 D 8+b	JF←$\overline{(HL).b}$, HL←HL+1	*	–	$\overline{J}$	–	4
	The inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the contents of the HL register pair is entered into the jump status flag. Then, the HL register pair is incremented by one.							
TEST (–HL). b	1110 0111 1101 1bbb	E7 D 8+b	HL←HL–1, JF←$\overline{(HL).b}$	*	–	$\overline{J}$	–	4
	The HL register pair is decremented by one. Then the inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the new contents of the HL register pair is entered into the jump status flag.							
TEST (HL+d). b	1110 0100 dddd dddd 1101 1bbb	E4 dd D 8+b	JF←$\overline{(HL+d).b}$	*	–	$\overline{J}$	–	5
	The inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair is entered into the jump status flag.							
TEST (HL+C). b	1110 0101 1101 1bbb	E5 D 8+b	JF←$\overline{(HL+C).b}$	*	–	$\overline{J}$	–	5
	The inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair is entered into the jump status flag.							
TEST (PC+A). b	1110 0001 1101 1bbb	E1 D 8+b	JF←$\overline{(PC+A).b}$	*	–	$\overline{J}$	–	5
	The inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter is entered into the jump status flag.							
TEST (pp). g	1110 1ggg 1001 111p	E 8+g 9 E+pp	JF←$\overline{(pp).g_{2\sim0}}$	*	–	$\overline{J}$	–	4
	The inverted data of the content of the bit specified by the contents of the register g's lower 3 bits of the memory address specified by the contents of the register pair pp (HL or DE) is entered into the jump status flag. This instruction is suitable for bit search processing. Example: When HL = 0123_H and A = 45_H and the memory contents at the address (0123_H) are 39_H, after executing the TEST (HL).A instruction, the results will become JF = 0 and CF = 1.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
CLR g. b	1110 1ggg 0100 1bbb	E 8+g 4 8+b	ZF←$\overline{\text{g. b}}$, g. b←0	Z	*	−	−	3
	After the inverted data of the content of the bit directly specified by the "b" in the object code of the register g is entered into the zero flag, the content of this bit is cleared to "0". Example: When A = $3C_H$, and the CLR A.2 instruction is executed, the results will become ZF = 0 and A = 38_H.							
CLR (x). b	0100 1bbb xxxx xxxx	4 8+b xx	ZF←$\overline{\text{(x). b}}$, (x). b←0	Z	*	−	−	5
	After the inverted data of the content of the bit specified by the "b" in the object code of the memory address (0000-00FF_H) directly specified by the "x" in the object code is entered into the zero flag, the content of the bit is cleared to "0".							
CLR (pp). b	1110 001p 0100 1bbb	E 2+pp 4 8+b	ZF←$\overline{\text{(pp). b}}$, (pp). b←0	Z	*	−	−	4
	After the inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the contents of the register pair pp (HL or DE) is entered into the zero flag, the content of the bit is cleared to "0".							
CLR (HL+). b	1110 0110 0100 1bbb	E6 4 8+b	ZF←$\overline{\text{(HL). b}}$, (HL). b←0, HL←HL+1	Z	*	−	−	5
	After the inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the contents of the HL register pair is entered into the zero flag, the content of the bit is cleared to "0". Then, the HL register pair is incremented by one.							
CLR (−HL). b	1110 0111 0100 1bbb	E7 4 8+b	HL←HL−1, ZF←$\overline{\text{(HL). b}}$, (HL). b←0	Z	*	−	−	5
	The HL register pair is decremented by one. After the inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the new contents of the HL register pair is entered into the zero flag, the content of the bit is cleared to "0".							
CLR (HL+d). b	1110 0100 dddd dddd 0100 1bbb	E4 dd 4 8+b	ZF←$\overline{\text{(HL+d). b}}$, (HL+d). b←0	Z	*	−	−	6
	After the inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair is entered into the zero flag, the content of the bit is cleared to "0".							
CLR (HL+C). b	1110 0101 0100 1bbb	E5 4 8+b	ZF←$\overline{\text{(HL+C). b}}$, (HL+C). b←0	Z	*	−	−	6
	After the inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair is entered into the zero flag, the content of the bit is cleared to "0".							
CLR (PC+A). b	1110 0001 0100 1bbb	E1 4 8+b	ZF←$\overline{\text{(PC+A). b}}$, (PC+A). b←0	Z	*	−	−	6
	After the inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter is entered into the zero flag, the content of the bit is cleared to "0".							
CLR (pp). g	1110 1ggg 1000 101p	E 8+g 8 A+pp	ZF←$\overline{\text{(pp). }g_{2\sim0}}$, (pp). $g_{2\sim0}$←0	Z	*	−	−	5
	After the inverted data of the contents of the bit specified by the register g's lower 3-bit contents of the memory address specified by the contents of the register pair pp (HL or DE) is entered into the zero flag, the content of the bit is cleared to "0". Example: When DE = 0125_H, W = 86_H and the memory contents at the address 0125_H are $6B_H$, after executing the CLR (DE).W instruction, the ZF will be cleared to "0" and the memory contents at the address 0125_H will become $2B_H$.							

Note : The bit test and clear instruction (CLR) is the read-modify-write instruction.

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
SET g. b	1110 1ggg 0100 0bbb	E 8+g 4b	ZF←$\overline{\text{g. b}}$, g. b←1	Z	*	−	−	3
	After the inverted data of the content of the bit directly specified by the "b" in the object code of the register g is entered into the zero flag, the content of the bit is set to "1".							
SET (x). b	0100 0bbb xxxx xxxx	4b xx	ZF←$\overline{\text{(x). b}}$, (x). b←1	Z	*	−	−	5
	After the inverted data of the content of the bit specified by the "b" in the object code of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code is entered into the zero flag, the content of the bit is set to "1".							
SET (pp). b	1110 001p 0100 0bbb	E 2+pp 4b	ZF←$\overline{\text{(pp). b}}$, (pp). b←1	Z	*	−	−	4
	After the inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the contents of the register pair pp (HL or DE) is entered into the zero flag, the content of the bit is set to "1".							
SET (HL+). b	1110 0110 0100 0bbb	E6 4b	ZF←$\overline{\text{(HL). b}}$, (HL). b←1, HL←HL+1	Z	*	−	−	5
	After the inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the HL register pair is entered into the zero flag, the content of the bit is set to "1". Then, the HL register pair is incremented by one.							
SET (−HL). b	1110 0111 0100 0bbb	E7 4b	HL←HL−1, ZF←$\overline{\text{(HL). b}}$, (HL). b←1	Z	*	−	−	5
	The HL register pair is decremented by one. After the inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the new content of the HL register pair is entered into the zero flag, the content of the bit is set to "1".							
SET (HL+d). b	1110 0100 dddd dddd 0100 0bbb	E4 dd 4b	ZF←$\overline{\text{(HL+d). b}}$, (HL+d). b←1	Z	*	−	−	6
	After the inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair is entered into the zero flag, the content of the bit is set to "1".							
SET (HL+C). b	1110 0101 0100 0bbb	E5 4b	ZF←$\overline{\text{(HL+C). b}}$, (HL+C). b←1	Z	*	−	−	6
	After the inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair is entered into the zero flag, the content of the bit is set to "1".							
SET (PC+A). b	1110 0001 0100 0bbb	E1 4b	ZF←$\overline{\text{(PC+A). b}}$, (PC+A). b←1	Z	*	−	−	6
	After the inverted data of the content of the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter is entered into the zero flag, the content of the bit is set to "1".							
SET (pp). g	1110 1ggg 1000 001p	E 8+g 8 2+pp	ZF←$\overline{\text{(pp). g}_{2\sim0}}$, (pp). g$_{2\sim0}$←1	Z	*	−	−	5
	After the inverted data of the content of the bit specified by the register g's lower 3-bit contents at the memory address specified by the contents of the register pair pp (HL or DE) is entered into the zero flag, the content of the bit is set to "1". Example: When DE = 0125$_H$, W = 82$_H$ and the memory contents at the address 0125$_H$ are 6B$_H$, after executing the SET (DE).W instruction, the ZF will be set to "1" and the memory contents at the address 0125$_H$ will become 6F$_H$.							

Note : The bit test and set instruction (SET) is the read-modify-write instruction.

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
CPL g. b	1110 1ggg 1100 0bbb	E 8+g C b	ZF←$\overline{\text{g. b}}$, g. b←$\overline{\text{g. b}}$	Z	*	—	—	3
	The content of the bit directly specified by the "b" in the object code of the register g is inverted. Example: When A = $3C_H$, after executing the CPL A.3 instruction, the results will become Z = 0 and A = 34_H.							
CPL (x). b	1110 0000 xxxx xxxx 1100 0bbb	E0 xx C b	ZF←$\overline{\text{(x). b}}$, (x). b←$\overline{\text{(x). b}}$	Z	*	—	—	5
	The content of the bit specified by the "b" in the object code of the memory address (0000-00FF_H) directly specified by the "x" in the object code is inverted.							
CPL (pp). b	1110 001p 1100 0bbb	E 2+pp C b	ZF←$\overline{\text{(pp). b}}$, (pp). b←$\overline{\text{(pp). b}}$	Z	*	—	—	4
	The content of the bit specified by the "b" in the object code of the memory address specified by the contents of the register pair pp (HL or DE) is inverted.							
CPL (HL+). b	1110 0110 1100 0bbb	E6 C b	ZF←$\overline{\text{(HL). b}}$, (HL). b←$\overline{\text{(HL). b}}$, HL←HL+1	Z	*	—	—	5
	The content of the bit specified the "b" in the object code of the memory address specified by the contents of the HL register pair is inverted. Then, the HL register pair is incremented by one.							
CPL (−HL). b	1110 0111 1100 0bbb	E7 C b	HL←HL−1, ZF←$\overline{\text{(HL). b}}$, (HL). b←$\overline{\text{(HL). b}}$	Z	*	—	—	5
	The HL register pair is decremented by one. Then the content of the bit specified by the "b" in the object code of the memory address specified by the new contents of the HL register pair is inverted.							
CPL (HL+d). b	1110 0100 dddd dddd 1100 0bbb	E4 d d C b	ZF←$\overline{\text{(HL+d). b}}$, (HL+d). b←$\overline{\text{(HL+d). b}}$	Z	*	—	—	6
	The content of bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair is inverted.							
CPL (HL+C). b	1110 0101 1100 0bbb	E5 C b	ZF←$\overline{\text{(HL+C). b}}$, (HL+C). b←$\overline{\text{(HL+C). b}}$	Z	*	—	—	6
	The content of bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair is inverted.							
CPL (PC+A). b	1110 0001 1100 0bbb	E1 C b	ZF←$\overline{\text{(PC+A). b}}$, (PC+A). b←$\overline{\text{(PC+A). b}}$	Z	*	—	—	6
	The content of bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter is inverted.							
CPL (pp). g	1110 1ggg 1001 001p	E 8+g 9 2+pp	ZF←$\overline{\text{(pp). }g_{2\sim0}}$, (pp). $g_{2\sim0}$←$\overline{\text{(pp). }g_{2\sim0}}$	Z	*	—	—	5
	The content of bit specified by the register g's lower 3-bit contents of the memory address specified by the contents of the register pair pp (HL or DE) is inverted. Example: When HL = 0125_H, B = 58_H and the memory contents at the address 0125_H are $6B_H$, after executing the CPL (HL).B instruction, the ZF will be cleared to "0" and the memory contents at the address 0125_H will become $6A_H$.							
LD CF, g. b	1110 1ggg 1101 1bbb	E 8+g D 8+b	CF←g. b	$\overline{C}$	—	*	—	2
	The content of the bit directly specified by the "b" in the object code of the register g is loaded into the carry flag. Example: When A = 01101101_B, after executing the LD CF, A.4 instruction, the results will become CF = 0 and JF = 1.							

Note : The bit test and complement instruction (CPL) is the read-modify-write instruction.

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
LD CF, (x). b	1101 1bbb xxxx xxxx	D 8+b xx	CF←(x). b	$\overline{C}$	–	*	–	4
	The content of the bit specified by the "b" in the object code (first byte) of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code (second byte) is loaded into the carry flag.							
LD CF, (pp). b	1110 001p 1101 1bbb	E 2+pp D 8+b	CF←(pp). b	$\overline{C}$	–	*	–	3
	The content of the bit specified by the "b" in the object code of the memory address specified by the contents of the register pair pp (HL or DE) is loaded into the carry flag.							
LD CF, (HL+). b	1110 0110 1101 1bbb	E6 D 8+b	CF←(HL). b, HL←HL+1	$\overline{C}$	–	*	–	4
	The content of the bit specified by the "b" in the object code of the memory address specified by the HL register pair is loaded into the carry flag. Then, the HL register pair is incremented by one.							
LD CF, (–HL). b	1110 0111 1101 1bbb	E7 D 8+b	HL←HL–1, CF←(HL). b	$\overline{C}$	–	*	–	4
	The HL register pair is decremented by one. Then the content of the bit specified by the "b" in the object code of the memory address specified by the new contents of the HL register pair is loaded into the carry flag.							
LD CF, (HL+d). b	1110 0100 dddd dddd 1101 1bbb	E4 dd D 8+b	CF←(HL+d). b	$\overline{C}$	–	*	–	5
	The content of the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair is loaded into the carry flag.							
LD CF, (HL+C). b	1110 0101 1101 1bbb	E5 D 8+b	CF←(HL+C). b	$\overline{C}$	–	*	–	5
	The content of the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair is loaded into the carry flag.							
LD CF, (PC+A). b	1110 0001 1101 1bbb	E1 D 8+b	CF←(PC+A). b	$\overline{C}$	–	*	–	5
	The content of the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter is loaded into the carry flag.							
LD CF, (pp). g	1110 1ggg 1001 111p	E 8+g 9 E+pp	CF←(pp). $g_{2\sim0}$	$\overline{C}$	–	*	–	4
	The content of the bit specified by the register g's lower 3-bit contents of the memory address specified by the contents of the register pair pp (HL or DE) is loaded into the carry flag.							
LD g. b, CF	1110 1ggg 1100 1bbb	E 8+g C 8+b	g. b←CF	1	–	–	–	2
	The content of the carry flag is entered into the bit directly specified by the "b" in the object code of the register g. Note that the content of the carry flag is not changed. Example 1: When A = 15$_H$ and CF = 1, after executing the LD A.5, CF instruction, the results will become A = 35$_H$ and CF = 1. Example 2: When B = 7E$_H$ and CF = 0, after executing the LD B.2, CF instruction, the results will become B = 7A$_H$ and CF = 0.							
LD (x). b, CF	1110 0000 xxxx xxxx 1100 1bbb	E0 xx C 8+b	(x). b←CF	1	–	–	–	5
	The content of the carry flag is stored into the bit specified by the "b" in the object code of the memory address (0000-00FF$_H$) directly specified by the "x" in the object code.							

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
LD (pp). b, CF	1110 001p 1100 1bbb	E 2+pp C 8+b	(pp). b←CF	1	–	–	–	4
	The content of the carry flag is stored into the bit specified by the "b" in the object code of the memory address specified by the contents of the register pair pp (HL or DE).							
LD (HL+). b, CF	1110 0110 1100 1bbb	E6 C 8+b	(HL). b←CF, HL←HL+1	1	–	–	–	5
	The content of the carry flag is stored into the bit specified by the "b" in the object code of the memory address specified by the contents of the HL register pair. Then, the HL register pair is incremented by one.							
LD (–HL). b, CF	1110 0111 1100 1bbb	E7 C 8+b	HL←HL–1, (HL). b←CF	1	–	–	–	5
	The HL register pair is decremented by one. Then the content of the carry flag is stored into the bit specified by the "b" in the object code of the memory address specified by the new contents of the HL register pair.							
LD (HL+d). b, CF	1110 0100 dddd dddd 1100 1bbb	E4 dd C 8+b	(HL+d). b←CF	1	–	–	–	6
	The content of the carry flag is stored into the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair.							
LD (HL+C). b, CF	1110 0101 1100 1bbb	E5 C 8+b	(HL+C). b←CF	1	–	–	–	6
	The content of the carry flag is stored into the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair.							
LD (PC+A). b, CF	1110 0001 1100 1bbb	E1 C 8+b	(PC+A). b←CF	1	–	–	–	6
	The content of the carry flag is stored into the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter.							
LD (pp). g, CF	1110 1ggg 1001 101p	E 8+g 9 A+pp	(pp). $g_{2\sim0}$←CF	1	–	–	–	5
	The content of the carry flag is stored into the bit specified by the lower 3 bits of register g of the memory address specified by the contents of the register pair pp (HL or DE).							
XOR CF, g. b	1110 1ggg 1101 0bbb	E 8+g Db	CF←CF∀g. b	$\overline{C}$	–	*	–	2
	The data that is the logical exclusive-OR of the content of the carry flag and the content of the bit directly specified by the "b" in the object code of the register g is derived. The result is entered into the carry flag. Example: When A = $5B_H$ and CF = 1, after executing the XOR CF, A.3 instruction, the results will become A = $5B_H$, CF = 0 and JF = 1.							
XOR CF, (x). b	1110 0000 xxxx xxxx 1101 0bbb	E0 xx Db	CF←CF∀(x). b	$\overline{C}$	–	*	–	4
	The data that is the logical exclusive-OR of the content of the carry flag and the content of the bit specified by the "b" in the object code of the memory address (0000-00FF_H) directly specified by the "x" in the object code is derived. The result is entered into the carry flag.							
XOR CF, (pp). b	1110 001p 1101 0bbb	E 2+pp Db	CF←CF∀(pp). b	$\overline{C}$	–	*	–	3
	The data that is the logical exclusive-OR of the content of the carry flag and the content of the bit specified by the "b" in the object code of the memory address specified by the contents of the register pair pp (HL or DE) is derived. The result is entered into the carry flag.							

Note : The bit store instruction is the read-modify-write instruction.

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
XOR CF, (HL+). b	1110 0110 1101 0bbb	E6 Db	CF←CF∀(HL). b, HL←HL+1	$\overline{C}$	–	*	–	4
	The data that is the logical exclusive-OR of the content of the carry flag and the content of the bit specified by the "b" in the object code of the memory address specified by the contents of the HL register pair is derived. The result is entered into the carry flag. Then, the HL register pair is incremented by one.							
XOR CF, (–HL). b	1110 0111 1101 0bbb	E7 Db	HL←HL–1, CF←CF∀(HL). b	$\overline{C}$	–	*	–	4
	The HL register pair is decremented by one. Then the data that is the logical exclusive-OR of the content of the carry flag and the content of the bit specified by the "b" in the object code of the memory address specified by the new contents of the HL register pair is derived. The result is entered into the carry flag.							
XOR CF, (HL+d). b	1110 0100 dddd dddd 1101 0bbb	E4 dd Db	CF←CF∀(HL+d). b	$\overline{C}$	–	*	–	5
	The data that is the logical exclusive-OR of the content of the carry flag and the content of the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair is derived. The result is entered into the carry flag.							
XOR CF, (HL+C). b	1110 0101 1101 0bbb	E5 Db	CF←CF∀(HL+C). b	$\overline{C}$	–	*	–	5
	The data that is the logical exclusive-OR of the content of the carry flag and the content of the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair is derived. The results is entered into the carry flag.							
XOR CF, (PC+A). b	1110 0001 1101 0bbb	E1 Db	CF←CF∀(PC+A). b	$\overline{C}$	–	*	–	5
	The data that is the logical exclusive-OR of the content of the carry flag and the content of the bit specified by the "b" in the object code of the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter is derived. The result is entered into the carry flag.							

Note: During SET/CLR/CPL (src).b or SET/CLR/CPL (pp).g or LD (src).b, CF or LD (pp).g, CF instruction, the contents of the output latch are read in if the I/O port except the tri-state port is accessed as the source memory (src)/(pp).

(5) FLAG MANIPULATION

MNEMONIC	OBJECT CODE		OPERATION	FLAG				CYCLE
	Binary	Hex-decimal		JF	ZF	CF	HF	
CLR CF	0000 1100	0C	CF←0	1	—	0	—	1
	The carry flag is cleared to "0".							
SET CF	0000 1101	0D	CF←1	0	—	1	—	1
	The carry flag is set to "1".							
CPL CF	0000 1110	0E	JF←CF, CF←$\overline{CF}$	*	—	*	—	1
	The content of the carry flag is entered into the jump status flag. Then, the content of the carry flag is inverted. Example: When CF = 0, after executing this instruction, the results will become JF = 0, CF = 1.							
DI	0100 1000 0011 1010	48 3A	ZF ← $\overline{IMF}$, IMF←0	Z	*	—	—	5
	The inverted data of the content of the interrupt master enable flag (IMF) is entered into the zero flag. The interrupt master enable flag is then cleared to "0" (acceptance of all maskable interrupts are disabled). Example: When IMF = 1, after executing this instruction, the results will become JF = ZF = 0, IMF = 0.							
EI	0100 0000 0011 1010	40 3A	ZF ← $\overline{IMF}$, IMF←1	Z	*	—	—	5
	The inverted data of the content of the interrupt master enable flag (IMF) is entered into the zero flag. Then the interrupt master enable flag is set to "1" (acceptance of all interrupts are enabled). Example: When IMF = 0, after executing this instruction, the results will become JF = ZF = 1, IMF = 1.							

(6) JUMP

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
JRS T, a	100d dddd	80+d	if JF=1 then PC←PC+d else null (a=$+2+d)	1	–	–	–	4/2
	If the jump status flag is "1", a jump is made to the memory address specified by the value obtained by adding the sign-extended 5-bit displacement "d" in the object code (– 16 to + 15) to the current contents of the program counter (PC = $ + 2). In this case, the JRS is the 4-cycle instruction. If the jump status flag is "0", the jump status flag is set to "1" and the CPU proceeds with the next sequential instruction. In this case, the JRS is the 2-cycle instruction. Example: When the JF is "1", the JRS T,$ + 9 instruction at the address $C134_H$ is executed, a jump is made to the address $C13D_H$.							
JRS F, a	101d dddd	A0+d	if JF=0 then PC←PC+d else null (a=$+2+d)	1	–	–	–	4/2
	If the jump status flag is "0", a jump is made to the memory address specified by the value obtained by adding the coded extended 5-bit displacement "d" in the object code (– 16 to + 15) to the contents of the program counter. In this case, the JRS is the 4-cycle instruction. If the jump status flag is "1", a jump is not made, and the CPU proceeds with the next sequential instruction. In this case, the JRS is the 2-cycle instruction.							
JR T, a	1101 0110 dddd dddd	D6 dd	if JF=1 then PC←PC+d else null (a=$+2+d)	1	–	–	–	4/2
	If the jump status flag is "1", a jump (4-cycle instruction) is made to the memory address specified by the value obtained by adding the sign-extended 8-bit displacement "d" in the object code (– 128 to + 127) to the current contents of the program counter (which indicate the next instruction). If the jump status flag is "0", the jump status flag is set to "1", and the CPU proceeds with the next sequential instruction. In this case, the JR is the 2-cycle instruction. Example: When the JF is "1", and the JR T,$ + 0F6H instruction at the address $C134_H$ is executed, a jump is made to the address $C12A_H$.							
JR F, a	1101 0111 dddd dddd	D7 dd	if JF=0 then PC←PC+d else null (a=$+2+d)	1	–	–	–	4/2
	If the jump status flag is "0", a jump (4-cycle instruciton) is made to the address specified by the value obtained by adding the sign-extended 8-bit displacement "d" in the object code (– 128 to + 127) to the current contents of the program counter. If the jump status flag is "1", a jump is not made, and the CPU proceeds with the next sequential instruction. In this case, the JR is the 2-cycle instruction.							
JR EQ, a or JR Z, a	1101 0000 dddd dddd	D0 dd	if ZF=1 then PC←PC+d else null (a=$+2+d)	1	–	–	–	4/2
	If the zero flag is "1", a jump (4-cycle instruction) is made to the memory address specified by the value obtained by adding the sign-extended 8-bit displacement "d" in the object code (– 128 to + 127) to the current contents of the program counter. If the zero flag is "0", the jump status flag is set to "1", and the CPU proceeds with the next sequential instruction. In this case, the JR is the 2-cycle instruction.							
JR NE, a or JR NZ, a	1101 0001 dddd dddd	D1 dd	if ZF=0 then PC←PC+d else null (a=$+2+d)	1	–	–	–	4/2
	If the zero flag is "0", a jump (4-cycle instruction) is made to the memory address specified by the value obtained by adding the sign-extended 8-bit displacement "d" in the object code (– 128 to + 127) to the current contents of the program counter. If the zero flag is "1", the jump status flag is set to "1", and the CPU proceeds with the next sequential instruction. In this case, the JR is the 2-cycle instruction.							
JR CS, a or JR LT, a	1101 0010 dddd dddd	D2 dd	if CF=1 then PC←PC+d else null (a=$+2+d)	1	–	–	–	4/2
	If the carry flag is "1", a jump (4-cycle instruction) is made to the memory address specified by the value obtained by adding the sign-extended 8-bit displacement "d" in the object code (– 128 to + 127) to the current contents of the program counter. If the zero flag is "0", the jump status flag is set to "1", and the CPU proceeds with the next sequential instruction. In this case, the JR is the 2-cycle instruction.							

