TOSHIBA

TLCS-870/C1 Series Instruction Set

TOSHIBA CORPORATION

Semiconductor Company

1. Overview

The basic machine instructions available with the TLCS-870/C series of microcontrollers consist of 133 types, providing a total of 732 instructions. The table below shows these instructions classifiled by type. The instructions for the TLCS-870/C series vary in length from one byte to a maximum of five bytes. The frequently used instructions have their object code length reduced, making it possible to create memory-efficient programs.

The TLCS-870/C series offers a simple instruction architecture based on a memory-mapped I/O system. Although only 42 kinds of mnemonics are available, it supports 8 addressing modes allowing for powerful memory manipulation.

In addition to the basic machine instruction, the TLCS-870/C series offers assembler-based extended machine instructions in order to increase coding efficency.

Table 1-1 Intsruction Set

Load/Store, Exchange	8-bit data Load/Store, Exchange		7 types	49 instructions
	16-bit data Load/Store, Exchange		7	43
	Flag Manipuration		5	5
	SP Manipuration		1	2
	Push, Pop		4	6
ALU	8-bit ALU	Compare	4	29
		Increment, Decrement	4	28
		Arithmetic	16	116
		Logical	12	87
		Decimal Adjust	2	2
	16-bit ALU	Compare	3	15
		Increment, Decrement	2	2
		Arithmetic	12	60
		Logical	9	45
		2's Complement	1	1
	Multiply, Divide		2	2
Shift/Rotate	Shift	8-bit (Logical)	2	2
		16-bit (Arithmetic)	2	2
	Rotate	8-bit	2	2
Nibble Manipulation	Swap, Nibble Rotate		3	27
Bit Manipulation	Set, Clear, Complement, Load/Store, Exchange, EXOR		18	162
Branch	Jump		6	24
	Call		4	16
	Return		3	3
Others	Software Interrupt, Other		2	2
Total			133 types	732 instructions

Table 1-2 Addressing Modes

Register Indirect	7 types
Direct	2
Register	1
Immediate	1
Relative	2
Absolute	1
Vector	1
Direct Bit	2
Register Indirect Bit	1
Total	18

1.1 Symbols Used in this Document

The symbols shown below are used in the descriptions of instructions and addressing modes in the following pages.

Symbol	Description	Symbol	Description
A	A register	r, g	8-bit register (SeeTable 1-3)
W	W register	rr, gg	16-bit register (SeeTable 1-4)
B	B register	n	4-bit or 8-bit immediate data
C	C register	mn	16-bit immediate data
D	D register	d	Signed 5-bit or 8-bit displacement (–16 to +15/–128 to +127)
E	E register	x, y	8-bit direct address (0x0000 to 0x00FF)
H	H register	vw, uz	16-bit direct address (0x0000 to 0xFFFF)
L	L register	(XX)	Memory contents at the address specified by XX
WA	WA register	(XX + 1, XX)	Two consective bytes from the memory location specified by XX
BC	BC register	b	Bit number (0 to 7)
DE	DE register	.b	Content of bit specified by b
HL	HL register	←	Load/store
IX	IX register	↔	Exchange
IY	IY register	+	Add
PC	Program Counter	–	Substract
SP	Stack Pointer	×	Multiply
PSW	Program Status Word	÷	Divide
JF	Jump Status flag	&	Bitwise AND
ZF	Zero flag	|	Bitwise OR
CF	Carry flag (1-bit accumulator)	^	Bitwise exclusive OR
HF	Halfcarry flag	null	No operation (Goes to instruction at the next address without performing anything.)
SF	Sign flag	$	Start address of instruction bein executed (The program Counter points to the address two or three bytes after $.)
VF	Overflow flag	(src)	Source memory
$\overline{CF}$	Inverse of carry flag	(dst)	Destination memory
IMF	Interrupt Master Enable flag	RBS	Register Bank Selector
NxtOp	Next operation addresses (Start address of the next instruction)		

Table 1-3

r, g	8-Bit Register
0	A
1	W
2	C
3	B
4	E
5	D
6	L
7	H

Table 1-4

rr, gg	16-Bit Register
0	WA
1	BC
2	DE
3	HL
4	IX
5	IY
6	SP
7	HL

Flag setting conditions		
*	The value specified by the operation is set.	
Z	Zero detection	
	• Load/Store	When the 8-bit source data is 0x00, the Z flag is set to 1. Otherwise, the Z flag is cleared to 0.
	• Exchange	When the contents of g register or (src) before the exchange is 0x00, the Z flag is set to 1. Otherwise, the Z flag is cleared to 0.
	• ALU	When the operation result is 0x00 (8-bit operation) or 0x0000 (16-bit operation), the Z flag is set to 1. Otherwise, the Z flag is cleared to 0. However, for multiply operation, this Z flag is set when the high-order 8 bits of the product are 0x00, for divide operation, this Z flag is set when the remainder is 0x00. Otherwise, the Z flag is cleared to 0.
	• Shift/Rotate	When the register contents is 0x00 after being shifted or rotated, the Z flag is set to 1. Otherwise, the Z flag is cleared to 0.
	• Other	When Z appears in the JF column, it means that the value set in ZF is also set in JF.
C	Carry	
	• Addition	When a carry occurs from the most significant bit, the C flag is set.
	• Subtraction	When a borrow occurs from the most significant bit, the C flag is set.
	• Division	When the divisor is 0x00 or the quotient is equal to or greater than 0x100, the C flag is set to 1. Otherwise, the flag is cleared to 0.
	• Other	When C appears in the JF column, it means that the value set in CF is also set in JF.
$\overline{C}$	An inverse of CF	
H	Half carry	
	• Half carry	Addition When a carry occurs from bit 3, the H flag is set.
	• Subtraction	When a borrow occurs from bit 3, the H flag is set.
S	Sign (most significant bit of data)	
V	Overflow	
$\overline{J}$	An inverse of JF	
1	A logic 1 is set.	
0	A logic 0 is set.	
U	An indeterminate value is set.	
–	The value before instruction execution is retained leaving the flag unchanged.	

1.2 Mnemonics

The following describes the rules on mnemonics of the TLCS-870/C1 series instructions.

Each mnemonics consists of an opcode and an operand (s). (Some instructions do not have any operand.) The opcode and the operands are separated with one or more spaces. If an instruction has two or more operands, each operand is separated with a comma.

If the instruction has two operands, source and destination, the destination operand is always placed before the source operand. If there are multiple source operands, the first operand is placed first followed by the other operand.

If the operand includes a bit specifier, the address and the bit specifier are separated with a period.

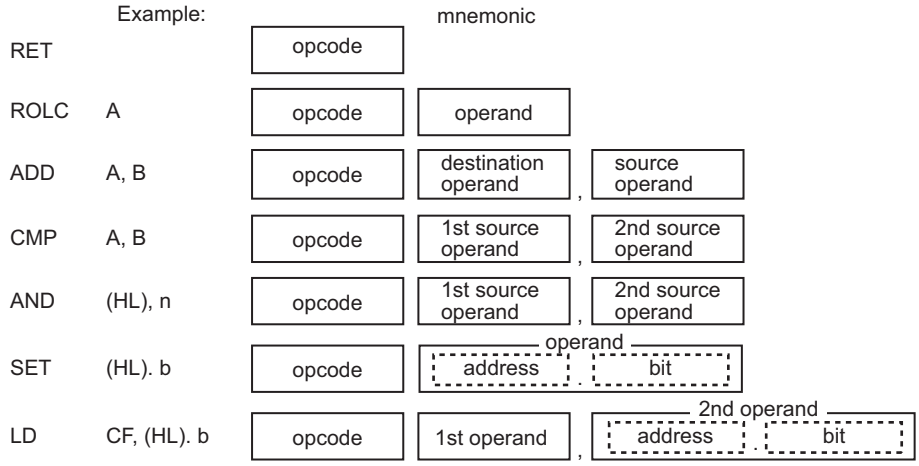


Table 1-5 Mnemonics

Mnemonics	Description
ADD	Add
ADDC	Add with carry
AND	Logical AND
CALL	Call
CALLV	Vector call
CLR	Clear bit/byte
CMP	Compare
CPL	1's complement bit
DAA	Decimal adjust for 8-bit addition
DAS	Decimal adjust for 8-bit subtraction
DEC	Decrement byte/word (Register)
DI*	Disable maskable interrupt
DIV	Divide byte quotient
EI*	Enable interrupt
INC	Increment byte/word (Register)
J*	Optimized jump
JP	Absolute jump
JR	Relative jump
JRS	Short relative jump
LD	Load bit/byte/word (Register)/effective address
LDW	Load word (Memory)
MUL	Multiply
NEG	Negate
NOP	No operation
OR	Logical OR
POP	Pop up
PUSH	Push down
RET	Return from subroutine
RETI	Return from maskable interrupt service routine
RETN	Return from non-maskable interrupt service routine
ROLC	Rotate left through carry
ROLD	Rotate left digit
RORC	Rotate right through carry
RORD	Rotate right digit
SET	Bit test and set
SHLC	Logical shift left
SHLCA	Arithmetic shift left
SHRC	Logical shift right
SHRCA	Arithmetic shift right
SUB	Subtract
SUBB	Subtract with borrow
SWAP	Swap nibble
SWI	Software interrupt
TEST*	Bit test
XCH	Exchange
XOR	Logical exclusive OR

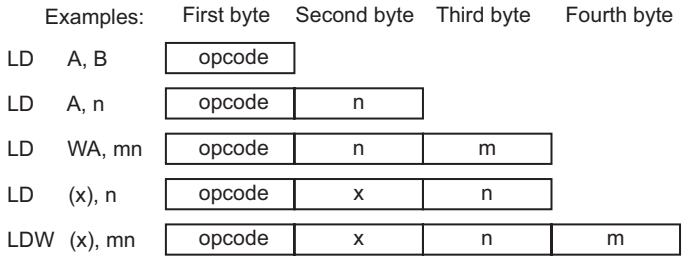
Note: *: Instructions marked with an asterisk (*) are extended assembler machine instructions.

1.3 Object Code Format

The TLCS-870/C1 series has instructions with 1-byte and 2-byte opcodes.

1.3.1 Format of Instructions with a 1-byte Opcode.

The opcode is placed in the first byte and the operand is placed in the second and subsequent bytes. If the operand is 2 bytes long, the lower byte is placed first followed by the upper byte. When the operand consists of source and destination, the source is placed after the destination if the source is an immediate (e.g., LD(x), n).



1.3.2 Format of Instructions with 2-byte Opcodes

The first opcode is placed in the first byte and then the operand specified by the first opcode is placed in the next byte. Then, the second opcode and the operand specified by the second opcode are placed in the following bytes.

The first opcode, which is used to specify an addressing mode, is called the "prefix code." There are two types of prefix: the register prefix to specify a register and the source/destination memory prefix to specify the source or destination memory address. Note that the first opcode in the five instructions shown below ignores the contents of a specified register:

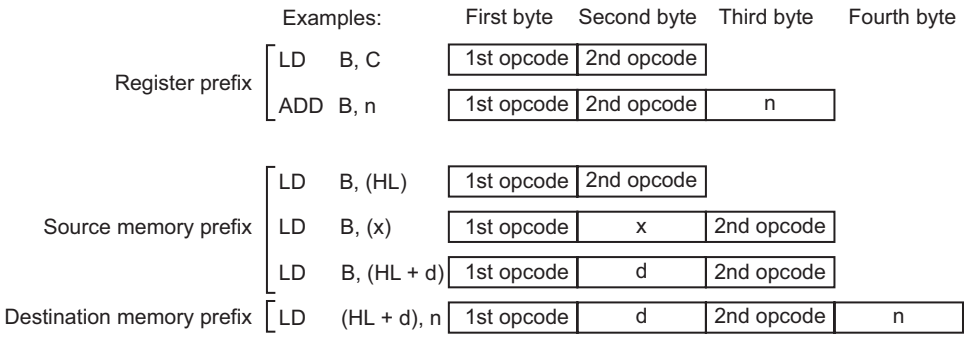
(1) RETN

(2) LD PSW, n

(3) PUSH PSW

(4) POP PSW

(5) JR M/P/SLT/SGE/SLE/SGT/VS/VC, a



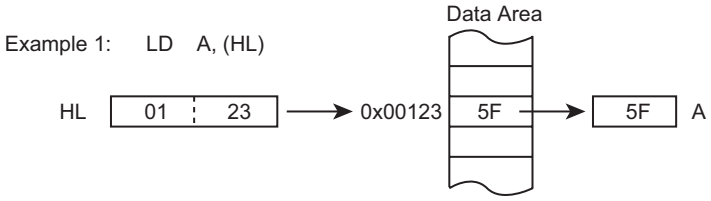
1.4 Addressing Mode

The TLCS-870/C series supports 17 addressing modes (or the methods for specifying addresses). Some instructions use a combination of multiple addressing modes.

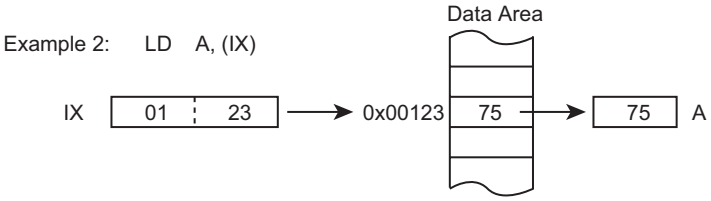
1.4.1 Register Indirect Addressing

1.4.1.1 Register Indirect Addressing (HL), (DE), (IX), (IY)

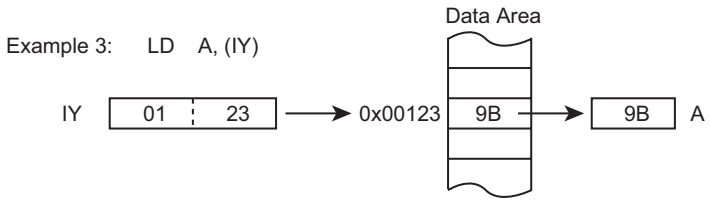
The effective address is specified by the contents of a 16-bit register pair HL, DE, IX or IY.



The contents of the memory location pointed to by HL, i.e., 0x5F, at address 0x00123, is loaded into the A register.



The contents of the memory location pointed to by IX, i.e., 0x75, at address 0x00123, is loaded into the A register.

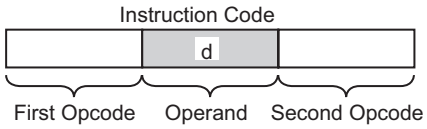


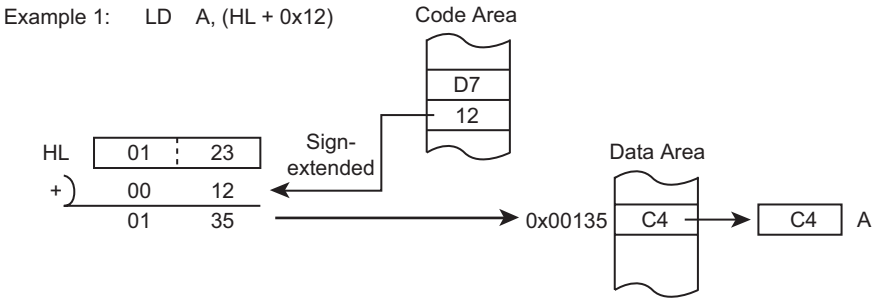
The contents of the memory location pointed to by IX, i.e., 0x9B, at address 0x00123, is loaded into the A register.

1.4.1.2 Register Indirect with 8-Bit Displacement Addressing (HL + d), (IX + d), (IY + d)

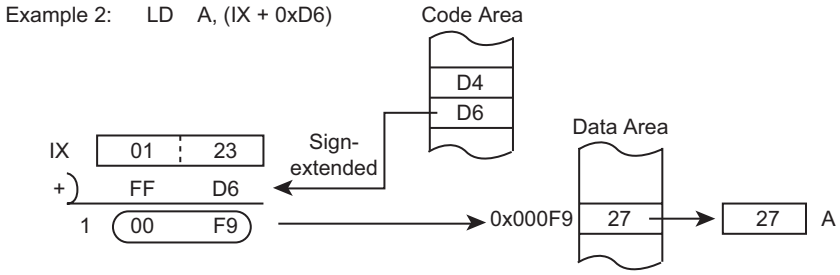
The effective address is formed by sign-extending the 8-bit displacement d in the instruction code (see the table below) and adding it to the contents of the 16-bit register HL, IX or IY. Note that the contents of the 16-bit register HL, IX or IY does not change.

Displacement d	Sign-extended Value
0x00 to 0x7F	0x0000 to 0x007F (0 to +127)
0x80 to 0xFF	0xFF80 to 0xFFFF (–128 to –1)





The displacement (0x12) is sign-extended (0x0012) and added to the contents of the HL register (0x0123) to form an effective address. The contents of the memory location specified by the effective address, i.e., 0xC4, at address 0x00135, is loaded into the A register.

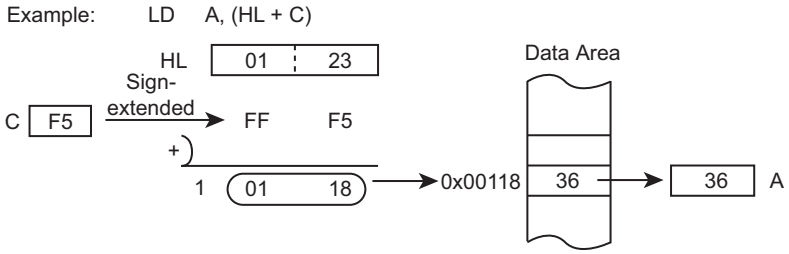


The displacement (0xD6) is sign-extended (0xFFD6) and added to the contents of the IX register (0x0123) to form an effective address. The contents of the memory location specified by the effective address, i.e., 0x27, at address 0x000F9, is loaded into the A register.

1.4.1.3 Register Indexed Addressing (HL + C)

The effective address is formed by sign-extending the contents of the C register (see the table below) and adding it to the contents of the HL register. Note that the contents of the HL and C registers do not change.

C Register	Sign-extended Value
0x00 to 0x7F	0x0000 to 0x007F (0 to +127)
0x80 to 0xFF	0xFF80 to 0xFFFF (–128 to –1)

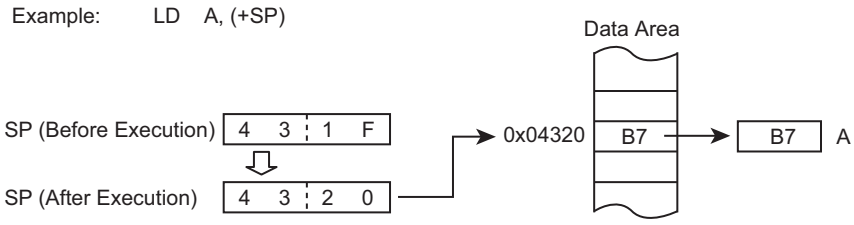


The contents of the C register (0xF5) is sign-extended (0xFFF5) and added to the contents of the HL register (0x0123) to from an effective address. The contents of the memory location specified by the effective address, i.e., 0x36, at address 0x00118, is loaded into the A register.

1.4.1.4 Stack Pointer Indirect with Auto-Pre-Increment Addressing (+SP)

The contents of the SP is incremented to form an effective address. Incrementing the SP does not affect the flag bits.

This addressing mode can only be used to specify the source memory address.

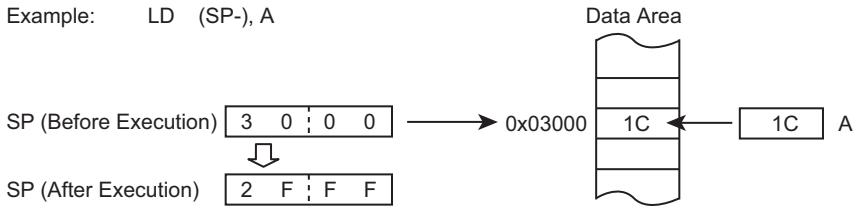


The contents of the SP is incremented first to form an effective address. The memory contents at the effective address, i.e., 0xB7, at address 0x04320, is then loaded into the A register.

1.4.1.5 Stack Pointer Indirect with Auto-Decrement Addressing (SP-)

The SP holds the effective address. After the data manipulation, the contents of the SP is automatically decremented. Decrementing the SP does not affect the flag bits. This addressing mode can only be used to specify the destination memory address.

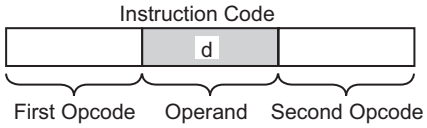
The contents of the A register (0x1C) is stored into the memory location specified by the contents of the SP, i.e., address 0x03000. The SP is then decremented to 0x2FFF.

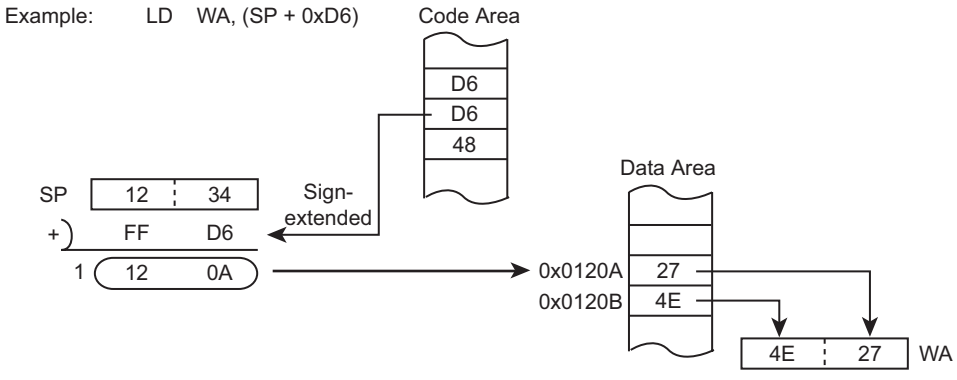


1.4.1.6 Stack Pointer Indirect with 8-Bit Displacement Offset Addressing (SP + d)

The effective address is formed by sign-extending the 8-bit displacement d in the instruction code (see the table below) and adding it to the contents of the Stack Pointer SP.