MNEMONIC	OBJECT CODE		OPERATION	FLAG				CYCLE
	Binary	Hex-decimal		JF	ZF	CF	HF	
JR CC, a or JR GE, a	1101 0011 dddd dddd	D3 dd	if CF=0 then PC←PC+d else null (a=$+2+d)	1	–	–	–	4/2
If the carry flag is "0", a jump (4-cycle instruction) is made to the memory address specified by the value obtained by adding the sign-extended 8-bit displacement "d" in the object code (– 128 to + 127) to the current contents of the program counter. If the carry flag is "1", the jump status flag is set to "1", and the CPU proceeds with the next instruction. In this case, the JR is the 2-cycle instruction.								
JR LE, a	1101 0100 dddd dddd	D4 dd	if (CF∨ZF)=1 then PC←PC+d else null (a=$+2+d)	1	–	–	–	4/2
If the carry flag or the zero flag is "1", a jump (4-cycle instruction) is made to the memory address specified by the value obtained by adding the sign-extended 8-bit displacement "d" in the object code (– 128 to + 127) to the current contents of the program counter. If both the carry flag and the zero flag are "0", the jump status flag is set to "1", and the CPU proceeds with the next instruction. In this case, the JR is the 2-cycle instruction.								
JR GT, a	1101 0101 dddd dddd	D5 dd	if (CF∨ZF)=0 then PC←PC+d else null (a=$+2+d)	1	–	–	–	4/2
If both the carry flag and the zero flag are "0", a jump (4-cycle instruction) is made to the memory address specified by the value obtained by adding the sign-extended 8-bit displacement "d" in the object code (– 128 to + 127) to the current contents of the program counter. If the carry flag or the zero flag is "1", the jump status flag is set to "1", and the CPU proceeds with the next instruction. In this case, the JR is the 2-cycle instruction.								
JR a	1111 1011 dddd dddd	FB dd	PC←PC+d (a=$+2+d)	1	–	–	–	4
An unconditional jump is made to the memory address specified by the value obtained by adding the sign-extended 8-bit displacement "d" in the object code (– 128 to + 127) to the current contents of the program counter (which indicate the next instruction). When d = FE_H, it provides an endless looping. Example: When the JR $ + 73H instruction at the address $D5A7_H$ is executed, a jump is made to the address $D61C_H$.								
JP mn	1111 1110 nnnn nnnn mmmm mmmm	FE nn mm	PC←mn	1	–	–	–	4
An unconditional jump is made to the memory address directly specified by the 16-bit data "mn" in the object code.								
JP gg	1110 10gg 1111 1110	E 8+gg FE	PC←gg	1	–	–	–	3
An unconditional jump is made to the memory address specified by the 16-bit register gg. Example: When HL = $E325_H$ and the JP HL instruction is executed, a jump is made to the address $E325_H$.								
JP (x)	1110 0000 xxxx xxxx 1111 1110	E0 xx FE	PC←(x+1, x)	1	–	–	–	6
An unconditional jump is made to the memory address stored in the 2 consecutive bytes starting from the memory address (0000-00FF_H) directly specified by the "x" in the object code. Example: When the memory contents of the address 0085_H are 27_H and the memory contents of the address 0086_H are $C3_H$, and the JP (85H) instruction is executed, a jump is made to the address $C327_H$.								
JP (pp)	1110 001p 1111 1110	E 2+pp FE	PC←(pp+1, pp)	1	–	–	–	5
An unconditional jump is made to the memory address stored in the 2 consecutive bytes starting from the memory address specified by the register pair pp (HL or DE). Example: When HL = 0125_H, the memory contents at the address 0125_H are 87_H and the memory contents at the address 0126_H are $E5_H$, and the JP (HL) instruction is executed, a jump is made to the address $E587_H$.								

MNEMONIC	OBJECT CODE		OPERATION	FLAG				CYCLE
	Binary	Hex-decimal		JF	ZF	CF	HF	
JP (HL+d)	1110 0100 dddd dddd 1111 1110	E4 dd FE	PC←(HL+d+1, HL+d)	1	–	–	–	7
	An unconditional jump is made to the memory address stored in the 2 consecutive bytes starting from the memory address specified by the value obtained by adding to sign-extended 8-bit data "d" in the object code to the contents of the HL register pair.							
JP (HL+C)	1110 0101 1111 1110	E5 FE	PC←(HL+C+1, HL+C)	1	–	–	–	7
	An unconditional jump is made to the memory address stored in the 2 consecutive bytes starting from the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair.							
JP (PC+A)	1110 0001 1111 1110	E1 FE	PC←(PC+A+1, PC+A)	1	–	–	–	7
	An unconditional jump is made to the memory address stored in the 2 consecutive bytes starting from the memory address specified by the value obtained by adding to sign-extended contents of the accumulator to the current contents of the program counter (which indicate the next instruction). This instruction is highly suited for multiple direction branch processing (refer to Section 1.8.5).							

Note: Expansion machine instruction "J" is a jump instruction used for optimization by relocatable macro assembler in place of "JRS", "JR" or "JP".

(7) CALL and RETURN

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
CALLV n	1100 nnnn	Cn	(SP, SP−1)←PC−1, SP←SP−2, PC←(n×2+FFC1$_H$, n×2+FFC0$_H$)	−	−	−	−	7
Vector format subroutine call is performed. That is, the value (the address where the CALLV instruction is located + 1, i.e., the return address) obtained by subtracting "1" from current contents of the program counter are stored at the address specified by the contents of the stack pointer, and the stack pointer is decremented twice. Then the contents (vector) in the 2 consecutive bytes starting from the address specified by the value obtained by adding FFC0$_H$ to the value obtained by doubling the 4-bit immediate data "n" ("n"is from 0 to 15) in the object code are loaded into the program counter. Therefore the subroutine may begin anywhere in the 64K bytes program memory space. Example: When SP = 0123$_H$, the contents at the address FFC8$_H$ are 45$_H$ and the contents at the address FFC9$_H$ are E6$_H$, and the CALLV 4H instruction at the address D1A4$_H$ is executed, D1$_H$ is written into the address 0123$_H$, A5$_H$ is written into the address 0122$_H$, SP = 0121$_H$, and the address E645$_H$ is called.								
CALLP n	1111 1101 nnnn nnnn	FD nn	(SP, SP−1)←PC, SP←SP−2, PC←FF00$_H$+n	−	−	−	−	6
Page (short-absolute) format subroutine call is performed. That is, the program counter contents (i.e., the return address) are stored at the address specified by the contents of the stack pointer, and the stack pointer is decremented twice. Then, the 8-bit immediate data "n" in the object code is loaded into the lower 8 bits of the program counter and FF$_H$ is loaded into the upper 8 bits of the program counter. Note that because the address from FFC0$_H$ to FFFF$_H$ are vector tables for the CALLV instruction and the interrupt/reset, the range 0≦n≦BF$_H$ is usually used. Example: When SP = 013F$_H$ and the CALLP 76H instruction at the address CA51$_H$ is executed, CA$_H$ is written into the address 013F$_H$, 53$_H$ is written into the address 013E$_H$, SP = 013D$_H$, and the address FF76$_H$ is called.								
CALL mn	1111 1100 nnnn nnnn mmmm mmmm	FC nn mm	(SP, SP−1)←PC+1, SP←SP−2, PC←mn	−	−	−	−	6
Long-absolute format subroutine call is performed. That is, the value (i.e., the return address) obtained by adding "1" to the current contents of the program counter is stored at the address specified by the contents of the stack pointer, and the stack pointer is decremented twice. Then, the 16-bit absolute address "mn" in the object code is loaded into the program counter. Therefore the subroutine may begin anywhere in the 64K bytes program memory space. Example: When SP = 0138$_H$ and the CALL 0CE05H instruction at the address E273$_H$ is executed, E2$_H$ is written into the address 0138$_H$, 76$_H$ is written into the address 0137$_H$, SP = 0136$_H$, and the address CE05$_H$ is called.								
CALL gg	1110 10gg 1111 1100	E 8+gg FC	(SP, SP−1)←PC, SP←SP−2, PC←gg	−	−	−	−	6
The current contents of the program counter (i.e., the return address) are stored at the address specified by the contents of the stack pointer and the stack pointer is decremented twice. Then, the contents of the 16-bit register gg are loaded into the program counter. Example: When SP = 0126$_H$, HL = C8A7$_H$ and the CALL HL instruction at the address D491$_H$ is executed, D4$_H$ is written into the address 0126$_H$ and 93$_H$ is written into the address 0125$_H$, SP = 0124$_H$ and the address C8A7$_H$ is called.								
CALL (x)	1110 0000 xxxx xxxx 1111 1100	E0 xx FC	(SP, SP−1)←PC+1, SP←SP−2, PC←(x+1, x)	−	−	−	−	9
The value (i.e., the return address) obtained by adding "1" to the contents of the program counter is stored in the memory address specified by the contents of the stack pointer and the stack pointer is decremented twice. Then, the contents in the 2 consecutive bytes starting from the memory address (0000-00FF$_H$) directly specified by the "x" in the object code are loaded into the program counter.								
CALL (pp)	1110 001p 1111 1100	E 2+pp FC	(SP, SP−1) ←PC, SP←SP−2, PC← (pp+1, pp)	−	−	−	−	8
The current contents of the program counter (i.e., the return address) are stored in the memory address specified by the contents of the stack pointer and the stack pointer is decremented twice. Then, the contents in the 2 consecutive bytes starting from the memory address specified by the contents of the register pair pp (HL or DE) are loaded into the program counter.								

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
CALL (HL+d)	1110 0100 dddd dddd 1111 1100	E4 dd FC	(SP, SP−1)←PC+1, SP←SP−2, PC←(HL+d+1, HL+d)	−	−	−	−	10
	The value (i.e., the return address) obtained by adding "1" to the contents of the program counter are stored in the memory address specified by the contents of the stack pointer, and the stack pointer is decremented twice. Then, the contents in the 2 consecutive bytes starting from the memory address specified by the value obtained by adding the sign-extended 8-bit data "d" in the object code to the contents of the HL register pair are loaded into the program counter.							
CALL (HL+C)	1110 0101 1111 1100	E5 FC	(SP, SP−1) ←PC, SP←SP−2, PC← (HL+C+1, HL+C)	−	−	−	−	10
	The current contents of the program counter (i.e., the return address) are stored in the memory address specified by the contents of the stack pointer, and the stack pointer is decremented twice. Then, the contents in the 2 consecutive bytes starting from the memory address specified by the value obtained by adding the sign-extended contents of the C register to the contents of the HL register pair are loaded into the program counter.							
CALL (PC+A)	1110 0001 1111 1100	E1 FC	(SP, SP−1) ←PC, SP←SP−2, PC← (PC+A+1, PC+A)	−	−	−	−	10
	The current contents of the program counter (i.e., the return address) are stored in the memory address specified by the contents of the stack pointer, and the stack pointer is decremented twice. Then, the contents in the 2 consecutive bytes starting from the memory address specified by the value obtained by adding the sign-extended contents of the accumulator to the contents of the program counter are loaded into the program counter. This instruction is suitable for multiple direction branch processing.							
RET	0000 0101	05	SP←SP+2, PC← (SP, SP−1)	−	−	−	−	6
	A return from the subroutine is performed. That is to say, the stack pointer is incremented by one, and the memory contents (i.e., the lower 8 bits of the return address) of the address specified by new contents of the stack pointer are loaded into the lower 8 bits of the program counter. The stack pointer is incremented again, and the memory contents (i. e., the upper 8bits of the return address) of the address specified by new contents of the stack pointer are loaded into the upper 8bits of the program counter. No flags are affected. Example: When SP = 0123_H, the contents at the address 0124_H are $3E_H$ and the contents at 0125_H are $C8_H$, and this instruction is executed, SP = 0125_H and a jump is made to the address $C83E_H$.							
RETI	0000 0100	04	SP←SP+3, PC← (SP−1, SP−2), PSW← (SP)	*	*	*	*	6
	A return from the maskable interrupt service routine is performed. That is, the stack pointer is incremented twice and the memory contents (i.e., the return address) in the 2 consecutive bytes starting from the memory address specified by that data are loaded into the program counter. Then, the stack pointer is further incremented and the contents of the memory address specified by that data are loaded into the program status word. The interrupt master enable flag is set to "1" during maskable interrupt subroutine (if this instruction is executed during non-maskable interrupt service, the interrupt master enable flag becomes undefined data). Example : When SP = 0123_H, the contents at the address 0124_H are $3E_H$, the contents at 0125_H are $C8_H$ and the contents at 0126_H are $E4_H$, after executing the RETI instruction during maskable interrupt service, the SP will be 0126_H, the PSW will be $E4_H$, the IMF will be set to "1" and a jump will be made to the address $C83E_H$.							
RETN	1110 1000 0000 0100	E8 04	SP←SP+3, PC← (SP−1, SP−2), PSW← (SP)	*	*	*	*	7
	A return from the non-maskable interrupt service routine is performed. That is, the stack pointer is incremented twice and the memory contents (i.e., the return address) in the 2 consecutive bytes starting from the memory address specified by that data are loaded into the program counter. Then, the stack pointer is further incremented and the contents of the memory address specified by that data are loaded into the program status word. The interrupt master enable flag is returned to the status it was in before the non-maskable interrupt accept (If the interrupt master enable flag is set to "1" in the interrupt service routine, then this flag remains "1").							

(8) PUSH, POP, SOFTWARE INTERRUPT and NO OPERATION

MNEMONIC	OBJECT CODE Binary	OBJECT CODE Hex-decimal	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
PUSH PSW	0000 0111	07	(SP)←PSW, SP←SP−1	−	−	−	−	2
The contents of the program status word are stored in the memory address specified by the contents of the stack pointer. The stack pointer is then decremented by one. No flags are affected.								
PUSH gg	1110 10gg 0000 0111	E 8+gg 07	(SP, SP−1) ←gg, SP←SP−2	−	−	−	−	4
The contents of the upper 8 bits of the register pair gg are stored in the memory address specified by the contents of the stack pointer, and the contents of the lower 8 bits of the register pair gg are stored in the memory address specified by the value obtained by subtracting 1 from the above address. Then, the stack pointer is decremented twice. Example: When SP = $013F_H$, WA = 1234_H, after executing the PUSH WA instruction, the contents of the address $013F_H$ will become 12_H and the contents of the address $013E_H$ will become 34_H. Also the SP will become $013D_H$.								
POP PSW	0000 0110	06	SP←SP+1, PSW←(SP)	*	*	*	*	3
The stack pointer is incremented by one, and the contents of the memory address specified by the contents of the incremented stack pointer are loaded into the program status word. Example: When the contents of the address 0137_H are 64_H, SP = 0136_H, after executing the POP PSW instruction, the results will become SP = 0137_H and PSW = 64_H (i.e., JF = HF = 0, ZF = CF = 1 and RBS = 4).								
POP gg	1110 10gg 0000 0110	E 8+gg 06	SP←SP+2, gg←(SP, SP−1)	−	−	−	−	5
The stack pointer is incremented twice, the contents of the memory address specified by the contents of the incremented stack pointer are loaded into the upper 8 bits of the register pair gg and the contents of the memory address obtained by subtracting "1" from the above address are loaded into the lower 8 bits of the register pair gg. Example: When the contents of the address 0124_H are 78_H, the contents of the address 0125_H are 56_H, SP = 0123_H , after executing the POP HL instruction, the results will become HL = 5678_H and SP = 0125_H.								
SWI	1111 1111	FF	(SP)←PSW, (SP−1, SP−2)←PC−1, SP←SP−3, IMF←0, PC←($FFFD_H$, $FFFC_H$)	−	−	−	−	9/1
Software interrupt is performed. That is, the program status word contents are stored in the memory address specified by the stack pointer contents, and the stack pointer is decremented by one. Then, the upper 8 bits of the return address (PC − 1) are stored in the memory address specified by new stack pointer contents (SP − 1), and the stack pointer is decremented again. The lower 8 bits of the return address are stored in the memory address specified by new stack pointer contents (SP − 2), and the stack pointer is decremented by one more times. The interrupt master enable flag (IMF) is cleared to "0". Then, the memory contents (vector) in the addresses $FFFC_H$ and $FFFD_H$ are loaded into the program counter. The contents in the assress $FFFC_H$ are the lower 8 bits of the software interrupt vector, and the contents in the address $FFFD_H$ are the upper. In this case, the SWI is the 9-cycle instruction. Software interrupt will not be generated during non-maskable interrupt service even if the SWI instruction is executed and the same operation will be performed as with the NOP instruction. In this case, the SWI is the 1-cycle instruction. Example: When SP = 0128_H, PSW = 46_H, the contents of the address $FFFC_H$ are 03_H, the contents of the address $FFFD_H$ are $E7_H$, after executing the SWI instruction at the address $CA74_H$, 46_H will be written into the address 0128_H, 75_H will be written into the address 0127_H and CA_H will be written into the address 0126_H. The results will become SP = 1025_H, IMF = 0, and a jump will be made to the address $E703_H$.								
NOP	0000 0000	00	No operation	−	−	−	−	1
CPU performs no operation during this machine cycle (i.e., moves to the next sequential instruction without performing any operation). No flags are affected.								

1.6 INSTRUCTION SET SUMMARY

This section provides TLCS-870 instruction set summary. Refer to section 1.5 for more detailed descriptions concerning instructions.

(1) DATA TRANSFER and EXCHANGE

MNEMONIC	OBJECT CODE (Binary)	OPERATION	FLAG				CYCLE	
			JF	ZF	CF	HF		
LD A, r	0101 0rrr	A ← r	1	Z	–	–	1	
LD r, A	0101 1rrr	r ← A	1	Z	–	–	1	
LD r, g	1110 1ggg 0101 1rrr	r ← g	1	Z	–	–	2	
LD rr, gg	1110 10gg 0001 01rr	rr ← gg	1	–	–	–	2	
LD SP, gg	1110 10gg 1111 1010	SP ← gg	1	–	–	–	3	
LD gg, SP	1110 10gg 1111 1011	gg ← SP	1	–	–	–	3	
LD A, (x)	0010 0010 xxxx xxxx	A ← (x)	1	Z	–	–	3	
LD r, (x)	1110 0000 xxxx xxxx 0101 1rrr	r ← (x)	1	Z	–	–	4	
LD A, (HL)	0010 0011	A ← (HL)	1	Z	–	–	2	
LD r, (pp)	1110 001p 0101 1rrr	r ← (pp)	1	Z	–	–	3	
LD r, (HL+)	1110 0110 0101 1rrr	r ← (HL): HL ← HL+1	1	Z	–	–	4	*1
LD r, (–HL)	1110 0111 0101 1rrr	HL ← HL–1: r ← (HL)	1	Z	–	–	4	
LD r, (HL+d)	1110 0100 dddd dddd 0101 1rrr	r ← (HL+d)	1	Z	–	–	5	
LD r, (HL+C)	1110 0101 0101 1rrr	r ← (HL+C)	1	Z	–	–	5	
LD r, (PC+A)	1110 0001 0101 1rrr	r ← (PC+A)	1	Z	–	–	5	
LD rr, (x)	1110 0000 xxxx xxxx 0001 01rr	rr ← (x+1, x)	1	–	–	–	5	
LD rr, (pp)	1110 001p 0001 01rr	rr ← (pp+1, pp)	1	–	–	–	4	
LD rr, (HL+d)	1110 0100 dddd dddd 0001 01rr	rr ← (HL+d+1, HL+d)	1	–	–	–	6	
LD rr, (HL+C)	1110 0101 0001 01rr	rr ← (HL+C+1, HL+C)	1	–	–	–	6	
LD rr, (PC+A)	1110 0001 0001 01rr	rr ← (PC+A+1, PC+A)	1	–	–	–	6	
LD (x), A	0010 1010 xxxx xxxx	(x) ← A	1	–	–	–	3	
LD (x), r	1111 0000 xxxx xxxx 0101 0rrr	(x) ← r	1	–	–	–	4	
LD (HL), A	0010 1011	(HL) ← A	1	–	–	–	2	
LD (pp), r	1111 001p 0101 0rrr	(pp) ← r	1	–	–	–	3	
LD (HL+), r	1111 0110 0101 0rrr	(HL) ← r: HL ← HL+1	1	–	–	–	4	*1
LD (–HL), r	1111 0111 0101 0rrr	HL ← HL–1: (HL) ← r	1	–	–	–	4	
LD (HL+d), r	1111 0100 dddd dddd 0101 0rrr	(HL+d) ← r	1	–	–	–	5	
LD (x), rr	1111 0000 xxxx xxxx 0001 00rr	(x+1, x) ← rr	1	–	–	–	5	
LD (pp), rr	1111 001p 0001 00rr	(pp+1, pp) ← rr	1	–	–	–	4	
LD (HL+d), rr	1111 0100 dddd dddd 0001 00rr	(HL+d+1, HL+d) ← rr	1	–	–	–	6	
LD (x), (y)	0010 0110 yyyy yyyy xxxx xxxx	(x) ← (y)	1	Z	–	–	5	
LD (x), (pp)	1110 001p 0010 0110 xxxx xxxx	(x) ← (pp)	1	U	–	–	5	
LD (x), (HL+d)	1110 0100 dddd dddd 0010 0110 xxxx xxxx	(x) ← (HL+d)	1	U	–	–	7	
LD (x), (HL+C)	1110 0101 0010 0110 xxxx xxxx	(x) ← (HL+C)	1	U	–	–	7	
LD (x), (PC+A)	1110 0001 0010 0110 xxxx xxxx	(x) ← (PC+A)	1	U	–	–	7	
LD (HL), (x)	1110 0000 xxxx xxxx 0010 0111	(HL) ← (x)	1	Z	–	–	5	
LD (HL), (pp)	1110 001p 0010 0111	(HL) ← (pp)	1	Z	–	–	4	
LD (HL), (HL+d)	1110 0100 dddd dddd 0010 0111	(HL) ← (HL+d)	1	Z	–	–	6	
LD (HL), (HL+C)	1110 0101 0010 0111	(HL) ← (HL+C)	1	Z	–	–	6	
LD (HL), (PC+A)	1110 0001 0010 0111	(HL) ← (PC+A)	1	Z	–	–	6	

*Note: *1 r ≠ H or L register.*

MNEMONIC	OBJECT CODE (Binary)		OPERATION	FLAG				CYCLE	
				JF	ZF	CF	HF		
LD r, n	0011 0rrr	nnnn nnnn	r ← n	1	–	–	–	2	
LD rr, mn	0001 01rr mmmm mmmm	nnnn nnnn	rr ← mn	1	–	–	–	3	
LD SP, mn	1111 1010 mmmm mmmm	nnnn nnnn	SP ← mn	1	–	–	–	3	
LD RBS, n	0000 1111	0000 nnnn	RBS ← n	1	–	–	–	2	
CLR (x)	0010 1110	xxxx xxxx	(x) ← 0	1	–	–	–	4	
LD (x), n	0010 1100 nnnn nnnn	xxxx xxxx	(x) ← n	1	–	–	–	4	
CLR (HL)	0010 1111		(HL) ← 0	1	–	–	–	2	
LD (HL), n	0010 1101	nnnn nnnn	(HL) ← n	1	–	–	–	3	
LD (DE), n	1111 0010 nnnn nnnn	0010 1100	(DE) ← n	1	–	–	–	4	
LD (HL+), n	1111 0110 nnnn nnnn	0010 1100	(HL) ← n: HL ← HL+1	1	–	–	–	5	
LD (−HL), n	1111 0111 nnnn nnnn	0010 1100	HL ← HL−1: (HL) ← n	1	–	–	–	5	
LD (HL+d), n	1111 0100 0010 1100	dddd dddd nnnn nnnn	(HL+d) ← n	1	–	–	–	6	
LDW (x), mn	0010 0100 nnnn nnnn	xxxx xxxx mmmm mmmm	(x+1, x) ← mn	1	–	–	–	6	
LDW (HL), mn	0010 0101 mmmm mmmm	nnnn nnnn	(HL+1, HL) ← mn	1	–	–	–	5	
XCH r, g	1110 1ggg	1010 1rrr	r ↔ g	1	Z	–	–	3	*2
XCH rr, gg	1110 10gg	0001 00rr	rr ↔ gg	1	–	–	–	3	
XCH r, (x)	1110 0000 1010 1rrr	xxxx xxxx	r ↔ (x)	1	Z	–	–	5	*2
XCH r, (pp)	1110 001p	1010 1rrr	r ↔ (pp)	1	Z	–	–	4	*2
XCH r, (HL+)	1110 0110	1010 1rrr	r ↔ (HL): HL ← HL+1	1	Z	–	–	5	* 1,2
XCH r, (−HL)	1110 0111	1010 1rrr	HL ← HL−1: r ↔ (HL)	1	Z	–	–	5	*2
XCH r, (HL+d)	1110 0100 1010 1rrr	dddd dddd	r ↔ (HL+d)	1	Z	–	–	6	*2
XCH r, (HL+C)	1110 0101	1010 1rrr	r ↔ (HL+C)	1	Z	–	–	6	*2
XCH r, (PC+A)	1110 0001	1010 1rrr	r ↔ (PC+A)	1	Z	–	–	6	*2

*Note: *2 During XCH instruction, the ZF is set to "1" if an original value of register g or source memory (src) before exchange is 00_H; otherwise the ZF is cleared to "0".*

(2) ARITHMETIC and LOGICAL OPERATION

MNEMONIC	OBJECT CODE (Binary)	OPERATION	FLAG				CYCLE
			JF	ZF	CF	HF	
CMP A, g	1110 1ggg 0110 0111	A − g	Z	Z	C	H	2
CMP g, A	1110 1ggg 0110 1111	g − A	Z	Z	C	H	3
CMP WA, gg	1110 10gg 0011 0111	WA − gg	Z	Z	C	U	4
CMP A, (x)	0111 1111 xxxx xxxx	A − (x)	Z	Z	C	H	4
CMP A, (pp)	1110 001p 0111 1111	A − (pp)	Z	Z	C	H	3
CMP A, (HL+)	1110 0110 0111 1111	A − (HL): HL ← HL+1	Z	Z	C	H	4
CMP A, (−HL)	1110 0111 0111 1111	HL ← HL−1: A − (HL)	Z	Z	C	H	4
CMP A, (HL+d)	1110 0100 dddd dddd 0111 1111	A − (HL+d)	Z	Z	C	H	5
CMP A, (HL+C)	1110 0101 0111 1111	A − (HL+C)	Z	Z	C	H	5
CMP A, (PC+A)	1110 0001 0111 1111	A − (PC+A)	Z	Z	C	H	5
CMP (x), (HL)	1110 0000 xxxx xxxx 0110 0111	(x) − (HL)	Z	Z	C	H	6
CMP (pp), (HL)	1110 001p 0110 0111	(pp) − (HL)	Z	Z	C	H	5
CMP (HL+d), (HL)	1110 0100 dddd dddd 0110 0111	(HL+d) − (HL)	Z	Z	C	H	7
CMP (HL+C), (HL)	1110 0101 0110 0111	(HL+C) − (HL)	Z	Z	C	H	7
CMP (PC+A), (HL)	1110 0001 0110 0111	(PC+A) − (HL)	Z	Z	C	H	7
CMP A, n	0111 0111 nnnn nnnn	A − n	Z	Z	C	H	2
CMP g, n	1110 1ggg 0111 0111 nnnn nnnn	g − n	Z	Z	C	H	3
CMP gg, mn	1110 10gg 0011 1111 nnnn nnnn mmmm mmmm	gg − mn	Z	Z	C	U	4
CMP (x), n	1110 0000 xxxx xxxx 0111 0111 nnnn nnnn	(x) − n	Z	Z	C	H	5
CMP (pp), n	1110 001p 0111 0111 nnnn nnnn	(pp) − n	Z	Z	C	H	4
CMP (HL+), n	1110 0110 0111 0111 nnnn nnnn	(HL) − n: HL ← HL+1	Z	Z	C	H	5
CMP (−HL), n	1110 0111 0111 0111 nnnn nnnn	HL ← HL−1: (HL) − n	Z	Z	C	H	5
CMP (HL+d), n	1110 0100 dddd dddd 0111 0111 nnnn nnnn	(HL+d) − n	Z	Z	C	H	6
CMP (HL+C), n	1110 0101 0111 0111 nnnn nnnn	(HL+C) − n	Z	Z	C	H	6
CMP (PC+A), n	1110 0001 0111 0111 nnnn nnnn	(PC+A) − n	Z	Z	C	H	6
MCMP (x), n	1110 0000 xxxx xxxx 0010 1111 nnnn nnnn	(x)∧n − A	Z	Z	C	H	6
MCMP (pp), n	1110 001p 0010 1111 nnnn nnnn	(pp)∧n − A	Z	Z	C	H	5
MCMP (HL+), n	1110 0110 0010 1111 nnnn nnnn	(HL)∧n − A: HL ← HL+1	Z	Z	C	H	6
MCMP (−HL), n	1110 0111 0010 1111 nnnn nnnn	HL ← HL−1: (HL)∧n − A	Z	Z	C	H	6
MCMP (HL+d), n	1110 0100 dddd dddd 0010 1111 nnnn nnnn	(HL+d)∧n − A	Z	Z	C	H	7
MCMP (HL+C), n	1110 0101 0010 1111 nnnn nnnn	(HL+C)∧n − A	Z	Z	C	H	7
MCMP (PC+A), n	1110 0001 0010 1111 nnnn nnnn	(PC+A)∧n − A	Z	Z	C	H	7
INC r	0110 0rrr	r ← r+1	C	Z	–	–	1
INC rr	0001 00rr	rr ← rr+1	C	Z	–	–	2
INC (x)	0010 0000 xxxx xxxx	(x) ← (x)+1	C	Z	–	–	5
INC (HL)	0010 0001	(HL) ← (HL)+1	C	Z	–	–	4
INC (DE)	1110 0010 0010 0000	(DE) ← (DE)+1	C	Z	–	–	4
INC (HL+)	1110 0110 0010 0000	(HL) ← (HL)+1: HL ← HL+1	C	Z	–	–	5
INC (−HL)	1110 0111 0010 0000	HL ← HL−1: (HL) ← (HL)+1	C	Z	–	–	5
INC (HL+d)	1110 0100 dddd dddd 0010 0000	(HL+d) ← (HL+d)+1	C	Z	–	–	6
INC (HL+C)	1110 0101 0010 0000	(HL+C) ← (HL+C)+1	C	Z	–	–	6
INC (PC+A)	1110 0001 0010 0000	(PC+A) ← (PC+A)+1	C	Z	–	–	6

MNEMONIC	OBJECT CODE (Binary)	OPERATION	FLAG				CYCLE
			JF	ZF	CF	HF	
DEC r	0110 1rrr	r ← r − 1	C	Z	−	−	1
DEC rr	0001 10rr	rr ← rr − 1	C	Z	−	−	2
DEC (x)	0010 1000 xxxx xxxx	(x) ← (x) − 1	C	Z	−	−	5
DEC (HL)	0010 1001	(HL) ← (HL) − 1	C	Z	−	−	4
DEC (DE)	1110 0010 0010 1000	(DE) ← (DE) − 1	C	Z	−	−	4
DEC (HL+)	1110 0110 0010 1000	(HL) ← (HL) − 1: HL ← HL + 1	C	Z	−	−	5
DEC (−HL)	1110 0111 0010 1000	HL ← HL − 1: (HL) ← (HL) − 1	C	Z	−	−	5
DEC (HL+d)	1110 0100 dddd dddd 0010 1000	(HL+d) ← (HL+d) − 1	C	Z	−	−	6
DEC (HL+C)	1110 0101 0010 1000	(HL+C) ← (HL+C) − 1	C	Z	−	−	6
DEC (PC+A)	1110 0001 0010 1000	(PC+A) ← (PC+A) − 1	C	Z	−	−	6
ADD A, g	1110 1ggg 0110 0001	A ← A + g	C	Z	C	H	2
ADD g, A	1110 1ggg 0110 1001	g ← g + A	C	Z	C	H	3
ADD WA, gg	1110 10gg 0011 0001	WA ← WA + gg	C	Z	C	U	4
ADD A, (x)	0111 1001 xxxx xxxx	A ← A + (x)	C	Z	C	H	4
ADD A, (pp)	1110 001p 0111 1001	A ← A + (pp)	C	Z	C	H	3
ADD A, (HL+)	1110 0110 0111 1001	A ← A + (HL): HL ← HL + 1	C	Z	C	H	4
ADD A, (−HL)	1110 0111 0111 1001	HL ← HL − 1: A ← A + (HL)	C	Z	C	H	4
ADD A, (HL+d)	1110 0100 dddd dddd 0111 1001	A ← A + (HL+d)	C	Z	C	H	5
ADD A, (HL+C)	1110 0101 0111 1001	A ← A + (HL+C)	C	Z	C	H	5
ADD A, (PC+A)	1110 0001 0111 1001	A ← A + (PC+A)	C	Z	C	H	5
ADD (x), (HL)	1110 0000 xxxx xxxx 0110 0001	(x) ← (x) + (HL)	C	Z	C	H	7
ADD (pp), (HL)	1110 001p 0110 0001	(pp) ← (pp) + (HL)	C	Z	C	H	6
ADD (HL+d), (HL)	1110 0100 dddd dddd 0110 0001	(HL+d) ← (HL+d) + (HL)	C	Z	C	H	8
ADD (HL+C), (HL)	1110 0101 0110 0001	(HL+C) ← (HL+C) + (HL)	C	Z	C	H	8
ADD (PC+A), (HL)	1110 0001 0110 0001	(PC+A) ← (PC+A) + (HL)	C	Z	C	H	8
ADD A, n	0111 0001 nnnn nnnn	A ← A + n	C	Z	C	H	2
ADD g, n	1110 1ggg 0111 0001 nnnn nnnn	g ← g + n	C	Z	C	H	3
ADD gg, mn	1110 10gg 0011 1001 nnnn nnnn mmmm mmmm	gg ← gg + mn	C	Z	C	U	4
ADD (x), n	1110 0000 xxxx xxxx 0111 0001 nnnn nnnn	(x) ← (x) + n	C	Z	C	H	6
ADD (pp), n	1110 001p 0111 0001 nnnn nnnn	(pp) ← (pp) + n	C	Z	C	H	5
ADD (HL+), n	1110 0110 0111 0001 nnnn nnnn	(HL) ← (HL) + n: HL ← HL + 1	C	Z	C	H	6
ADD (−HL), n	1110 0111 0111 0001 nnnn nnnn	HL ← HL − 1: (HL) ← (HL) + n	C	Z	C	H	6
ADD (HL+d), n	1110 0100 dddd dddd 0111 0001 nnnn nnnn	(HL+d) ← (HL+d) + n	C	Z	C	H	7
ADD (HL+C), n	1110 0101 0111 0001 nnnn nnnn	(HL+C) ← (HL+C) + n	C	Z	C	H	7
ADD (PC+A), n	1110 0001 0111 0001 nnnn nnnn	(PC+A) ← (PC+A) + n	C	Z	C	H	7
ADDC A, g	1110 1ggg 0110 0000	A ← A + g + CF	C	Z	C	H	2
ADDC g, A	1110 1ggg 0110 1000	g ← g + A + CF	C	Z	C	H	3
ADDC WA, gg	1110 10gg 0011 0000	WA ← WA + gg + CF	C	Z	C	U	4
ADDC A, (x)	0111 1000 xxxx xxxx	A ← A + (x) + CF	C	Z	C	H	4
ADDC A, (pp)	1110 001p 0111 1000	A ← A + (pp) + CF	C	Z	C	H	3
ADDC A, (HL+)	1110 0110 0111 1000	A ← A + (HL) + CF: HL ← HL + 1	C	Z	C	H	4
ADDC A, (−HL)	1110 0111 0111 1000	HL ← HL − 1: A ← A + (HL) + CF	C	Z	C	H	4
ADDC A, (HL+d)	1110 0100 dddd dddd 0111 1000	A ← A + (HL+d) + CF	C	Z	C	H	5
ADDC A, (HL+C)	1110 0101 0111 1000	A ← A + (HL+C) + CF	C	Z	C	H	5
ADDC A, (PC+A)	1110 0001 0111 1000	A ← A + (PC+A) + CF	C	Z	C	H	5

MNEMONIC	OBJECT CODE (Binary)	OPERATION	FLAG				CYCLE
			JF	ZF	CF	HF	
ADDC (x), (HL)	1110 0000 xxxx xxxx 0110 0000	(x) ← (x) + (HL) + CF	C	Z	C	H	7
ADDC (pp), (HL)	1110 001p 0110 0000	(pp) ← (pp) + (HL) + CF	C	Z	C	H	6
ADDC (HL+d), (HL)	1110 0100 dddd dddd 0110 0000	(HL+d) ← (HL+d) + (HL) + CF	C	Z	C	H	8
ADDC (HL+C), (HL)	1110 0101 0110 0000	(HL+C) ← (HL+C) + (HL) + CF	C	Z	C	H	8
ADDC (PC+A), (HL)	1110 0001 0110 0000	(PC+A) ← (PC+A) + (HL) + CF	C	Z	C	H	8
ADDC A, n	0111 0000 nnnn nnnn	A ← A + n + CF	C	Z	C	H	2
ADDC g, n	1110 1ggg 0111 0000 nnnn nnnn	g ← g + n + CF	C	Z	C	H	3
ADDC gg, mn	1110 10gg 0011 1000 nnnn nnnn mmmm mmmm	gg ← gg + mn + CF	C	Z	C	U	4
ADDC (x), n	1110 0000 xxxx xxxx 0111 0000 nnnn nnnn	(x) ← (x) + n + CF	C	Z	C	H	6
ADDC (pp), n	1110 001p 0111 0000 nnnn nnnn	(pp) ← (pp) + n + CF	C	Z	C	H	5
ADDC (HL+), n	1110 0110 0111 0000 nnnn nnnn	(HL) ← (HL) + n + CF: HL ← HL + 1	C	Z	C	H	6
ADDC (−HL), n	1110 0111 0111 0000 nnnn nnnn	HL ← HL − 1: (HL) ← (HL) + n + CF	C	Z	C	H	6
ADDC (HL+d), n	1110 0100 dddd dddd 0111 0000 nnnn nnnn	(HL+d) ← (HL+d) + n + CF	C	Z	C	H	7
ADDC (HL+C), n	1110 0101 0111 0000 nnnn nnnn	(HL+C) ← (HL+C) + n + CF	C	Z	C	H	7
ADDC (PC+A), n	1110 0001 0111 0000 nnnn nnnn	(PC+A) ← (PC+A) + n + CF	C	Z	C	H	7
SUB A, g	1110 1ggg 0110 0011	A ← A − g	C	Z	C	H	2
SUB g, A	1110 1ggg 0110 1011	g ← g − A	C	Z	C	H	3
SUB WA, gg	1110 10gg 0011 0011	WA ← WA − gg	C	Z	C	U	4
SUB A, (x)	0111 1011 xxxx xxxx	A ← A − (x)	C	Z	C	H	4
SUB A, (pp)	1110 001p 0111 1011	A ← A − (pp)	C	Z	C	H	3
SUB A, (HL+)	1110 0110 0111 1011	A ← A − (HL): HL ← HL + 1	C	Z	C	H	4
SUB A, (−HL)	1110 0111 0111 1011	HL ← HL − 1: A ← A − (HL)	C	Z	C	H	4
SUB A, (HL+d)	1110 0100 dddd dddd 0111 1011	A ← A − (HL+d)	C	Z	C	H	5
SUB A, (HL+C)	1110 0101 0111 1011	A ← A − (HL+C)	C	Z	C	H	5
SUB A, (PC+A)	1110 0001 0111 1011	A ← A − (PC+A)	C	Z	C	H	5
SUB (x), (HL)	1110 0000 xxxx xxxx 0110 0011	(x) ← (x) − (HL)	C	Z	C	H	7
SUB (pp), (HL)	1110 001p 0110 0011	(pp) ← (pp) − (HL)	C	Z	C	H	6
SUB (HL+d), (HL)	1110 0100 dddd dddd 0110 0011	(HL+d) ← (HL+d) − (HL)	C	Z	C	H	8
SUB (HL+C), (HL)	1110 0101 0110 0011	(HL+C) ← (HL+C) − (HL)	C	Z	C	H	8
SUB (PC+A), (HL)	1110 0001 0110 0011	(PC+A) ← (PC+A) − (HL)	C	Z	C	H	8
SUB A, n	0111 0011 nnnn nnnn	A ← A − n	C	Z	C	H	2
SUB g, n	1110 1ggg 0111 0011 nnnn nnnn	g ← g − n	C	Z	C	H	3
SUB gg, mn	1110 10gg 0011 1011 nnnn nnnn mmmm mmmm	gg ← gg − mn	C	Z	C	U	4
SUB (x), n	1110 0000 xxxx xxxx 0111 0011 nnnn nnnn	(x) ← (x) − n	C	Z	C	H	6
SUB (pp), n	1110 001p 0111 0011 nnnn nnnn	(pp) ← (pp) − n	C	Z	C	H	5
SUB (HL+), n	1110 0110 0111 0011 nnnn nnnn	(HL) ← (HL) − n: HL ← HL + 1	C	Z	C	H	6
SUB (−HL), n	1110 0111 0111 0011 nnnn nnnn	HL ← HL − 1: (HL) ← (HL) − n	C	Z	C	H	6
SUB (HL+d), n	1110 0100 dddd dddd 0111 0011 nnnn nnnn	(HL+d) ← (HL+d) − n	C	Z	C	H	7
SUB (HL+C), n	1110 0101 0111 0011 nnnn nnnn	(HL+C) ← (HL+C) − n	C	Z	C	H	7
SUB (PC+A), n	1110 0001 0111 0011 nnnn nnnn	(PC+A) ← (PC+A) − n	C	Z	C	H	7

MNEMONIC	OBJECT CODE (Binary)	OPERATION	FLAG				CYCLE
			JF	ZF	CF	HF	
SUBB A, g	1110 1ggg 0110 0010	A ← A − g − CF	C	Z	C	H	2
SUBB g, A	1110 1ggg 0110 1010	g ← g − A − CF	C	Z	C	H	3
SUBB WA, gg	1110 10gg 0011 0010	WA ← WA − gg − CF	C	Z	C	U	4
SUBB A, (x)	0111 1010 xxxx xxxx	A ← A − (x) − CF	C	Z	C	H	4
SUBB A, (pp)	1110 001p 0111 1010	A ← A − (pp) − CF	C	Z	C	H	3
SUBB A, (HL+)	1110 0110 0111 1010	A ← A − (HL) − CF: HL ← HL+1	C	Z	C	H	4
SUBB A, (−HL)	1110 0111 0111 1010	HL ← HL−1: A ← A − (HL) − CF	C	Z	C	H	4
SUBB A, (HL+d)	1110 0100 dddd dddd 0111 1010	A ← A − (HL+d) − CF	C	Z	C	H	5
SUBB A, (HL+C)	1110 0101 0111 1010	A ← A − (HL+C) − CF	C	Z	C	H	5
SUBB A, (PC+A)	1110 0001 0111 1010	A ← A − (PC+A) − CF	C	Z	C	H	5
SUBB (x), (HL)	1110 0000 xxxx xxxx 0110 0010	(x) ← (x) − (HL) − CF	C	Z	C	H	7
SUBB (pp), (HL)	1110 001p 0110 0010	(pp) ← (pp) − (HL) − CF	C	Z	C	H	6
SUBB (HL+d), (HL)	1110 0100 dddd dddd 0110 0010	(HL+d) ← (HL+d) − (HL) − CF	C	Z	C	H	8
SUBB (HL+C), (HL)	1110 0101 0110 0010	(HL+C) ← (HL+C) − (HL) − CF	C	Z	C	H	8
SUBB (PC+A), (HL)	1110 0001 0110 0010	(PC+A) ← (PC+A) − (HL) − CF	C	Z	C	H	8
SUBB A, n	0111 0010 nnnn nnnn	A ← A − n − CF	C	Z	C	H	2
SUBB g, n	1110 1ggg 0111 0010 nnnn nnnn	g ← g − n − CF	C	Z	C	H	3
SUBB gg, mn	1110 10gg 0011 1010 nnnn nnnn mmmm mmmm	gg ← gg − mn − CF	C	Z	C	U	4
SUBB (x), n	1110 0000 xxxx xxxx 0111 0010 nnnn nnnn	(x) ← (x) − n − CF	C	Z	C	H	6
SUBB (pp), n	1110 001p 0111 0010 nnnn nnnn	(pp) ← (pp) − n − CF	C	Z	C	H	5
SUBB (HL+), n	1110 0110 0111 0010 nnnn nnnn	(HL) ← (HL) − n − CF: HL ← HL+1	C	Z	C	H	6
SUBB (−HL), n	1110 0111 0111 0010 nnnn nnnn	HL ← HL−1: (HL) ← (HL) − n − CF	C	Z	C	H	6
SUBB (HL+d), n	1110 0100 dddd dddd 0111 0010 nnnn nnnn	(HL+d) ← (HL+d) − n − CF	C	Z	C	H	7
SUBB (HL+C), n	1110 0101 0111 0010 nnnn nnnn	(HL+C) ← (HL+C) − n − CF	C	Z	C	H	7
SUBB (PC+A), n	1110 0001 0111 0010 nnnn nnnn	(PC+A) ← (PC+A) − n − CF	C	Z	C	H	7
AND A, g	1110 1ggg 0110 0100	A ← A ∧ g	Z	Z	−	−	2
AND g, A	1110 1ggg 0110 1100	g ← g ∧ A	Z	Z	−	−	3
AND WA, gg	1110 10gg 0011 0100	WA ← WA ∧ gg	Z	Z	−	−	4
AND A, (x)	0111 1100 xxxx xxxx	A ← A ∧ (x)	Z	Z	−	−	4
AND A, (pp)	1110 001p 0111 1100	A ← A ∧ (pp)	Z	Z	−	−	3
AND A, (HL+)	1110 0110 0111 1100	A ← A ∧ (HL): HL ← HL+1	Z	Z	−	−	4
AND A, (−HL)	1110 0111 0111 1100	HL ← HL−1: A ← A ∧ (HL)	Z	Z	−	−	4
AND A, (HL+d)	1110 0100 dddd dddd 0111 1100	A ← A ∧ (HL+d)	Z	Z	−	−	5
AND A, (HL+C)	1110 0101 0111 1100	A ← A ∧ (HL+C)	Z	Z	−	−	5
AND A, (PC+A)	1110 0001 0111 1100	A ← A ∧ (PC+A)	Z	Z	−	−	5
AND (x), (HL)	1110 0000 xxxx xxxx 0110 0100	(x) ← (x) ∧ (HL)	Z	Z	−	−	7
AND (pp), (HL)	1110 001p 0110 0100	(pp) ← (pp) ∧ (HL)	Z	Z	−	−	6
AND (HL+d), (HL)	1110 0100 dddd dddd 0110 0100	(HL+d) ← (HL+d) ∧ (HL)	Z	Z	−	−	8
AND (HL+C), (HL)	1110 0101 0110 0100	(HL+C) ← (HL+C) ∧ (HL)	Z	Z	−	−	8
AND (PC+A), (HL)	1110 0001 0110 0100	(PC+A) ← (PC+A) ∧ (HL)	Z	Z	−	−	8
AND A, n	0111 0100 nnnn nnnn	A ← A ∧ n	Z	Z	−	−	2
AND g, n	1110 1ggg 0111 0100 nnnn nnnn	g ← g ∧ n	Z	Z	−	−	3
AND gg, mn	1110 10gg 0011 1100 nnnn nnnn mmmm mmmm	gg ← gg ∧ mn	Z	Z	−	−	4