Displacement d	Sign-extended Value
0x00 to 0x7F	0x0000 to 0x007F (0 to +127)
0x80 to 0xFF	0xFF80 to 0xFFFF (–128 to –1)



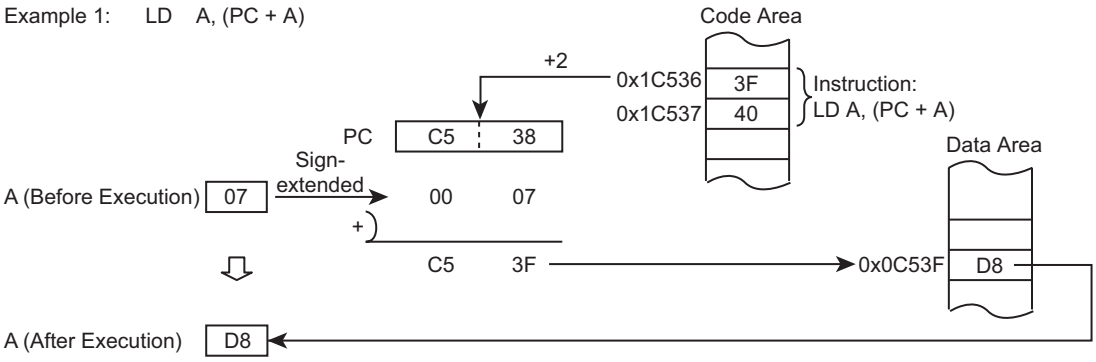


The displacement (0xD6) is sign-extended (0xFFD6) and added to the contents of SP (0x1234) to form an effective address (0x0120A). Two consecutive bytes from the memory location specified by the effective address, i.e., 0x4E27, are loaded into the WA register.

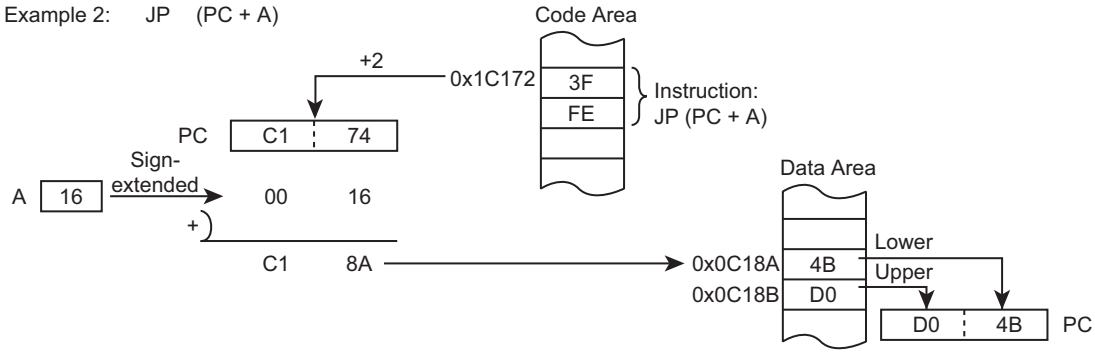
1.4.1.7 PC-Relative Register Indirect Addressing (PC + A)

The effective address is formed by sign-extending the contents of the A register (see the table below) and adding it to the contents of the Program Counter (start address of the current instruction + 2). This addressing mode can only be used to specify the source address. This addressing mode simplifies the coding of BCD to 7-segment conversions, table lookups, table searches and multi-way (n-way) branches.

Accumulator	Sign-extended Value
0x00 to 0x7F	0x0000 to 0x007F (0 to +127)
0x80 to 0xFF	0xFF80 to 0xFFFF (–128 to –1)



The contents of the A register (0x07) is sign-extended (0x0007) and added to the contents of the Program Counter (0xC538) to form an effective address (0x0C53F). The contents of the memory location specified by the effective address, i.e., 0xD8 is loaded into the A register.



The contents of the A register (0x16) is sign-extended (0x0016) and added to the contents of the Program Counter (0xC174) to form an effective address (0x0C18A). Two consecutive bytes from the memory location specified by the effective address, 0xD04B, are loaded into the Program Counter. That is, a jump is taken to the destination address, 0xD04B.

Note: The meaning of the operand (PC + A) differs between the 870/C and 870/C1.

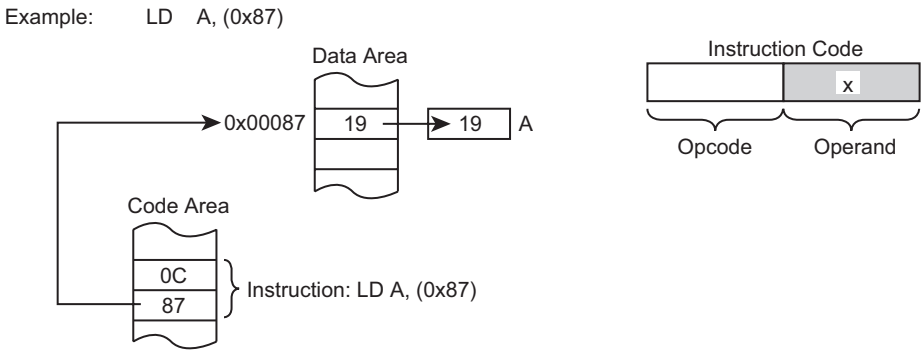
In the 870/C1, the area that is accessed using the addressing mode using the operand (PC + A) is not a code area but a data area. Therefore, the addressing mode using the operand (PC + A) in the 870/C1 is not compatible with that in the 870/C.

Care should be taken when porting programs from the 870/C to the 870/C1.

1.4.2 Direct Addressing

1.4.2.1 8-Bit Direct Addressing (x)

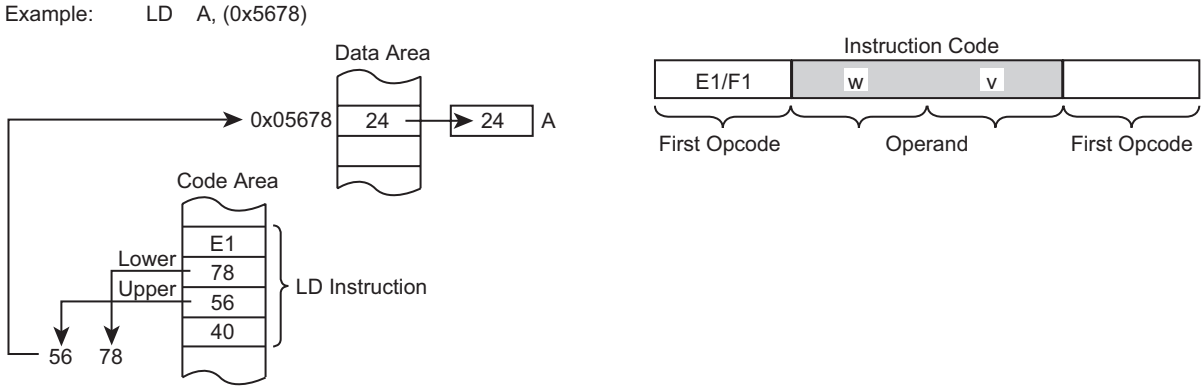
The address is specified directly by the 8-bit value x in the instruction code. The address is in the range of 0x00000 to 0x000FF.



The contents of the memory location specified directly by the value x in the instruction code (0x87), i.e, 0x19, at address 0x00087, is loaded into the A register.

1.4.2.2 16-Bit Direct Addressing (vw)

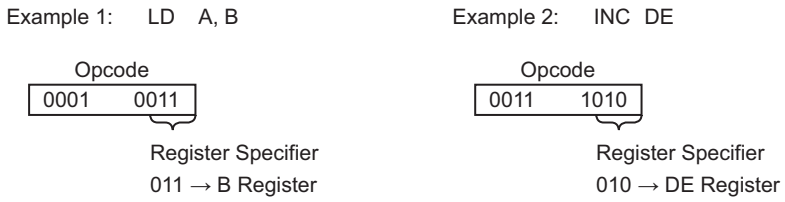
The effective address is specified directly by the 16-bit value vw in the instruction code. The address is in the range of 0x00000 to 0x0FFFF.



The contents of the memory location specified directly by the value vw in the instruction code (0x5678), i.e., 0x24, at address 0x05678, is loaded into the A register.

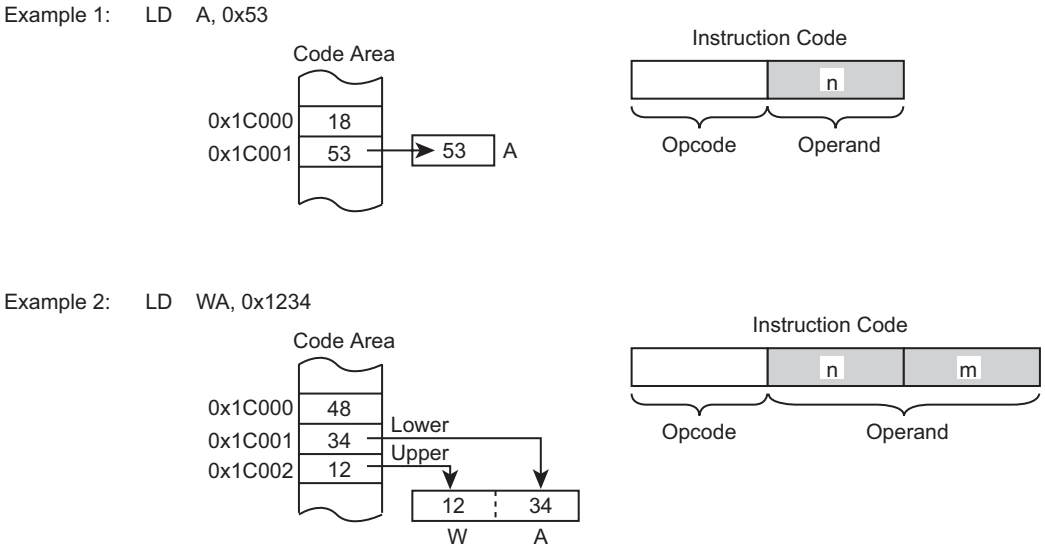
1.4.3 Register Addressing (r or rr)

The register specifier in the instruction code (opcode) specifies which register is to be accessed.



1.4.4 Immediate Addressing (n or mn)

A specified operation is performed directly on the immediate operand in the instruction code. Note that when the operand is a 16-bit immediate, it is stored with the low-order byte first, followed by the high-order byte.



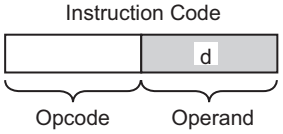
Note: In the assembler source program, do not enclose the immediate with parentheses. Otherwise, it is interpreted as the direct addressing mode.

1.4.5 Relative Addressing

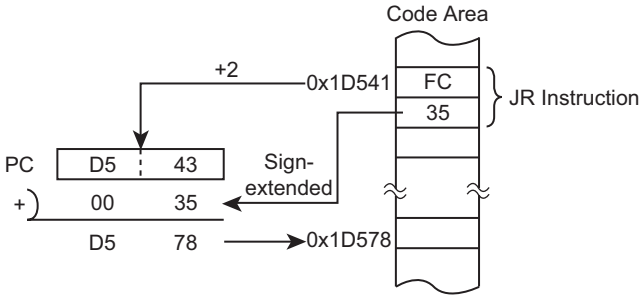
1.4.5.1 PC-Relative with 8-Bit Displacement Addressing

The effective address is formed by sign-extending the 8-bit displacement d in the instruction code (see the table below) and adding it to the contents of the Program Counter (start address of the current instruction + 2 or 3).

Displacement d	Sign-extended Value
0x00 to 0x7F	0x0000 to 0x007F (0 to +127)
0x80 to 0xFF	0xFF80 to 0xFFFF (–128 to –1)



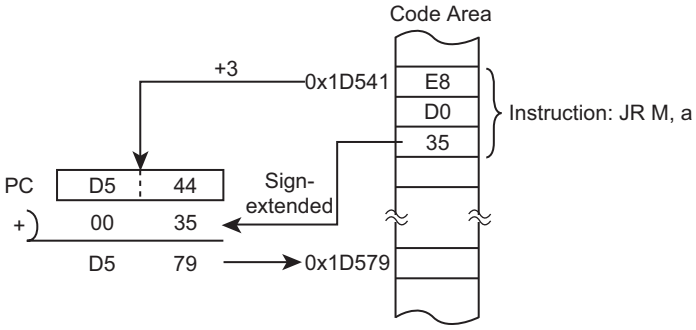
Example 1: JR $ + 2 + 35H or JR 0x0D578



The displacement (0x35) is sign-extended and added to the contents of the Program Counter (0xD543) to form an effective address. Program control is then transferred to the effective address, 0x1D578.

Note: $ denotes the start address of the current instruction.

Example 2: JR M, $ + 3 + 35H or JR M, 0x0D579



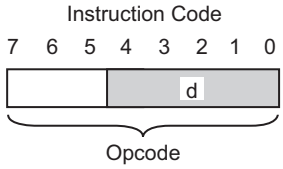
The displacement (0x35) is sign-extended and added to the contents of the Program Counter (0xD544) to form an effective address. Program control is then transferred to the effective address, 0x1D579, if the sign flag is 1.

Note: $ denotes the start address of the current instruction.

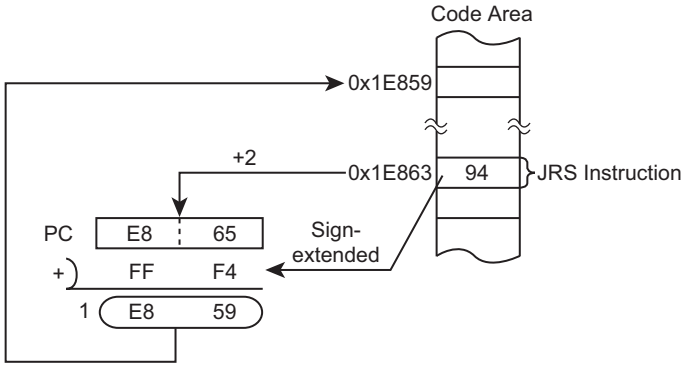
1.4.5.2 PC-Relative with 5-Bit Displacement Addressing

The effective address is formed be sign-extending the 5-bit displacement d in the opcode (see the table below) and adding it to the contents of the Program Counter (start address of the current instruction + 2).

Displacement d	Sign-extended Value
0x00 to 0x0F	0x0000 to 0x000F (0 to +15)
0x10 to 0x1F	0xFFF0 to 0xFFFF (–16 to –1)



Example: JRS T, $ + 2 + 14H or JRS T, 0x0E859



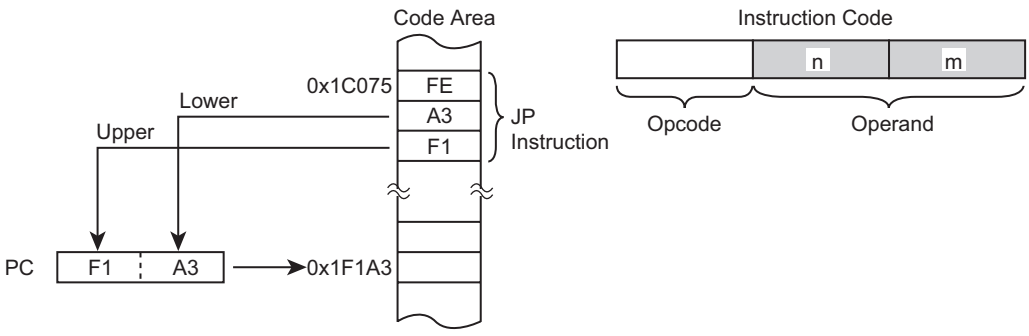
The displacement (0x14) is sign-extended (0xFFF4) and added to the contents of the Program Counter (0xE865) to form an effective address. Program control is then transferred to the effective address, 0x1E859, if the Jump Status flag is 1.

Note: $ denotes the start address of the current instruction.

1.4.6 Absolute Addressing

The effective address is specified by a 16-bit value in the instruction code (stored with the low-order byte first, followed by the high-order byte).

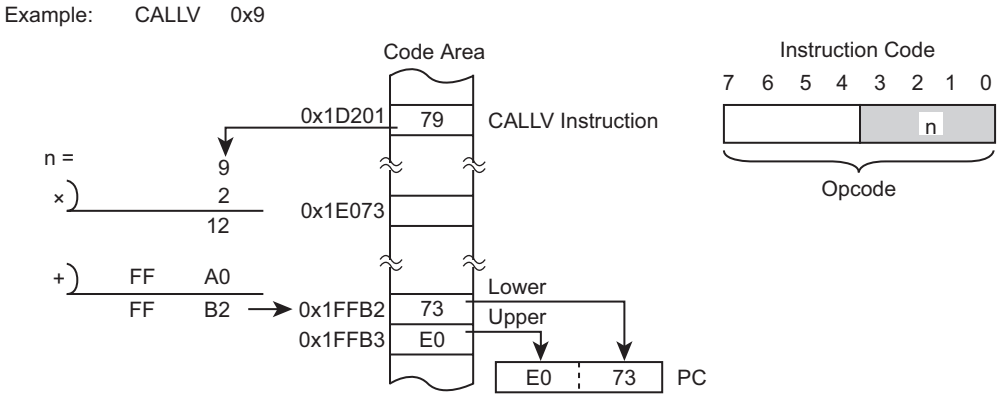
Example: JP 0x0F1A3



Program control is transferred to the address specified by the operand, 0x1F1A3.

1.4.7 Vector Addressing

The 4-bit operand is multiplied by 2 and added to the top address of the vector call table to form a pointer to a location where a 16-bit jump destination address (vector address) is located.

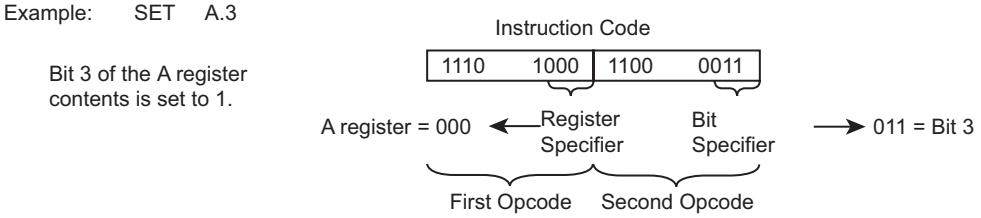


The operand n (0x9) is multiplied by 2 and added to the top address of the vector call table (0xFFA0) to form a pointer to a location (0x1FFB2) where a jump destination address, which is formed from the contents of two consecutive bytes (0x1E073), is located.

1.4.8 Direct Bit Addressing

1.4.8.1 Register Bit Addressing

The register and bit specifiers in the instruction code specify a bit position in a register whose value should be tested or changed.



1.4.8.2 Memory Bit Addressing

In Memory Bit addressing mode, the bit specifier in the instruction code specifies the bit in the memory location pointed to by (HL), (DE), (IX), (IY), (HL + d), (IX + d), (IY + d), (HL + C), (+SP), (SP + d), (PC + A), (x) or (vw). A bit manipulation is performed on the specified bit.

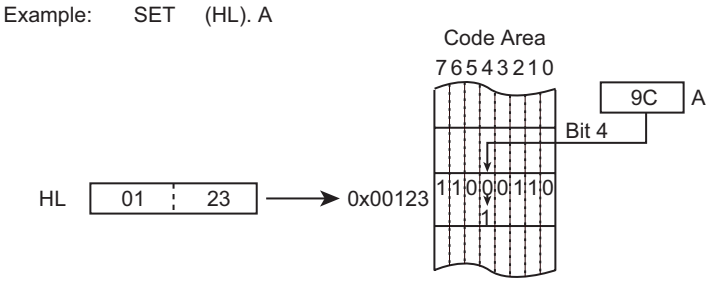
Example 1: SET (HL). 1

Example 2: SET (HL + 0x57). 6

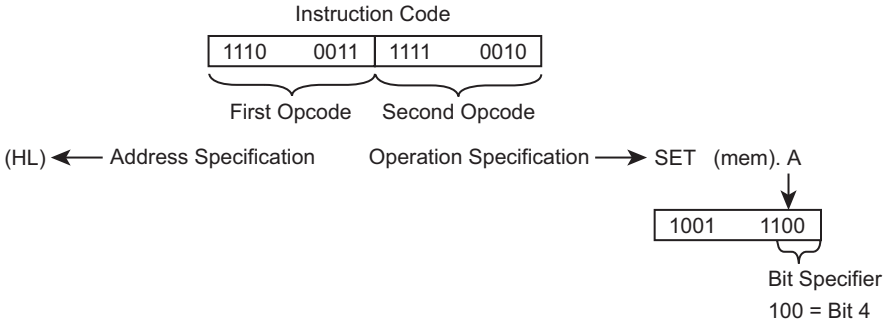
Example 3: SET (0x00058). 3

1.4.9 Register Indirect Bit Addressing

In Memory Bit addressing mode, low-order 3 bits of the A register specify the bit in the memory location pointed to by (HL), (DE), (IX), (IY), (HL + d), (IX + d), (IY + d), (HL + C), (+SP), (SP + d), (PC + A), (x) or (vw). A bit manipulation is performed on the specified bit.