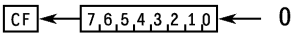
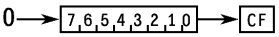
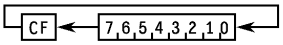
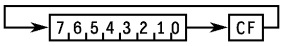
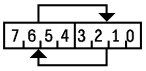
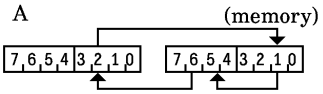
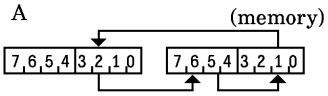
MNEMONIC	OBJECT CODE (Binary)	OPERATION	FLAG				CYCLE
			JF	ZF	CF	HF	
AND (x), n	1110 0000 xxxx xxxx 0111 0100 nnnn nnnn	(x) ← (x) ∧ n	Z	Z	–	–	6
AND (pp), n	1110 001p 0111 0100 nnnn nnnn	(pp) ← (pp) ∧ n	Z	Z	–	–	5
AND (HL+), n	1110 0110 0111 0100 nnnn nnnn	(HL) ← (HL) ∧ n : HL ← HL+1	Z	Z	–	–	6
AND (−HL), n	1110 0111 0111 0100 nnnn nnnn	HL ← HL−1: (HL) ← (HL) ∧ n	Z	Z	–	–	6
AND (HL+d), n	1110 0100 dddd dddd 0111 0100 nnnn nnnn	(HL+d) ← (HL+d) ∧ n	Z	Z	–	–	7
AND (HL+C), n	1110 0101 0111 0100 nnnn nnnn	(HL+C) ← (HL+C) ∧ n	Z	Z	–	–	7
AND (PC+A), n	1110 0001 0111 0100 nnnn nnnn	(PC+A) ← (PC+A) ∧ n	Z	Z	–	–	7
OR A, g	1110 1ggg 0110 0110	A ← A ∨ g	Z	Z	–	–	2
OR g, A	1110 1ggg 0110 1110	g ← g ∨ A	Z	Z	–	–	3
OR WA, gg	1110 10gg 0011 0110	WA ← WA ∨ gg	Z	Z	–	–	4
OR A, (x)	0111 1110 xxxx xxxx	A ← A ∨ (x)	Z	Z	–	–	4
OR A, (pp)	1110 001p 0111 1110	A ← A ∨ (pp)	Z	Z	–	–	3
OR A, (HL+)	1110 0110 0111 1110	A ← A ∨ (HL): HL ← HL+1	Z	Z	–	–	4
OR A, (−HL)	1110 0111 0111 1110	HL ← HL−1: A ← A ∨ (HL)	Z	Z	–	–	4
OR A, (HL+d)	1110 0100 dddd dddd 0111 1110	A ← A ∨ (HL+d)	Z	Z	–	–	5
OR A, (HL+C)	1110 0101 0111 1110	A ← A ∨ (HL+C)	Z	Z	–	–	5
OR A, (PC+A)	1110 0001 0111 1110	A ← A ∨ (PC+A)	Z	Z	–	–	5
OR (x), (HL)	1110 0000 xxxx xxxx 0110 0110	(x) ← (x) ∨ (HL)	Z	Z	–	–	7
OR (pp), (HL)	1110 001p 0110 0110	(pp) ← (pp) ∨ (HL)	Z	Z	–	–	6
OR (HL+d), (HL)	1110 0100 dddd dddd 0110 0110	(HL+d) ← (HL+d) ∨ (HL)	Z	Z	–	–	8
OR (HL+C), (HL)	1110 0101 0110 0110	(HL+C) ← (HL+C) ∨ (HL)	Z	Z	–	–	8
OR (PC+A), (HL)	1110 0001 0110 0110	(PC+A) ← (PC+A) ∨ (HL)	Z	Z	–	–	8
OR A, n	0111 0110 nnnn nnnn	A ← A ∨ n	Z	Z	–	–	2
OR g, n	1110 1ggg 0111 0110 nnnn nnnn	g ← g ∨ n	Z	Z	–	–	3
OR gg, mn	1110 10gg 0011 1110 nnnn nnnn mmmm mmmm	gg ← gg ∨ mn	Z	Z	–	–	4
OR (x), n	1110 0000 xxxx xxxx 0111 0110 nnnn nnnn	(x) ← (x) ∨ n	Z	Z	–	–	6
OR (pp), n	1110 001p 0111 0110 nnnn nnnn	(pp) ← (pp) ∨ n	Z	Z	–	–	5
OR (HL+), n	1110 0110 0111 0110 nnnn nnnn	(HL) ← (HL) ∨ n: HL ← HL+1	Z	Z	–	–	6
OR (−HL), n	1110 0111 0111 0110 nnnn nnnn	HL ← HL−1: (HL) ← (HL) ∨ n	Z	Z	–	–	6
OR (HL+d), n	1110 0100 dddd dddd 0111 0110 nnnn nnnn	(HL+d) ← (HL+d) ∨ n	Z	Z	–	–	7
OR (HL+C), n	1110 0101 0111 0110 nnnn nnnn	(HL+C) ← (HL+C) ∨ n	Z	Z	–	–	7
OR (PC+A), n	1110 0001 0111 0110 nnnn nnnn	(PC+A) ← (PC+A) ∨ n	Z	Z	–	–	7
XOR A, g	1110 1ggg 0110 0101	A ← A ∀ g	Z	Z	–	–	2
XOR g, A	1110 1ggg 0110 1101	g ← g ∀ A	Z	Z	–	–	3
XOR WA, gg	1110 10gg 0011 0101	WA ← WA ∀ gg	Z	Z	–	–	4
XOR A, (x)	0111 1101 xxxx xxxx	A ← A ∀ (x)	Z	Z	–	–	4
XOR A, (pp)	1110 001p 0111 1101	A ← A ∀ (pp)	Z	Z	–	–	3
XOR A, (HL+)	1110 0110 0111 1101	A ← A ∀ (HL): HL ← HL+1	Z	Z	–	–	4
XOR A, (−HL)	1110 0111 0111 1101	HL ← HL−1: A ← A ∀ (HL)	Z	Z	–	–	4
XOR A, (HL+d)	1110 0100 dddd dddd 0111 1101	A ← A ∀ (HL+d)	Z	Z	–	–	5
XOR A, (HL+C)	1110 0101 0111 1101	A ← A ∀ (HL+C)	Z	Z	–	–	5
XOR A, (PC+A)	1110 0001 0111 1101	A ← A ∀ (PC+A)	Z	Z	–	–	5

MNEMONIC	OBJECT CODE (Binary)	OPERATION	FLAG				CYCLE	
			JF	ZF	CF	HF		
XOR (x), (HL)	1110 0000 xxxx xxxx 0110 0101	(x) ← (x) ∀ (HL)	Z	Z	–	–	7	
XOR (pp), (HL)	1110 001p 0110 0101	(pp) ← (pp) ∀ (HL)	Z	Z	–	–	6	
XOR (HL+d), (HL)	1110 0100 dddd dddd 0110 0101	(HL+d) ← (HL+d) ∀ (HL)	Z	Z	–	–	8	
XOR (HL+C), (HL)	1110 0101 0110 0101	(HL+C) ← (HL+C) ∀ (HL)	Z	Z	–	–	8	
XOR (PC+A), (HL)	1110 0001 0110 0101	(PC+A) ← (PC+A) ∀ (HL)	Z	Z	–	–	8	
XOR A, n	0111 0101 nnnn nnnn	A ← A ∀ n	Z	Z	–	–	2	
XOR g, n	1110 1ggg 0111 0101 nnnn nnnn	g ← g ∀ n	Z	Z	–	–	3	
XOR gg, mn	1110 10gg 0011 1101 nnnn nnnn mmmm mmmm	gg ← gg ∀ mn	Z	Z	–	–	4	
XOR (x), n	1110 0000 xxxx xxxx 0111 0101 nnnn nnnn	(x) ← (x) ∀ n	Z	Z	–	–	6	
XOR (pp), n	1110 001p 0111 0101 nnnn nnnn	(pp) ← (pp) ∀ n	Z	Z	–	–	5	
XOR (HL+), n	1110 0110 0111 0101 nnnn nnnn	(HL) ← (HL) ∀ n: HL ← HL+1	Z	Z	–	–	6	
XOR (–HL), n	1110 0111 0111 0101 nnnn nnnn	HL ← HL–1: (HL) ← (HL) ∀ n	Z	Z	–	–	6	
XOR (HL+d), n	1110 0100 dddd dddd 0111 0101 nnnn nnnn	(HL+d) ← (HL+d) ∀ n	Z	Z	–	–	7	
XOR (HL+C), n	1110 0101 0111 0101 nnnn nnnn	(HL+C) ← (HL+C) ∀ n	Z	Z	–	–	7	
XOR (PC+A), n	1110 0001 0111 0101 nnnn nnnn	(PC+A) ← (PC+A) ∀ n	Z	Z	–	–	7	
DAA A	0000 1010	Decimal adjust A for addition	C	Z	C	H	2	
DAA g	1110 1ggg 0000 1010	Decimal adjust g for addition	C	Z	C	H	3	
DAS A	0000 1011	Decimal adjust A for subtraction	C	Z	C	H	2	
DAS g	1110 1ggg 0000 1011	Decimal adjust g for subtraction	C	Z	C	H	3	
MUL W, A	0000 0010	WA ← W×A	Z	Z	–	–	7	*3
MUL B, C	1110 1001 0000 0010	BC ← B×C	Z	Z	–	–	8	*3
MUL D, E	1110 1010 0000 0010	DE ← D×E	Z	Z	–	–	8	*3
MUL H, L	1110 1011 0000 0010	HL ← H×L	Z	Z	–	–	8	*3
DIV WA, C	0000 0011	A ← WA ÷ C, W ← remainder	Z	Z	C	–	7	*4
DIV DE, C	1110 1010 0000 0011	E ← DE ÷ C, D ← remainder	Z	Z	C	–	8	*4
DIV HL, C	1110 1011 0000 0011	L ← HL ÷ C, H ← remainder	Z	Z	C	–	8	*4

Notes: *3 *During MUL instruction, the ZF is set to "1" if the high byte of the product is 00_H; otherwise the ZF is cleared to "0".*

*4 *During DIV instruction, the ZF is set to "1" if the remainder is 00_H; otherwise the ZF is cleared to "0". The CF is set to "1" if the divisor (C register contents) is 00_H or the quotient is greater than FF_H; otherwise the CF is cleared to "0".*

(3) SHIFT, ROTATE and NIBBLE MANIPULATION

MNEMONIC	OBJECT CODE (Binary)		OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE
SHLC A	0001 1100			C	Z	*	–	1
SHLC g	1110 1ggg	0001 1100		C	Z	*	–	2
SHRC A	0001 1101			C	Z	*	–	1
SHRC g	1110 1ggg	0001 1101		C	Z	*	–	2
ROLC A	0001 1110			C	Z	*	–	1
ROLC g	1110 1ggg	0001 1110		C	Z	*	–	2
RORC A	0001 1111			C	Z	*	–	1
RORC g	1110 1ggg	0001 1111		C	Z	*	–	2
SWAP A	0000 0001			1	–	–	–	3
SWAP g	1110 1ggg	0000 0001	(Swap upper and lower nibbles)	1	–	–	–	4
ROLD A, (x)	1110 0000 0000 1000	xxxx xxxx		1	–	–	–	8
ROLD A, (pp)	1110 001p	0000 1000		1	–	–	–	7
ROLD A, (HL+)	1110 0110	0000 1000		1	–	–	–	8
ROLD A, (–HL)	1110 0111	0000 1000		1	–	–	–	8
ROLD A, (HL+d)	1110 0100 0000 1000	dddd dddd	*Notes: (HL+) HL←HL+1* *(–HL) HL←HL–1*	1	–	–	–	9
ROLD A, (HL+C)	1110 0101	0000 1000		1	–	–	–	9
RORD A, (x)	1110 0000 0000 1001	xxxx xxxx		1	–	–	–	8
RORD A, (pp)	1110 001p	0000 1001		1	–	–	–	7
RORD A, (HL+)	1110 0110	0000 1001		1	–	–	–	8
RORD A, (–HL)	1110 0111	0000 1001		1	–	–	–	8
RORD A, (HL+d)	1110 0100 0000 1001	dddd dddd	*Notes: (HL+) HL←HL+1* *(–HL) HL←HL–1*	1	–	–	–	9
RORD A, (HL+C)	1110 0101	0000 1001		1	–	–	–	9

(4) BIT MANIPULATION and FLAG MANIPURATION

MNEMONIC	OBJECT CODE (Binary)	OPERATION	FLAG				CYCLE	
			JF	ZF	CF	HF		
TEST g.b	1110 1ggg 1101 1bbb	JF ← $\overline{g.b}$	*	–	$\overline{J}$	–	2	*5
TEST (x).b	1101 1bbb xxxx xxxx	JF ← $\overline{(x).b}$	*	–	$\overline{J}$	–	4	*5
TEST (pp).b	1110 001p 1101 1bbb	JF ← $\overline{(pp).b}$	*	–	$\overline{J}$	–	3	*5
TEST (HL+).b	1110 0110 1101 1bbb	JF ← $\overline{(HL).b}$: HL ← HL+1	*	–	$\overline{J}$	–	4	*5
TEST (−HL).b	1110 0111 1101 1bbb	HL ← HL−1: JF ← $\overline{(HL).b}$	*	–	$\overline{J}$	–	4	*5
TEST (HL+d).b	1110 0100 dddd dddd 1101 1bbb	JF ← $\overline{(HL+d).b}$	*	–	$\overline{J}$	–	5	*5
TEST (HL+C).b	1110 0101 1101 1bbb	JF ← $\overline{(HL+C).b}$	*	–	$\overline{J}$	–	5	*5
TEST (PC+A).b	1110 0001 1101 1bbb	JF ← $\overline{(PC+A).b}$	*	–	$\overline{J}$	–	5	*5
TEST (pp).g	1110 1ggg 1001 111p	JF ← $\overline{(pp).g_{2-0}}$	*	–	$\overline{J}$	–	4	*5
CLR g.b	1110 1ggg 0100 1bbb	ZF ← $\overline{g.b}$: g.b ← 0	Z	*	–	–	3	
CLR (x).b	0100 1bbb xxxx xxxx	ZF ← $\overline{(x).b}$: (x).b ← 0	Z	*	–	–	5	
CLR (pp).b	1110 001p 0100 1bbb	ZF ← $\overline{(pp).b}$: (pp).b ← 0	Z	*	–	–	4	
CLR (HL+).b	1110 0110 0100 1bbb	ZF ← $\overline{(HL).b}$: (HL).b ← 0 : HL ← HL+1	Z	*	–	–	5	
CLR (−HL).b	1110 0111 0100 1bbb	HL ← HL−1: ZF ← $\overline{(HL).b}$: (HL).b ← 0	Z	*	–	–	5	
CLR (HL+d).b	1110 0100 dddd dddd 0100 1bbb	ZF ← $\overline{(HL+d).b}$: (HL+d).b ← 0	Z	*	–	–	6	
CLR (HL+C).b	1110 0101 0100 1bbb	ZF ← $\overline{(HL+C).b}$: (HL+C).b ← 0	Z	*	–	–	6	
CLR (PC+A).b	1110 0001 0100 1bbb	ZF ← $\overline{(PC+A).b}$: (PC+A).b ← 0	Z	*	–	–	6	
CLR (pp).g	1110 1ggg 1000 101p	ZF ← $\overline{(pp).g_{2-0}}$: (pp).g_{2-0} ← 0	Z	*	–	–	5	
SET g.b	1110 1ggg 0100 0bbb	ZF ← $\overline{g.b}$: g.b ← 1	Z	*	–	–	3	
SET (x).b	0100 0bbb xxxx xxxx	ZF ← $\overline{(x).b}$: (x).b ← 1	Z	*	–	–	5	
SET (pp).b	1110 001p 0100 0bbb	ZF ← $\overline{(pp).b}$: (pp).b ← 1	Z	*	–	–	4	
SET (HL+).b	1110 0110 0100 0bbb	ZF ← $\overline{(HL).b}$: (HL).b ← 1 : HL ← HL+1	Z	*	–	–	5	
SET (−HL).b	1110 0111 0100 0bbb	HL ← HL−1: ZF ← $\overline{(HL).b}$: (HL).b ← 1	Z	*	–	–	5	
SET (HL+d).b	1110 0100 dddd dddd 0100 0bbb	ZF ← $\overline{(HL+d).b}$: (HL+d).b ← 1	Z	*	–	–	6	
SET (HL+C).b	1110 0101 0100 0bbb	ZF ← $\overline{(HL+C).b}$: (HL+C).b ← 1	Z	*	–	–	6	
SET (PC+A).b	1110 0001 0100 0bbb	ZF ← $\overline{(PC+A).b}$: (PC+A).b ← 1	Z	*	–	–	6	
SET (pp).g	1110 1ggg 1000 001p	ZF ← $\overline{(pp).g_{2-0}}$: (pp).g_{2-0} ← 1	Z	*	–	–	5	
CPL g.b	1110 1ggg 1100 0bbb	ZF ← $\overline{g.b}$: g.b ← $\overline{g.b}$	Z	*	–	–	3	
CPL (x).b	1110 0000 xxxx xxxx 1100 0bbb	ZF ← $\overline{(x).b}$: (x).b ← $\overline{(x).b}$	Z	*	–	–	5	
CPL (pp).b	1110 001p 1100 0bbb	ZF ← $\overline{(pp).b}$: (pp).b ← $\overline{(pp).b}$	Z	*	–	–	4	
CPL (HL+).b	1110 0110 1100 0bbb	ZF ← $\overline{(HL).b}$: (HL).b ← $\overline{(HL).b}$: HL ← HL+1	Z	*	–	–	5	
CPL (−HL).b	1110 0111 1100 0bbb	HL ← HL−1: ZF ← $\overline{(HL).b}$: (HL).b ← $\overline{(HL).b}$	Z	*	–	–	5	
CPL (HL+d).b	1110 0100 dddd dddd 1100 0bbb	ZF ← $\overline{(HL+d).b}$: (HL+d).b ← $\overline{(HL+d).b}$	Z	*	–	–	6	
CPL (HL+C).b	1110 0101 1100 0bbb	ZF ← $\overline{(HL+C).b}$: (HL+C).b ← $\overline{(HL+C).b}$	Z	*	–	–	6	
CPL (PC+A).b	1110 0001 1100 0bbb	ZF ← $\overline{(PC+A).b}$: (PC+A).b ← $\overline{(PC+A).b}$	Z	*	–	–	6	
CPL (pp).g	1110 1ggg 1001 001p	ZF ← $\overline{(pp).g_{2-0}}$: (pp).g_{2-0} ← $\overline{(pp).g_{2-0}}$	Z	*	–	–	5	
LD CF, g.b	1110 1ggg 1101 1bbb	CF ← g.b	$\overline{C}$	–	*	–	2	
LD CF, (x).b	1101 1bbb xxxx xxxx	CF ← (x).b	$\overline{C}$	–	*	–	4	
LD CF, (pp).b	1110 001p 1101 1bbb	CF ← (pp).b	$\overline{C}$	–	*	–	3	
LD CF, (HL+).b	1110 0110 1101 1bbb	CF ← (HL).b: HL ← HL+1	$\overline{C}$	–	*	–	4	
LD CF, (−HL).b	1110 0111 1101 1bbb	HL ← HL−1: CF ← (HL).b	$\overline{C}$	–	*	–	4	
LD CF, (HL+d).b	1110 0100 dddd dddd 1101 1bbb	CF ← (HL+d).b	$\overline{C}$	–	*	–	5	
LD CF, (HL+C).b	1110 0101 1101 1bbb	CF ← (HL+C).b	$\overline{C}$	–	*	–	5	
LD CF, (PC+A).b	1110 0001 1101 1bbb	CF ← (PC+A).b	$\overline{C}$	–	*	–	5	
LD CF, (pp).g	1110 1ggg 1001 111p	CF ← (pp).g_{2-0}	$\overline{C}$	–	*	–	4	

*Note: *5 Assembler expansion machine instruction.*

MNEMONIC	OBJECT CODE (Binary)	OPERATION	FLAG				CYCLE	
			JF	ZF	CF	HF		
LD g.b, CF	1110 1ggg 1100 1bbb	g.b ← CF	1	–	–	–	2	
LD (x).b, CF	1110 0000 xxxx xxxx 1100 1bbb	(x).b ← CF	1	–	–	–	5	
LD (pp).b, CF	1110 001p 1100 1bbb	(pp).b ← CF	1	–	–	–	4	
LD (HL+).b, CF	1110 0110 1100 1bbb	(HL).b ← CF: HL ← HL+1	1	–	–	–	5	
LD (−HL).b, CF	1110 0111 1100 1bbb	HL ← HL−1: (HL).b ← CF	1	–	–	–	5	
LD (HL+d).b, CF	1110 0100 dddd dddd 1100 1bbb	(HL+d).b ← CF	1	–	–	–	6	
LD (HL+C).b, CF	1110 0101 1100 1bbb	(HL+C).b ← CF	1	–	–	–	6	
LD (PC+A).b, CF	1110 0001 1100 1bbb	(PC+A).b ← CF	1	–	–	–	6	
LD (pp).g, CF	1110 1ggg 1001 101p	(pp).g_{2-0} ← CF	1	–	–	–	5	
XOR CF, g.b	1110 1ggg 1101 0bbb	CF ← CF∀g.b	$\overline{C}$	–	*	–	2	
XOR CF, (x).b	1110 0000 xxxx xxxx 1101 0bbb	CF ← CF∀(x).b	$\overline{C}$	–	*	–	4	
XOR CF, (pp).b	1110 001p 1101 0bbb	CF ← CF∀(pp).b	$\overline{C}$	–	*	–	3	
XOR CF, (HL+).b	1110 0110 1101 0bbb	CF ← CF∀(HL).b: HL ← HL+1	$\overline{C}$	–	*	–	4	
XOR CF, (−HL).b	1110 0111 1101 0bbb	HL ← HL−1: CF ← CF∀(HL).b	$\overline{C}$	–	*	–	4	
XOR CF, (HL+d).b	1110 0100 dddd dddd 1101 0bbb	CF ← CF∀(HL+d).b	$\overline{C}$	–	*	–	5	
XOR CF, (HL+C).b	1110 0101 1101 0bbb	CF ← CF∀(HL+C).b	$\overline{C}$	–	*	–	5	
XOR CF, (PC+A).b	1110 0001 1101 0bbb	CF ← CF∀(PC+A).b	$\overline{C}$	–	*	–	5	
CLR CF	0000 1100	CF ← 0	1	–	0	–	1	
SET CF	0000 1101	CF ← 1	0	–	1	–	1	
CPL CF	0000 1110	JF ← $\overline{CF}$: CF ← $\overline{CF}$	*	–	*	–	1	
DI	0100 1000 0011 1010	ZF ← $\overline{IMF}$: IMF ← 0	Z	*	–	–	5	*5
EI	0100 0000 0011 1010	ZF ← $\overline{IMF}$: IMF ← 1	Z	*	–	–	5	*5

*Note: *5 Assembler expansion machine instruction.*

(5) JUMP

MNEMONIC	OBJECT CODE (Binary)	OPERATION	FLAG				CYCLE	
			JF	ZF	CF	HF		
JRS T,a	100d dddd	if JF = 1 then PC←PC+d (a = $+2+d) else null	1	–	–	–	4/2	*6
JRS F,a	101d dddd	if JF = 0 then PC←PC+d else null	1	–	–	–	4/2	*6
JR T,a	1101 0110 dddd dddd	if JF = 1 then PC←PC+d (a = $+2+d) else null	1	–	–	–	4/2	*6
JR F,a	1101 0111 dddd dddd	if JF = 0 then PC←PC+d else null	1	–	–	–	4/2	*6
JR EQ,a / Z,a	1101 0000 dddd dddd	if ZF = 1 then PC←PC+d else null	1	–	–	–	4/2	*6
JR NE,a / NZ,a	1101 0001 dddd dddd	if ZF = 0 then PC←PC+d else null	1	–	–	–	4/2	*6
JR CS,a / LT,a	1101 0010 dddd dddd	if CF = 1 then PC←PC+d else null	1	–	–	–	4/2	*6
JR CC,a / GE,a	1101 0011 dddd dddd	if CF = 0 then PC←PC+d else null	1	–	–	–	4/2	*6
JR LE,a	1101 0100 dddd dddd	if (CF∨ZF) = 1 then PC←PC+d else null	1	–	–	–	4/2	*6
JR GT,a	1101 0101 dddd dddd	if (CF∨ZF) = 0 then PC←PC+d else null	1	–	–	–	4/2	*6
JR a	1111 1011 dddd dddd	PC←PC+d (a = $+2+d)	1	–	–	–	4	
JP mn	1111 1110 nnnn nnnn mmmm mmmm	PC←mn	1	–	–	–	4	
JP gg	1110 10gg 1111 1110	PC←gg	1	–	–	–	3	
JP (x)	1110 0000 xxxx xxxx 1111 1110	PC←(x+1,x)	1	–	–	–	6	
JP (pp)	1110 001p 1111 1110	PC←(pp+1,pp)	1	–	–	–	5	
JP (HL+d)	1110 0100 dddd dddd 1111 1110	PC←(HL+d+1,HL+d)	1	–	–	–	7	
JP (HL+C)	1110 0101 1111 1110	PC←(HL+C+1,HL+C)	1	–	–	–	7	
JP (PC+A)	1110 0001 1111 1110	PC←(PC+A+1,PC+A)	1	–	–	–	7	

*Note: *6 A description of "4/2" in the cycle field is provided below.*
"4" means the number of cycles when the condition is satisfied; "2" means the number of cycles when the condition is not satisfied.