The data in the memory location pointed to by the contents of the HL register (0x0123) is 0y11000110. Since the low-order 3 bits of the A register (0y100) indicates bit 4, bit 4 of the data is set and the value is changed to 0y11010110.



1.1 Symbols Used in this Document

2. Instruction Set Details

2.1 Load/Store and Exchange Instructions

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
LD A,r	0 0 0 1 0 r r r	1 Z – – – –	1	A ← r
	Load the contents of 8-bit register r into A. The Zero flag is set to 1 when r = 0x00; otherwise, it is cleared to 0.			
LD r,A	0 1 0 0 0 r r r	1 Z – – – –	1	r ← A
	Load the contents of A into 8-bit register r. The Zero flag is set to 1 when A = 0x00; otherwise, it is cleared to 0.			
LD r,g	1 1 1 0 1 g g g 0 1 0 0 0 r r r	1 Z – – – –	2	r ← g
	Load the contents of an 8-bit register g into 8-bit register r. The Zero flag is set to 1 when g = 0x00; otherwise, it is cleared to 0.			
LD rr,gg	1 1 1 0 1 g g g 0 1 0 0 1 r r r	1 – – – – –	2	rr ← gg
	Load the contents of a 16-bit register pair gg (WA, BC, DE or HL) into 16-bit register pair rr. Example: Assume DE contains 0x1234. Then, the instruction “LD HL, DE” loads 0x1234 into HL.			
LD A,(x)	0 0 0 0 1 1 0 0 x x x x x x x x	1 Z – – – –	3	A ← (x)
	Load the contents of the memory location directly addressed by x into A. The Zero flag is set to 1 when the loaded value is 0x00. (0x0000 ≤ x ≤ 0x00FF)			
LD A,(HL)	0 0 0 0 1 1 0 1	1 Z – – – –	2	A ← (HL)
	Load the contents of the memory location addressed by HL into A. The Zero flag is set to 1 when the loaded value is 0x00. (0x0000 ≤ (HL) ≤ 0xFFFF)			
LD r,(x)	1 1 1 0 0 0 0 0 x x x x x x x x 0 1 0 0 0 r r r	1 Z – – – –	4	r ← (x)
	Load the contents of the memory location directly addressed by x into 8-bit register r. The Zero flag is set to 1 when the loaded value is 0x00. (0x0000 ≤ x ≤ 0x00FF)			
LD r,(vw)	1 1 1 0 0 0 0 1 w w w w w w w w v v v v v v v v 0 1 0 0 0 r r r	1 Z – – – –	5	r ← (vw)
	Load the contents of the memory location directly addressed by vw into 8-bit register r. The Zero flag is set to 1 when the loaded value is 0x00. (0x0000 ≤ vw ≤ 0xFFFF)			
LD r,(DE)	1 1 1 0 0 0 1 0 0 1 0 0 0 r r r	1 Z – – – –	3	r ← (DE)
	Load the contents of the memory location addressed by DE into 8-bit register r. The Zero flag is set to 1 when the loaded value is 0x00. (0x0000 ≤ (DE) ≤ 0xFFFF)			
LD r,(HL)	1 1 1 0 0 0 1 1 0 1 0 0 0 r r r	1 Z – – – –	3	r ← (HL)
	Load the contents of the memory location addressed by HL into 8-bit register r. The Zero flag is set to 1 when the loaded value is 0x00. (0x0000 ≤ (HL) ≤ 0xFFFF)			
LD r,(IX)	1 1 1 0 0 1 0 0 0 1 0 0 0 r r r	1 Z – – – –	3	r ← (IX)
	Load the contents of the memory location addressed by the Index register IX into 8-bit register r. The Zero flag is set to 1 when the loaded value is 0x00. (0x0000 ≤ (IX) ≤ 0xFFFF)			
LD r,(IY)	1 1 1 0 0 1 0 1 0 1 0 0 0 r r r	1 Z – – – –	3	r ← (IY)
	Load the contents of the memory location addressed by the Index register IY into 8-bit register r. The Zero flag is set to 1 when the loaded value is 0x00. (0x0000 ≤ IY ≤ 0xFFFF)			
LD r,(IX+d)	1 1 0 1 0 1 0 0 d d d d d d d d 0 1 0 0 0 r r r	1 Z – – – –	5	r ← (IX+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Load the contents of the memory location at the EA into 8-bit register r. The Zero flag is set to 1 when the loaded value is 0x00.			
LD r,(IY+d)	1 1 0 1 0 1 0 1 d d d d d d d d 0 1 0 0 0 r r r	1 Z – – – –	5	r ← (IY+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Load the contents of the memory location at the EA into 8-bit register r. The Zero flag is set to 1 when the loaded value is 0x00.			
LD r,(SP+d)	1 1 0 1 0 1 1 0 d d d d d d d d 0 1 0 0 0 r r r	1 Z – – – –	5	r ← (SP+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Load the contents of the memory location at the EA into 8-bit register r. The Zero flag is set to 1 when the loaded value is 0x00.			
LD r,(HL+d)	1 1 0 1 0 1 1 1 d d d d d d d d 0 1 0 0 0 r r r	1 Z – – – –	5	r ← (HL + d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Load the contents of the memory location at the EA into 8-bit register r. The Zero flag is set to 1 when the loaded value is 0x00.			
LD r,(HL+C)	1 1 1 0 0 1 1 1 0 1 0 0 0 r r r	1 Z – – – –	5	r ← (HL+C)
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Load the contents of the memory location at the EA into 8-bit register r. The Zero flag is set to 1 when the loaded value is 0x00.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
LD r,(+SP)	1 1 1 0 0 1 1 0 | 0 1 0 0 0 r r r	1 Z – – – –	4	SP ← SP+1:r ← (SP)
	Load the contents of the memory location at (SP+1) into 8-bit register r. The Zero flag is set to 1 when the loaded value is 0x00. This instruction is used to pop 8-bit data off the stack.			
LD r,(PC+A)[Note]	0 1 0 0 1 1 1 1 | 0 1 0 0 0 r r r	1 Z – – – –	5	r ← (PC+A)
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Load the contents of the memory location at the EA into 8-bit register r. The Zero flag is set to 1 when the loaded value is 0x00. This instruction is useful for code conversions.			
LD rr,(x)	1 1 1 0 0 0 0 0 | x x x x x x x x | 0 1 0 0 1 r r r	1 – – – – –	5	rr ← (x+1, x)
	Load 2 consecutive bytes from the memory location directly addressed by x into 16-bit register pair rr. (0x0000 ≤ x ≤ 0x00FF) Example: Assume the memory locations at addresses 0x0072 and 0x0073 contain 0x8E and 0x59 respectively. Then, the instruction “LD WA, (0x72)” loads 0x59 and 0x8E into the registers W and A respectively.			
LD rr,(vw)	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v 0 1 0 0 1 r r r	1 – – – – –	6	rr ← (vw+1, vw)
	Load 2 consecutive bytes from the memory location directly addressed by vw into 16-bit register pair rr. This instruction loads the contents of the memory location at address 0x1000 into the high-order byte of rr, when vw = 0x0FFF. (0x0000 ≤ vw ≤ 0xFFFF)			
LD rr,(DE)	1 1 1 0 0 0 1 0 | 0 1 0 0 1 r r r	1 – – – – –	4	rr ← (DE+1, DE)
	Load 2 consecutive bytes from the memory location addressed by DE into 16-bit register pair rr.			
LD rr,(HL)	1 1 1 0 0 0 1 1 | 0 1 0 0 1 r r r	1 – – – – –	4	rr ← (HL+1, HL)
	Load 2 consecutive bytes from the memory location addressed by HL into 16-bit register pair rr.			
LD rr,(IX)	1 1 1 0 0 1 0 0 | 0 1 0 0 1 r r r	1 – – – – –	4	rr ← (IX+1, IX)
	Load 2 consecutive bytes from the memory location addressed by IX into 16-bit register pair rr.			
LD rr,(IY)	1 1 1 0 0 1 0 1 | 0 1 0 0 1 r r r	1 – – – – –	4	rr ← (IY+1, IY)
	Load 2 consecutive bytes from the memory location addressed by IY into 16-bit register pair rr.			
LD rr,(IX+d)	1 1 0 1 0 1 0 0 | d d d d d d d d | 0 1 0 0 1 r r r	1 – – – – –	6	rr ← (IX+d+1, IX+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Load 2 consecutive bytes from the memory location addressed by the EA into 16-bit register pair rr.			
LD rr,(IY+d)	1 1 0 1 0 1 0 1 | d d d d d d d d | 0 1 0 0 1 r r r	1 – – – – –	6	rr ← (IY+d+1, IY+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Load 2 consecutive bytes from the memory location addressed by the EA into 16-bit register pair rr.			
LD rr,(SP+d)	1 1 0 1 0 1 1 0 | d d d d d d d d | 0 1 0 0 1 r r r	1 – – – – –	6	rr ← (SP+d+1, SP+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Load 2 consecutive bytes from the memory location addressed by the EA into 16-bit register pair rr. Example: Assume SP contains 0x51E4 and the memory locations at addresses 0x5216 and 0x5217 contain 0x9F and 0xC3 respectively. Then, the instruction “LD WA, (SP + 0x32)” loads 0x9F and 0xC3 into A and W respectively.			
LD rr,(HL+d)	1 1 0 1 0 1 1 1 | d d d d d d d d | 0 1 0 0 1 r r r	1 – – – – –	6	rr ← (HL+d+1, HL+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Load 2 consecutive bytes from the memory location addressed by the EA into 16-bit register pair rr.			
LD rr,(HL+C)	1 1 1 0 0 1 1 1 | 0 1 0 0 1 r r r	1 – – – – –	6	rr ← (HL+C+1, HL+C)
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Load 2 consecutive bytes from the memory location addressed by the EA into 16-bit register pair rr.			
LD rr,(+SP)	1 1 1 0 0 1 1 0 | 0 1 0 0 1 r r r	1 – – – – –	5	SP ← SP+1:rr ← (SP+1, SP)
	Load 2 consecutive bytes from the memory location addressed by (SP+1) into 16-bit register pair rr.			
LD rr,(PC+A)[Note]	0 1 0 0 1 1 1 1 | 0 1 0 0 1 r r r	1 – – – – –	6	rr ← (PC+A+1, PC+A)
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Load 2 consecutive bytes from the memory location addressed by the EA into 16-bit register pair rr.			
LD (x),A	0 0 0 0 1 1 1 0 | x x x x x x x x	1 – – – – –	3	(x) ← A
	Store the contents of A into the memory location directly addressed by x. (0x0000 ≤ x ≤ 0x00FF)			
LD (HL),A	0 0 0 0 1 1 1 1	1 – – – – –	2	(HL) ← A
	Store the contents of A into the memory location addressed by HL. (0x0000 ≤ (HL) ≤ 0xFFFF)			
LD (x),r	1 1 1 1 0 0 0 0 | x x x x x x x x | 0 1 1 1 1 r r r	1 – – – – –	4	(x) ← r
	Store the contents of 8-bit register r into the memory location directly addressed by x. (0x0000 ≤ x ≤ 0x00FF)			
LD (vw),r	1 1 1 1 0 0 0 1 | w w w w w w w w | v v v v v v v v 0 1 1 1 1 r r r	1 – – – – –	5	(vw) ← r
	Store the contents of 8-bit register r into the memory location directly addressed by vw. (0x0000 ≤ vw ≤ 0xFFFF)			
LD (DE),r	1 1 1 1 0 0 1 0 | 0 1 1 1 1 r r r	1 – – – – –	3	(DE) ← r
	Store the contents of 8-bit register r into the memory location addressed by DE. (0x0000 ≤ (DE) ≤ 0xFFFF)			
LD (HL),r	1 1 1 1 0 0 1 1 | 0 1 1 1 1 r r r	1 – – – – –	3	(HL) ← r
	Store the contents of 8-bit register r into the memory location addressed by HL. (0x0000 ≤ (HL) ≤ 0xFFFF)			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
LD (IX),r	1 1 1 1 0 1 0 0 | 0 1 1 1 1 r r r	1 – – – – –	3	(IX) ← r
	Store the contents of 8-bit register r into the memory location addressed by IX. (0x0000 ≤ (IX) ≤ 0xFFFF)			
LD (IY),r	1 1 1 1 0 1 0 1 | 0 1 1 1 1 r r r	1 – – – – –	3	(IY) ← r
	Store the contents of 8-bit register r into the memory location addressed by IY. (0x0000 ≤ (IY) ≤ 0xFFFF)			
LD (IX+d),r	0 1 0 1 0 1 0 0 | d d d d d d d d | 0 1 1 1 1 r r r	1 – – – – –	4	(IX+d) ← r
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Store the contents of the memory location at the EA into the 8-bit register r.			
LD (IY+d),r	0 1 0 1 0 1 0 1 | d d d d d d d d | 0 1 1 1 1 r r r	1 – – – – –	4	(IY+d) ← r
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Store the contents of the memory location at the EA into the 8-bit register r.			
LD (SP+d),r	0 1 0 1 0 1 1 0 | d d d d d d d d | 0 1 1 1 1 r r r	1 – – – – –	4	(SP+d) ← r
	Sign-extend the 8-bit data d in the instruction code and add the result to SP to form an effective address (EA). Store the contents of the memory location at the EA into the 8-bit register r.			
LD (HL+d),r	0 1 0 1 0 1 1 1 | d d d d d d d d | 0 1 1 1 1 r r r	1 – – – – –	4	(HL+d) ← r
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Store the contents of the memory location at the EA into the 8-bit register r.			
LD (HL+C),r	1 1 1 1 0 1 1 1 | 0 1 1 1 1 r r r	1 – – – – –	5	(HL+C) ← r
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Store the contents of the memory location at the EA into the 8-bit register r.			
LD (SP–),r	1 1 1 1 0 1 1 0 | 0 1 1 1 1 r r r	1 – – – – –	4	(SP) ← r:SP ← SP–1
	Store the contents of 8-bit register r into the memory location addressed by SP. Then, decrement the contents of SP. This instruction is used to push 8-bit data onto the stack.			
LD (x),rr	1 1 1 1 0 0 0 0 | x x x x x x x x | 0 1 1 0 1 r r r	1 – – – – –	5	(x+1, x) ← rr
	Store the contents of 16-bit register pair rr into 2 consecutive memory locations directly addressed by x with the low-order byte first, followed by the high-order byte. (0x0000 ≤ x ≤ 0x00FF)			
LD (vw),rr	1 1 1 1 0 0 0 1 | w w w w w w w w | v v v v v v v v | 0 1 1 0 1 r r r	1 – – – – –	6	(vw+1, vw) ← rr
	Store the contents of 16-bit register pair rr into 2 consecutive memory locations directly addressed by vw with the low-order byte first, followed by the high-order byte. (0x0000 ≤ vw ≤ 0xFFFF)			
LD (DE),rr	1 1 1 1 0 0 1 0 | 0 1 1 0 1 r r r	1 – – – – –	4	(DE+1, DE) ← rr
	Store the contents of 16-bit register pair rr into 2 consecutive memory locations addressed by DE with the low-order byte first, followed by the high-order byte.			
LD (HL),rr	1 1 1 1 0 0 1 1 | 0 1 1 0 1 r r r	1 – – – – –	4	(HL+1, HL) ← rr
	Store the contents of 16-bit register pair rr into 2 consecutive memory locations addressed by HL with the low-order byte first, followed by the high-order byte.			
LD (IX),rr	1 1 1 1 0 1 0 0 | 0 1 1 0 1 r r r	1 – – – – –	4	(IX+1, IX) ← rr
	Store the contents of 16-bit register pair rr into 2 consecutive memory locations addressed by IX with the low-order byte first, followed by the high-order byte.			
LD (IY),rr	1 1 1 1 0 1 0 1 | 0 1 1 0 1 r r r	1 – – – – –	4	(IY+1, IY) ← rr
	Store the contents of 16-bit register pair rr into 2 consecutive memory locations addressed by IY with the low-order byte first, followed by the high-order byte.			
LD (IX+d),rr	0 1 0 1 0 1 0 0 | d d d d d d d d | 0 1 1 0 1 r r r	1 – – – – –	5	(IX+d+1, IX+d) ← rr
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Store the contents of 16-bit register pair rr into 2 consecutive memory locations at the EA with the low-order byte first, followed by the high-order byte.			
LD (IY+d),rr	0 1 0 1 0 1 0 1 | d d d d d d d d | 0 1 1 0 1 r r r	1 – – – – –	5	(IY+d+1, IY+d) ← rr
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Store the contents of 16-bit register pair rr into 2 consecutive memory locations at the EA with the low-order byte first, followed by the high-order byte.			
LD (SP+d),rr	0 1 0 1 0 1 1 0 | d d d d d d d d | 0 1 1 0 1 r r r	1 – – – – –	5	(SP+d+1, SP+d) ← rr
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Store the contents of 16-bit register pair rr into 2 consecutive memory locations at the EA with the low-order byte first, followed by the high-order byte.			
LD (HL+d),rr	0 1 0 1 0 1 1 1 | d d d d d d d d | 0 1 1 0 1 r r r	1 – – – – –	5	(HL+d+1, HL+d) ← rr
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Store the contents of 16-bit register pair rr into 2 consecutive memory locations addressed by the EA with the low-order byte first, followed by the high-order byte.			
LD (HL+C),rr	1 1 1 1 0 1 1 1 | 0 1 1 0 1 r r r	1 – – – – –	6	(HL+C+1, HL+C) ← rr
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Store the contents of 16-bit register pair rr into 2 consecutive memory locations addressed by the EA with the low-order byte first, followed by the high-order byte.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
LD (SP−),rr	1 1 1 1 0 1 1 0 | 0 1 1 0 1 r r r	1 – – – – –	5	(SP+1, SP) ← rr:SP ← SP−1
	Store the contents of 16-bit register pair rr into 2 consecutive memory locations addressed by SP. Then, decrement the contents of SP.			
LD r,n	0 0 0 1 1 r r r | n n n n n n n n	1 – – – – –	2	r ← n
	Load the immediate n in the instruction code into 8-bit register r. Example: The instruction “LD A, 0x53” loads 0x53 into A.			
LD rr,mn	0 1 0 0 1 r r r | n n n n n n n n | m m m m m m m m	1 – – – – –	3	rr ← mn
	Load the immediate mn in the instruction code into 16-bit register pair rr. Example: The instruction “LD WA, 0x1234” loads 0x34 and 0x12 into A and W respectively.			
LD (x),n	0 0 0 0 1 0 1 0 | x x x x x x x x | n n n n n n n n	1 – – – – –	4	(x) ← n
	Store the immediate n in the instruction code into the memory location directly addressed by x. (0x0000 ≤ x ≤ 0x00FF)			
LD (vw),n	1 1 1 1 0 0 0 1 | w w w w w w w w | v v v v v v v v 1 1 1 1 1 0 0 1 | n n n n n n n n	1 – – – – –	6	(vw) ← n
	Store the immediate n in the instruction code into the memory location directly addressed by vw. (0x0000 ≤ vw ≤ 0xFFFF)			
LD (DE),n	1 1 1 1 0 0 1 0 | 1 1 1 1 1 0 0 1 | n n n n n n n n	1 – – – – –	4	(DE) ← n
	Store the immediate n in the instruction code into the memory location addressed by DE.			
LD (HL),n	0 0 0 0 1 0 1 1 | n n n n n n n n	1 – – – – –	3	(HL) ← n
	Store the immediate n in the instruction code into the memory location addressed by HL.			
LD (IX),n	1 1 1 1 0 1 0 0 | 1 1 1 1 1 0 0 1 | n n n n n n n n	1 – – – – –	4	(IX) ← n
	Store the immediate n in the instruction code into the memory location addressed by IX.			
LD (IY),n	1 1 1 1 0 1 0 1 | 1 1 1 1 1 0 0 1 | n n n n n n n n	1 – – – – –	4	(IY) ← n
	Store the immediate n in the instruction code into the memory location addressed by IY.			
LD (IX+d),n	0 1 0 1 0 1 0 0 | d d d d d d d d | 1 1 1 1 1 0 0 1 n n n n n n n n	1 – – – – –	5	(IX+d) ← n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Store the immediate n in the instruction code into the memory location at the EA.			
LD (IY+d),n	0 1 0 1 0 1 0 1 | d d d d d d d d | 1 1 1 1 1 0 0 1 n n n n n n n n	1 – – – – –	5	(IY+d) ← n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Store the immediate n in the instruction code into the memory location at the EA.			
LD (SP+d),n	0 1 0 1 0 1 1 0 | d d d d d d d d | 1 1 1 1 1 0 0 1 n n n n n n n n	1 – – – – –	5	(SP+d) ← n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Store the immediate n in the instruction code into the memory location at the EA.			
LD (HL+d),n	0 1 0 1 0 1 1 1 | d d d d d d d d | 1 1 1 1 1 0 0 1 n n n n n n n n	1 – – – – –	5	(HL+d) ← n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Store the immediate n in the instruction code into the memory location at the EA.			
LD (HL+C),n	1 1 1 1 0 1 1 1 | 1 1 1 1 1 0 0 1 | n n n n n n n n	1 – – – – –	6	(HL+C) ← n
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Store the immediate n in the instruction code into the memory location at the EA.			
LD (SP−),n	1 1 1 1 0 1 1 0 | 1 1 1 1 1 0 0 1 | n n n n n n n n	1 – – – – –	5	(SP) ← n:SP ← SP−1
	Store the immediate n in the instruction code into the memory location addressed by SP. Then, decrement the contents of the SP.			
LDW (x),mn	0 0 0 0 1 0 0 0 | x x x x x x x x | n n n n n n n n m m m m m m m m	1 – – – – –	6	(x+1, x) ← mn
	Store the 16-bit immediate mn into 2 consecutive memory locations directly addressed by x with the low-order byte first, followed by the high-order byte. (0x0000 ≤ x ≤ 0x00FF) Example: The instruction “LDW (0x73), 0x1234” stores 0x34 and 0x12 into memory locations at addresses 0x0073 and 0x0074 respectively.			
LDW (HL),mn	0 0 0 0 1 0 0 1 | n n n n n n n n | m m m m m m m m	1 – – – – –	5	(HL+1, HL) ← mn
	Store the 16-bit immediate mn into 2 consecutive memory locations addressed by HL with the low-order byte first, followed by the high-order byte.			
PUSH rr[#1]	0 1 0 1 0 0 r r	– – – – – –	3	(SP, SP−1) ← rr:SP ← SP−2
	Store the high-order byte of rr into the memory location addressed by SP, and the low-order byte into the memory location at (SP-1). Then, decrement the contents of SP by 2. rr must be WA, BC, DE or HL. Example: Assume SP and IX contain 0x013F and 0x1234 respectively. Then, the instruction “PUSH IX” stores 0x12 and 0x34 into the memory locations at addresses 0x013F and 0x013E respectively. At the same time, the contents of SP is decremented to 0x013D.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
PUSH gg[#2]	1 1 1 0 1 g g g | 1 1 0 1 1 0 0 0	– – – – – –	4	(SP, SP–1) ← gg:SP ← SP–2
	Store the high-order byte of gg into the memory location addressed by SP, and the low-order byte into the memory location at (SP-1). Then, decrement the contents of SP by 2. Example: Assume SP and IX contain 0x013F and 0x1234 respectively. Then, the instruction “PUSH IX” stores 0x12 and 0x34 into the memory locations at addresses 0x013F and 0x013E respectively. At the same time, the contents of SP is decremented to 0x013D.			
POP rr[#1]	1 1 0 1 0 0 r r	– – – – – –	4	SP ← SP+2:rr ← (SP, SP–1)
	Load the contents of the memory location at (SP+2) into the high-order 8 bits of rr. Load the contents of the memory location at (SP+1) into the low-order 8 bits of rr. rr must be WA, BC, DE or HL.			
POP gg[#2]	1 1 1 0 1 g g g | 1 1 0 1 1 0 0 1	– – – – – –	5	SP ← SP+2:gg ← (SP, SP–1)
	Load the contents of the memory location at (SP+2) into the high-order 8 bits of gg. Load the contents of the memory location at (SP+1) into the low-order 8 bits of gg.			
PUSH PSW	1 1 1 0 1 0 0 0 | 1 1 0 1 1 1 0 0	– – – – – –	3	(SP) ← PSW:SP ← SP–1
	Store the contents of PSW into the memory location addressed by SP. Then, decrement the contents of SP. Example: Assume SP contains 0x2345 and PSW contains 0x62 (JF = HF = SF = VF =0, ZF = CF = RBS = 1). Then, the instruction “PUSH PSW” stores 0x62 into the memory location at address 0x2345. At the same time, contents of SP is decremented to 0x2344.			
POP PSW	1 1 1 0 1 0 0 0 | 1 1 0 1 1 1 0 1	* * * * * *	4	SP ← SP+1:PSW ← (SP)
	Load the contents of the memory location at (SP+1) into PSW. Example: Assume the memory location at address 0x0137 and SP contain 0x63 and 0x0136 respectively. Then, the instruction “POP PSW” increments the contents of SP to 0x0137 and loads 0x62 (JF = HF = SF = VF =0, ZF = CF = RBS = 1) into PSW.			
LD PSW,n	1 1 1 0 1 0 0 0 | 1 1 0 1 1 1 1 0 | n n n n n n n n	* * * * * *	3	PSW ← n.7-1
	Load the immediate n in the instruction code into PSW. Each bit in the immediate n, beginning with the most significant bit, corresponds to the JF, ZF, CF, HF, SF, VF and RBS respectively; the least significant bit is ignored. Example: The “LD FLAG, 0y00110100” sets the flag bits as follows: JF = 0, ZF = 0, CF = 1, HF = 1, SF = 0, VF = 1 and RBS = 0.			
LD RBS,0	1 1 1 1 1 0 0 1 | 0 0 0 0 0 0 0 0	– – – – – –	2	RBS ← 0
	Load a 0 into the RBS bit in PSW to select the register bank BANK0. This instruction does not affect the flag bits.			
LD RBS,1	1 1 1 1 1 0 0 1 | 0 0 0 0 0 0 1 0	– – – – – –	2	RBS ← 1
	Load a 1 into the RBS bit in PSW to select the register bank BANK1. This instruction does not affect the flag bits.			
LD SP,SP+d	0 0 1 1 0 1 1 1 | d d d d d d d d	1 – – – – –	2	SP ← SP+d
	Add the 8-bit displacement d in the instruction code to the contents of SP and write back the sum into SP. This instruction operates like an add immediate instruction for SP. Example: Assume SP contains 0x2345. The instruction “LD SP, SP + 4” loads 0x2349 into SP and sets JF to 1.			
LD SP,SP–d	0 0 1 1 1 1 1 1 | d d d d d d d d	1 – – – – –	2	SP ← SP–d
	Subtract the 8-bit displacement d in the instruction code from the contents of SP and write back the result into SP. This instruction operates like an subtract immediate instruction for SP.			
XCH r,g	1 1 1 0 1 g g g | 0 1 1 1 0 r r r	1 Z – – – –	3	r ↔ g
	Exchange the contents of 8-bit register r with that of 8-bit register g. The Zero flag is set to 1 when the contents of g before the instruction execution is 0x00; otherwise, it is cleared to 0. Example: Assume A and B contain 0x3C and 0x5C respectively. Then, the instruction “XCH A, B” stores 0x5F and 0x3C into A and B, and clears ZF to 0.			
XCH rr,gg	1 1 1 0 1 g g g | 0 1 1 1 1 r r r	1 – – – – –	3	rr ↔ gg
	Exchange the contents of 16-bit register pair rr with that of 16-bit register pair gg. The Zero flag remains unchanged. Example: Assume HL and DE contain 0x0123 and 0x9587 respectively. Then, the instruction “XCH HL, DE” stores 0x9587 and 0x0123 into HL and DE.			
XCH r,(x)	1 1 1 0 0 0 0 0 | x x x x x x x x | 0 1 1 1 0 r r r	1 Z – – – –	5	r ↔ (x)
	Exchange the contents of the memory location directly addressed by x with that of 8-bit register r. The Zero flag is set to 1 when the value stored into r register is 0x00. (0x0000 ≤ x ≤ 0x00FF)			
XCH r,(vw)	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v 0 1 1 1 0 r r r	1 Z – – – –	6	r ↔ (vw)
	Exchange the contents of the memory location directly addressed by vw with that of 8-bit register r. The Zero flag is set to 1 when the value stored into r register is 0x00. (0x0000 ≤ vw ≤ 0xFFFF)			
XCH r,(DE)	1 1 1 0 0 0 1 0 | 0 1 1 1 0 r r r	1 Z – – – –	4	r ↔ (DE)
	Exchange the contents of the memory location addressed by DE with that of 8-bit register r. The Zero flag is set to 1 when the value stored into r register is 0x00.			
XCH r,(HL)	1 1 1 0 0 0 1 1 | 0 1 1 1 0 r r r	1 Z – – – –	4	r ↔ (HL)
	Exchange the contents of the memory location addressed by HL with that of 8-bit register r. The Zero flag is set to 1 when the value stored into r register is 0x00.			
XCH r,(IX)	1 1 1 0 0 1 0 0 | 0 1 1 1 0 r r r	1 Z – – – –	4	r ↔ (IX)
	Exchange the contents of the memory location addressed by IX with that of 8-bit register r. The Zero flag is set to 1 when the value stored into r register is 0x00.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
XCH r,(IY)	1 1 1 0 0 1 0 1 | 0 1 1 1 0 r r r	1 Z – – – –	4	r ↔ (IY)
	Exchange the contents of the memory location addressed by IY with that of 8-bit register r. The Zero flag is set to 1 when the value stored into r register is 0x00.			
XCH r,(IX+d)	1 1 0 1 0 1 0 0 | d d d d d d d d | 0 1 1 1 0 r r r	1 Z – – – –	6	r ↔ (IX+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Exchange the contents of the memory location at the EA with that of 8-bit register r. The Zero flag is set to 1 when the value stored into r register is 0x00.			
XCH r,(IY+d)	1 1 0 1 0 1 0 1 | d d d d d d d d | 0 1 1 1 0 r r r	1 Z – – – –	6	r ↔ (IY+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Exchange the contents of the memory location at the EA with that of 8-bit register r. The Zero flag is set to 1 when the value stored into r register is 0x00.			
XCH r,(SP+d)	1 1 0 1 0 1 1 0 | d d d d d d d d | 0 1 1 1 0 r r r	1 Z – – – –	6	r ↔ (SP+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Exchange the contents of the memory location at the EA with that of 8-bit register r. The Zero flag is set to 1 when the value stored into r register is 0x00.			
XCH r,(HL+d)	1 1 0 1 0 1 1 1 | d d d d d d d d | 0 1 1 1 0 r r r	1 Z – – – –	6	r ↔ (HL+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Exchange the contents of the memory location at the EA with that of 8-bit register r. The Zero flag is set to 1 when the value stored into r register is 0x00.			
XCH r,(HL+C)	1 1 1 0 0 1 1 1 | 0 1 1 1 0 r r r	1 Z – – – –	6	r ↔ (HL+C)
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Exchange the contents of the memory location at the EA with that of 8-bit register r. The Zero flag is set to 1 when the value stored into r register is 0x00.			
XCH r, (+SP)	1 1 1 0 0 1 1 0 | 0 1 1 1 0 r r r	1 Z – – – –	5	SP ← SP+1:r ↔ (SP)
	Exchange the contents of the memory location at (SP+1) with that of 8-bit register r. The Zero flag is set to 1 when the value stored into r register is 0x00.			
XCH r,(PC+A)[Note]	0 1 0 0 1 1 1 1 | 0 1 1 1 0 r r r	1 Z – – – –	6	r ↔ (PC+A)
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Exchange the contents of the memory location at the EA with that of 8-bit register r. The Zero flag is set to 1 when the value stored into r register is 0x00.			
XCH rr,(x)	1 1 1 0 0 0 0 0 | x x x x x x x x | 1 1 0 1 1 r r r	1 – – – – –	7	rr ↔ (x+1, x)
	Exchange the 16-bit memory word directly addressed by x with the contents of 16-bit register pair rr. (0x0000 ≤ x ≤ 0x00FF)			
XCH rr,(vw)	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v 1 1 0 1 1 r r r	1 – – – – –	8	rr ↔ (vw+1, vw)
	Exchange the 16-bit memory word directly addressed by vw with the contents of 16-bit register pair rr. (0x0000 ≤ vw ≤ 0xFFFF)			
XCH rr,(DE)	1 1 1 0 0 0 1 0 | 1 1 0 1 1 r r r	1 – – – – –	6	rr ↔ (DE+1, DE)
	Exchange the 16-bit memory word addressed by DE with the contents of 16-bit register pair rr.			
XCH rr,(HL)	1 1 1 0 0 0 1 1 | 1 1 0 1 1 r r r	1 – – – – –	6	rr ↔ (HL+1, HL)
	Exchange the 16-bit memory word addressed by HL with the contents of 16-bit register pair rr.			
XCH rr,(IX)	1 1 1 0 0 1 0 0 | 1 1 0 1 1 r r r	1 – – – – –	6	rr ↔ (IX+1, IX)
	Exchange the 16-bit memory word addressed by IX with the contents of 16-bit register pair rr.			
XCH rr,(IY)	1 1 1 0 0 1 0 1 | 1 1 0 1 1 r r r	1 – – – – –	6	rr ↔ (IY+1, IY)
	Exchange the 16-bit memory word addressed by IY with the contents of 16-bit register pair rr.			
XCH rr,(IX+d)	1 1 0 1 0 1 0 0 | d d d d d d d d | 1 1 0 1 1 r r r	1 – – – – –	8	rr ↔ (IX+d+1, IX+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Exchange the 16-bit memory word addressed by the EA with the contents of 16-bit register pair rr.			
XCH rr,(IY+d)	1 1 0 1 0 1 0 1 | d d d d d d d d | 1 1 0 1 1 r r r	1 – – – – –	8	rr ↔ (IY+d+1, IY+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Exchange the 16-bit memory word addressed by the EA with the contents of 16-bit register pair rr.			
XCH rr,(SP+d)	1 1 0 1 0 1 1 0 | d d d d d d d d | 1 1 0 1 1 r r r	1 – – – – –	8	rr ↔ (SP+d+1, SP+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Exchange the 16-bit memory word addressed by the EA with the contents of 16-bit register pair rr.			
XCH rr,(HL+d)	1 1 0 1 0 1 1 1 | d d d d d d d d | 1 1 0 1 1 r r r	1 – – – – –	8	rr ↔ (HL+d+1, HL+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Exchange the 16-bit memory word addressed by the EA with the contents of 16-bit register pair rr.			
XCH rr,(HL+C)	1 1 1 0 0 1 1 1 | 1 1 0 1 1 r r r	1 – – – – –	8	rr ↔ (HL+C+1, HL+C)
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Exchange the 16-bit memory word addressed by the EA with the contents of 16-bit register pair rr.			
XCH rr,(+SP)	1 1 1 0 0 1 1 0 | 1 1 0 1 1 r r r	1 – – – – –	7	SP ← SP+1:rr ↔ (SP+1, SP)
	Exchange the 16-bit memory word addressed by (SP+1) with the contents of 16-bit register pair rr.			
XCH rr,(PC+A)[Note]	0 1 0 0 1 1 1 1 | 1 1 0 1 1 r r r	1 – – – – –	8	rr ↔ (PC+A+1, PC+A)
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Exchange the 16-bit memory word addressed by the EA with the contents of 16-bit register pair rr.			