(6) CALL, RETURN, PUSH, POP, SOFTWARE INTERRUPT and NO OPERATION

MNEMONIC	OBJECT CODE (Binary)	OPERATION	FLAG JF	FLAG ZF	FLAG CF	FLAG HF	CYCLE	
CALLV n	1100 nnnn	(SP, SP−1) ← PC−1: SP ← SP−2: PC ← (n×2+FFC1$_H$, n×2+FFC0$_H$)	−	−	−	−	7	
CALLP n	1111 1101 nnnn nnnn	(SP, SP−1) ← PC: SP ← SP−2: PC ← FF00$_H$+n	−	−	−	−	6	
CALL mn	1111 1100 mmmm mmmm nnnn nnnn	(SP, SP−1) ← PC+1: SP ← SP−2: PC ← mn	−	−	−	−	6	
CALL gg	1110 10gg 1111 1100	(SP, SP−1) ← PC: SP ← SP−2: PC ← gg	−	−	−	−	6	
CALL (x)	1110 0000 xxxx xxxx 1111 1100	(SP, SP−1) ← PC+1: SP ← SP−2: PC ← (x+1, x)	−	−	−	−	9	
CALL (pp)	1110 001p 1111 1100	(SP, SP−1) ← PC: SP ← SP−2: PC ← (pp+1, pp)	−	−	−	−	8	
CALL (HL+d)	1110 0100 dddd dddd 1111 1100	(SP, SP−1) ← PC+1: SP ← SP−2: PC ← (HL+d+1, HL+d)	−	−	−	−	10	
CALL (HL+C)	1110 0101 1111 1100	(SP, SP−1) ← PC: SP ← SP−2: PC ← (HL+C+1, HL+C)	−	−	−	−	10	
CALL (PC+A)	1110 0001 1111 1100	(SP, SP−1) ← PC: SP ← SP−2: PC ← (PC+A+1, PC+A)	−	−	−	−	10	
RET	0000 0101	SP ← SP+2: PC ← (SP, SP−1)	−	−	−	−	6	
RETI	0000 0100	SP ← SP+3: PC ← (SP−1, SP−2): PSW ← (SP) Return from maskable interrupt service	*	*	*	*	6	
RETN	1110 1000 0000 0100	SP ← SP+3: PC ← (SP−1, SP−2): PSW ← (SP) Return from non-maskable interrupt service	*	*	*	*	7	
PUSH PSW	0000 0111	(SP) ← PSW: SP ← SP−1	−	−	−	−	2	
PUSH gg	1110 10gg 0000 0111	(SP, SP−1) ← gg: SP ← SP−2	−	−	−	−	4	
POP PSW	0000 0110	SP ← SP+1: PSW ← (SP)	*	*	*	*	3	
POP gg	1110 10gg 0000 0110	SP ← SP+2: gg ← (SP, SP−1)	−	−	−	−	5	
SWI	1111 1111	Software interrupt	−	−	−	−	9/1	*7
NOP	0000 0000	No operation	−	−	−	−	1	

*Notes: *7 A description of "9/1" in the cycle field is provided below.*

"9" means the number of cycles when the SWI instruction is executed except during the non-maskable interrupt service;

"1" means the number of cycles when this instruction is executed during the non-maskable interrupt service.

**8 The following instructions have dummy fetch cycle.*

① *JRS cc, a (cc = T/F)*
② *JR cc, a (cc = T/F/EQ/NE/Z/NZ/CS/CC/LT/GE/LE/GT)*
③ *JR a*
④ *JP gg*
⑤ *CALLV n*
⑥ *CALL gg*
⑦ *RET*
⑧ *RETI*
⑨ *RETN*
⑩ *SWI*

1.7 INSTRUCTION CODE MAPS

These tables are opcode maps for the instructions of the TLCS-870 Series.

(1) 1st OPCODE

H \ L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	SWAP A	MUL W,A	DIV WA,C	RETI	RET	POP PSW	PUSH PSW			DAA A	DAS A	CLR CF	SET CF	CPL CF	LD RBS, n
1	INC rr: WA	INC rr: BC	INC rr: DE	INC rr: HL	LD rr,mn: WA	LD rr,mn: BC	LD rr,mn: DE	LD rr,mn: HL	DEC rr: WA	DEC rr: BC	DEC rr: DE	DEC rr: HL	SHLC A	SHRC A	ROLC A	RORC A
2	INC (mem): (x)	INC (mem): (HL)	LD A,(mem): (x)	LD A,(mem): (HL)	LDW (mem), mn: (x)	LDW (mem), mn: (HL)	LD (x),(y)		DEC (mem): (x)	DEC (mem): (HL)	LD (mem),A: (x)	LD (mem),A: (HL)	LD (mem),n: (x)	LD (mem),n: (HL)	CLR (mem): (x)	CLR (mem): (HL)
3	LD r, n: A	LD r, n: W	LD r, n: C	LD r, n: B	LD r, n: E	LD r, n: D	LD r, n: L	LD r, n: H								
4	SET (x).b: 0	SET (x).b: 1	SET (x).b: 2	SET (x).b: 3	SET (x).b: 4	SET (x).b: 5	SET (x).b: 6	SET (x).b: 7	CLR (x).b: 0	CLR (x).b: 1	CLR (x).b: 2	CLR (x).b: 3	CLR (x).b: 4	CLR (x).b: 5	CLR (x).b: 6	CLR (x).b: 7
5	LD A, r: A	LD A, r: W	LD A, r: C	LD A, r: B	LD A, r: E	LD A, r: D	LD A, r: L	LD A, r: H	LD r, A: A	LD r, A: W	LD r, A: C	LD r, A: B	LD r, A: E	LD r, A: D	LD r, A: L	LD r, A: H
6	INC r: A	INC r: W	INC r: C	INC r: B	INC r: E	INC r: D	INC r: L	INC r: H	DEC r: A	DEC r: W	DEC r: C	DEC r: B	DEC r: E	DEC r: D	DEC r: L	DEC r: H
7	ADDC A, n	ADD A, n	SUBB A, n	SUB A, n	AND A, n	XOR A, n	OR A, n	CMP A, n	ADDC A, (x)	ADD A, (x)	SUBB A, (x)	SUB A, (x)	AND A, (x)	XOR A, (x)	OR A, (x)	CMP A, (x)
8	JRS T, a															
9																
A	JRS F, a															
B																
C	CALLV n															
D	JR cc, a: EQ / Z	JR cc, a: NE / NZ	JR cc, a: LT / CS	JR cc, a: GE / CC	JR cc, a: LE	JR cc, a: GT	JR cc, a: T	JR cc, a: F	LD CF, (x).b: 0	LD CF, (x).b: 1	LD CF, (x).b: 2	LD CF, (x).b: 3	LD CF, (x).b: 4	LD CF, (x).b: 5	LD CF, (x).b: 6	LD CF, (x).b: 7
E	source memory prefix : (src): (x)	source memory prefix : (src): (PC + A)	source memory prefix : (src): (DE)	source memory prefix : (src): (HL)	source memory prefix : (src): (HL + d)	source memory prefix : (src): (HL + C)	source memory prefix : (src): (HL +)	source memory prefix : (src): (– HL)	register prefix : g / gg: A / WA	register prefix : g / gg: W / BC	register prefix : g / gg: C / DE	register prefix : g / gg: B / HL	register prefix : g / gg: E	register prefix : g / gg: D	register prefix : g / gg: L	register prefix : g / gg: H
F	destination memory prefix : (dst): (x)		destination memory prefix : (dst): (DE)	destination memory prefix : (dst): (HL)	destination memory prefix : (dst): (HL + d)		destination memory prefix : (dst): (HL +)	destination memory prefix : (dst): (– HL)			LD SP, mn	JR a	CALL mn	CALLP n	JP mn	SWI

Note: Blank code is undefined.

(2) 2nd OPCODE (Register Prefix)

H \ L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		SWAP g	MUL ggH,ggL	DIV gg,C	RETN *		POP gg	PUSH gg			DAA g	DAS g				
1	XCH rr, gg WA : BC : DE : HL				LD rr, gg WA : BC : DE : HL								SHLC g	SHRC g	ROLC g	RORC g
2																
3	ADDC WA, gg	ADD WA, gg	SUBB WA, gg	SUB WA, gg	AND WA, gg	XOR WA, gg	OR WA, gg	CMP WA, gg	ADDC gg, mn	ADD gg, mn	SUBB gg, mn	SUB gg, mn	AND gg, mn	XOR gg, mn	OR gg, mn	CMP gg, mn
4	SET g.b 0 : 1 : 2 : 3 : 4 : 5 : 6 : 7								CLR g.b 0 : 1 : 2 : 3 : 4 : 5 : 6 : 7							
5									LD r, g A : W : C : B : E : D : L : H							
6	ADDC A, g	ADD A, g	SUBB A, g	SUB A, g	AND A, g	XOR A, g	OR A, g	CMP A, g	ADDC g, A	ADD g, A	SUBB g, A	SUB g, A	AND g, A	XOR g, A	OR g, A	CMP g, A
7	ADDC g, n	ADD g, n	SUBB g, n	SUB g, n	AND g, n	XOR g, n	OR g, n	CMP g, n								
8			SET (mem).g (DE) : (HL)								CLR (mem).g (DE) : (HL)					
9			CPL (mem).g (DE) : (HL)								LD (mem).g, CF (DE) : (HL)				LD CF, (mem).g (DE) : (HL)	
A									XCH r, g A : W : C : B : E : D : L : H							
B																
C	CPL g.b 0 : 1 : 2 : 3 : 4 : 5 : 6 : 7								LD g.b, CF 0 : 1 : 2 : 3 : 4 : 5 : 6 : 7							
D	XOR CF, g.b 0 : 1 : 2 : 3 : 4 : 5 : 6 : 7								LD CF, g.b 0 : 1 : 2 : 3 : 4 : 5 : 6 : 7							
E																
F											LD SP, gg	LD gg, SP	CALL gg		JP gg	

Notes: *1. Blank code is undefined.*
2. The 1st opcodes are shown below.

g	1st opcode
W	E9
A	E8
B	EB
C	EA
D	ED
E	EC
H	EF
L	EE

gg	1st opcode
WA	E8
BC	E9
DE	EA
HL	EB

*	1st opcode
	E8

(3) 2nd OPCODE (Source / Destination Memory Prefix)

H \ L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0									ROLD A, (src)	RORD A, (src)						
1	LD (dst), rr WA BC DE HL				LD rr, (src) WA BC DE HL											
2	INC (src)						LD (x),(src)	LD (HL),(src)	DEC (src)				LD (dst), n			MCMP (src), n
3																
4	SET (src).b 0 1 2 3 4 5 6 7								CLR (src).b 0 1 2 3 4 5 6 7							
5	LD (dst), r A W C B E D L H								LD r, (src) A W C B E D L H							
6	ADDC (src), (HL)	ADD (src), (HL)	SUBB (src), (HL)	SUB (src), (HL)	AND (src), (HL)	XOR (src), (HL)	OR (src), (HL)	CMP (src), (HL)								
7	ADDC (src), n	ADD (src), n	SUBB (src), n	SUB (src), n	AND (src), n	XOR (src), n	OR (src), n	CMP (src), n	ADDC A, (src)	ADD A, (src)	SUBB A, (src)	SUB A, (src)	AND A, (src)	XOR A, (src)	OR A, (src)	CMP A, (src)
8																
9																
A									XCH r, (src) A W C B E D L H							
B																
C	CPL (src).b 0 1 2 3 4 5 6 7								LD (src).b, CF 0 1 2 3 4 5 6 7							
D	XOR CF, (src).b 0 1 2 3 4 5 6 7								LD CF, (src).b 0 1 2 3 4 5 6 7							
E																
F													CALL (src)		JP (src)	

Notes: *1. Blank code is undefined.*
2. The 1st opcodes are shown below.

(src)	1st opcode
(x)	E0
(PC+A)	E1
(DE)	E2
(HL)	E3
(HL+d)	E4
(HL+C)	E5
(HL+)	E6
(−HL)	E7

(dst)	1st opcode
(x)	F0
(DE)	F2
(HL)	F3
(HL+d)	F4
(HL+)	F6
(−HL)	F7

1.8 EXAMPLES OF INSTRUCTION USAGE

This section explains how to use the TLCS-870 Series instructions more effectively and more expansively.

1.8.1 Data Transfer / Exchange Instructions

(1) LD A, (x)

The instructions that load 8-bit data from memory to register have a zero test function. For example, the instruction [LD A, (x)] reads data from memory address "x" to the accumulator and enters whether or not the data was 00_H in the ZF.

By using the zero test function, the comparison instructions [CMP A,00H] can be omitted after transfers and the number of program steps can be reduced.

Example : Jump to SNEXT if 00_H is read from I/O port P0.

```
        LD      A, (P0)          ; A←Port P0
        JR      Z , SNEXT        ; If A = 00H then jump to SNEXT
        :
SNEXT : ———
```

(2) LD (HL +) , n

This instruction is used for storing immediate data to addresses within a specific memory area.

Example : Stores data to RAM addresses $00FC_H$ to 0102_H.

```
LD      HL, 00FCH
LD      (HL + ) , 11110000B
LD      (HL + ) , 11011000B
LD      (HL + ) , 11111111B
LD      (HL + ) , 00001111B
LD      (HL + ) , 10100110B
LD      (HL + ) , 00111001B
LD      (HL + ) , 11100100B
```

Note: Use the instruction [LD (x), n] when the memory address range is 0000_H to $00FF_H$.

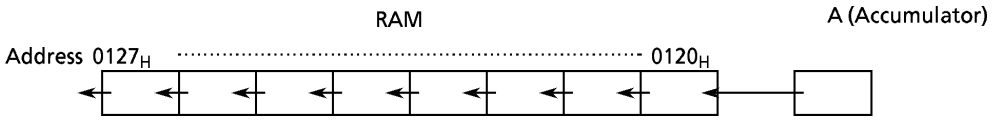
Example : Initializes I/O ports.

```
LD      (00H) , 11111111B    ; Port P0←11111111B
LD      (01H) , 11111111B    ; Port P1←11111111B
LD      (02H) , 00000111B    ; Port P2←00000111B
LD      (03H) , 11111111B    ; Port P3←11111111B
```

(3) XCH r, (HL +)

This instruction is used for byte data shifts.

Example : Shifts the contents of the accumulator into RAM addresses 0120_H to 0127_H.



```
                LD      HL, 0120H
                LD      C, 07H
SSHIFT :        XCH     A, (HL + )
                DEC     C
                JRS     F, SSHIFT
```

(4) LD (x), (HL + C) and LD (x), (PC + A)

These instructions are suitable for code conversion.

Example : This subroutine reads from the ROM table the 7-segment codes corresponding to the BCD data (00_H - 09_H) stored in the accumulator and outputs to port P3.

① [LD (x), (HL + C)] instruction

```
                LD      HL, TABLE
                LD      C, A
                LD      (P3), (HL + C)
                RET
                 ⋮
TABLE:          DB      11000000B, 11111001B, 10100100B, 10110000B,
                        10011001B, 10010010B, 10000010B, 11011000B,
                        10000000B, 10011000B
```

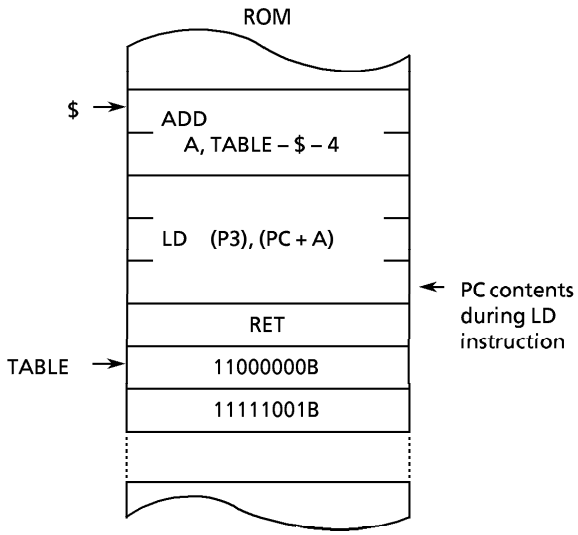
② [LD (x), (PC + A)] instruction

```
                ADD     A, TABLE – $ – 4      ; Adds the offset value
                LD      (P3), (PC + A)
                RET
TABLE:          DB      11000000B, 11111001B, 10100100B, 10110000B,
                        10011001B, 10010010B, 10000010B, 11011000B,
                        10000000B, 10011000B
```

How to calculate offset value

The $ contains the header address of the ADD instruction. The PC contents are $ + 4 during the LD instruction; therefore, the offset value "n" is calculated as follows:

$$PC + A + n = TABLE + A$$
$$\therefore \quad n = TABLE - PC$$
$$= TABLE - (\$ + 4)$$
$$= TABLE - \$ - 4$$



1.8.2 Arithmetic and Logical Operations

(1) MCMP (x) , n

This instruction is used to compare multiple bits of memory data.

Example : Checks whether or not the lower 4 bits at RAM address 96_H are 1010_B.

```
        LD      A, 00001010B
        MCMP    (96H) , 00001111B
        JRS     F , SNE
SEQ :   ————————                    ; Equal
        ⋮
SNE :   ————————                    ; Not Equal
```

(2) DAA r and DAS r

The DAA instruction is used for decimal adjust when adding BCD data.
The DAS instruction is used for decimal adjust when subtracting BCD data.

Example : Increments the 4-digit BCD data stored in the WA register pair.

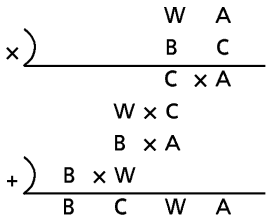
```
        ADD     A, 1                ; A ← A + 1
        DAA     A
        ADDC    W, 0                ; W ← W + CF
        DAA     W
```

(3) MUL r_H , r_L

This instruction multiplies the contents of the 8-bit registers r_H and r_L, and obtains a 16-bit product. In the case of multiple-digit multiplication, calculates as in ordinary multiplication.

Example : Multiplies the 16-bit data stored at RAM addresses 0072_H - 0073_H and the 16-bit data stored in the WA register pair, and obtains a 32-bit result (the lower 16 bits of the product are stored into the WA register pair and the upper 16 bits are stored into the BC register pair).

```
        LD      BC, (72H)           ; BC ← RAM (0073H, 0072H)
        LD      DE, BC              ; HL ← C×A, BC ←B×W
        LD      HL , WA
        XCH     H , C
        MUL     H , L
        MUL     B, C
        XCH     A , E               ; WA ← W×C, DE ←B×A
        MUL     W, A
        MUL     D, E
        ADD     WA, DE              ; WA ← WA + DE
        ADDC    B, 00H              ; B ←B + CF
        LD      D, C                ; WA ←  WA + CH
        LD      E, H
        ADD     WA, DE
        ADDC    B, 00H              ; B ←B + CF
        LD      C, W                ; C ←W
        LD      W, A                ; W←A
        LD      A , L               ; A ←L
```



(4) DIV WA , C

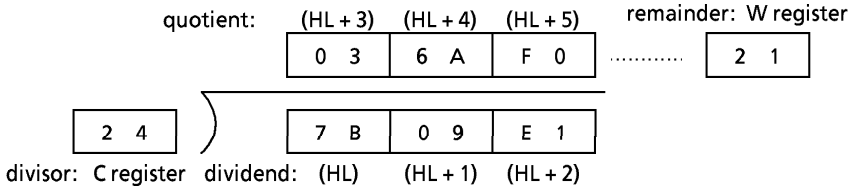
This instruction divides the 2-byte data by the 1-byte data and obtains the quotient and remainder for the 1-byte length. An overflow error occurs when the quotient exceeds 1 byte; therefore, the division is performed in two stages (see Example 1). When multiple byte data are divided by 1-byte data, the quotient is determined by dividing from the most significant digit, in the same sequence as for manual calculation (see Example 2).

The DIV instruction is convenient for performing radix conversions such as converting binary to BCD (see Example 3).

Example 1: Calculates 16-bit data (WA register pair) ÷ 8-bit data (C register), and obtains a 16-bit quotient (WA register pair) and 8-bit remainder (B register).

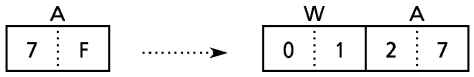
```
        LD      B , 00H         ; If C = 0 then jump to SERROR
        CMP     C, B
        JR      T, SERROR
        XCH     A , W
        XCH     W, B
        DIV     WA, C
        XCH     A, B
        DIV     WA, C           ; A ← low byte of quotient
        XCH     W, B            ; W ← high byte of quotient
        ⋮
SERROR: ————                    ; Divided by zero Error
```

Example 2: Divides the dividend stored in the 3-byte memory starting from the address specified by the contents of the HL register pair by the contents of the C register, stores the quotient to 3 bytes of memory following the dividend and stores the remainder to the W register.




```
        LD      A , (HL + )     ; A ← most significant digit of dividend
        LD      W, 00H
        DIV     WA, C           ; A ← most significant digit of quotient
        LD      (HL + 2) , A
        LD      A , (HL + )
        DIV     WA, C
        LD      (HL + 2) , A
        LD      A , (HL)        ; A ← least significant digit of dividend
        DIV     WA, C           ; A ← least significant digit of quotient, W ← remainder
        LD      (HL + 3) , A
```

Example 3: Converts the binary stored into the A register to packed BCD (binary coded decimal), and stores the result into the WA register.