#1 rr must be the register pair WA, BC, DE or HL.
#2 gg must be the register IX or IY.

Note: There are restrictions on instructions that uses the operand (PC + A). For more details, see "1.4 Addressing Mode".

2.2 ALU Instructions

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
CMP A,n	0 1 1 0 0 1 1 1 n n n n n n n n	Z Z C H S V	2	A–n
	Compare the contents of A with the immediate n in the instruction code. The Carry flag is set to 1 when A < n; otherwise, it is cleared to 0.			
CMP g,n	1 1 1 0 1 g g g 0 1 1 0 0 1 1 1 n n n n n n n n	Z Z C H S V	3	g–n
	Compare the contents of 8-bit register g with the immediate n in the instruction code. The Carry flag is set to 1 when g < n; otherwise, it is cleared to 0.			
CMP gg,mn	1 1 1 0 1 g g g 0 1 1 0 1 1 1 1 n n n n n n n n m m m m m m m m	Z Z C U S V	4	gg–mn
	Compare the contents of 16-bit register gg with the immediate mn in the instruction code. The Carry flag is set to 1 when gg < n; otherwise, it is cleared to 0.			
CMP r,g	1 1 1 0 1 g g g 0 0 r r r 1 1 1	Z Z C H S V	2	r–g
	Compare the contents of 8-bit register r with that of 8-bit register g. The Carry flag is set to 1 when r < g; otherwise, it is cleared to 0.			
CMP rr,gg	1 1 1 0 1 g g g 1 0 r r r 1 1 1	Z Z C U S V	3	rr–gg
	Compare the contents of 16-bit register rr with that of 16-bit register gg.			
CMP r,(x)	1 1 1 0 0 0 0 0 x x x x x x x x 0 0 r r r 1 1 1	Z Z C H S V	4	r–(x)
	Compare the contents of the memory location directly addressed by x with the contents of 8-bit register r. (0x0000 ≤ x ≤ 0x00FF)			
CMP r,(vw)	1 1 1 0 0 0 0 1 w w w w w w w w v v v v v v v v 0 0 r r r 1 1 1	Z Z C H S V	5	r–(vw)
	Compare the contents of the memory location directly addressed by vw with the contents of 8-bit register r. (0x0000 ≤ vw ≤ 0xFFFF)			
CMP r,(DE)	1 1 1 0 0 0 1 0 0 0 r r r 1 1 1	Z Z C H S V	3	r–(DE)
	Compare the contents of the memory location addressed by DE with the contents of 8-bit register r.			
CMP r,(HL)	1 1 1 0 0 0 1 1 0 0 r r r 1 1 1	Z Z C H S V	3	r–(HL)
	Compare the contents of the memory location addressed by HL with the contents of 8-bit register r.			
CMP r,(IX)	1 1 1 0 0 1 0 0 0 0 r r r 1 1 1	Z Z C H S V	3	r–(IX)
	Compare the contents of the memory location addressed by IX with the contents of 8-bit register r.			
CMP r,(IY)	1 1 1 0 0 1 0 1 0 0 r r r 1 1 1	Z Z C H S V	3	r–(IY)
	Compare the contents of the memory location addressed by IY with the contents of 8-bit register r.			
CMP r,(IX+d)	1 1 0 1 0 1 0 0 d d d d d d d d 0 0 r r r 1 1 1	Z Z C H S V	5	r–(IX+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Compare the contents of the memory location at the EA with the contents of 8-bit register r.			
CMP r,(IY+d)	1 1 0 1 0 1 0 1 d d d d d d d d 0 0 r r r 1 1 1	Z Z C H S V	5	r–(IY+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Compare the contents of the memory location at the EA with the contents of 8-bit register r.			
CMP r,(SP+d)	1 1 0 1 0 1 1 0 d d d d d d d d 0 0 r r r 1 1 1	Z Z C H S V	5	r–(SP+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Compare the contents of the memory location at the EA with the contents of 8-bit register r.			
CMP r,(HL+d)	1 1 0 1 0 1 1 1 d d d d d d d d 0 0 r r r 1 1 1	Z Z C H S V	5	r–(HL+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Compare the contents of the memory location at the EA with the contents of 8-bit register r.			
CMP r,(HL+C)	1 1 1 0 0 1 1 1 0 0 r r r 1 1 1	Z Z C H S V	5	r–(HL+C)
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Compare the contents of the memory location at the EA with the contents of 8-bit register r.			
CMP r,(+SP)	1 1 1 0 0 1 1 0 0 0 r r r 1 1 1	Z Z C H S V	4	SP ← SP+1:r–(SP)
	Compare the contents of the memory location at (SP+1) with the contents of 8-bit register r.			
CMP r,(PC+A)[Note]	0 1 0 0 1 1 1 1 0 0 r r r 1 1 1	Z Z C H S V	5	r–(PC+A)
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Compare the contents of the memory location at the EA with the contents of 8-bit register r.			
CMP (x),n	0 0 0 0 0 1 1 1 x x x x x x x x n n n n n n n n	Z Z C H S V	4	(x)–n
	Compare the contents of the memory location directly addressed by x with the immediate n in the instruction code. (0x0000 ≤ x ≤ 0x00FF)			
CMP (vw),n	1 1 1 0 0 0 0 1 w w w w w w w w v v v v v v v v 0 1 1 0 0 1 1 1 n n n n n n n n	Z Z C H S V	6	(vw)–n
	Compare the contents of the memory location directly addressed by vw with the immediate n in the instruction code. (0x0000 ≤ vw ≤ 0xFFFF)			
CMP (DE),n	1 1 1 0 0 0 1 0 0 1 1 0 0 1 1 1 n n n n n n n n	Z Z C H S V	4	(DE)–n
	Compare the contents of the memory location addressed by DE with the immediate n in the instruction code.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
CMP (HL),n	1 1 1 0 0 0 1 1 | 0 1 1 0 0 1 1 1 | n n n n n n n n	Z Z C H S V	4	(HL)–n
	Compare the contents of the memory location addressed by HL with the immediate n in the instruction code.			
CMP (IX),n	1 1 1 0 0 1 0 0 | 0 1 1 0 0 1 1 1 | n n n n n n n n	Z Z C H S V	4	(IX)–n
	Compare the contents of the memory location addressed by IX with the immediate n in the instruction code.			
CMP (IY),n	1 1 1 0 0 1 0 1 | 0 1 1 0 0 1 1 1 | n n n n n n n n	Z Z C H S V	4	(IY)–n
	Compare the contents of the memory location addressed by IY with the immediate n in the instruction code.			
CMP (IX+d),n	1 1 0 1 0 1 0 0 | d d d d d d d d | 0 1 1 0 0 1 1 1 | n n n n n n n n	Z Z C H S V	6	(IX+d)–n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Compare the contents of the memory location at the EA with the immediate n in the instruction code.			
CMP (IY+d),n	1 1 0 1 0 1 0 1 | d d d d d d d d | 0 1 1 0 0 1 1 1 | n n n n n n n n	Z Z C H S V	6	(IY+d)–n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Compare the contents of the memory location at the EA with the immediate n in the instruction code.			
CMP (SP+d),n	1 1 0 1 0 1 1 0 | d d d d d d d d | 0 1 1 0 0 1 1 1 | n n n n n n n n	Z Z C H S V	6	(SP+d)–n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Compare the contents of the memory location at the EA with the immediate n in the instruction code.			
CMP (HL+d),n	1 1 0 1 0 1 1 1 | d d d d d d d d | 0 1 1 0 0 1 1 1 | n n n n n n n n	Z Z C H S V	6	(HL+d)–n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Compare the contents of the memory location at the EA with the immediate n in the instruction code.			
CMP (HL+C),n	1 1 1 0 0 1 1 1 | 0 1 1 0 0 1 1 1 | n n n n n n n n	Z Z C H S V	6	(HL+C)–n
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Compare the contents of the memory location at the EA with the immediate n in the instruction code.			
CMP (+SP),n	1 1 1 0 0 1 1 0 | 0 1 1 0 0 1 1 1 | n n n n n n n n	Z Z C H S V	5	SP ← SP+1:(SP)–n
	Compare the contents of the memory location at (SP+1) and the immediate n in the instruction code.			
CMP (PC+A),n[Note]	0 1 0 0 1 1 1 1 | 0 1 1 0 0 1 1 1 | n n n n n n n n	Z Z C H S V	6	(PC+A)–n
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Compare the contents of the memory location at the EA with the immediate n in the instruction code.			
CMP rr,(x)	1 1 1 0 0 0 0 0 | x x x x x x x x | 1 0 r r r 1 1 1	Z Z C U S V	5	rr–(x)
	Compare the 16-bit memory word directly addressed by x with the contents of 16-bit register rr. (0x0000 ≤ x ≤ 0x00FF)			
CMP rr,(vw)	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v | 1 0 r r r 1 1 1	Z Z C U S V	6	rr–(vw)
	Compare the 16-bit memory word directly addressed by vw with the contents of 16-bit register rr. (0x0000 ≤ vw ≤ 0xFFFF)			
CMP rr,(DE)	1 1 1 0 0 0 1 0 | 1 0 r r r 1 1 1	Z Z C U S V	4	rr–(DE)
	Compare the 16-bit memory word addressed by DE with the contents of 16-bit register rr.			
CMP rr,(HL)	1 1 1 0 0 0 1 1 | 1 0 r r r 1 1 1	Z Z C U S V	4	rr–(HL)
	Compare the 16-bit memory word addressed by HL with the contents of 16-bit register rr.			
CMP rr,(IX)	1 1 1 0 0 1 0 0 | 1 0 r r r 1 1 1	Z Z C U S V	4	rr–(IX)
	Compare the 16-bit memory word addressed by IX with the contents of 16-bit register rr.			
CMP rr,(IY)	1 1 1 0 0 1 0 1 | 1 0 r r r 1 1 1	Z Z C U S V	4	rr–(IY)
	Compare the 16-bit memory word addressed by IY with the contents of 16-bit register rr.			
CMP rr,(IX+d)	1 1 0 1 0 1 0 0 | d d d d d d d d | 1 0 r r r 1 1 1	Z Z C U S V	6	rr–(IX+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Compare the 16-bit memory word addressed by the EA with the contents of 16-bit register rr.			
CMP rr,(IY+d)	1 1 0 1 0 1 0 1 | d d d d d d d d | 1 0 r r r 1 1 1	Z Z C U S V	6	rr–(IY+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Compare the 16-bit memory word addressed by the EA with the contents of 16-bit register rr.			
CMP rr,(SP+d)	1 1 0 1 0 1 1 0 | d d d d d d d d | 1 0 r r r 1 1 1	Z Z C U S V	6	rr–(SP+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Compare the 16-bit memory word addressed by the EA with the contents of 16-bit register rr.			
CMP rr,(HL+d)	1 1 0 1 0 1 1 1 | d d d d d d d d | 1 0 r r r 1 1 1	Z Z C U S V	6	rr–(HL+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Compare the 16-bit memory word addressed by the EA with the contents of 16-bit register rr.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
CMP rr,(HL+C)	1 1 1 0 0 1 1 1 1 0 r r r 1 1 1	Z Z C U S V	6	rr–(HL+C)
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Compare the 16-bit memory word addressed by the EA with the contents of 16-bit register rr.			
CMP rr,(+SP)	1 1 1 0 0 1 1 0 1 0 r r r 1 1 1	Z Z C U S V	5	SP ← SP+1:rr–(SP)
	Compare the 16-bit memory word addressed by (SP+1) with the contents of 16-bit register rr.			
CMP rr,(PC+A)[Note]	0 1 0 0 1 1 1 1 1 0 r r r 1 1 1	Z Z C U S V	6	rr–(PC+A)
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Compare the 16-bit memory word addressed by the EA with the contents of 16-bit register rr.			
ADD A,n	0 1 1 0 0 0 0 1 n n n n n n n n	C Z C H S V	2	A ← A+n
	Add the immediate n in the instruction code to the contents of A and write back the sum into A.			
ADD g,n	1 1 1 0 1 g g g 0 1 1 0 0 0 0 1 n n n n n n n n	C Z C H S V	3	g ← g+n
	Add the immediate n in the instruction code to the contents of 8-bit register g and write back the sum into g.			
ADD gg,mn	1 1 1 0 1 g g g 0 1 1 0 1 0 0 1 n n n n n n n n m m m m m m m m	C Z C U S V	4	gg ← gg+mn
	Add the immediate mn in the instruction code to the contents of 16-bit register gg and write back the sum into gg.			
ADD r,g	1 1 1 0 1 g g g 0 0 r r r 0 0 1	C Z C H S V	2	r ← r+g
	Add the contents of 8-bit register g to that of 8-bit register r and write back the sum into r.			
ADD rr,gg	1 1 1 0 1 g g g 1 0 r r r 0 0 1	C Z C U S V	3	rr ← rr+gg
	Add the contents of 16-bit register gg to that of 16-bit register rr and write back the sum into rr.			
ADD r,(x)	1 1 1 0 0 0 0 0 x x x x x x x x 0 0 r r r 0 0 1	C Z C H S V	4	r ← r+(x)
	Add the contents of the memory location directly addressed by x to that of 8-bit register r and write back the sum into r. (0x0000 ≤ x ≤ 0x00FF)			
ADD r,(vw)	1 1 1 0 0 0 0 1 w w w w w w w w v v v v v v v v 0 0 r r r 0 0 1	C Z C H S V	5	r ← r+(vw)
	Add the contents of the memory location directly addressed by vw to that of 8-bit register r and write back the sum into r. (0x0000 ≤ vw ≤ 0xFFFF)			
ADD r,(DE)	1 1 1 0 0 0 1 0 0 0 r r r 0 0 1	C Z C H S V	3	r ← r+(DE)
	Add the contents of the memory location addressed by DE to that of 8-bit register r and write back the sum into r.			
ADD r,(HL)	1 1 1 0 0 0 1 1 0 0 r r r 0 0 1	C Z C H S V	3	r ← r+(HL)
	Add the contents of the memory location addressed by HL to that of 8-bit register r and write back the sum into r.			
ADD r,(IX)	1 1 1 0 0 1 0 0 0 0 r r r 0 0 1	C Z C H S V	3	r ← r+(IX)
	Add the contents of the memory location addressed by IX to that of 8-bit register r and write back the sum into r.			
ADD r,(IY)	1 1 1 0 0 1 0 1 0 0 r r r 0 0 1	C Z C H S V	3	r ← r+(IY)
	Add the contents of the memory location addressed by IY to that of 8-bit register r and write back the sum into r.			
ADD r,(IX+d)	1 1 0 1 0 1 0 0 d d d d d d d d 0 0 r r r 0 0 1	C Z C H S V	5	r ← r+(IX+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Add the contents of the memory location at the EA to that of 8-bit register r and write back the sum into r.			
ADD r,(IY+d)	1 1 0 1 0 1 0 1 d d d d d d d d 0 0 r r r 0 0 1	C Z C H S V	5	r ← r+(IY+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Add the contents of the memory location at the EA to that of 8-bit register r and write back the sum into r.			
ADD r,(SP+d)	1 1 0 1 0 1 1 0 d d d d d d d d 0 0 r r r 0 0 1	C Z C H S V	5	r ← r+(SP+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Add the contents of the memory location at the EA to that of 8-bit register r and write back the sum into r.			
ADD r,(HL+d)	1 1 0 1 0 1 1 1 d d d d d d d d 0 0 r r r 0 0 1	C Z C H S V	5	r ← r+(HL+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Add the contents of the memory location at the EA to that of 8-bit register r and write back the sum into r.			
ADD r,(HL+C)	1 1 1 0 0 1 1 1 0 0 r r r 0 0 1	C Z C H S V	5	r ← r+(HL+C)
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Add the contents of the memory location at the EA to that of 8-bit register r and write back the sum into r.			
ADD r,(+SP)	1 1 1 0 0 1 1 0 0 0 r r r 0 0 1	C Z C H S V	4	SP ← SP+1:r ← r+(SP)
	Add the contents of the memory location at (SP+1) to the contents of 8-bit register r and write back the sum into r.			
ADD r,(PC+A)[Note]	0 1 0 0 1 1 1 1 0 0 r r r 0 0 1	C Z C H S V	5	r ← r+(PC+A)
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Add the contents of the memory location at the EA to that of 8-bit register r and write back the sum into r.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
ADD (x),n	1 1 1 0 0 0 0 0 x x x x x x x x 0 1 1 0 0 0 0 1 n n n n n n n n	C Z C H S V	6	(x) ← (x)+n
	Add the immediate n in the instruction code to the contents of the memory location directly addressed by x and write back the sum into the same location. (0x0000 ≤ x ≤ 0x00FF)			
ADD (vw),n	1 1 1 0 0 0 0 1 w w w w w w w w v v v v v v v v 0 1 1 0 0 0 0 1 n n n n n n n n	C Z C H S V	7	(vw) ← (vw)+n
	Add the immediate n in the instruction code to the contents of the memory location directly addressed by vw and write back the sum into the same location. (0x0000 ≤ vw ≤ 0xFFFF)			
ADD (DE),n	1 1 1 0 0 0 1 0 0 1 1 0 0 0 0 1 n n n n n n n n	C Z C H S V	5	(DE) ← (DE)+n
	Add the immediate n in the instruction code to the contents of the memory location addressed by DE and write back the sum into the same location.			
ADD (HL),n	1 1 1 0 0 0 1 1 0 1 1 0 0 0 0 1 n n n n n n n n	C Z C H S V	5	(HL) ← (HL)+n
	Add the immediate n in the instruction code to the contents of the memory location addressed by HL and write back the sum into the same location.			
ADD (IX),n	1 1 1 0 0 1 0 0 0 1 1 0 0 0 0 1 n n n n n n n n	C Z C H S V	5	(IX) ← (IX)+n
	Add the immediate n in the instruction code to the contents of the memory location addressed by IX and write back the sum into the same location.			
ADD (IY),n	1 1 1 0 0 1 0 1 0 1 1 0 0 0 0 1 n n n n n n n n	C Z C H S V	5	(IY) ← (IY)+n
	Add the immediate n in the instruction code to the contents of the memory location addressed by IY and write back the sum into the same location.			
ADD (IX+d),n	1 1 0 1 0 1 0 0 d d d d d d d d 0 1 1 0 0 0 0 1 n n n n n n n n	C Z C H S V	7	(IX+d) ← (IX+d)+n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Add the immediate n in the instruction code to the contents of the memory location at the EA and write back the sum into the same location.			
ADD (IY+d),n	1 1 0 1 0 1 0 1 d d d d d d d d 0 1 1 0 0 0 0 1 n n n n n n n n	C Z C H S V	7	(IY+d) ← (IY+d)+n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Add the immediate n in the instruction code to the contents of the memory location at the EA and write back the sum into the same location.			
ADD (SP+d),n	1 1 0 1 0 1 1 0 d d d d d d d d 0 1 1 0 0 0 0 1 n n n n n n n n	C Z C H S V	7	(SP+d) ← (SP+d)+n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Add the immediate n in the instruction code to the contents of the memory location at the EA and write back the sum into the same location.			
ADD (HL+d),n	1 1 0 1 0 1 1 1 d d d d d d d d 0 1 1 0 0 0 0 1 n n n n n n n n	C Z C H S V	7	(HL+d) ← (HL+d)+n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Add the immediate n in the instruction code to the contents of the memory location at the EA and write back the sum into the same location.			
ADD (HL+C),n	1 1 1 0 0 1 1 1 0 1 1 0 0 0 0 1 n n n n n n n n	C Z C H S V	7	(HL+C) ← (HL+C)+n
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Add the immediate n in the instruction code to the contents of the memory location at the EA and write back the sum into the same location.			
ADD (+SP),n	1 1 1 0 0 1 1 0 0 1 1 0 0 0 0 1 n n n n n n n n	C Z C H S V	6	SP ← SP+1:(SP) ← (SP)+n
	Add the immediate n in the instruction code to the contents of the memory location at (SP+1) and write back the sum into the same location.			
ADD (PC+A),n[Note]	0 1 0 0 1 1 1 1 0 1 1 0 0 0 0 1 n n n n n n n n	C Z C H S V	7	(PC+A) ← (PC+A)+n
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Add the immediate n in the instruction code to the contents of the memory location at the EA and write back the sum into the same location.			
ADD rr,(x)	1 1 1 0 0 0 0 0 x x x x x x x x 1 0 r r r 0 0 1	C Z C U S V	5	rr ← rr+(x+1, x)
	Add the 16-bit memory word directly addressed by x to the contents of 16-bit register rr and write back the sum into rr. (0x0000 ≤ x ≤ 0x00FF)			
ADD rr,(vw)	1 1 1 0 0 0 0 1 w w w w w w w w v v v v v v v v 1 0 r r r 0 0 1	C Z C U S V	6	rr ← rr+(vw+1, vw)
	Add the 16-bit memory word directly addressed by vw to the contents of 16-bit register rr and write back the sum into rr. (0x0000 ≤ vw ≤ 0xFFFF)			
ADD rr,(DE)	1 1 1 0 0 0 1 0 1 0 r r r 0 0 1	C Z C U S V	4	rr ← rr+(DE+1, DE)
	Add the 16-bit memory word addressed by DE to the contents of 16-bit register rr and write back the sum into rr.			
ADD rr,(HL)	1 1 1 0 0 0 1 1 1 0 r r r 0 0 1	C Z C U S V	4	rr ← rr+(HL+1, HL)
	Add the 16-bit memory word addressed by HL to the contents of 16-bit register rr and write back the sum into rr.			
ADD rr,(IX)	1 1 1 0 0 1 0 0 1 0 r r r 0 0 1	C Z C U S V	4	rr ← rr+(IX+1, IX)
	Add the 16-bit memory word addressed by IX to the contents of 16-bit register rr and write back the sum into rr.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
ADD rr,(IY)	1 1 1 0 0 1 0 1 1 0 r r r 0 0 1	C Z C U S V	4	rr ← rr+(IY+1, IY)
	Add the 16-bit memory word addressed by IY to the contents of 16-bit register rr and write back the sum into rr.			
ADD rr,(IX+d)	1 1 0 1 0 1 0 0 d d d d d d d d 1 0 r r r 0 0 1	C Z C U S V	6	rr ← rr+(IX+d+1, IX+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Add the 16-bit memory word addressed by the EA to the contents of 16-bit register rr and write back the sum into rr.			
ADD rr,(IY+d)	1 1 0 1 0 1 0 1 d d d d d d d d 1 0 r r r 0 0 1	C Z C U S V	6	rr ← rr+(IY+d+1, IY+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Add the 16-bit memory word addressed by the EA to the contents of 16-bit register rr and write back the sum into rr.			
ADD rr,(SP+d)	1 1 0 1 0 1 1 0 d d d d d d d d 1 0 r r r 0 0 1	C Z C U S V	6	rr ← rr+(SP+d+1, SP+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Add the 16-bit memory word addressed by the EA to the contents of 16-bit register rr and write back the sum into rr.			
ADD rr,(HL+d)	1 1 0 1 0 1 1 1 d d d d d d d d 1 0 r r r 0 0 1	C Z C U S V	6	rr ← rr+(HL+d+1, HL+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Add the 16-bit memory word addressed by the EA to the contents of 16-bit register rr and write back the sum into rr.			
ADD rr,(HL+C)	1 1 1 0 0 1 1 1 1 0 r r r 0 0 1	C Z C U S V	6	rr ← rr+(HL+C+1, HL+C)
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Add the 16-bit memory word addressed by the EA to the contents of 16-bit register rr and write back the sum into rr.			
ADD rr,(+SP)	1 1 1 0 0 1 1 0 1 0 r r r 0 0 1	C Z C U S V	5	SP ← SP+1:rr ← rr+(SP+1, SP)
	Add the 16-bit memory word addressed by (SP+1) to the contents of 16-bit register rr and write back the sum into rr.			
ADD rr,(PC+A)[Note]	0 1 0 0 1 1 1 1 1 0 r r r 0 0 1	C Z C U S V	6	rr ← rr+(PC+A+1, PC+A)
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Add the 16-bit memory word addressed by the EA to the contents of 16-bit register rr and write back the sum into rr.			
ADDC A,n	0 1 1 0 0 0 0 0 n n n n n n n n	C Z C H S V	2	A ← A+n+CF
	Add both the immediate n in the instruction code and the contents of the Carry flag to A and write back the sum into A.			
ADDC g,n	1 1 1 0 1 g g g 0 1 1 0 0 0 0 0 n n n n n n n n	C Z C H S V	3	g ← g+n+CF
	Add both the immediate n in the instruction code and the contents of the Carry flag to the contents of 8-bit register g and write back the sum into g.			
ADDC gg,mn	1 1 1 0 1 g g g 0 1 1 0 1 0 0 0 n n n n n n n n m m m m m m m m	C Z C U S V	4	gg ← gg+mn+CF
	Add both the immediate mn in the instruction code and the contents of the Carry flag to the contents of 16-bit register gg and write back the sum into gg.			
ADDC r,g	1 1 1 0 1 g g g 0 0 r r r 0 0 0	C Z C H S V	2	r ← r+g+CF
	Add both the contents of 8-bit register g and that of the Carry flag to the contents of 8-bit register r and write back the sum into r.			
ADDC rr,gg	1 1 1 0 1 g g g 1 0 r r r 0 0 0	C Z C U S V	3	rr ← rr+gg+CF
	Add both the contents of 16-bit register gg and that of the Carry flag to the contents of 16-bit register rr and write back the sum into rr.			
ADDC r,(x)	1 1 1 0 0 0 0 0 x x x x x x x x 0 0 r r r 0 0 0	C Z C H S V	4	r ← r+(x)+CF
	Add both the contents of the memory location directly addressed by x and that of the Carry flag to the contents of 8-bit register r and write back the sum into r. (0x0000 ≤ x ≤ 0x00FF)			
ADDC r, (vw)	1 1 1 0 0 0 0 1 w w w w w w w w v v v v v v v v 0 0 r r r 0 0 0	C Z C H S V	5	r ← r+(vw)+CF
	Add both the contents of the memory location directly addressed by vw and that of the Carry flag to the contents of 8-bit register r and write back the sum into r. (0x0000 ≤ vw ≤ 0xFFFF)			
ADDC r,(DE)	1 1 1 0 0 0 1 0 0 0 r r r 0 0 0	C Z C H S V	3	r ← r+(DE)+CF
	Add both the contents of the memory location addressed by DE and that of the Carry flag to the contents of 8-bit register r and write back the sum into r.			
ADDC r,(HL)	1 1 1 0 0 0 1 1 0 0 r r r 0 0 0	C Z C H S V	3	r ← r+(HL)+CF
	Add both the contents of the memory location addressed by HL and that of the Carry flag to the contents of 8-bit register r and write back the sum into r.			
ADDC r,(IX)	1 1 1 0 0 1 0 0 0 0 r r r 0 0 0	C Z C H S V	3	r ← r+(IX)+CF
	Add both the contents of the memory location addressed by IX and that of the Carry flag to the contents of 8-bit register r and write back the sum into r.			
ADDC r,(IY)	1 1 1 0 0 1 0 1 0 0 r r r 0 0 0	C Z C H S V	3	r ← r+(IY)+CF
	Add both the contents of the memory location addressed by IY and that of the Carry flag to the contents of 8-bit register r and write back the sum into r.			
ADDC r,(IX+d)	1 1 0 1 0 1 0 0 d d d d d d d d 0 0 r r r 0 0 0	C Z C H S V	5	r ← r+(IX+d)+CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Add both the contents of the memory location at the EA and that of the Carry flag to the contents of 8-bit register r and write back the sum into r.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
ADDC r,(IY+d)	1 1 0 1 0 1 0 1 d d d d d d d d 0 0 r r r 0 0 0	C Z C H S V	5	r ← r+(IY+d)+CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Add both the contents of the memory location at the EA and that of the Carry flag to the contents of 8-bit register r and write back the sum into r.			
ADDC r,(SP+d)	1 1 0 1 0 1 1 0 d d d d d d d d 0 0 r r r 0 0 0	C Z C H S V	5	r ← r+(SP+d)+CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Add both the contents of the memory location at the EA and that of the Carry flag to the contents of 8-bit register r and write back the sum into r.			
ADDC r,(HL+d)	1 1 0 1 0 1 1 1 d d d d d d d d 0 0 r r r 0 0 0	C Z C H S V	5	r ← r+(HL+d)+CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Add both the contents of the memory location at the EA and that of the Carry flag to the contents of 8-bit register r and write back the sum into r.			
ADDC r,(HL+C)	1 1 1 0 0 1 1 1 0 0 r r r 0 0 0	C Z C H S V	5	r ← r+(HL+C)+CF
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Add both the contents of the memory location at the EA and that of the Carry flag to the contents of 8-bit register r and write back the sum into r.			
ADDC r,(+SP)	1 1 1 0 0 1 1 0 0 0 r r r 0 0 0	C Z C H S V	4	SP ← SP+1:r ← r+(SP)+CF
	Add both the contents of the memory location at (SP+1) and that of the Carry flag to the contents of 8-bit register r and write back the sum into r.			
ADDC r,(PC+A)[Note]	0 1 0 0 1 1 1 1 0 0 r r r 0 0 0	C Z C H S V	5	r ← r+(PC+A)+CF
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Add both the contents of the memory location at the EA and that of the Carry flag to the contents of 8-bit register r and write back the sum into r.			
ADDC (x),n	1 1 1 0 0 0 0 0 x x x x x x x x 0 1 1 0 0 0 0 0 n n n n n n n n	C Z C H S V	6	(x) ← (x)+n+CF
	Add both the immediate n in the instruction code and the contents of the Carry flag to the contents of the memory location directly addressed by x. Then write back the sum into the same location. (0x0000 ≤ x ≤ 0x00FF)			
ADDC (vw),n	1 1 1 0 0 0 0 1 w w w w w w w w v v v v v v v v 0 1 1 0 0 0 0 0 n n n n n n n n	C Z C H S V	7	(vw) ← (vw)+n+CF
	Add both the immediate n in the instruction code and the contents of the Carry flag to the contents of the memory location directly addressed by vw. Then write back the sum into the same location. (0x0000 ≤ vw ≤ 0xFFFF)			
ADDC (DE),n	1 1 1 0 0 0 1 0 0 1 1 0 0 0 0 0 n n n n n n n n	C Z C H S V	5	(DE) ← (DE)+n+CF
	Add both the immediate n in the instruction code and the contents of the Carry flag to the contents of the memory location addressed by DE. Then write back the sum into the same location.			
ADDC (HL),n	1 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0 n n n n n n n n	C Z C H S V	5	(HL) ← (HL)+n +CF
	Add both the immediate n in the instruction code and the contents of the Carry flag to the contents of the memory location addressed by HL. Then write back the sum into the same location.			
ADDC (IX),n	1 1 1 0 0 1 0 0 0 1 1 0 0 0 0 0 n n n n n n n n	C Z C H S V	5	(IX) ← (IX)+n+CF
	Add both the immediate n in the instruction code and the contents of the Carry flag to the contents of the memory location addressed by IX. Then write back the sum into the same location.			
ADDC (IY),n	1 1 1 0 0 1 0 1 0 1 1 0 0 0 0 0 n n n n n n n n	C Z C H S V	5	(IY) ← (IY)+n+CF
	Add both the immediate n in the instruction code and the contents of the Carry flag to the contents of the memory location addressed by IY. Then write back the sum into the same location.			
ADDC (IX+d),n	1 1 0 1 0 1 0 0 d d d d d d d d 0 1 1 0 0 0 0 0 n n n n n n n n	C Z C H S V	7	(IX+d) ← (IX+d)+n+CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Add both the immediate n in the instruction code and the contents of the Carry flag to the contents of the memory location at the EA. Then write back the sum into the same location.			
ADDC (IY+d),n	1 1 0 1 0 1 0 1 d d d d d d d d 0 1 1 0 0 0 0 0 n n n n n n n n	C Z C H S V	7	(IY+d) ← (IY+d)+n+CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Add both the immediate n in the instruction code and the contents of the Carry flag to the contents of the memory location at the EA. Then write back the sum into the same location.			
ADDC (SP+d),n	1 1 0 1 0 1 1 0 d d d d d d d d 0 1 1 0 0 0 0 0 n n n n n n n n	C Z C H S V	7	(SP+d) ← (SP+d)+n+CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Add both the immediate n in the instruction code and the contents of the Carry flag to the contents of the memory location at the EA. Then write back the sum into the same location.			
ADDC (HL+d),n	1 1 0 1 0 1 1 1 d d d d d d d d 0 1 1 0 0 0 0 0 n n n n n n n n	C Z C H S V	7	(HL+d) ← (HL+d)+n+CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Add both the immediate n in the instruction code and the contents of the Carry flag to the contents of the memory location at the EA. Then write back the sum into the same location.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
ADDC (HL+C),n	1 1 1 0 0 1 1 1 | 0 1 1 0 0 0 0 0 | n n n n n n n n	C Z C H S V	7	(HL+C) ← (HL+C)+n+CF
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Add both the immediate n in the instruction code and the contents of the Carry flag to the contents of the memory location at the EA. Then write back the sum into the same location.			
ADDC (+SP),n	1 1 1 0 0 1 1 0 | 0 1 1 0 0 0 0 0 | n n n n n n n n	C Z C H S V	6	SP ← SP+1:(SP) ← (SP)+n+CF
	Add both the immediate n in the instruction code and the contents of the Carry flag to the contents of the memory location at (SP+1) and write back the sum into the same location.			
ADDC (PC+A),n[Note]	0 1 0 0 1 1 1 1 | 0 1 1 0 0 0 0 0 | n n n n n n n n	C Z C H S V	7	(PC+A) ← (PC+A)+n+CF
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Add both the immediate n in the instruction code and the contents of the Carry flag to the contents of the memory location at the EA and write back the sum into the same location.			
ADDC rr,(x)	1 1 1 0 0 0 0 0 | x x x x x x x x | 1 0 r r r 0 0 0	C Z C U S V	5	rr ← rr+(x+1, x)+CF
	Add both the 16-bit memory word directly addressed by x and the contents of the Carry flag to the contents of 16-bit register rr. Then write back the sum into rr. (0x0000 ≤ x ≤ 0x00FF)			
ADDC rr,(vw)	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v | 1 0 r r r 0 0 0	C Z C U S V	6	rr ← rr+(vw+1, vw)+CF
	Add both the 16-bit memory word directly addressed by vw and the contents of the Carry flag to the contents of 16-bit register rr. Then write back the sum into rr. (0x0000 ≤ vw ≤ 0xFFFF)			
ADDC rr,(DE)	1 1 1 0 0 0 1 0 | 1 0 r r r 0 0 0	C Z C U S V	4	rr ← rr+(DE+1, DE)+CF
	Add both the 16-bit memory word addressed by DE and the contents of the Carry flag to the contents of 16-bit register rr. Then write back the sum into rr.			
ADDC rr,(HL)	1 1 1 0 0 0 1 1 | 1 0 r r r 0 0 0	C Z C U S V	4	rr ← rr+(HL+1, HL)+CF
	Add both the 16-bit memory word addressed by HL and the contents of the Carry flag to the contents of 16-bit register rr. Then write back the sum into rr.			
ADDC rr,(IX)	1 1 1 0 0 1 0 0 | 1 0 r r r 0 0 0	C Z C U S V	4	rr ← rr+(IX+1, IX)+CF
	Add both the 16-bit memory word addressed by IX and the contents of the Carry flag to the contents of 16-bit register rr. Then write back the sum into rr.			
ADDC rr,(IY)	1 1 1 0 0 1 0 1 | 1 0 r r r 0 0 0	C Z C U S V	4	rr ← rr+(IY+1,IY)+CF
	Add both the 16-bit memory word addressed by IY and the contents of the Carry flag to the contents of 16-bit register rr. Then write back the sum into rr.			
ADDC rr,(IX+d)	1 1 0 1 0 1 0 0 | d d d d d d d d | 1 0 r r r 0 0 0	C Z C U S V	6	rr ← rr+(IX+d+1, IX+d)+CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Add both the 16-bit memory word addressed by the EA and the contents of the Carry flag to the contents of 16-bit register rr. Then write back the sum into rr.			
ADDC rr,(IY+d)	1 1 0 1 0 1 0 1 | d d d d d d d d | 1 0 r r r 0 0 0	C Z C U S V	6	rr ← rr+(IY+d+1, IY+d)+CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Add both the 16-bit memory word addressed by the EA and the contents of the Carry flag to the contents of 16-bit register rr. Then write back the sum into rr.			
ADDC rr,(SP+d)	1 1 0 1 0 1 1 0 | d d d d d d d d | 1 0 r r r 0 0 0	C Z C U S V	6	rr ← rr+(SP+d+1, SP+d)+CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Add both the 16-bit memory word addressed by the EA and the contents of the Carry flag to the contents of 16-bit register rr. Then write back the sum into rr.			
ADDC rr,(HL+d)	1 1 0 1 0 1 1 1 | d d d d d d d d | 1 0 r r r 0 0 0	C Z C U S V	6	rr ← rr+(HL+d+1, HL+d)+CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Add both the 16-bit memory word addressed by the EA and the contents of the Carry flag to the contents of 16-bit register rr. Then write back the sum into rr.			
ADDC rr,(HL+C)	1 1 1 0 0 1 1 1 | 1 0 r r r 0 0 0	C Z C U S V	6	rr ← rr+(HL+C+1, HL+C)+CF
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Add both the 16-bit memory word addressed by the EA and the contents of the Carry flag to the contents of 16-bit register rr. Then write back the sum into rr.			
ADDC rr,(+SP)	1 1 1 0 0 1 1 0 | 1 0 r r r 0 0 0	C Z C U S V	5	SP ← SP+1:rr ← rr+(SP+1, SP)+CF
	Add both the 16-bit memory word addressed by (SP+1) and the contents of the Carry flag to the contents of 16-bit register rr. Then write back the sum into rr.			
ADDC rr,(PC+A)[Note]	0 1 0 0 1 1 1 1 | 1 0 r r r 0 0 0	C Z C U S V	6	rr ← rr+(PC+A+1, PC+A)+CF
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Add both the 16-bit memory word addressed by the EA and the contents of the Carry flag to the contents of 16-bit register rr. Then write back the sum into rr.			
SUB A,n	0 1 1 0 0 0 1 1 | n n n n n n n n	C Z C H S V	2	A ← A–n
	Subtract the immediate n in the instruction code from the contents of A and write back the difference into A.			
SUB g,n	1 1 1 0 1 g g g | 0 1 1 0 0 0 1 1 | n n n n n n n n	C Z C H S V	3	g ← g–n
	Subtract the immediate n in the instruction code from the contents of 8-bit register g and write back the difference into g.			
SUB gg,mn	1 1 1 0 1 g g g | 0 1 1 0 1 0 1 1 | n n n n n n n n | m m m m m m m m	C Z C U S V	4	gg ← gg–mn
	Subtract the immediate mn in the instruction code from the contents of 16-bit register gg and write back the difference into gg.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
SUB r,g	1 1 1 0 1 g g g | 0 0 r r r 0 1 1	C Z C H S V	2	r ← r–g
	Subtract the contents of 8-bit register g from that of 8-bit register r and write back the difference into r.			
SUB rr,gg	1 1 1 0 1 g g g | 1 0 r r r 0 1 1	C Z C U S V	3	rr ← rr–gg
	Subtract the contents of 16-bit register gg from that of 16-bit register rr and write back the difference into rr.			
SUB r,(x)	1 1 1 0 0 0 0 0 | x x x x x x x x | 0 0 r r r 0 1 1	C Z C H S V	4	r ← r–(x)
	Subtract the contents of the memory location directly addressed by x from that of 8-bit register r and write back the difference into r. (0x0000 ≤ x ≤ 0x00FF)			
SUB r,(vw)	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v 0 0 r r r 0 1 1	C Z C H S V	5	r ← r–(vw)
	Subtract the contents of the memory location directly addressed by vw from that of 8-bit register r and write back the difference into r. (0x0000 ≤ vw ≤ 0xFFFF)			
SUB r,(DE)	1 1 1 0 0 0 1 0 | 0 0 r r r 0 1 1	C Z C H S V	3	r ← r–(DE)
	Subtract the contents of the memory location addressed by DE from that of 8-bit register r and write back the difference into r.			
SUB r,(HL)	1 1 1 0 0 0 1 1 | 0 0 r r r 0 1 1	C Z C H S V	3	r ← r–(HL)
	Subtract the contents of the memory location addressed by HL from that of 8-bit register r and write back the difference into r.			
SUB r,(IX)	1 1 1 0 0 1 0 0 | 0 0 r r r 0 1 1	C Z C H S V	3	r ← r–(IX)
	Subtract the contents of the memory location addressed by IX from that of 8-bit register r and write back the difference into r.			
SUB r,(IY)	1 1 1 0 0 1 0 1 | 0 0 r r r 0 1 1	C Z C H S V	3	r ← r–(IY)
	Subtract the contents of the memory location addressed by IY from that of 8-bit register r and write back the difference into r.			
SUB r,(IX+d)	1 1 0 1 0 1 0 0 | d d d d d d d d | 0 0 r r r 0 1 1	C Z C H S V	5	r ← r–(IX+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Subtract the contents of the memory location at the EA from that of 8-bit register r and write back the difference into r.			
SUB r,(IY+d)	1 1 0 1 0 1 0 1 | d d d d d d d d | 0 0 r r r 0 1 1	C Z C H S V	5	r ← r–(IY+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Subtract the contents of the memory location at the EA from that of 8-bit register r and write back the difference into r.			
SUB r,(SP+d)	1 1 0 1 0 1 1 0 | d d d d d d d d | 0 0 r r r 0 1 1	C Z C H S V	5	r ← r–(SP+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Subtract the contents of the memory location at the EA from that of 8-bit register r and write back the difference into r.			
SUB r,(HL+d)	1 1 0 1 0 1 1 1 | d d d d d d d d | 0 0 r r r 0 1 1	C Z C H S V	5	r ← r–(HL+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Subtract the contents of the memory location at the EA from that of 8-bit register r and write back the difference into r.			
SUB r,(HL+C)	1 1 1 0 0 1 1 1 | 0 0 r r r 0 1 1	C Z C H S V	5	r ← r–(HL)+C
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Subtract the contents of the memory location at the EA from that of 8-bit register r and write back the difference into r.			
SUB r,(+SP)	1 1 1 0 0 1 1 0 | 0 0 r r r 0 1 1	C Z C H S V	4	SP ← SP+1:r ← r–(SP)
	Subtract the contents of the memory location at (SP+1) from that of 8-bit register r and write back the difference into r.			
SUB r,(PC+A)[Note]	0 1 0 0 1 1 1 1 | 0 0 r r r 0 1 1	C Z C H S V	5	r ← r–(PC+A)
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Subtract the contents of the memory location at the EA from that of 8-bit register r and write back the difference into r.			
SUB (x),n	1 1 1 0 0 0 0 0 | x x x x x x x x | 0 1 1 0 0 0 1 1 n n n n n n n n	C Z C H S V	6	(x) ← (x)–n
	Subtract the immediate n in the instruction code from the contents of the memory location directly addressed by x and write back the difference into the same location. (0x0000 ≤ x ≤ 0x00FF)			
SUB (vw),n	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v 0 1 1 0 0 0 1 1 | n n n n n n n n	C Z C H S V	7	(vw) ← (vw)–n
	Subtract the immediate n in the instruction code from the contents of the memory location directly addressed by vw and write back the difference into the same location. (0x0000 ≤ vw ≤ 0xFFFF)			
SUB (DE),n	1 1 1 0 0 0 1 0 | 0 1 1 0 0 0 1 1 | n n n n n n n n	C Z C H S V	5	(DE) ← (DE)–n
	Subtract the immediate n in the instruction code from the contents of the memory location addressed by DE and write back the difference into the same location.			
SUB (HL),n	1 1 1 0 0 0 1 1 | 0 1 1 0 0 0 1 1 | n n n n n n n n	C Z C H S V	5	(HL) ← (HL)–n
	Subtract the immediate n in the instruction code from the contents of the memory location addressed by HL and write back the difference into the same location.			
SUB (IX),n	1 1 1 0 0 1 0 0 | 0 1 1 0 0 0 1 1 | n n n n n n n n	C Z C H S V	5	(IX) ← (IX)–n
	Subtract the immediate n in the instruction code from the contents of the memory location addressed by IX and write back the difference into the same location.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
SUB (IY),n	1 1 1 0 0 1 0 1 | 0 1 1 0 0 0 1 1 | n n n n n n n n	C Z C H S V	5	(IY) ← (IY)–n
	Subtract the immediate n in the instruction code from the contents of the memory location addressed by IY and write back the difference into the same location.			
SUB (IX+d),n	1 1 0 1 0 1 0 0 | d d d d d d d d | 0 1 1 0 0 0 1 1 | n n n n n n n n	C Z C H S V	7	(IX+d) ← (IX+d)–n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Subtract the immediate n in the instruction code from the contents of the memory location at the EA and write back the difference into the same location.			
SUB (IY+d),n	1 1 0 1 0 1 0 1 | d d d d d d d d | 0 1 1 0 0 0 1 1 | n n n n n n n n	C Z C H S V	7	(IY+d) ← (IY+d)–n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Subtract the immediate n in the instruction code from the contents of the memory location at the EA and write back the difference into the same location.			
SUB (SP+d),n	1 1 0 1 0 1 1 0 | d d d d d d d d | 0 1 1 0 0 0 1 1 | n n n n n n n n	C Z C H S V	7	(SP+d) ← (SP+d)–n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Subtract the immediate n in the instruction code from the contents of the memory location at the EA and write back the difference into the same location.			
SUB (HL+d),n	1 1 0 1 0 1 1 1 | d d d d d d d d | 0 1 1 0 0 0 1 1 | n n n n n n n n	C Z C H S V	7	(HL+d) ← (HL+d)–n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Subtract the immediate n in the instruction code from the contents of the memory location at the EA and write back the difference into the same location.			
SUB (HL+C),n	1 1 1 0 0 1 1 1 | 0 1 1 0 0 0 1 1 | n n n n n n n n	C Z C H S V	7	(HL+C) ← (HL+C)–n
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Subtract the immediate n in the instruction code from the contents of the memory location at the EA and write back the difference into the same location.			
SUB (+SP),n	1 1 1 0 0 1 1 0 | 0 1 1 0 0 0 1 1 | n n n n n n n n	C Z C H S V	6	SP ← SP+1:(SP) ← (SP)–n
	Subtract the immediate n in the instruction code from the contents of the memory location at (SP+1) and write back the difference into the same location.			
SUB (PC+A),n[Note]	0 1 0 0 1 1 1 1 | 0 1 1 0 0 0 1 1 | n n n n n n n n	C Z C H S V	7	(PC+A) ← (PC+A)–n
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Subtract the immediate n in the instruction code from the contents of the memory location at the EA and write back the difference into the same location.			
SUB rr,(x)	1 1 1 0 0 0 0 0 | x x x x x x x x | 1 0 r r r 0 1 1	C Z C U S V	5	rr ← rr–(x+1, x)
	Subtract the 16-bit memory word directly addressed by x from the contents of 16-bit register rr and write back the difference into rr. (0x0000 ≤ x ≤ 0x00FF)			
SUB rr,(vw)	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v | 1 0 r r r 0 1 1	C Z C U S V	6	rr ← rr–(vw+1, vw)
	Subtract the 16-bit memory word directly addressed by vw from the contents of 16-bit register rr and write back the difference into rr. (0x0000 ≤ vw ≤ 0xFFFF)			
SUB rr,(DE)	1 1 1 0 0 0 1 0 | 1 0 r r r 0 1 1	C Z C U S V	4	rr ← rr–(DE+1, DE)
	Subtract the 16-bit memory word addressed by DE from the contents of 16-bit register rr and write back the difference into rr.			
SUB rr,(HL)	1 1 1 0 0 0 1 1 | 1 0 r r r 0 1 1	C Z C U S V	4	rr ← rr–(HL+1, HL)
	Subtract the 16-bit memory word addressed by HL from the contents of 16-bit register rr and write back the difference into rr.			
SUB rr,(IX)	1 1 1 0 0 1 0 0 | 1 0 r r r 0 1 1	C Z C U S V	4	rr ← rr–(IX+1, IX)
	Subtract the 16-bit memory word addressed by IX from the contents of 16-bit register rr and write back the difference into rr.			
SUB rr,(IY)	1 1 1 0 0 1 0 1 | 1 0 r r r 0 1 1	C Z C U S V	4	rr ← rr–(IY+1, IY)
	Subtract the 16-bit memory word addressed by IY from the contents of 16-bit register rr and write back the difference into rr.			