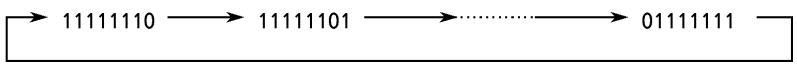

```
LD      BC, 000AH       ; B ← 0, C ← 10
LD      W, B            ; B ← least significant digit (1)
DIV     WA, C
XCH     W, B
DIV     WA, C           ; A ← middle significant digit (10)
XCH     A, W              W ← most significant digit (100)
SWAP    A
ADD     A, B            ; A ← middle significant digit (10) and
                              least significant digit (1)
```

1.8.3 Shift, Rotate and Nibble Rotate Instructions

(1) ROLC A

This instruction cyclically rotates to the left the 9 bits of data obtained by linking the contents of the accumulator and the carry flag. To rotate only the contents of the accumulator, combine with the [LD CF, A.7] instruction.

Example : Rotates the contents of the accumulator as shown below.



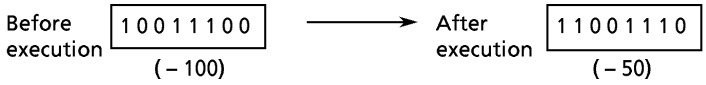

```
LD      A, 11111110B    ; Sets initial value to A
 ⋮
LD      CF, A.7         ; Rotation
ROLC    A
```

(2) RORC A

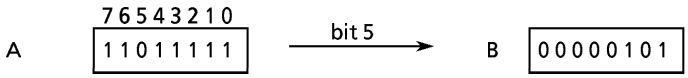
This instruction is used in combination with the [LD CF, A.7] instruction for arithmetic right shifts (refer to Example 1).

Example 1: Halves a signed 8-bit data stored in the accumulator.

```
LD      CF, A.7
RORC    A
```



Example 2: Searches "0" bit in the accumulator, and obtains the position of "0" bit in the B register. If there are two or more "0" bits, jumps to SERROR. If there are no "0" bits, jumps to SNON.

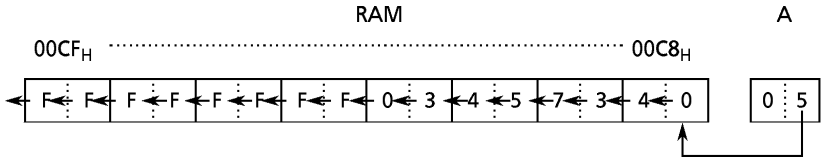


```
                LD      B, 11111111B
                CMP     A, B
                JRS     T, SNON
SSEARCH:        INC     B
                RORC    A
                JRS     T, SSEARCH
                CMP     A, 11111111B
                JRS     T, SERROR
                ⋮
SNON:           ————
                ⋮
SERROR:         ————
```

(3) ROLD A, (HL+)

This instruction is suitable for BCD data shift.

Example : Shifts in the BCD data stored in the lower nibble of the accumulator to RAM addresses $00C8_H$-$00CF_H$.




```
                LD      HL, 00C8H
                LD      C, 07H
SSHIFT :        ROLD    A, (HL+)
                DEC     C
                JRS     F, SSHIFT
```

1.8.4 Bit Manipulation and Flag Manipulation Instructions

(1) CLR (x). b

Bit set/clear/complement instructions have a bit test function. For example, [CLR (x).b] instruction loads the complement of the value of the bit specified by "b" of memory address "x" to the ZF and the JF, and then it clears the specified bit to "0".

Example : To clear bit 3 at RAM address $E9_H$ to "0" and call a subroutine if it is set to "1".

```
                CLR     (0E9H). 3
                JRS     T, SSKIP
                CALL    PSUB
SSKIP:          ————
```

(2) TEST (HL). A

This instruction is suitable for bit search.

Example : To search the contents of RAM address 0135_H starting from the most significant bit, and stores the positions of bits set to "1" to the accumulator. The accumulator is set to FF_H when there are no "1" bits.

```
                LD      HL, 0135H       ; HL  ← memory address
                LD      A, 7            ; A   ← bit number
SSEARCH:        TEST    (HL). A
                JRS     F, SEND
                DEC     A
                JRS     F,SSEARCH
SEND:           ————
```

1.8.5 Jump, Call, Return, Other Instructions

(1) JRS T, a

When performing an unconditional jump immediately after instructions such as transfer/exchange with which the JF is set to "1", and the jump destination is within the range of -14 to + 17 of the address where the jump instruction is located, the object code can be reduced by one byte by using the [JRS T, a] instruction instead of the [JR a] instruction.

Example : Unconditional jumps to SLABEL

```
①               LD      A, 5CH          ②               LD      A, 5CH
                JRS     T, SLABEL                       JR      SLABEL
                 ⋮                                       ⋮
SLABEL:         ————            SLABEL:         ————
        3 bytes                         4 bytes
```

(2) JP (PC + A) and CALL (PC + A)

These instructions are used for multiple direction branching in accordance with the accumulator contents.

Example : To jump as shown below for various accumulator contents; otherwise, to jump to SLABEL0.

When 01_H : jumps to SLABEL1
When 02_H : jumps to SLABEL2
When 03_H : jumps to SLABEL3
When 04_H : jumps to SLABEL4

```
                CMP     A, 05H
                JR      LT, SKIP
                LD      A, 00H
SKIP:           SHLC    A
                JP      (PC + A)
                DW      SLABEL0, SLABEL1, SLABEL2, SLABEL3, SLABEL4
```

(3) CALLV n and CALLP n

The TLCS-870 Series has three types of subroutine call instruction: CALL, CALLP and CALLV. Memory efficiency can be improved by using the CALLV instruction in subroutines called from 3 or more locations.

Example 1: Usage of CALLV instruction

```
                CSEG    REL             CSAMPLE
                  :
                CALLV   PSUB1
                  :
                CALLV   PSUB2
VSUB1:          .....................................  ; Subroutine PSUB1
                RET
                  :
VSUB2:          .....................................  ; Subroutine PSUB2
                RET
                  :
                CSEG    ABS             VECTOR TABLE (0FFC0H)
PSUB1:          DW      VSUB1
PSUB2:          DW      VSUB2
                  :
                END
```

Example 2: Usage of CALLP instruction

```
                CSEG    REL             CSAMPLE
                  :
                CALLP   PSUB1
                  :
                CALLP   PSUB2
                  :
                CSEG    ABS             PAGE CALL (0FF00H)
PSUB1:          .....................................  ; Subroutine PSUB1
                RET
                  :
PSUB2:          .....................................  ; Subroutine PSUB2
                RET
                  :
                END
```

1.9 OBJECT CODE TABLE

(1) DATA TRANSFER and EXCHANGE

① Transfer 8-bit register to register LD r_1, r_2 (Operation : $r_1 \leftarrow r_2$)

r_1 \ r_2	W	A	B	C	D	E	H	L
W	E9 59	59	EB 59	EA 59	ED 59	EC 59	EF 59	EE 59
A	51	50	53	52	55	54	57	56
B	E9 5B	5B	EB 5B	EA 5B	ED 5B	EC 5B	EF 5B	EE 5B
C	E9 5A	5A	EB 5A	EA 5A	ED 5A	EC 5A	EF 5A	EE 5A
D	E9 5D	5D	EB 5D	EA 5D	ED 5D	EC 5D	EF 5D	EE 5D
E	E9 5C	5C	EB 5C	EA 5C	ED 5C	EC 5C	EF 5C	EE 5C
H	E9 5F	5F	EB 5F	EA 5F	ED 5F	EC 5F	EF 5F	EE 5F
L	E9 5E	5E	EB 5E	EA 5E	ED 5E	EC 5E	EF 5E	EE 5E

② Transfer 16-bit register to register LD rr_1, rr_2 (Operation : $rr_1 \leftarrow rr_2$)

rr_1 \ rr_2	WA	BC	DE	HL	SP
WA	E8 14	E9 14	EA 14	EB 14	E8 FB
BC	E8 15	E9 15	EA 15	EB 15	E9 FB
DE	E8 16	E9 16	EA 16	EB 16	EA FB
HL	E8 17	E9 17	EA 17	EB 17	EB FB
SP	E8 FA	E9 FA	EA FA	EB FA	—

③ Transfer 8-bit memory to register LD r, mem (Operation : r ← mem)

r \ mem	(x)	(DE)	(HL)	(HL+)	(−HL)	(HL+d)	(HL+C)	(PC+A)
W	E0 xx 59	E2 59	E3 59	E6 59	E7 59	E4 dd 59	E5 59	E1 59
A	22 xx	E2 58	23	E6 58	E7 58	E4 dd 58	E5 58	E1 58
B	E0 xx 5B	E2 5B	E3 5B	E6 5B	E7 5B	E4 dd 5B	E5 5B	E1 5B
C	E0 xx 5A	E2 5A	E3 5A	E6 5A	E7 5A	E4 dd 5A	E5 5A	E1 5A
D	E0 xx 5D	E2 5D	E3 5D	E6 5D	E7 5D	E4 dd 5D	E5 5D	E1 5D
E	E0 xx 5C	E2 5C	E3 5C	E6 5C	E7 5C	E4 dd 5C	E5 5C	E1 5C
H	E0 xx 5F	E2 5F	E3 5F	—	E7 5F	E4 dd 5F	E5 5F	E1 5F
L	E0 xx 5E	E2 5E	E3 5E	—	E7 5E	E4 dd 5E	E5 5E	E1 5E

④ Transfer 16-bit memory to register LD rr, mem (Operation : rr ← mem)

rr \ mem	(x)	(DE)	(HL)	(HL+)	(−HL)	(HL+d)	(HL+C)	(PC+A)
WA	E0 xx 14	E2 14	E3 14	—	—	E4 dd 14	E5 14	E1 14
BC	E0 xx 15	E2 15	E3 15	—	—	E4 dd 15	E5 15	E1 15
DE	E0 xx 16	E2 16	E3 16	—	—	E4 dd 16	E5 16	E1 16
HL	E0 xx 17	E2 17	E3 17	—	—	E4 dd 17	E5 17	E1 17

⑤ Transfer 8-bit register to memory LD mem, r (Operation : mem ← r)

mem \ r	W	A	B	C	D	E	H	L
(x)	F0 xx 51	2A xx	F0 xx 53	F0 xx 52	F0 xx 55	F0 xx 54	F0 xx 57	F0 xx 56
(DE)	F2 51	F2 50	F2 53	F2 52	F2 55	F2 54	F2 57	F2 56
(HL)	F3 51	2B	F3 53	F3 52	F3 55	F3 54	F3 57	F3 56
(HL+)	F6 51	F6 50	F6 53	F6 52	F6 55	F6 54	—	—
(−HL)	F7 51	F7 50	F7 53	F7 52	F7 55	F7 54	F7 57	F7 56
(HL+d)	F4 dd 51	F4 dd 50	F4 dd 53	F4 dd 52	F4 dd 55	F4 dd 54	F4 dd 57	F4 dd 56

⑥ Transfer 16-bit register to memory LD mem, rr (Operation : mem ← rr)

mem \ rr	WA	BC	DE	HL
(x)	F0 xx 10	F0 xx 11	F0 xx 12	F0 xx 13
(DE)	F2 10	F2 11	F2 12	F2 13
(HL)	F3 10	F3 11	F3 12	F3 13
(HL+d)	F4 dd 10	F4 dd 11	F4 dd 12	F4 dd 13

⑦ Transfer 8-bit memory to memory LD mem1, mem2 (Operation : mem1 ← mem2)

mem1 \ mem2	(y)	(DE)	(HL)	(HL+d)	(HL+C)	(PC+A)
(x)	26 yy xx	E2 26 xx	E3 26 xx	E4 dd 26 xx	E5 26 xx	E1 26 xx
(HL)	E0 yy 27	E2 27	E3 27	E4 dd 27	E5 27	E1 27

⑧ Transfer immediate data to register (Operation : r ← n / rr ← mn)

Instr. \ r	W	A	B	C	D	E	H	L
LD r, n	31 nn	30 nn	33 nn	32 nn	35 nn	34 nn	37 nn	36 nn

Instr. \ rr	WA	BC	DE	HL	SP
LD rr, mn	14 nn mm	15 nn mm	16 nn mm	17 nn mm	FA nn mm

⑨ Change register bank (Operation : RBS ← n)

LD RBS, n	0F0n

⑩ Transfer immediate data to memory

Instr. \ mem	(x)	(DE)	(HL)	(HL+)	(−HL)	(HL+d)	Operation
CLR mem	2E xx	—	2F	—	—	—	mem ← 0
LD mem, n	2C xx nn	F2 2C nn	2D nn	F6 2C nn	F7 2C nn	F4 dd 2C nn	mem ← n
LDW mem,mn	24 xx nn mm	—	25 nn mm	—	—	—	mem ← mn

⑪ Exchange 8-bit register with register XCH r_1, r_2 (Operation : $r_1 \leftrightarrow r_2$)

r_1 \ r_2	W	A	B	C	D	E	H	L
W	E9 A9	E8 A9	EB A9	EA A9	ED A9	EC A9	EF A9	EE A9
A	E9 A8	E8 A8	EB A8	EA A8	ED A8	EC A8	EF A8	EE A8
B	E9 AB	E8 AB	EB AB	EA AB	ED AB	EC AB	EF AB	EE AB
C	E9 AA	E8 AA	EB AA	EA AA	ED AA	EC AA	EF AA	EE AA
D	E9 AD	E8 AD	EB AD	EA AD	ED AD	EC AD	EF AD	EE AD
E	E9 AC	E8 AC	EB AC	EA AC	ED AC	EC AC	EF AC	EE AC
H	E9 AF	E8 AF	EB AF	EA AF	ED AF	EC AF	EF AF	EE AF
L	E9 AE	E8 AE	EB AE	EA AE	ED AE	EC AE	EF AE	EE AE

⑫ Exchange 16-bit register with register XCH rr_1, rr_2 (Operation : $rr_1 \leftrightarrow rr_2$)

rr_1 \ rr_2	WA	BC	DE	HL
WA	E8 10	E9 10	EA 10	EB 10
BC	E8 11	E9 11	EA 11	EB 11
DE	E8 12	E9 12	EA 12	EB 12
HL	E8 13	E9 13	EA 13	EB 13

⑬ Exchange 8-bit register with memory XCH r, mem (Operation : r ↔ mem)

r \ mem	(x)	(DE)	(HL)	(HL+)	(−HL)	(HL+d)	(HL+C)	(PC+A)
W	E0 xx A9	E2 A9	E3 A9	E6 A9	E7 A9	E4 dd A9	E5 A9	E1 A9
A	E0 xx A8	E2 A8	E3 A8	E6 A8	E7 A8	E4 dd A8	E5 A8	E1 A8
B	E0 xx AB	E2 AB	E3 AB	E6 AB	E7 AB	E4 dd AB	E5 AB	E1 AB
C	E0 xx AA	E2 AA	E3 AA	E6 AA	E7 AA	E4 dd AA	E5 AA	E1 AA
D	E0 xx AD	E2 AD	E3 AD	E6 AD	E7 AD	E4 dd AD	E5 AD	E1 AD
E	E0 xx AC	E2 AC	E3 AC	E6 AC	E7 AC	E4 dd AC	E5 AC	E1 AC
H	E0 xx AF	E2 AF	E3 AF	—	E7 AF	E4 dd AF	E5 AF	E1 AF
L	E0 xx AE	E2 AE	E3 AE	—	E7 AE	E4 dd AE	E5 AE	E1 AE

(2) ARITHMETIC and LOGICAL OPERATION

① Operation 8-bit register to register

Instr. \ r	W	A	B	C	D	E	H	L
CMP A, r	E9 67	E8 67	EB 67	EA 67	ED 67	EC 67	EF 67	EE 67
ADD A, r	E9 61	E8 61	EB 61	EA 61	ED 61	EC 61	EF 61	EE 61
ADDC A, r	E9 60	E8 60	EB 60	EA 60	ED 60	EC 60	EF 60	EE 60
SUB A, r	E9 63	E8 63	EB 63	EA 63	ED 63	EC 63	EF 63	EE 63
SUBB A, r	E9 62	E8 62	EB 62	EA 62	ED 62	EC 62	EF 62	EE 62
AND A, r	E9 64	E8 64	EB 64	EA 64	ED 64	EC 64	EF 64	EE 64
OR A, r	E9 66	E8 66	EB 66	EA 66	ED 66	EC 66	EF 66	EE 66
XOR A, r	E9 65	E8 65	EB 65	EA 65	ED 65	EC 65	EF 65	EE 65

Instr. \ r	W	A	B	C	D	E	H	L
CMP r, A	E9 6F	—	EB 6F	EA 6F	ED 6F	EC 6F	EF 6F	EE 6F
ADD r, A	E9 69	—	EB 69	EA 69	ED 69	EC 69	EF 69	EE 69
ADDC r, A	E9 68	—	EB 68	EA 68	ED 68	EC 68	EF 68	EE 68
SUB r, A	E9 6B	—	EB 6B	EA 6B	ED 6B	EC 6B	EF 6B	EE 6B
SUBB r, A	E9 6A	—	EB 6A	EA 6A	ED 6A	EC 6A	EF 6A	EE 6A
AND r, A	E9 6C	—	EB 6C	EA 6C	ED 6C	EC 6C	EF 6C	EE 6C
OR r, A	E9 6E	—	EB 6E	EA 6E	ED 6E	EC 6E	EF 6E	EE 6E
XOR r, A	E9 6D	—	EB 6D	EA 6D	ED 6D	EC 6D	EF 6D	EE 6D

② Operation 16-bit register to 16-bit accumulator

Instr. \ rr	WA	BC	DE	HL
CMP WA, rr	E8 37	E9 37	EA 37	EB 37
ADD WA, rr	E8 31	E9 31	EA 31	EB 31
ADDC WA, rr	E8 30	E9 30	EA 30	EB 30
SUB WA, rr	E8 33	E9 33	EA 33	EB 33
SUBB WA, rr	E8 32	E9 32	EA 32	EB 32
AND WA, rr	E8 34	E9 34	EA 34	EB 34
OR WA, rr	E8 36	E9 36	EA 36	EB 36
XOR WA, rr	E8 35	E9 35	EA 35	EB 35

③ Operation memory to 8-bit accumulator

Instr. \ mem	(x)	(DE)	(HL)	(HL+)	(−HL)	(HL+d)	(HL+C)	(PC+A)
CMP A, mem	7F xx	E2 7F	E3 7F	E6 7F	E7 7F	E4 dd 7F	E5 7F	E1 7F
ADD A, mem	79 xx	E2 79	E3 79	E6 79	E7 79	E4 dd 79	E5 79	E1 79
ADDC A, mem	78 xx	E2 78	E3 78	E6 78	E7 78	E4 dd 78	E5 78	E1 78
SUB A, mem	7B xx	E2 7B	E3 7B	E6 7B	E7 7B	E4 dd 7B	E5 7B	E1 7B
SUBB A, mem	7A xx	E2 7A	E3 7A	E6 7A	E7 7A	E4 dd 7A	E5 7A	E1 7A
AND A, mem	7C xx	E2 7C	E3 7C	E6 7C	E7 7C	E4 dd 7C	E5 7C	E1 7C
OR A, mem	7E xx	E2 7E	E3 7E	E6 7E	E7 7E	E4 dd 7E	E5 7E	E1 7E
XOR A, mem	7D xx	E2 7D	E3 7D	E6 7D	E7 7D	E4 dd 7D	E5 7D	E1 7D

④ Operation 8-bit memory to memory

Instr. \ mem	(x)	(DE)	(HL)	(HL+)	(−HL)	(HL+d)	(HL+C)	(PC+A)
CMP mem, (HL)	E0 xx 67	E2 67	E3 67	—	—	E4 dd 67	E5 67	E1 67
ADD mem, (HL)	E0 xx 61	E2 61	E3 61	—	—	E4 dd 61	E5 61	E1 61
ADDC mem, (HL)	E0 xx 60	E2 60	E3 60	—	—	E4 dd 60	E5 60	E1 60
SUB mem, (HL)	E0 xx 63	E2 63	E3 63	—	—	E4 dd 63	E5 63	E1 63
SUBB mem, (HL)	E0 xx 62	E2 62	E3 62	—	—	E4 dd 62	E5 62	E1 62
AND mem, (HL)	E0 xx 64	E2 64	E3 64	—	—	E4 dd 64	E5 64	E1 64
OR mem, (HL)	E0 xx 66	E2 66	E3 66	—	—	E4 dd 66	E5 66	E1 66
XOR mem, (HL)	E0 xx 65	E2 65	E3 65	—	—	E4 dd 65	E5 65	E1 65

⑤ Operation immediate data to 8-bit register

Instr. \ r	W	A	B	C	D	E	H	L
CMP r, n	E9 77 nn	77 nn	EB 77 nn	EA 77 nn	ED 77 nn	EC 77 nn	EF 77 nn	EE 77 nn
ADD r, n	E9 71 nn	71 nn	EB 71 nn	EA 71 nn	ED 71 nn	EC 71 nn	EF 71 nn	EE 71 nn
ADDC r, n	E9 70 nn	70 nn	EB 70 nn	EA 70 nn	ED 70 nn	EC 70 nn	EF 70 nn	EE 70 nn
SUB r, n	E9 73 nn	73 nn	EB 73 nn	EA 73 nn	ED 73 nn	EC 73 nn	EF 73 nn	EE 73 nn
SUBB r, n	E9 72 nn	72 nn	EB 72 nn	EA 72 nn	ED 72 nn	EC 72 nn	EF 72 nn	EE 72 nn
AND r, n	E9 74 nn	74 nn	EB 74 nn	EA 74 nn	ED 74 nn	EC 74 nn	EF 74 nn	EE 74 nn
OR r, n	E9 76 nn	76 nn	EB 76 nn	EA 76 nn	ED 76 nn	EC 76 nn	EF 76 nn	EE 76 nn
XOR r, n	E9 75 nn	75 nn	EB 75 nn	EA 75 nn	ED 75 nn	EC 75 nn	EF 75 nn	EE 75 nn

⑥ Operation immediate data to 16-bit register

Instr. \ rr	WA	BC	DE	HL
CMP rr, mn	E8 3F nn mm	E9 3F nn mm	EA 3F nn mm	EB 3F nn mm
ADD rr, mn	E8 39 nn mm	E9 39 nn mm	EA 39 nn mm	EB 39 nn mm
ADDC rr, mn	E8 38 nn mm	E9 38 nn mm	EA 38 nn mm	EB 38 nn mm
SUB rr, mn	E8 3B nn mm	E9 3B nn mm	EA 3B nn mm	EB 3B nn mm
SUBB rr, mn	E8 3A nn mm	E9 3A nn mm	EA 3A nn mm	EB 3A nn mm
AND rr, mn	E8 3C nn mm	E9 3C nn mm	EA 3C nn mm	EB 3C nn mm
OR rr, mn	E8 3E nn mm	E9 3E nn mm	EA 3E nn mm	EB 3E nn mm
XOR rr, mn	E8 3D nn mm	E9 3D nn mm	EA 3D nn mm	EB 3D nn mm

⑦ Operation immediate data to 8-bit memory, and Masked compare 8-bit memory to accumulator