SUB rr,(IX+d)	1 1 0 1 0 1 0 0 | d d d d d d d d | 1 0 r r r 0 1 1	C Z C U S V	6	rr ← rr–(IX+d+1, IX+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Subtract the 16-bit memory word addressed by the EA from the contents of 16-bit register rr and write back the difference into rr.			
SUB rr,(IY+d)	1 1 0 1 0 1 0 1 | d d d d d d d d | 1 0 r r r 0 1 1	C Z C U S V	6	rr ← rr–(IY+d+1, IY+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Subtract the 16-bit memory word addressed by the EA from the contents of 16-bit register rr and write back the difference into rr.			
SUB rr,(SP+d)	1 1 0 1 0 1 1 0 | d d d d d d d d | 1 0 r r r 0 1 1	C Z C U S V	6	rr ← rr–(SP+d+1, SP+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Subtract the 16-bit memory word addressed by the EA from the contents of 16-bit register rr and write back the difference into rr.			
SUB rr,(HL+d)	1 1 0 1 0 1 1 1 | d d d d d d d d | 1 0 r r r 0 1 1	C Z C U S V	6	rr ← rr–(HL+d+1, HL+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Subtract the 16-bit memory word addressed by the EA from the contents of 16-bit register rr and write back the difference into rr.			
SUB rr,(HL+C)	1 1 1 0 0 1 1 1 | 1 0 r r r 0 1 1	C Z C U S V	6	rr ← rr–(HL+C+1, HL+C)
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Subtract the 16-bit memory word addressed by the EA from the contents of 16-bit register rr and write back the difference into rr.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
SUB rr,(+SP)	1 1 1 0 0 1 1 0 | 1 0 r r r 0 1 1	C Z C U S V	5	SP ← SP+1:rr ← rr–(SP+1, SP)
	Subtract the 16-bit memory word addressed by (SP+1) from the contents of 16-bit register rr and write back the difference into rr.			
SUB rr,(PC+A)[Note]	0 1 0 0 1 1 1 1 | 1 0 r r r 0 1 1	C Z C U S V	6	rr ← rr–(PC+A+1, PC+A)
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Subtract the 16-bit memory word addressed by the EA from the contents of 16-bit register rr and write back the difference into rr.			
SUBB A,n	0 1 1 0 0 0 1 0 | n n n n n n n n	C Z C H S V	2	A ← A–n–CF
	Subtract both the immediate n in the instruction code and the contents of the Carry flag from the contents of A and write back the difference into A.			
SUBB g,n	1 1 1 0 1 g g g | 0 1 1 0 0 0 1 0 | n n n n n n n n	C Z C H S V	3	g ← g–n–CF
	Subtract both the immediate n in the instruction code and the contents of the Carry flag from the contents of 8-bit register g and write back the difference into g.			
SUBB gg,mn	1 1 1 0 1 g g g | 0 1 1 0 1 0 1 0 | n n n n n n n n m m m m m m m m	C Z C U S V	4	gg ← gg–mn–CF
	Subtract both the immediate mn in the instruction code and the contents of the Carry flag from the contents of 16-bit register gg and write back the difference into gg.			
SUBB r,g	1 1 1 0 1 g g g | 0 0 r r r 0 1 0	C Z C H S V	2	r ← r–g–CF
	Subtract both the contents of 8-bit register g and that of the Carry flag from the contents of 8-bit register r and write back the difference into r.			
SUBB rr,gg	1 1 1 0 1 g g g | 1 0 r r r 0 1 0	C Z C U S V	3	rr ← rr–gg–CF
	Subtract both the contents of 16-bit register gg and that of the Carry flag from the contents of 16-bit register rr and write back the difference into rr.			
SUBB r,(x)	1 1 1 0 0 0 0 0 | x x x x x x x x | 0 0 r r r 0 1 0	C Z C H S V	4	r ← r–(x)–CF
	Subtract both the contents of the memory location directly addressed by x and that of the Carry flag from the contents of 8-bit register r and write back the difference into r. (0x0000 ≤ x ≤ 0x00FF)			
SUBB r,(vw)	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v 0 0 r r r 0 1 0	C Z C H S V	5	r ← r–(vw)–CF
	Subtract both the contents of the memory location directly addressed by vw and that of the Carry flag from the contents of 8-bit register r and write back the difference into r. (0x0000 ≤ vw ≤ 0xFFFF)			
SUBB r,(DE)	1 1 1 0 0 0 1 0 | 0 0 r r r 0 1 0	C Z C H S V	3	r ← r–(DE)–CF
	Subtract both the contents of the memory location addressed by DE and that of the Carry flag from the contents of 8-bit register r and write back the difference into r.			
SUBB r,(HL)	1 1 1 0 0 0 1 1 | 0 0 r r r 0 1 0	C Z C H S V	3	r ← r–(HL)–CF
	Subtract both the contents of the memory location addressed by HL and that of the Carry flag from the contents of 8-bit register r and write back the difference into r.			
SUBB r,(IX)	1 1 1 0 0 1 0 0 | 0 0 r r r 0 1 0	C Z C H S V	3	r ← r–(IX)–CF
	Subtract both the contents of the memory location addressed by IX and that of the Carry flag from the contents of 8-bit register r and write back the difference into r.			
SUBB r,(IY)	1 1 1 0 0 1 0 1 | 0 0 r r r 0 1 0	C Z C H S V	3	r ← r–(IY)–CF
	Subtract both the contents of the memory location addressed by IY and that of the Carry flag from the contents of 8-bit register r and write back the difference into r.			
SUBB r,(IX+d)	1 1 0 1 0 1 0 0 | d d d d d d d d | 0 0 r r r 0 1 0	C Z C H S V	5	r ← r–(IX+d)–CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Subtract both the contents of the memory location at the EA and that of the Carry flag from the contents of 8-bit register r and write back the difference into r.			
SUBB r,(IY+d)	1 1 0 1 0 1 0 1 | d d d d d d d d | 0 0 r r r 0 1 0	C Z C H S V	5	r ← r–(IX+d)–CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Subtract both the contents of the memory location at the EA and that of the Carry flag from the contents of 8-bit register r and write back the difference into r.			
SUBB r,(SP+d)	1 1 0 1 0 1 1 0 | d d d d d d d d | 0 0 r r r 0 1 0	C Z C H S V	5	r ← r–(SP+d)–CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Subtract both the contents of the memory location at the EA and that of the Carry flag from the contents of 8-bit register r and write back the difference into r.			
SUBB r,(HL+d)	1 1 0 1 0 1 1 1 | d d d d d d d d | 0 0 r r r 0 1 0	C Z C H S V	5	r ← r–(HL+d)–CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Subtract both the contents of the memory location at the EA and that of the Carry flag from the contents of 8-bit register r and write back the difference into r.			
SUBB r,(HL+C)	1 1 1 0 0 1 1 1 | 0 0 r r r 0 1 0	C Z C H S V	5	r ← r–(HL+C)–CF
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Subtract both the contents of the memory location at the EA and that of the Carry flag from the contents of 8-bit register r and write back the difference into r.			
SUBB r,(+SP)	1 1 1 0 0 1 1 0 | 0 0 r r r 0 1 0	C Z C H S V	4	SP ← SP+1:r ← r–(SP)–CF
	Subtract both the contents of the memory location at (SP+1) and that of the Carry flag from the contents of 8-bit register r and write back the difference into r.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
SUBB r,(PC+A)[Note]	0 1 0 0 1 1 1 1 | 0 0 r r r 0 1 0	C Z C H S V	5	r ← r–(PC+A)–CF
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Subtract both the contents of the memory location at the EA and that of the Carry flag from the contents of 8-bit register r and write back the difference into r.			
SUBB (x),n	1 1 1 0 0 0 0 0 | x x x x x x x x | 0 1 1 0 0 0 1 0 | n n n n n n n n	C Z C H S V	6	(x) ← (x)–n–CF
	Subtract both the immediate n in the instruction code and the contents of the Carry flag from the contents of the memory location directly addressed by x. Then write back the difference into the same location. (0x0000 ≤ x ≤ 0x00FF)			
SUBB (vw),n	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v | 0 1 1 0 0 0 1 0 | n n n n n n n n	C Z C H S V	7	(vw) ← (vw)–n–CF
	Subtract both the immediate n in the instruction code and the contents of the Carry flag from the contents of the memory location directly addressed by vw. Then write back the difference into the same location. (0x0000 ≤ vw ≤ 0xFFFF)			
SUBB (DE),n	1 1 1 0 0 0 1 0 | 0 1 1 0 0 0 1 0 | n n n n n n n n	C Z C H S V	5	(DE) ← (DE)–n–CF
	Subtract both the immediate n in the instruction code and the contents of the Carry flag from the contents of the memory location addressed by DE. Then write back the difference into the same location.			
SUBB (HL),n	1 1 1 0 0 0 1 1 | 0 1 1 0 0 0 1 0 | n n n n n n n n	C Z C H S V	5	(HL) ← (HL)–n–CF
	Subtract both the immediate n in the instruction code and the contents of the Carry flag from the contents of the memory location addressed by HL. Then write back the difference into the same location.			
SUBB (IX),n	1 1 1 0 0 1 0 0 | 0 1 1 0 0 0 1 0 | n n n n n n n n	C Z C H S V	5	(IX) ← (IX)–n–CF
	Subtract both the immediate n in the instruction code and the contents of the Carry flag from the contents of the memory location addressed by IX. Then write back the difference into the same location.			
SUBB (IY),n	1 1 1 0 0 1 0 1 | 0 1 1 0 0 0 1 0 | n n n n n n n n	C Z C H S V	5	(IY) ← (IY)–n–CF
	Subtract both the immediate n in the instruction code and the contents of the Carry flag from the contents of the memory location addressed by IY. Then write back the difference into the same location.			
SUBB (IX+d),n	1 1 0 1 0 1 0 0 | d d d d d d d d | 0 1 1 0 0 0 1 0 | n n n n n n n n	C Z C H S V	7	(IX+d) ← (IX+d)–n–CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Subtract both the immediate n in the instruction code and the contents of the Carry flag from the contents of the memory location at the EA. Then write back the difference into the same location.			
SUBB (IY+d),n	1 1 0 1 0 1 0 1 | d d d d d d d d | 0 1 1 0 0 0 1 0 | n n n n n n n n	C Z C H S V	7	(IY+d) ← (IY+d)–n–CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Subtract both the immediate n in the instruction code and the contents of the Carry flag from the contents of the memory location at the EA. Then write back the difference into the same location.			
SUBB (SP+d),n	1 1 0 1 0 1 1 0 | d d d d d d d d | 0 1 1 0 0 0 1 0 | n n n n n n n n	C Z C H S V	7	(SP+d) ← (SP+d)–n–CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Subtract both the immediate n in the instruction code and the contents of the Carry flag from the contents of the memory location at the EA. Then write back the difference into the same location.			
SUBB (HL+d),n	1 1 0 1 0 1 1 1 | d d d d d d d d | 0 1 1 0 0 0 1 0 | n n n n n n n n	C Z C H S V	7	(HL+d) ← (HL+d)–n–CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Subtract both the immediate n in the instruction code and the contents of the Carry flag from the contents of the memory location at the EA. Then write back the difference into the same location.			
SUBB (HL+C),n	1 1 1 0 0 1 1 1 | 0 1 1 0 0 0 1 0 | n n n n n n n n	C Z C H S V	7	(HL+C) ← (HL+C)–n–CF
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Subtract both the immediate n in the instruction code and the contents of the Carry flag from the contents of the memory location at the EA. Then write back the difference into the same location.			
SUBB (+SP),n	1 1 1 0 0 1 1 0 | 0 1 1 0 0 0 1 0 | n n n n n n n n	C Z C H S V	6	SP ← SP+1:(SP) ← (SP)–n–CF
	Subtract both the immediate n in the instruction code and the contents of the Carry flag from the contents of the memory location at (SP+1) and write back the difference into the same location.			
SUBB (PC+A),n[Note]	0 1 0 0 1 1 1 1 | 0 1 1 0 0 0 1 0 | n n n n n n n n	C Z C H S V	7	(PC+A) ← (PC+A)–n–CF
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Subtract both the immediate n in the instruction code and the contents of the Carry flag from the contents of the memory location at the EA. Then write back the difference into the same location.			
SUBB rr,(x)	1 1 1 0 0 0 0 0 | x x x x x x x x | 1 0 r r r 0 1 0	C Z C U S V	5	rr ← rr–(x+1, x)–CF
	Subtract both the 16-bit memory word directly addressed by x and the contents of the Carry flag from the contents of 16-bit register rr. Then write back the difference into rr. (0x0000 ≤ x ≤ 0x00FF)			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
SUBB rr,(vw)	1 1 1 0 0 0 0 1 w w w w w w w w v v v v v v v v 1 0 r r r 0 1 0	C Z C U S V	6	rr ← rr–(vw+1, vw)–CF
	Subtract both the 16-bit memory word directly addressed by vw and the contents of the Carry flag from the contents of 16-bit register rr. Then write back the difference into rr. (0x0000 ≤ vw ≤ 0xFFFF)			
SUBB rr,(DE)	1 1 1 0 0 0 1 0 1 0 r r r 0 1 0	C Z C U S V	4	rr ← rr–(DE+1, DE)–CF
	Subtract both the 16-bit memory word directly addressed by vw and the contents of the Carry flag from the contents of 16-bit register rr. Then write back the difference into rr. (0x0000 ≤ vw ≤ 0xFFFF)			
SUBB rr,(HL)	1 1 1 0 0 0 1 1 1 0 r r r 0 1 0	C Z C U S V	4	rr ← rr–(HL+1, HL)–CF
	Subtract both the 16-bit memory word addressed by HL and the contents of the Carry flag from the contents of 16-bit register rr. Then write back the difference into rr.			
SUBB rr,(IX)	1 1 1 0 0 1 0 0 1 0 r r r 0 1 0	C Z C U S V	4	rr ← rr–(IX+1, IX)–CF
	Subtract both the 16-bit memory word addressed by IX and the contents of the Carry flag from the contents of 16-bit register rr. Then write back the difference into rr.			
SUBB rr,(IY)	1 1 1 0 0 1 0 1 1 0 r r r 0 1 0	C Z C U S V	4	rr ← rr–(IY+1, IY)–CF
	Subtract both the 16-bit memory word addressed by IY and the contents of the Carry flag from the contents of 16-bit register rr. Then write back the difference into rr.			
SUBB rr,(IX+d)	1 1 0 1 0 1 0 0 d d d d d d d d 1 0 r r r 0 1 0	C Z C U S V	6	rr ← rr–(IX+d+1, IX+d)–CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Subtract both the 16-bit memory word addressed by the EA and the contents of the Carry flag from the contents of 16-bit register rr. Then write back the difference into rr.			
SUBB rr,(IY+d)	1 1 0 1 0 1 0 1 d d d d d d d d 1 0 r r r 0 1 0	C Z C U S V	6	rr ← rr–(IY+d+1, IY+d)–CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Subtract both the 16-bit memory word addressed by the EA and the contents of the Carry flag from the contents of 16-bit register rr. Then write back the difference into rr.			
SUBB rr,(SP+d)	1 1 0 1 0 1 1 0 d d d d d d d d 1 0 r r r 0 1 0	C Z C U S V	6	rr ← rr–(SP+d+1, SP+d)–CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Subtract both the 16-bit memory word addressed by the EA and the contents of the Carry flag from the contents of 16-bit register rr. Then write back the difference into rr.			
SUBB rr,(HL+d)	1 1 0 1 0 1 1 1 d d d d d d d d 1 0 r r r 0 1 0	C Z C U S V	6	rr ← rr–(HL+d+1, HL+d)–CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Subtract both the 16-bit memory word addressed by the EA and the contents of the Carry flag from the contents of 16-bit register rr. Then write back the difference into rr.			
SUBB rr,(HL+C)	1 1 1 0 0 1 1 1 1 0 r r r 0 1 0	C Z C U S V	6	rr ← rr–(HL+C+1, HL+C)–CF
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Subtract both the 16-bit memory word addressed by the EA and the contents of the Carry flag from the contents of 16-bit register rr. Then write back the difference into rr.			
SUBB rr,(+SP)	1 1 1 0 0 1 1 0 1 0 r r r 0 1 0	C Z C U S V	5	SP ← SP+1:rr ← rr–(SP+1, SP)–CF
	Subtract both the 16-bit memory word addressed by (SP+1) and the contents of the Carry flag from the contents of 16-bit register rr. Then write back the difference into rr.			
SUBB rr,(PC+A)[Note]	0 1 0 0 1 1 1 1 1 0 r r r 0 1 0	C Z C U S V	6	rr ← rr–(PC+A+1, PC+A)–CF
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Subtract both the 16-bit memory word addressed by the EA and the contents of the Carry flag from the contents of 16-bit register rr. Then write back the difference into rr.			
AND A,n	0 1 1 0 0 1 0 0 n n n n n n n n	Z Z – – – –	2	A ← A&n
	Combine the contents of A with the immediate n in the instruction code in a bitwise logical AND operation and write back the result into A.			
AND g,n	1 1 1 0 1 g g g 0 1 1 0 0 1 0 0 n n n n n n n n	Z Z – – – –	3	g ← g&n
	Combine the contents of 8-bit register g with the immediate n in the instruction code in a bitwise logical AND operation and write back the result into g.			
AND gg,mn	1 1 1 0 1 g g g 0 1 1 0 1 1 0 0 n n n n n n n n m m m m m m m m	Z Z – – – –	4	gg ← gg&mn
	Combine the contents of 16-bit register gg with the immediate mn in the instruction code in a bitwise logical AND operation and write back the result into gg.			
AND r,g	1 1 1 0 1 g g g 0 0 r r r 1 0 0	Z Z – – – –	2	r ← r&g
	Combine the contents of 8-bit register r with that of 8-bit register g in a bitwise logical AND operation and write back the result into r.			
AND rr,gg	1 1 1 0 1 g g g 1 0 r r r 1 0 0	Z Z – – – –	3	rr ← rr&gg
	Combine the contents of 16-bit register rr with that of 16-bit register gg in a bitwise logical AND operation and write back the result into rr.			
AND r,(x)	1 1 1 0 0 0 0 0 x x x x x x x x 0 0 r r r 1 0 0	Z Z – – – –	4	r ← r&(x)
	Combine the contents of 8-bit register r with that of the memory location directly addressed by x in a bitwise logical AND operation and write back the result into r. (0x0000 ≤ x ≤ 0x00FF)			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
AND r,(vw)	1 1 1 0 0 0 0 1 w w w w w w w w v v v v v v v v 0 0 r r r 1 0 0	Z Z – – – –	5	r ← r&(vw)
	Combine the contents of 8-bit register r with that of the memory location directly addressed by vw in a bitwise logical AND operation and write back the result into r. (0x0000 ≤ vw ≤ 0xFFFF)			
AND r,(DE)	1 1 1 0 0 0 1 0 0 0 r r r 1 0 0	Z Z – – – –	3	r ← r&(DE)
	Combine the contents of 8-bit register r with that of the memory location addressed by DE in a bitwise logical AND operation and write back the result into r.			
AND r,(HL)	1 1 1 0 0 0 1 1 0 0 r r r 1 0 0	Z Z – – – –	3	r ← r&(HL)
	Combine the contents of 8-bit register r with that of the memory location addressed by HL in a bitwise logical AND operation and write back the result into r.			
AND r,(IX)	1 1 1 0 0 1 0 0 0 0 r r r 1 0 0	Z Z – – – –	3	r ← r&(IX)
	Combine the contents of 8-bit register r with that of the memory location addressed by IX in a bitwise logical AND operation and write back the result into r.			
AND r,(IY)	1 1 1 0 0 1 0 1 0 0 r r r 1 0 0	Z Z – – – –	3	r ← r&(IY)
	Combine the contents of 8-bit register r with that of the memory location addressed by IY in a bitwise logical AND operation and write back the result into r.			
AND r,(IX+d)	1 1 0 1 0 1 0 0 d d d d d d d d 0 0 r r r 1 0 0	Z Z – – – –	5	r ← r&(IX+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical AND operation and write back the result into r.			
AND r,(IY+d)	1 1 0 1 0 1 0 1 d d d d d d d d 0 0 r r r 1 0 0	Z Z – – – –	5	r ← r&(IY+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical AND operation and write back the result into r.			
AND r,(SP+d)	1 1 0 1 0 1 1 0 d d d d d d d d 0 0 r r r 1 0 0	Z Z – – – –	5	r ← r&(SP+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical AND operation and write back the result into r.			
AND r,(HL+d)	1 1 0 1 0 1 1 1 d d d d d d d d 0 0 r r r 1 0 0	Z Z – – – –	5	r ← r&(HL+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical AND operation and write back the result into r.			
AND r,(HL+C)	1 1 1 0 0 1 1 1 0 0 r r r 1 0 0	Z Z – – – –	5	r ← r&(HL+C)
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical AND operation and write back the result into r.			
AND r,(+SP)	1 1 1 0 0 1 1 0 0 0 r r r 1 0 0	Z Z – – – –	4	SP ← SP+1:r ← r&(SP)
	Combine the contents of 8-bit register r with that of the memory location at (SP+1) in a bitwise logical AND operation and write back the result into r.			
AND r,(PC+A)[Note]	0 1 0 0 1 1 1 1 0 0 r r r 1 0 0	Z Z – – – –	5	r ← r&(PC+A)
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical AND operation and write back the result into r.			
AND (X),n	1 1 1 0 0 0 0 0 x x x x x x x x 0 1 1 0 0 1 0 0 n n n n n n n n	Z Z – – – –	6	(x) ← (x)&n
	Combine the contents of the memory location directly addressed by x with the immediate n in the instruction code in a bitwise logical AND operation. Then write back the result into the same location. (0x0000 £ x £ 0x00FF)			
AND (vw),n	1 1 1 0 0 0 0 1 w w w w w w w w v v v v v v v v 0 1 1 0 0 1 0 0 n n n n n n n n	Z Z – – – –	7	(vw) ← (vw)&n
	Combine the contents of the memory location directly addressed by vw with the immediate n in the instruction code in a bitwise logical AND operation. Then write back the result into the same location. (0x0000 ≤ vw ≤ 0xFFFF)			
AND (DE),n	1 1 1 0 0 0 1 0 0 1 1 0 0 1 0 0 n n n n n n n n	Z Z – – – –	5	(DE) ← (DE)&n
	Combine the contents of the memory location addressed by DE with the immediate n in the instruction code in a bitwise logical AND operation. Then write back the result into the same location.			
AND (HL),n	1 1 1 0 0 0 1 1 0 1 1 0 0 1 0 0 n n n n n n n n	Z Z – – – –	5	(HL) ← (HL)&n
	Combine the contents of the memory location addressed by HL with the immediate n in the instruction code in a bitwise logical AND operation. Then write back the result into the same location.			
AND (IX),n	1 1 1 0 0 1 0 0 0 1 1 0 0 1 0 0 n n n n n n n n	Z Z – – – –	5	(IX) ← (IX)&n
	Combine the contents of the memory location addressed by IX with the immediate n in the instruction code in a bitwise logical AND operation. Then write back the result into the same location.			
AND (IY),n	1 1 1 0 0 1 0 1 0 1 1 0 0 1 0 0 n n n n n n n n	Z Z – – – –	5	(IY) ← (IY)&n
	Combine the contents of the memory location addressed by IY with the immediate n in the instruction code in a bitwise logical AND operation. Then write back the result into the same location.			