Instr. \ mem	(x)	(DE)	(HL)	(HL+)	(−HL)	(HL+d)	(HL+C)	(PC+A)
CMP mem, n	E0 xx 77 nn	E2 77 nn	E3 77 nn	E6 77 nn	E7 77 nn	E4 dd 77 nn	E5 77 nn	E1 77 nn
MCMP mem, n	E0 xx 2F nn	E2 2F nn	E3 2F nn	E6 2F nn	E7 2F nn	E4 dd 2F nn	E5 2F nn	E1 2F nn
ADD mem, n	E0 xx 71 nn	E2 71 nn	E3 71 nn	E6 71 nn	E7 71 nn	E4 dd 71 nn	E5 71 nn	E1 71 nn
ADDC mem, n	E0 xx 70 nn	E2 70 nn	E3 70 nn	E6 70 nn	E7 70 nn	E4 dd 70 nn	E5 70 nn	E1 70 nn
SUB mem, n	E0 xx 73 nn	E2 73 nn	E3 73 nn	E6 73 nn	E7 73 nn	E4 dd 73 nn	E5 73 nn	E1 73 nn
SUBB mem, n	E0 xx 72 nn	E2 72 nn	E3 72 nn	E6 72 nn	E7 72 nn	E4 dd 72 nn	E5 72 nn	E1 72 nn
AND mem, n	E0 xx 74 nn	E2 74 nn	E3 74 nn	E6 74 nn	E7 74 nn	E4 dd 74 nn	E5 74 nn	E1 74 nn
OR mem, n	E0 xx 76 nn	E2 76 nn	E3 76 nn	E6 76 nn	E7 76 nn	E4 dd 76 nn	E5 76 nn	E1 76 nn
XOR mem, n	E0 xx 75 nn	E2 75 nn	E3 75 nn	E6 75 nn	E7 75 nn	E4 dd 75 nn	E5 75 nn	E1 75 nn

⑧ Increment/decrement 8-bit register

Instr. \ r	W	A	B	C	D	E	H	L
INC r	61	60	63	62	65	64	67	66
DEC r	69	68	6B	6A	6D	6C	6F	6E

⑨ Increment/decrement 16-bit register

Instr. \ rr	WA	BC	DE	HL	SP
INC rr	10	11	12	13	—
DEC rr	18	19	1A	1B	—

⑩ Increment/decrement 8-bit memory

Instr. \ mem	(x)	(DE)	(HL)	(HL+)	(−HL)	(HL+d)	(HL+C)	(PC+A)
INC mem	20 xx	E2 20	21	E6 20	E7 20	E4 dd 20	E5 20	E1 20
DEC mem	28 xx	E2 28	29	E6 28	E7 28	E4 dd 28	E5 28	E1 28

⑪ Decimal adjust 8-bit register

Instr. \ r	W	A	B	C	D	E	H	L
DAA r	E9 0A	0A	EB 0A	EA 0A	ED 0A	EC 0A	EF 0A	EE 0A
DAS r	E9 0B	0B	EB 0B	EA 0B	ED 0B	EC 0B	EF 0B	EE 0B

⑫ Multiply 8-bit register and 8-bit register

Instr. \ $r_1 \times r_2$	W×A	B×C	D×E	H×L
MUL r_1, r_2	02	E9 02	EA 02	EB 02

⑬ Divide 16-bit register by C register

Instr. \ rr	WA	BC	DE	HL
DIV rr, C	03	—	EA 03	EB 03

(3) SHIFT, ROTATE and NIBBLE MANIPULATION

① Shift 8-bit register, and Rotate 8-bit register through carry flag

Instr. \ r	W	A	B	C	D	E	H	L
SHLC r	E9 1C	1C	EB 1C	EA 1C	ED 1C	EC 1C	EF 1C	EE 1C
SHRC r	E9 1D	1D	EB 1D	EA 1D	ED 1D	EC 1D	EF 1D	EE 1D
ROLC r	E9 1E	1E	EB 1E	EA 1E	ED 1E	EC 1E	EF 1E	EE 1E
RORC r	E9 1F	1F	EB 1F	EA 1F	ED 1F	EC 1F	EF 1F	EE 1F

② Swap nibbles within 8-bit register

Instr. \ r	W	A	B	C	D	E	H	L
SWAP r	E9 01	01	EB 01	EA 01	ED 01	EC 01	EF 01	EE 01

③ Rotate nibbles 8-bit memory through lower nibble in accumulator

Instr. \ mem	(x)	(DE)	(HL)	(HL+)	(−HL)	(HL+d)	(HL+C)
ROLD A, mem	E0 xx 08	E2 08	E3 08	E6 08	E7 08	E4 dd 08	E5 08
RORD A, mem	E0 xx 09	E2 09	E3 09	E6 09	E7 09	E4 dd 09	E5 09

(4) BIT MANIPULATION and FLAG MANIPULATION

① Test direct bit in register TEST r. b

r \ b	7	6	5	4	3	2	1	0
W	E9 DF	E9 DE	E9 DD	E9 DC	E9 DB	E9 DA	E9 D9	E9 D8
A	E8 DF	E8 DE	E8 DD	E8 DC	E8 DB	E8 DA	E8 D9	E8 D8
B	EB DF	EB DE	EB DD	EB DC	EB DB	EB DA	EB D9	EB D8
C	EA DF	EA DE	EA DD	EA DC	EA DB	EA DA	EA D9	EA D8
D	ED DF	ED DE	ED DD	ED DC	ED DB	ED DA	ED D9	ED D8
E	EC DF	EC DE	EC DD	EC DC	EC DB	EC DA	EC D9	EC D8
H	EF DF	EF DE	EF DD	EF DC	EF DB	EF DA	EF D9	EF D8
L	EE DF	EE DE	EE DD	EE DC	EE DB	EE DA	EE D9	EE D8

② Test direct bit in memory TEST mem. b

mem \ b	7	6	5	4	3	2	1	0
(x)	DF xx	DE xx	DD xx	DC xx	DB xx	DA xx	D9 xx	D8 xx
(DE)	E2 DF	E2 DE	E2 DD	E2 DC	E2 DB	E2 DA	E2 D9	E2 D8
(HL)	E3 DF	E3 DE	E3 DD	E3 DC	E3 DB	E3 DA	E3 D9	E3 D8
(HL+)	E6 DF	E6 DE	E6 DD	E6 DC	E6 DB	E6 DA	E6 D9	E6 D8
(−HL)	E7 DF	E7 DE	E7 DD	E7 DC	E7 DB	E7 DA	E7 D9	E7 D8
(HL+d)	E4 dd DF	E4 dd DE	E4 dd DD	E4 dd DC	E4 dd DB	E4 dd DA	E4 dd D9	E4 dd D8
(HL+C)	E5 DF	E5 DE	E5 DD	E5 DC	E5 DB	E5 DA	E5 D9	E5 D8
(PC+A)	E1 DF	E1 DE	E1 DD	E1 DC	E1 DB	E1 DA	E1 D9	E1 D8

③ Test register-indirect bit in memory TEST mem. r

mem \ r	W	A	B	C	D	E	H	L
(DE)	E9 9E	E8 9E	EB 9E	EA 9E	ED 9E	EC 9E	EF 9E	EE 9E
(HL)	E9 9F	E8 9F	EB 9F	EA 9F	ED 9F	EC 9F	EF 9F	EE 9F

④ Clear direct bit in register CLR r. b

r \ b	7	6	5	4	3	2	1	0
W	E9 4F	E9 4E	E9 4D	E9 4C	E9 4B	E9 4A	E9 49	E9 48
A	E8 4F	E8 4E	E8 4D	E8 4C	E8 4B	E8 4A	E8 49	E8 48
B	EB 4F	EB 4E	EB 4D	EB 4C	EB 4B	EB 4A	EB 49	EB 48
C	EA 4F	EA 4E	EA 4D	EA 4C	EA 4B	EA 4A	EA 49	EA 48
D	ED 4F	ED 4E	ED 4D	ED 4C	ED 4B	ED 4A	ED 49	ED 48
E	EC 4F	EC 4E	EC 4D	EC 4C	EC 4B	EC 4A	EC 49	EC 48
H	EF 4F	EF 4E	EF 4D	EF 4C	EF 4B	EF 4A	EF 49	EF 48
L	EE 4F	EE 4E	EE 4D	EE 4C	EE 4B	EE 4A	EE 49	EE 48

⑤ Clear direct bit in memory CLR mem. b

mem \ b	7	6	5	4	3	2	1	0
(x)	4F xx	4E xx	4D xx	4C xx	4B xx	4A xx	49 xx	48 xx
(DE)	E2 4F	E2 4E	E2 4D	E2 4C	E2 4B	E2 4A	E2 49	E2 48
(HL)	E3 4F	E3 4E	E3 4D	E3 4C	E3 4B	E3 4A	E3 49	E3 48
(HL+)	E6 4F	E6 4E	E6 4D	E6 4C	E6 4B	E6 4A	E6 49	E6 48
(−HL)	E7 4F	E7 4E	E7 4D	E7 4C	E7 4B	E7 4A	E7 49	E7 48
(HL+d)	E4 dd 4F	E4 dd 4E	E4 dd 4D	E4 dd 4C	E4 dd 4B	E4 dd 4A	E4 dd 49	E4 dd 48
(HL+C)	E5 4F	E5 4E	E5 4D	E5 4C	E5 4B	E5 4A	E5 49	E5 48
(PC+A)	E1 4F	E1 4E	E1 4D	E1 4C	E1 4B	E1 4A	E1 49	E1 48

⑥ Clear register-indirect bit in memory CLR mem. r

mem \ r	W	A	B	C	D	E	H	L
(DE)	E9 8A	E8 8A	EB 8A	EA 8A	ED 8A	EC 8A	EF 8A	EE 8A
(HL)	E9 8B	E8 8B	EB 8B	EA 8B	ED 8B	EC 8B	EF 8B	EE 8B

⑦ Set direct bit in register SET r. b

r \ b	7	6	5	4	3	2	1	0
W	E9 47	E9 46	E9 45	E9 44	E9 43	E9 42	E9 41	E9 40
A	E8 47	E8 46	E8 45	E8 44	E8 43	E8 42	E8 41	E8 40
B	EB 47	EB 46	EB 45	EB 44	EB 43	EB 42	EB 41	EB 40
C	EA 47	EA 46	EA 45	EA 44	EA 43	EA 42	EA 41	EA 40
D	ED 47	ED 46	ED 45	ED 44	ED 43	ED 42	ED 41	ED 40
E	EC 47	EC 46	EC 45	EC 44	EC 43	EC 42	EC 41	EC 40
H	EF 47	EF 46	EF 45	EF 44	EF 43	EF 42	EF 41	EF 40
L	EE 47	EE 46	EE 45	EE 44	EE 43	EE 42	EE 41	EE 40

⑧ Set direct bit in memory SET mem. b

mem \ b	7	6	5	4	3	2	1	0
(x)	47 xx	46 xx	45 xx	44 xx	43 xx	42 xx	41 xx	40 xx
(DE)	E2 47	E2 46	E2 45	E2 44	E2 43	E2 42	E2 41	E2 40
(HL)	E3 47	E3 46	E3 45	E3 44	E3 43	E3 42	E3 41	E3 40
(HL+)	E6 47	E6 46	E6 45	E6 44	E6 43	E6 42	E6 41	E6 40
(−HL)	E7 47	E7 46	E7 45	E7 44	E7 43	E7 42	E7 41	E7 40
(HL+d)	E4 dd 47	E4 dd 46	E4 dd 45	E4 dd 44	E4 dd 43	E4 dd 42	E4 dd 41	E4 dd 40
(HL+C)	E5 47	E5 46	E5 45	E5 44	E5 43	E5 42	E5 41	E5 40
(PC+A)	E1 47	E1 46	E1 45	E1 44	E1 43	E1 42	E1 41	E1 40

⑨ Set register-indirect bit in memory SET mem. r

mem \ r	W	A	B	C	D	E	H	L
(DE)	E9 82	E8 82	EB 82	EA 82	ED 82	EC 82	EF 82	EE 82
(HL)	E9 83	E8 83	EB 83	EA 83	ED 83	EC 83	EF 83	EE 83

⑩ Complement direct bit in register CPL r. b

r \ b	7	6	5	4	3	2	1	0
W	E9 C7	E9 C6	E9 C5	E9 C4	E9 C3	E9 C2	E9 C1	E9 C0
A	E8 C7	E8 C6	E8 C5	E8 C4	E8 C3	E8 C2	E8 C1	E8 C0
B	EB C7	EB C6	EB C5	EB C4	EB C3	EB C2	EB C1	EB C0
C	EA C7	EA C6	EA C5	EA C4	EA C3	EA C2	EA C1	EA C0
D	ED C7	ED C6	ED C5	ED C4	ED C3	ED C2	ED C1	ED C0
E	EC C7	EC C6	EC C5	EC C4	EC C3	EC C2	EC C1	EC C0
H	EF C7	EF C6	EF C5	EF C4	EF C3	EF C2	EF C1	EF C0
L	EE C7	EE C6	EE C5	EE C4	EE C3	EE C2	EE C1	EE C0

⑪ Complement direct bit in memory CPL mem. b

mem \ b	7	6	5	4	3	2	1	0
(x)	E0 xx C7	E0 xx C6	E0 xx C5	E0 xx C4	E0 xx C3	E0 xx C2	E0 xx C1	E0 xx C0
(DE)	E2 C7	E2 C6	E2 C5	E2 C4	E2 C3	E2 C2	E2 C1	E2 C0
(HL)	E3 C7	E3 C6	E3 C5	E3 C4	E3 C3	E3 C2	E3 C1	E3 C0
(HL+)	E6 C7	E6 C6	E6 C5	E6 C4	E6 C3	E6 C2	E6 C1	E6 C0
(−HL)	E7 C7	E7 C6	E7 C5	E7 C4	E7 C3	E7 C2	E7 C1	E7 C0
(HL+d)	E4 dd C7	E4 dd C6	E4 dd C5	E4 dd C4	E4 dd C3	E4 dd C2	E4 dd C1	E4 dd C0
(HL+C)	E5 C7	E5 C6	E5 C5	E5 C4	E5 C3	E5 C2	E5 C1	E5 C0
(PC+A)	E1 C7	E1 C6	E1 C5	E1 C4	E1 C3	E1 C2	E1 C1	E1 C0

⑫ Complement register-indirect bit in memory CPL mem. r

mem \ r	W	A	B	C	D	E	H	L
(DE)	E9 92	E8 92	EB 92	EA 92	ED 92	EC 92	EF 92	EE 92
(HL)	E9 93	E8 93	EB 93	EA 93	ED 93	EC 93	EF 93	EE 93

⑬ Load direct bit in register to carry flag LD CF, r. b

r \ b	7	6	5	4	3	2	1	0
W	E9 DF	E9 DE	E9 DD	E9 DC	E9 DB	E9 DA	E9 D9	E9 D8
A	E8 DF	E8 DE	E8 DD	E8 DC	E8 DB	E8 DA	E8 D9	E8 D8
B	EB DF	EB DE	EB DD	EB DC	EB DB	EB DA	EB D9	EB D8
C	EA DF	EA DE	EA DD	EA DC	EA DB	EA DA	EA D9	EA D8
D	ED DF	ED DE	ED DD	ED DC	ED DB	ED DA	ED D9	ED D8
E	EC DF	EC DE	EC DD	EC DC	EC DB	EC DA	EC D9	EC D8
H	EF DF	EF DE	EF DD	EF DC	EF DB	EF DA	EF D9	EF D8
L	EE DF	EE DE	EE DD	EE DC	EE DB	EE DA	EE D9	EE D8

⑭ Load direct bit in memory to carry flag LD CF, mem. b

mem \ b	7	6	5	4	3	2	1	0
(x)	DF xx	DE xx	DD xx	DC xx	DB xx	DA xx	D9 xx	D8 xx
(DE)	E2 DF	E2 DE	E2 DD	E2 DC	E2 DB	E2 DA	E2 D9	E2 D8
(HL)	E3 DF	E3 DE	E3 DD	E3 DC	E3 DB	E3 DA	E3 D9	E3 D8
(HL+)	E6 DF	E6 DE	E6 DD	E6 DC	E6 DB	E6 DA	E6 D9	E6 D8
(−HL)	E7 DF	E7 DE	E7 DD	E7 DC	E7 DB	E7 DA	E7 D9	E7 D8
(HL+d)	E4 dd DF	E4 dd DE	E4 dd DD	E4 dd DC	E4 dd DB	E4 dd DA	E4 dd D9	E4 dd D8
(HL+C)	E5 DF	E5 DE	E5 DD	E5 DC	E5 DB	E5 DA	E5 D9	E5 D8
(PC+A)	E1 DF	E1 DE	E1 DD	E1 DC	E1 DB	E1 DA	E1 D9	E1 D8

⑮ Load register-indirect bit in memory to carry flag LD CF, mem. r

mem \ r	W	A	B	C	D	E	H	L
(DE)	E9 9E	E8 9E	EB 9E	EA 9E	ED 9E	EC 9E	EF 9E	EE 9E
(HL)	E9 9F	E8 9F	EB 9F	EA 9F	ED 9F	EC 9F	EF 9F	EE 9F

⑯ Store carry flag to direct bit in register LD r. b, CF

r \ b	7	6	5	4	3	2	1	0
W	E9 CF	E9 CE	E9 CD	E9 CC	E9 CB	E9 CA	E9 C9	E9 C8
A	E8 CF	E8 CE	E8 CD	E8 CC	E8 CB	E8 CA	E8 C9	E8 C8
B	EB CF	EB CE	EB CD	EB CC	EB CB	EB CA	EB C9	EB C8
C	EA CF	EA CE	EA CD	EA CC	EA CB	EA CA	EA C9	EA C8
D	ED CF	ED CE	ED CD	ED CC	ED CB	ED CA	ED C9	ED C8
E	EC CF	EC CE	EC CD	EC CC	EC CB	EC CA	EC C9	EC C8
H	EF CF	EF CE	EF CD	EF CC	EF CB	EF CA	EF C9	EF C8
L	EE CF	EE CE	EE CD	EE CC	EE CB	EE CA	EE C9	EE C8

⑰ Store carry flag to direct bit in memory LD mem. b, CF

mem \ b	7	6	5	4	3	2	1	0
(x)	E0 xx CF	E0 xx CE	E0 xx CD	E0 xx CC	E0 xx CB	E0 xx CA	E0 xx C9	E0 xx C8
(DE)	E2 CF	E2 CE	E2 CD	E2 CC	E2 CB	E2 CA	E2 C9	E2 C8
(HL)	E3 CF	E3 CE	E3 CD	E3 CC	E3 CB	E3 CA	E3 C9	E3 C8
(HL+)	E6 CF	E6 CE	E6 CD	E6 CC	E6 CB	E6 CA	E6 C9	E6 C8
(−HL)	E7 CF	E7 CE	E7 CD	E7 CC	E7 CB	E7 CA	E7 C9	E7 C8
(HL+d)	E4 dd CF	E4 dd CE	E4 dd CD	E4 dd CC	E4 dd CB	E4 dd CA	E4 dd C9	E4 dd C8
(HL+C)	E5 CF	E5 CE	E5 CD	E5 CC	E5 CB	E5 CA	E5 C9	E5 C8
(PC+A)	E1 CF	E1 CE	E1 CD	E1 CC	E1 CB	E1 CA	E1 C9	E1 C8

⑱ Store carry flag to register-indirect bit in memory LD mem. r, CF

mem \ r	W	A	B	C	D	E	H	L
(DE)	E9 9A	E8 9A	EB 9A	EA 9A	ED 9A	EC 9A	EF 9A	EE 9A
(HL)	E9 9B	E8 9B	EB 9B	EA 9B	ED 9B	EC 9B	EF 9B	EE 9B

⑲ Logical Exclusive-OR direct bit in register to carry flag XOR CF, r. b

r \ b	7	6	5	4	3	2	1	0
W	E9 D7	E9 D6	E9 D5	E9 D4	E9 D3	E9 D2	E9 D1	E9 D0
A	E8 D7	E8 D6	E8 D5	E8 D4	E8 D3	E8 D2	E8 D1	E8 D0
B	EB D7	EB D6	EB D5	EB D4	EB D3	EB D2	EB D1	EB D0
C	EA D7	EA D6	EA D5	EA D4	EA D3	EA D2	EA D1	EA D0
D	ED D7	ED D6	ED D5	ED D4	ED D3	ED D2	ED D1	ED D0
E	EC D7	EC D6	EC D5	EC D4	EC D3	EC D2	EC D1	EC D0
H	EF D7	EF D6	EF D5	EF D4	EF D3	EF D2	EF D1	EF D0
L	EE D7	EE D6	EE D5	EE D4	EE D3	EE D2	EE D1	EE D0

⑳ Logical Exclusive-OR direct bit in memory to carry flag XOR CF, mem. b

mem \ b	7	6	5	4	3	2	1	0
(x)	E0 xx D7	E0 xx D6	E0 xx D5	E0 xx D4	E0 xx D3	E0 xx D2	E0 xx D1	E0 xx D0
(DE)	E2 D7	E2 D6	E2 D5	E2 D4	E2 D3	E2 D2	E2 D1	E2 D0
(HL)	E3 D7	E3 D6	E3 D5	E3 D4	E3 D3	E3 D2	E3 D1	E3 D0
(HL+)	E6 D7	E6 D6	E6 D5	E6 D4	E6 D3	E6 D2	E6 D1	E6 D0
(−HL)	E7 D7	E7 D6	E7 D5	E7 D4	E7 D3	E7 D2	E7 D1	E7 D0
(HL+d)	E4 dd D7	E4 dd D6	E4 dd D5	E4 dd D4	E4 dd D3	E4 dd D2	E4 dd D1	E4 dd D0
(HL+C)	E5 D7	E5 D6	E5 D5	E5 D4	E5 D3	E5 D2	E5 D1	E5 D0
(PC+A)	E1 D7	E1 D6	E1 D5	E1 D4	E1 D3	E1 D2	E1 D1	E1 D0

㉑ Manipulate carry flag

CLR CF	0C
SET CF	0D
CPL CF	0E

㉒ Manipulate interrupt master enable flag

DI	48 3A
EI	40 3A

(5) JUMP

① Short relative jump

JRS T, a	d=0~+15	$8d_{3-0}$
	d=−16~−1	$9d_{3-0}$
JRS F, a	d=0~+15	Ad_{3-0}
	d=−16~−1	Bd_{3-0}

Note 1: a = $ + 2 + d, d = − 16~ + 15
Note 2: d_{3-0} means lower 4 bits of displacement "d".

d	−16	−15	−14	−13	−12	.....	+12	+13	+14	+15
d_{3-0}	0	1	2	3	4	.....	C	D	E	F

② Relative jump

Note: a = $ + 2 + d, d = − 128~ + 127

JR T, a	D6 dd
JR F, a	D7 dd
JR EQ, a	D0 dd
JR Z, a	
JR NE, a	D1 dd
JR NZ, a	
JR CS, a	D2 dd
JR LT, a	
JR CC, a	D3 dd
JR GE, a	
JR LE, a	D4 dd
JR GT, a	D5 dd
JR a	FB dd

③ Absolute long jump

JP mn	FE nn mm

④ Register-indirect jump

Instr. \ rr	WA	BC	DE	HL
JP rr	E8 FE	E9 FE	EA FE	EB FE

⑤ Memory-indirect jump

Instr. \ mem	(x)	(DE)	(HL)	(HL+d)	(HL+C)	(PC+A)
JP mem	E0 xx FE	E2 FE	E3 FE	E4 dd FE	E5 FE	E1 FE

(6) CALL, RETURN and OTHERS

① Vector call

CALLV n	Cn

② Paged call

CALLP n	FD nn

③ Absolute long call

CALL mn	FC nn mm

④ Register-indirect call

Instr. \ rr	WA	BC	DE	HL
CALL rr	E8 FC	E9 FC	EA FC	EB FC

⑤ Memory-indirect call

Instr. \ mem	(x)	(DE)	(HL)	(HL+d)	(HL+C)	(PC+A)
CALL mem	E0 xx FC	E2 FC	E3 FC	E4 dd FC	E5 FC	E1 FC

⑥ Return from subroutine

RET	05

⑦ Return from interrupt service routine

RETI	04	*for Maskable interrupt*
RETN	E8 04	*for Non-maskable interrupt*

⑧ Push down, and Pop up

Instr. \ rr	WA	BC	DE	HL	PSW
PUSH rr	E8 07	E9 07	EA 07	EB 07	07
POP rr	E8 06	E9 06	EA 06	EB 06	06

Note: PSW is an 8-bit register.

⑨ Software interrupt

SWI	FF

⑩ No operation

NOP	00

1.10 INSTRUCTION TABLE

ADD	(-HL),n	4-28	4-70	4-95
ADD	(HL+),n	4-28	4-70	4-95
ADD	(HL+C),(HL)	4-27	4-70	4-94
ADD	(HL+C),n	4-29	4-70	4-95
ADD	(HL+d),(HL)	4-27	4-70	4-94
ADD	(HL+d),n	4-28	4-70	4-95
ADD	(PC+A),(HL)	4-28	4-70	4-94
ADD	(PC+A),n	4-29	4-70	4-95
ADD	(pp),(HL)	4-27	4-70	4-94
ADD	(pp),n	4-28	4-70	4-95
ADD	(x),(HL)	4-27	4-70	4-94
ADD	(x),n	4-28	4-70	4-95
ADD	A,(-HL)	4-27	4-70	4-94
ADD	A,(HL+)	4-27	4-70	4-94
ADD	A,(HL+C)	4-27	4-70	4-94
ADD	A,(HL+d)	4-27	4-70	4-94
ADD	A,(PC+A)	4-27	4-70	4-94
ADD	A,(pp)	4-26	4-70	4-94
ADD	A,(x)	4-26	4-70	4-94
ADD	A,g	4-26	4-70	4-93
ADD	A,n	4-28	4-70	4-94
ADD	g,A	4-26	4-70	4-93
ADD	g,n	4-28	4-70	4-94
ADD	gg,mn	4-28	4-70	4-95
ADD	WA,gg	4-26	4-70	4-93
ADDC	(-HL),n	4-31	4-71	4-95
ADDC	(HL+),n	4-31	4-71	4-95
ADDC	(HL+C),(HL)	4-30	4-71	4-94
ADDC	(HL+C),n	4-31	4-71	4-95
ADDC	(HL+d),(HL)	4-30	4-71	4-94
ADDC	(HL+d),n	4-31	4-71	4-95
ADDC	(PC+A),(HL)	4-30	4-71	4-94
ADDC	(PC+A),n	4-31	4-71	4-95
ADDC	(pp),(HL)	4-30	4-71	4-94
ADDC	(pp),n	4-31	4-71	4-95
ADDC	(x),(HL)	4-30	4-71	4-94
ADDC	(x),n	4-31	4-71	4-95
ADDC	A,(-HL)	4-29	4-70	4-94
ADDC	A,(HL+)	4-29	4-70	4-94
ADDC	A,(HL+C)	4-30	4-70	4-94
ADDC	A,(HL+d)	4-29	4-70	4-94
ADDC	A,(PC+A)	4-30	4-70	4-94
ADDC	A,(pp)	4-29	4-70	4-94
ADDC	A,(x)	4-29	4-70	4-94
ADDC	A,g	4-29	4-70	4-93
ADDC	A,n	4-30	4-71	4-94
ADDC	g,A	4-29	4-70	4-93

ADDC	g,n	4-30	4-71	4-94
ADDC	gg,mn	4-31	4-71	4-95
ADDC	WA,gg	4-29	4-70	4-93
AND	(-HL),n	4-39	4-73	4-95
AND	(HL+),n	4-39	4-73	4-95
AND	(HL+C),(HL)	4-38	4-72	4-94
AND	(HL+C),n	4-40	4-73	4-95
AND	(HL+d),(HL)	4-38	4-72	4-94
AND	(HL+d),n	4-40	4-73	4-95
AND	(PC+A),(HL)	4-39	4-72	4-94
AND	(PC+A),n	4-40	4-73	4-95
AND	(pp),(HL)	4-38	4-72	4-94
AND	(pp),n	4-39	4-73	4-95
AND	(x),(HL)	4-38	4-72	4-94
AND	(x),n	4-39	4-73	4-95
AND	A,(-HL)	4-38	4-72	4-94
AND	A,(HL+)	4-38	4-72	4-94
AND	A,(HL+C)	4-38	4-72	4-94
AND	A,(HL+d)	4-38	4-72	4-94
AND	A,(PC+A)	4-38	4-72	4-94
AND	A,(pp)	4-37	4-72	4-94
AND	A,(x)	4-37	4-72	4-94
AND	A,g	4-37	4-72	4-93
AND	A,n	4-39	4-72	4-94
AND	g,A	4-37	4-72	4-93
AND	g,n	4-39	4-72	4-94
AND	gg,mn	4-39	4-72	4-95
AND	WA,gg	4-37	4-72	4-93
CALL	(HL+C)	4-65	4-78	4-105
CALL	(HL+d)	4-65	4-78	4-105
CALL	(PC+A)	4-65	4-78	4-105
CALL	(pp)	4-64	4-78	4-105
CALL	(x)	4-64	4-78	4-105
CALL	gg	4-64	4-78	4-105
CALL	mn	4-64	4-78	4-105
CALLP	n	4-64	4-78	4-105
CALLV	n	4-64	4-78	4-105
CLR	(-HL).b	4-54	4-76	4-98
CLR	(HL)	4-18	4-68	4-91
CLR	(HL+).b	4-54	4-76	4-98
CLR	(HL+C).b	4-54	4-76	4-98
CLR	(HL+d).b	4-54	4-76	4-98
CLR	(PC+A).b	4-54	4-76	4-98
CLR	(pp).b	4-54	4-76	4-98
CLR	(pp).g	4-54	4-76	4-98
CLR	(x)	4-18	4-68	4-91
CLR	(x).b	4-54	4-76	4-98

CLR	CF	4-60	4-77	4-103
CLR	g.b	4-54	4-76	4-98
CMP	(-HL),n	4-23	4-69	4-95
CMP	(HL+),n	4-23	4-69	4-95
CMP	(HL+C),(HL)	4-22	4-69	4-94
CMP	(HL+C),n	4-23	4-69	4-95
CMP	(HL+d),(HL)	4-22	4-69	4-94
CMP	(HL+d),n	4-23	4-69	4-95
CMP	(PC+A),(HL)	4-22	4-69	4-94
CMP	(PC+A),n	4-23	4-69	4-95
CMP	(pp),(HL)	4-22	4-69	4-94
CMP	(pp),n	4-22	4-69	4-95
CMP	(x),(HL)	4-21	4-69	4-94
CMP	(x),n	4-22	4-69	4-94
CMP	A,(-HL)	4-21	4-69	4-94
CMP	A,(HL+)	4-21	4-69	4-94
CMP	A,(HL+C)	4-21	4-69	4-94
CMP	A,(HL+d)	4-21	4-69	4-94
CMP	A,(PC+A)	4-21	4-69	4-94
CMP	A,(pp)	4-21	4-69	4-94
CMP	A,(x)	4-21	4-69	4-94
CMP	A,g	4-21	4-69	4-93
CMP	A,n	4-22	4-69	4-94
CMP	g,A	4-21	4-69	4-93
CMP	g,n	4-22	4-69	4-94
CMP	gg,mn	4-22	4-69	4-95
CMP	WA,gg	4-21	4-69	4-93
CPL	(-HL).b	4-56	4-76	4-100
CPL	(HL+).b	4-56	4-76	4-100
CPL	(HL+C).b	4-56	4-76	4-100
CPL	(HL+d).b	4-56	4-76	4-100
CPL	(PC+A).b	4-56	4-76	4-100
CPL	(pp).b	4-56	4-76	4-100
CPL	(pp).g	4-56	4-76	4-100
CPL	(x).b	4-56	4-76	4-100
CPL	CF	4-60	4-77	4-103
CPL	g.b	4-56	4-76	4-100
DAA	A	4-47	4-74	4-96
DAA	g	4-47	4-74	4-96
DAS	A	4-47	4-74	4-96
DAS	g	4-47	4-74	4-96
DEC	(-HL)	4-26	4-70	4-95
DEC	(DE)	4-25	4-70	4-95
DEC	(HL)	4-25	4-70	4-95
DEC	(HL+)	4-26	4-70	4-95
DEC	(HL+C)	4-26	4-70	4-95
DEC	(HL+d)	4-26	4-70	4-95

DEC	(PC+A)	4-26	4-70	4-95
DEC	(x)	4-25	4-70	4-95
DEC	r	4-25	4-70	4-95
DEC	rr	4-25	4-70	4-95
DI		4-60	4-77	4-103
DIV	DE,C	4-48	4-74	4-96
DIV	HL,C	4-48	4-74	4-96
DIV	WA,C	4-48	4-74	4-96
EI		4-60	4-77	4-103
INC	(-HL)	4-25	4-69	4-95
INC	(DE)	4-24	4-69	4-95
INC	(HL)	4-24	4-69	4-95
INC	(HL+)	4-25	4-69	4-95
INC	(HL+C)	4-25	4-69	4-95
INC	(HL+d)	4-25	4-69	4-95
INC	(PC+A)	4-25	4-69	4-95
INC	(x)	4-24	4-69	4-95
INC	r	4-24	4-69	4-95
INC	rr	4-24	4-69	4-95
JP	(HL+C)	4-63	4-77	4-104
JP	(HL+d)	4-63	4-77	4-104
JP	(PC+A)	4-63	4-77	4-104
JP	(pp)	4-62	4-77	4-104
JP	(x)	4-62	4-77	4-104
JP	gg	4-62	4-77	4-104
JP	mn	4-62	4-77	4-104
JR	a	4-62	4-77	4-104
JR	CC,a	4-62	4-77	4-104
JR	CS,a	4-61	4-77	4-104
JR	EQ,a	4-61	4-77	4-104
JR	F,a	4-61	4-77	4-104
JR	GE,a	4-62	4-77	4-104
JR	GT,a	4-62	4-77	4-104
JR	LE,a	4-62	4-77	4-104
JR	LT,a	4-61	4-77	4-104
JR	NE,a	4-61	4-77	4-104
JR	NZ,a	4-61	4-77	4-104
JR	T,a	4-61	4-77	4-104
JR	Z,a	4-61	4-77	4-104
JRS	F,a	4-61	4-77	4-104
JRS	T,a	4-61	4-77	4-104
LD	(-HL),n	4-19	4-68	4-91
LD	(-HL),r	4-16	4-67	4-91
LD	(-HL).b,CF	4-58	4-77	4-102
LD	(DE),n	4-18	4-68	4-91
LD	(HL),(HL+C)	4-18	4-67	4-91
LD	(HL),(HL+d)	4-17	4-67	4-91

LD	(HL),(PC+A)	4-18	4-67	4-91
LD	(HL),(pp)	4-17	4-67	4-91
LD	(HL),(x)	4-17	4-67	4-91
LD	(HL),A	4-16	4-67	4-91
LD	(HL),n	4-18	4-68	4-91
LD	(HL+),n	4-19	4-68	4-91
LD	(HL+),r	4-16	4-67	4-91
LD	(HL+).b,CF	4-58	4-77	4-102
LD	(HL+C).b,CF	4-58	4-77	4-102
LD	(HL+d),n	4-19	4-68	4-91
LD	(HL+d),r	4-16	4-67	4-91
LD	(HL+d),rr	4-16	4-67	4-91
LD	(HL+d).b,CF	4-58	4-77	4-102
LD	(PC+A).b,CF	4-58	4-77	4-102
LD	(pp),r	4-16	4-67	4-91
LD	(pp),rr	4-16	4-67	4-91
LD	(pp).b,CF	4-58	4-77	4-102
LD	(pp).g,CF	4-58	4-77	4-102
LD	(x),(HL+C)	4-17	4-67	4-91
LD	(x),(HL+d)	4-17	4-67	4-91
LD	(x),(PC+A)	4-17	4-67	4-91
LD	(x),(pp)	4-17	4-67	4-91
LD	(x),(y)	4-17	4-67	4-91
LD	(x),A	4-16	4-67	4-91
LD	(x),n	4-18	4-68	4-91
LD	(x),r	4-16	4-67	4-91
LD	(x),rr	4-16	4-67	4-91
LD	(x).b,CF	4-57	4-77	4-102
LD	A,(HL)	4-14	4-67	4-90
LD	A,(x)	4-14	4-67	4-90
LD	A,r	4-14	4-67	4-90
LD	CF,(-HL).b	4-57	4-76	4-101
LD	CF,(HL+).b	4-57	4-76	4-101
LD	CF,(HL+C).b	4-57	4-76	4-101
LD	CF,(HL+d).b	4-57	4-76	4-101
LD	CF,(PC+A).b	4-57	4-76	4-101
LD	CF,(pp).b	4-57	4-76	4-101
LD	CF,(pp).g	4-57	4-76	4-101
LD	CF,(x).b	4-57	4-76	4-101
LD	CF,g.b	4-56	4-76	4-101
LD	g.b,CF	4-57	4-77	4-102
LD	gg,SP	4-14	4-67	4-90
LD	r,(-HL)	4-15	4-67	4-90
LD	r,(HL+)	4-14	4-67	4-90
LD	r,(HL+C)	4-15	4-67	4-90
LD	r,(HL+d)	4-15	4-67	4-90
LD	r,(PC+A)	4-15	4-67	4-90

LD	r,(pp)	4-14	4-67	4-90
LD	r,(x)	4-14	4-67	4-90
LD	r,A	4-14	4-67	4-90
LD	r,g	4-14	4-67	4-90
LD	r,n	4-18	4-68	4-91
LD	RBS,n	4-18	4-68	4-91
LD	rr,(HL+C)	4-15	4-67	4-90
LD	rr,(HL+d)	4-15	4-67	4-90
LD	rr,(PC+A)	4-15	4-67	4-90
LD	rr,(pp)	4-15	4-67	4-90
LD	rr,(x)	4-15	4-67	4-90
LD	rr,gg	4-14	4-67	4-90
LD	rr,mn	4-18	4-68	4-91
LD	SP,gg	4-14	4-67	4-90
LD	SP,mn	4-18	4-68	4-91
LDW	(HL),mn	4-19	4-68	4-91
LDW	(x),mn	4-19	4-68	4-91
MCMP	(-HL),n	4-24	4-69	4-95
MCMP	(HL+),n	4-23	4-69	4-95
MCMP	(HL+C),n	4-24	4-69	4-95
MCMP	(HL+d),n	4-24	4-69	4-95
MCMP	(PC+A),n	4-24	4-69	4-95
MCMP	(pp),n	4-23	4-69	4-95
MCMP	(x),n	4-23	4-69	4-95
MUL	B,C	4-48	4-74	4-96
MUL	D,E	4-48	4-74	4-96
MUL	H,L	4-48	4-74	4-96
MUL	W,A	4-48	4-74	4-96
NOP		4-66	4-78	4-105
OR	(-HL),n	4-42	4-73	4-95
OR	(HL+),n	4-42	4-73	4-95
OR	(HL+C),(HL)	4-41	4-73	4-94
OR	(HL+C),n	4-43	4-73	4-95
OR	(HL+d),(HL)	4-41	4-73	4-94
OR	(HL+d),n	4-42	4-73	4-95
OR	(PC+A),(HL)	4-41	4-73	4-94
OR	(PC+A),n	4-43	4-73	4-95
OR	(pp),(HL)	4-41	4-73	4-94
OR	(pp),n	4-42	4-73	4-95
OR	(x),(HL)	4-41	4-73	4-94
OR	(x),n	4-42	4-73	4-95
OR	A,(-HL)	4-41	4-73	4-94
OR	A,(HL+)	4-40	4-73	4-94
OR	A,(HL+C)	4-41	4-73	4-94
OR	A,(HL+d)	4-41	4-73	4-94
OR	A,(PC+A)	4-41	4-73	4-94
OR	A,(pp)	4-40	4-73	4-94

OR	A,(x)	4-40	4-73	4-94
OR	A,g	4-40	4-73	4-93
OR	A,n	4-42	4-73	4-94
OR	g,A	4-40	4-73	4-93
OR	g,n	4-42	4-73	4-94
OR	gg,mn	4-42	4-73	4-95
OR	WA,gg	4-40	4-73	4-93
POP	gg	4-66	4-78	4-105
POP	PSW	4-66	4-78	4-105
PUSH	gg	4-66	4-78	4-105
PUSH	PSW	4-66	4-78	4-105
RET		4-65	4-78	4-105
RETI		4-65	4-78	4-105
RETN		4-65	4-78	4-105
ROLC	A	4-49	4-75	4-96
ROLC	g	4-49	4-75	4-96
ROLD	A,(-HL)	4-50	4-75	4-96
ROLD	A,(HL+)	4-50	4-75	4-96
ROLD	A,(HL+C)	4-51	4-75	4-96
ROLD	A,(HL+d)	4-51	4-75	4-96
ROLD	A,(pp)	4-50	4-75	4-96
ROLD	A,(x)	4-50	4-75	4-96
RORC	A	4-49	4-75	4-96
RORC	g	4-50	4-75	4-96
RORD	A,(-HL)	4-51	4-75	4-96
RORD	A,(HL+)	4-51	4-75	4-96
RORD	A,(HL+C)	4-52	4-75	4-96
RORD	A,(HL+d)	4-52	4-75	4-96
RORD	A,(pp)	4-51	4-75	4-96
RORD	A,(x)	4-51	4-75	4-96
SET	(-HL).b	4-55	4-76	4-99
SET	(HL+).b	4-55	4-76	4-99
SET	(HL+C).b	4-55	4-76	4-99
SET	(HL+d).b	4-55	4-76	4-99
SET	(PC+A).b	4-55	4-76	4-99
SET	(pp).b	4-55	4-76	4-99
SET	(pp).g	4-55	4-76	4-99
SET	(x).b	4-55	4-76	4-99
SET	CF	4-60	4-77	4-103
SET	g.b	4-55	4-76	4-99
SHLC	A	4-49	4-75	4-96
SHLC	g	4-49	4-75	4-96
SHRC	A	4-49	4-75	4-96
SHRC	g	4-49	4-75	4-96
SUB	(-HL),n	4-34	4-71	4-95
SUB	(HL+),n	4-34	4-71	4-95
SUB	(HL+C),(HL)	4-33	4-71	4-94

SUB	(HL+C),n	4-34	4-71	4-95
SUB	(HL+d),(HL)	4-33	4-71	4-94
SUB	(HL+d),n	4-34	4-71	4-95
SUB	(PC+A),(HL)	4-33	4-71	4-94
SUB	(PC+A),n	4-34	4-71	4-95
SUB	(pp),(HL)	4-33	4-71	4-94
SUB	(pp),n	4-34	4-71	4-95
SUB	(x),(HL)	4-33	4-71	4-94
SUB	(x),n	4-33	4-71	4-95
SUB	A,(-HL)	4-32	4-71	4-94
SUB	A,(HL+)	4-32	4-71	4-94
SUB	A,(HL+C)	4-32	4-71	4-94
SUB	A,(HL+d)	4-32	4-71	4-94
SUB	A,(PC+A)	4-32	4-71	4-94
SUB	A,(pp)	4-32	4-71	4-94
SUB	A,(x)	4-32	4-71	4-94
SUB	A,g	4-32	4-71	4-93
SUB	A,n	4-33	4-71	4-94
SUB	g,A	4-32	4-71	4-93
SUB	g,n	4-33	4-71	4-94
SUB	gg,mn	4-33	4-71	4-95
SUB	WA,gg	4-32	4-71	4-93
SUBB	(-HL),n	4-37	4-72	4-95
SUBB	(HL+),n	4-36	4-72	4-95
SUBB	(HL+C),(HL)	4-36	4-72	4-94
SUBB	(HL+C),n	4-37	4-72	4-95
SUBB	(HL+d),(HL)	4-36	4-72	4-94
SUBB	(HL+d),n	4-37	4-72	4-95
SUBB	(PC+A),(HL)	4-36	4-72	4-94
SUBB	(PC+A),n	4-37	4-72	4-95
SUBB	(pp),(HL)	4-35	4-72	4-94
SUBB	(pp),n	4-36	4-72	4-95
SUBB	(x),(HL)	4-35	4-72	4-94
SUBB	(x),n	4-36	4-72	4-95
SUBB	A,(-HL)	4-35	4-72	4-94
SUBB	A,(HL+)	4-35	4-72	4-94
SUBB	A,(HL+C)	4-35	4-72	4-94
SUBB	A,(HL+d)	4-35	4-72	4-94
SUBB	A,(PC+A)	4-35	4-72	4-94
SUBB	A,(pp)	4-35	4-72	4-94
SUBB	A,(x)	4-35	4-72	4-94
SUBB	A,g	4-34	4-72	4-93
SUBB	A,n	4-36	4-72	4-94
SUBB	g,A	4-34	4-72	4-93
SUBB	g,n	4-36	4-72	4-94
SUBB	gg,mn	4-36	4-72	4-94
SUBB	WA,gg	4-34	4-72	4-93

SWAP	A	4-50	4-75	4-96
SWAP	g	4-50	4-75	4-96
SWI		4-66	4-78	4-105
TEST	(-HL).b	4-53	4-76	4-97
TEST	(HL+).b	4-53	4-76	4-97
TEST	(HL+C).b	4-53	4-76	4-97
TEST	(HL+d).b	4-53	4-76	4-97
TEST	(PC+A).b	4-53	4-76	4-97
TEST	(pp).b	4-53	4-76	4-97
TEST	(pp).g	4-53	4-76	4-97
TEST	(x).b	4-53	4-76	4-97
TEST	g.b	4-53	4-76	4-97
XCH	r,(-HL)	4-20	4-68	4-92
XCH	r,(HL+)	4-20	4-68	4-92
XCH	r,(HL+C)	4-20	4-68	4-92
XCH	r,(HL+d)	4-20	4-68	4-92
XCH	r,(PC+A)	4-20	4-68	4-92
XCH	r,(pp)	4-20	4-68	4-92
XCH	r,(x)	4-19	4-68	4-92
XCH	r,g	4-19	4-68	4-92
XCH	rr,gg	4-19	4-68	4-92
XOR	(-HL),n	4-45	4-74	4-95
XOR	(HL+),n	4-45	4-74	4-95
XOR	(HL+C),(HL)	4-44	4-74	4-94
XOR	(HL+C),n	4-46	4-74	4-95
XOR	(HL+d),(HL)	4-44	4-74	4-94
XOR	(HL+d),n	4-45	4-74	4-95
XOR	(PC+A),(HL)	4-44	4-74	4-94
XOR	(PC+A),n	4-46	4-74	4-95
XOR	(pp),(HL)	4-44	4-74	4-94
XOR	(pp),n	4-45	4-74	4-95
XOR	(x),(HL)	4-44	4-74	4-94
XOR	(x),n	4-45	4-74	4-95
XOR	A,(-HL)	4-43	4-73	4-94
XOR	A,(HL+)	4-43	4-73	4-94
XOR	A,(HL+C)	4-44	4-73	4-94
XOR	A,(HL+d)	4-44	4-73	4-94
XOR	A,(PC+A)	4-44	4-73	4-94
XOR	A,(pp)	4-43	4-73	4-94
XOR	A,(x)	4-43	4-73	4-94
XOR	A,g	4-43	4-73	4-93
XOR	A,n	4-45	4-74	4-94
XOR	CF,(-HL).b	4-59	4-77	4-103
XOR	CF,(HL+).b	4-59	4-77	4-103
XOR	CF,(HL+C).b	4-59	4-77	4-103
XOR	CF,(HL+d).b	4-59	4-77	4-103
XOR	CF,(PC+A).b	4-59	4-77	4-103

XOR	CF,(pp).b	4-58	4-77	4-103
XOR	CF,(x).b	4-58	4-77	4-103
XOR	CF,g.b	4-58	4-77	4-103
XOR	g,A	4-43	4-73	4-93
XOR	g,n	4-45	4-74	4-94
XOR	gg,mn	4-45	4-74	4-95
XOR	WA,gg	4-43	4-73	4-93