2.2 ALU Instructions

Mnemonic	Instruction Code (Binary)				Flag J Z C H S V	Cycle	Operation
AND (IX+d),n	1 1 0 1 0 1 0 0 n n n n n n n n	d d d d d d d d	0 1 1 0 0 1 0 0		Z Z – – – –	7	(IX+d) ← (IX+d)&n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical AND operation. Then write back the result into the same location.						
AND (IY+d),n	1 1 0 1 0 1 0 1 n n n n n n n n	d d d d d d d d	0 1 1 0 0 1 0 0		Z Z – – – –	7	(IY+d) ← (IY+d)&n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical AND operation. Then write back the result into the same location.						
AND (SP+d),n	1 1 0 1 0 1 1 0 n n n n n n n n	d d d d d d d d	0 1 1 0 0 1 0 0		Z Z – – – –	7	(SP+d) ← (SP+d)&n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical AND operation. Then write back the result into the same location.						
AND (HL+d),n	1 1 0 1 0 1 1 1 n n n n n n n n	d d d d d d d d	0 1 1 0 0 1 0 0		Z Z – – – –	7	(HL+d) ← (HL+d)&n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical AND operation. Then write back the result into the same location.						
AND (HL+C),n	1 1 1 0 0 1 1 1	0 1 1 0 0 1 0 0	n n n n n n n n		Z Z – – – –	7	(HL+C) ← (HL+C)&n
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical AND operation. Then write back the result into the same location.						
AND (+SP),n	1 1 1 0 0 1 1 0	0 1 1 0 0 1 0 0	n n n n n n n n		Z Z – – – –	6	SP ← SP+1:(SP) ← (SP)&n
	Combine the contents of the memory location at (SP+1) with the immediate n in the instruction code in a bitwise logical AND operation and write back the result into the same location.						
AND (PC+A),n[Note]	0 1 0 0 1 1 1 1	0 1 1 0 0 1 0 0	n n n n n n n n		Z Z – – – –	7	(PC+A) ← (PC+A)&n
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical AND operation. Then write back the result into the same location.						
AND rr,(x)	1 1 1 0 0 0 0 0	x x x x x x x x	1 0 r r r 1 0 0		Z Z – – – –	5	rr ← rr&(x)
	Combine the contents of 16-bit register rr with the 16-bit memory word directly addressed by x in a bitwise logical AND operation. Then write back the result into rr. (0x0000 ≤ x ≤ 0x00FF)						
AND rr,(vw)	1 1 1 0 0 0 0 1 1 0 r r r 1 0 0	w w w w w w w w	v v v v v v v v		Z Z – – – –	6	rr ← rr&(vw)
	Combine the contents of 16-bit register rr with the 16-bit memory word directly addressed by vw in a bitwise logical AND operation. Then write back the result into rr. (0x0000 ≤ vw ≤ 0xFFFF)						
AND rr,(DE)	1 1 1 0 0 0 1 0	1 0 r r r 1 0 0			Z Z – – – –	4	rr ← rr&(DE)
	Combine the contents of 16-bit register rr with the 16-bit memory word addressed by DE in a bitwise logical AND operation. Then write back the result into rr.						
AND rr,(HL)	1 1 1 0 0 0 1 1	1 0 r r r 1 0 0			Z Z – – – –	4	rr ← rr&(HL)
	Combine the contents of 16-bit register rr with the 16-bit memory word addressed by HL in a bitwise logical AND operation. Then write back the result into rr.						
AND rr,(IX)	1 1 1 0 0 1 0 0	1 0 r r r 1 0 0			Z Z – – – –	4	rr ← rr&(IX)
	Combine the contents of 16-bit register rr with the 16-bit memory word addressed by IX in a bitwise logical AND operation. Then write back the result into rr.						
AND rr,(IY)	1 1 1 0 0 1 0 1	1 0 r r r 1 0 0			Z Z – – – –	4	rr ← rr&(IY)
	Combine the contents of 16-bit register rr with the 16-bit memory word addressed by IY in a bitwise logical AND operation. Then write back the result into rr.						
AND rr,(IX+d)	1 1 0 1 0 1 0 0	d d d d d d d d	1 0 r r r 1 0 0		Z Z – – – –	6	rr ← rr&(IX+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical AND operation. Then write back the result into rr.						
AND rr,(IY+d)	1 1 0 1 0 1 0 1	d d d d d d d d	1 0 r r r 1 0 0		Z Z – – – –	6	rr ← rr&(IY+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical AND operation. Then write back the result into rr.						
AND rr,(SP+d)	1 1 0 1 0 1 1 0	d d d d d d d d	1 0 r r r 1 0 0		Z Z – – – –	6	rr ← rr&(SP+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical AND operation. Then write back the result into rr.						

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
AND rr,(HL+d)	1 1 0 1 0 1 1 1 | d d d d d d d d | 1 0 r r r 1 0 0	Z Z – – – –	6	rr ← rr&(HL+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical AND operation. Then write back the result into rr.			
AND rr,(HL+C)	1 1 1 0 0 1 1 1 | 1 0 r r r 1 0 0	Z Z – – – –	6	rr ← rr&(HL+C)
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical AND operation. Then write back the result into rr.			
AND rr,(+SP)	1 1 1 0 0 1 1 0 | 1 0 r r r 1 0 0	Z Z – – – –	5	SP ← SP+1:rr ← rr&(SP)
	Combine the contents of 16-bit register rr with the 16-bit memory word addressed by (SP+1) in a bitwise logical AND operation. Then write back the result into rr.			
AND rr,(PC+A)[Note]	0 1 0 0 1 1 1 1 | 1 0 r r r 1 0 0	Z Z – – – –	6	rr ← rr&(PC+A)
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical AND operation. Then write back the result into rr.			
OR A,n	0 1 1 0 0 1 1 0 | n n n n n n n n	Z Z – – – –	2	A ← A | n
	Combine the contents of A with the immediate n in the instruction code in a bitwise logical OR operation and write back the result into A.			
OR g,n	1 1 1 0 1 g g g | 0 1 1 0 0 1 1 0 | n n n n n n n n	Z Z – – – –	3	g ← g | n
	Combine the contents of 8-bit register g with the immediate n in the instruction code in a bitwise logical OR operation and write back the result into g.			
OR gg,mn	1 1 1 0 1 g g g | 0 1 1 0 1 1 1 0 | n n n n n n n n | m m m m m m m m	Z Z – – – –	4	gg ← gg | mn
	Combine the contents of 16-bit register gg with the immediate mn in the instruction code in a bitwise logical OR operation and write back the result into gg.			
OR r,g	1 1 1 0 1 g g g | 0 0 r r r 1 1 0	Z Z – – – –	2	r ← r | g
	Combine the contents of 8-bit register r with that of 8-bit register g in a bitwise logical OR operation and write back the result into r.			
OR rr,gg	1 1 1 0 1 g g g | 1 0 r r r 1 1 0	Z Z – – – –	3	rr ← rr | gg
	Combine the contents of 16-bit register rr with that of 16-bit register gg in a bitwise logical OR operation and write back the result into rr.			
OR r,(x)	1 1 1 0 0 0 0 0 | x x x x x x x x | 0 0 r r r 1 1 0	Z Z – – – –	4	r ← r | (x)
	Combine the contents of 8-bit register r with that of the memory location directly addressed by x in a bitwise logical OR operation and write back the result into r. (0x0000 ≤ x ≤ 0x00FF)			
OR r,(vw)	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v | 0 0 r r r 1 1 0	Z Z – – – –	5	r ← r | (vw)
	Combine the contents of 8-bit register r with that of the memory location directly addressed by vw in a bitwise logical OR operation and write back the result into r. (0x0000 ≤ vw ≤ 0xFFFF)			
OR r,(DE)	1 1 1 0 0 0 1 0 | 0 0 r r r 1 1 0	Z Z – – – –	3	r ← r | (DE)
	Combine the contents of 8-bit register r with that of the memory location addressed by DE in a bitwise logical OR operation and write back the result into r.			
OR r,(HL)	1 1 1 0 0 0 1 1 | 0 0 r r r 1 1 0	Z Z – – – –	3	r ← r | (HL)
	Combine the contents of 8-bit register r with that of the memory location addressed by HL in a bitwise logical OR operation and write back the result into r.			
OR r,(IX)	1 1 1 0 0 1 0 0 | 0 0 r r r 1 1 0	Z Z – – – –	3	r ← r | (IX)
	Combine the contents of 8-bit register r with that of the memory location addressed by IX in a bitwise logical OR operation and write back the result into r.			
OR r,(IY)	1 1 1 0 0 1 0 1 | 0 0 r r r 1 1 0	Z Z – – – –	3	r ← r | (IY)
	Combine the contents of 8-bit register r with that of the memory location addressed by IY in a bitwise logical OR operation and write back the result into r.			
OR r,(IX+d)	1 1 0 1 0 1 0 0 | d d d d d d d d | 0 0 r r r 1 1 0	Z Z – – – –	5	r ← r | (IX+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical OR operation and write back the result into r.			
OR r,(IY+d)	1 1 0 1 0 1 0 1 | d d d d d d d d | 0 0 r r r 1 1 0	Z Z – – – –	5	r ← r | (IY+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical OR operation and write back the result into r.			
OR r,(SP+d)	1 1 0 1 0 1 1 0 | d d d d d d d d | 0 0 r r r 1 1 0	Z Z – – – –	5	r ← r | (SP+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical OR operation and write back the result into r.			
OR r,(HL+d)	1 1 0 1 0 1 1 1 | d d d d d d d d | 0 0 r r r 1 1 0	Z Z – – – –	5	r ← r | (HL+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical OR operation and write back the result into r.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
OR r,(HL+C)	1 1 1 0 0 1 1 1 | 0 0 r r r 1 1 0	Z Z – – – –	5	r ← r | (HL+C)
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical OR operation and write back the result into r.			
OR r,(+SP)	1 1 1 0 0 1 1 0 | 0 0 r r r 1 1 0	Z Z – – – –	4	SP ← SP+1:r ← r | (SP)
	Combine the contents of 8-bit register r with that of the memory location at (SP+1) in a bitwise logical OR operation and write back the result into r.			
OR r,(PC+A)[Note]	0 1 0 0 1 1 1 1 | 0 0 r r r 1 1 0	Z Z – – – –	5	r ← r | (PC+A)
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical OR operation and write back the result into r.			
OR (x),n	1 1 1 0 0 0 0 0 | x x x x x x x x | 0 1 1 0 0 1 1 0 | n n n n n n n n	Z Z – – – –	6	(x) ← (x) | n
	Combine the contents of the memory location directly addressed by x with the immediate n in the instruction code in a bitwise logical OR operation. Then write back the result into the same location. (0x0000 ≤ x ≤ 0x00FF)			
OR (vw),n	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v | 0 1 1 0 0 1 1 0 | n n n n n n n n	Z Z – – – –	7	(vw) ← (vw) | n
	Combine the contents of the memory location directly addressed by vw with the immediate n in the instruction code in a bitwise logical OR operation. Then write back the result into the same location. (0x0000 ≤ vw ≤ 0xFFFF)			
OR (DE),n	1 1 1 0 0 0 1 0 | 0 1 1 0 0 1 1 0 | n n n n n n n n	Z Z – – – –	5	(DE) ← (DE) | n
	Combine the contents of the memory location addressed by DE with the immediate n in the instruction code in a bitwise logical OR operation. Then write back the result into the same location.			
OR (HL),n	1 1 1 0 0 0 1 1 | 0 1 1 0 0 1 1 0 | n n n n n n n n	Z Z – – – –	5	(HL) ← (HL) | n
	Combine the contents of the memory location addressed by HL with the immediate n in the instruction code in a bitwise logical OR operation. Then write back the result into the same location.			
OR (IX),n	1 1 1 0 0 1 0 0 | 0 1 1 0 0 1 1 0 | n n n n n n n n	Z Z – – – –	5	(IX) ← (IX) | n
	Combine the contents of the memory location addressed by IX with the immediate n in the instruction code in a bitwise logical OR operation. Then write back the result into the same location.			
OR (IY),n	1 1 1 0 0 1 0 1 | 0 1 1 0 0 1 1 0 | n n n n n n n n	Z Z – – – –	5	(IY) ← (IY) | n
	Combine the contents of the memory location addressed by IY with the immediate n in the instruction code in a bitwise logical OR operation. Then write back the result into the same location.			
OR (IX+d),n	1 1 0 1 0 1 0 0 | d d d d d d d d | 0 1 1 0 0 1 1 0 | n n n n n n n n	Z Z – – – –	7	(IX+d) ← (IX+d) | n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical OR operation. Then write back the result into the same location.			
OR (IY+d),n	1 1 0 1 0 1 0 1 | d d d d d d d d | 0 1 1 0 0 1 1 0 | n n n n n n n n	Z Z – – – –	7	(IY+d) ← (IY+d) | n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical OR operation. Then write back the result into the same location.			
OR (SP+d),n	1 1 0 1 0 1 1 0 | d d d d d d d d | 0 1 1 0 0 1 1 0 | n n n n n n n n	Z Z – – – –	7	(SP+d) ← (SP+d) | n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical OR operation. Then write back the result into the same location.			
OR (HL+d),n	1 1 0 1 0 1 1 1 | d d d d d d d d | 0 1 1 0 0 1 1 0 | n n n n n n n n	Z Z – – – –	7	(HL+d) ← (HL+d) | n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical OR operation. Then write back the result into the same location.			
OR (HL+C),n	1 1 1 0 0 1 1 1 | 0 1 1 0 0 1 1 0 | n n n n n n n n	Z Z – – – –	7	(HL+C) ← (HL+C) | n
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical OR operation. Then write back the result into the same location.			
OR (+SP),n	1 1 1 0 0 1 1 0 | 0 1 1 0 0 1 1 0 | n n n n n n n n	Z Z – – – –	6	SP ← SP+1:(SP) ← (SP) | n
	Combine the contents of the memory location at (SP+1) with the immediate n in the instruction code in a bitwise logical OR operation and write back the result into the same location.			
OR (PC+A),n[Note]	0 1 0 0 1 1 1 1 | 0 1 1 0 0 1 1 0 | n n n n n n n n	Z Z – – – –	7	(PC+A) ← (PC+A) | n
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical OR operation. Then write back the result into the same location.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
OR rr,(x)	1 1 1 0 0 0 0 0 | x x x x x x x x | 1 0 r r r 1 1 0	Z Z – – – –	5	rr ← rr | (x)
	Combine the contents of 16-bit register rr with the 16-bit memory word directly addressed by x in a bitwise logical OR operation. Then write back the result into rr. (0x0000 ≤ x ≤ 0x00FF)			
OR rr,(vw)	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v | 1 0 r r r 1 1 0	Z Z – – – –	6	rr ← rr | (vw)
	Combine the contents of 16-bit register rr with the 16-bit memory word directly addressed by vw in a bitwise logical OR operation. Then write back the result into rr. (0x0000 ≤ vw ≤ 0xFFFF)			
OR rr,(DE)	1 1 1 0 0 0 1 0 | 1 0 r r r 1 1 0	Z Z – – – –	4	rr ← rr | (DE)
	Combine the contents of 16-bit register rr with the 16-bit memory word addressed by DE in a bitwise logical OR operation. Then write back the result into rr.			
OR rr,(HL)	1 1 1 0 0 0 1 1 | 1 0 r r r 1 1 0	Z Z – – – –	4	rr ← rr | (HL)
	Combine the contents of 16-bit register rr with the 16-bit memory word addressed by HL in a bitwise logical OR operation. Then write back the result into rr.			
OR rr,(IX)	1 1 1 0 0 1 0 0 | 1 0 r r r 1 1 0	Z Z – – – –	4	rr ← rr | (IX)
	Combine the contents of 16-bit register rr with the 16-bit memory word addressed by IX in a bitwise logical OR operation. Then write back the result into rr.			
OR rr,(IY)	1 1 1 0 0 1 0 1 | 1 0 r r r 1 1 0	Z Z – – – –	4	rr ← rr | (IY)
	Combine the contents of 16-bit register rr with the 16-bit memory word addressed by IY in a bitwise logical OR operation. Then write back the result into rr.			
OR rr,(IX+d)	1 1 0 1 0 1 0 0 | d d d d d d d d | 1 0 r r r 1 1 0	Z Z – – – –	6	rr ← rr | (IX+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical OR operation. Then write back the result into rr.			
OR rr,(IY+d)	1 1 0 1 0 1 0 1 | d d d d d d d d | 1 0 r r r 1 1 0	Z Z – – – –	6	rr ← rr | (IY+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical OR operation. Then write back the result into rr.			
OR rr,(SP+d)	1 1 0 1 0 1 1 0 | d d d d d d d d | 1 0 r r r 1 1 0	Z Z – – – –	6	rr ← rr | (SP+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical OR operation. Then write back the result into rr.			
OR rr,(HL+d)	1 1 0 1 0 1 1 1 | d d d d d d d d | 1 0 r r r 1 1 0	Z Z – – – –	6	rr ← rr | (HL+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical OR operation. Then write back the result into rr.			
OR rr,(HL+C)	1 1 1 0 0 1 1 1 | 1 0 r r r 1 1 0	Z Z – – – –	6	rr ← rr | (HL+C)
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical OR operation. Then write back the result into rr.			
OR rr,(+SP)	1 1 1 0 0 1 1 0 | 1 0 r r r 1 1 0	Z Z – – – –	5	SP ← SP+1:rr ← rr | (SP)
	Combine the contents of 16-bit register rr with the 16-bit memory word addressed by (SP+1) in a bitwise logical OR operation. Then write back the result into rr.			
OR rr,(PC+A)[Note]	0 1 0 0 1 1 1 1 | 1 0 r r r 1 1 0	Z Z – – – –	6	rr ← rr | (PC+A)
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical OR operation. Then write back the result into rr.			
XOR A,n	0 1 1 0 0 1 0 1 | n n n n n n n n	Z Z – – – –	2	A ←A^n
	Combine the contents of A with the immediate n in the instruction code in a bitwise logical exclusive-OR operation and write back the result into A. This instruction replaces the contents of A with its 1's complement when n = 0xFF. Example: Assume A contains 0x69. Then, the instruction "XOR A, 0xFF" replaces the contents of A with 0x96 and clears ZF to 0.			
XOR g,n	1 1 1 0 1 g g g | 0 1 1 0 0 1 0 1 | n n n n n n n n	Z Z – – – –	3	g ←g^n
	Combine the contents of 8-bit register g with the immediate n in the instruction code in a bitwise logical exclusive-OR operation and write back the result into g. This instruction replaces the contents of g with its 1's complement when n = 0xFF.			
XOR gg,mn	1 1 1 0 1 g g g | 0 1 1 0 1 1 0 1 | n n n n n n n n | m m m m m m m m	Z Z – – – –	4	gg ← gg ^ mn
	Combine the contents of 16-bit register gg with the immediate mn in the instruction code in a bitwise logical exclusive-OR operation and write back the result into gg. This instruction replaces the contents of gg with its 1's complement when mn = 0xFFFF.			
XOR r,g	1 1 1 0 1 g g g | 0 0 r r r 1 0 1	Z Z – – – –	2	r ←r^g
	Combine the contents of 8-bit register r with that of 8-bit register g in a bitwise logical exclusive-OR operation and write back the result into r.			

Mnemonic	Instruction Code (Binary)				Flag J Z C H S V	Cycle	Operation
XOR rr,gg	1 1 1 0 1 g g g	1 0 r r r 1 0 1			Z Z – – – –	3	rr ←rr^gg
	Combine the contents of 16-bit register rr with that of 16-bit register gg in a bitwise logical exclusive-OR operation and write back the result into rr.						
XOR r,(x)	1 1 1 0 0 0 0 0	x x x x x x x x	0 0 r r r 1 0 1		Z Z – – – –	4	r ←r ^ (x)
	Combine the contents of 8-bit register r with that of the memory location directly addressed by x in a bitwise logical exclusive-OR operation and write back the result into r. (0x0000 ≤ x ≤ 0x00FF)						
XOR r,(vw)	1 1 1 0 0 0 0 1 0 0 r r r 1 0 1	w w w w w w w w	v v v v v v v v		Z Z – – – –	5	r ← r^(vw)
	Combine the contents of 8-bit register r with that of the memory location directly addressed by vw in a bitwise logical exclusive-OR operation and write back the result into r. (0x0000 ≤ vw ≤ 0xFFFF)						
XOR r,(DE)	1 1 1 0 0 0 1 0	0 0 r r r 1 0 1			Z Z – – – –	3	r ←r ^ (DE)
	Combine the contents of 8-bit register r with that of the memory location addressed by DE in a bitwise logical exclusive-OR operation and write back the result into r.						
XOR r,(HL)	1 1 1 0 0 0 1 1	0 0 r r r 1 0 1			Z Z – – – –	3	r ←r^(HL)
	Combine the contents of 8-bit register r with that of the memory location addressed by HL in a bitwise logical exclusive-OR operation and write back the result into r.						
XOR r,(IX)	1 1 1 0 0 1 0 0	0 0 r r r 1 0 1			Z Z – – – –	3	r ←r ^ (IX)
	Combine the contents of 8-bit register r with that of the memory location addressed by IX in a bitwise logical exclusive-OR operation and write back the result into r.						
XOR r,(IY)	1 1 1 0 0 1 0 1	0 0 r r r 1 0 1			Z Z – – – –	3	r ←r^(IY)
	Combine the contents of 8-bit register r with that of the memory location addressed by IY in a bitwise logical exclusive-OR operation and write back the result into r.						
XOR r,(IX+d)	1 1 0 1 0 1 0 0	d d d d d d d d	0 0 r r r 1 0 1		Z Z – – – –	5	r ←r^(IX+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical exclusive-OR operation and write back the result into r.						
XOR r,(IY+d)	1 1 0 1 0 1 0 1	d d d d d d d d	0 0 r r r 1 0 1		Z Z – – – –	5	r ←r ^ (IY+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical exclusive-OR operation and write back the result into r.						
XOR r,(SP+d)	1 1 0 1 0 1 1 0	d d d d d d d d	0 0 r r r 1 0 1		Z Z – – – –	5	r ←r ^ (SP+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical exclusive-OR operation and write back the result into r.						
XOR r,(HL+d)	1 1 0 1 0 1 1 1	d d d d d d d d	0 0 r r r 1 0 1		Z Z – – – –	5	r ←r^(HL+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical exclusive-OR operation and write back the result into r.						
XOR r,(HL+C)	1 1 1 0 0 1 1 1	0 0 r r r 1 0 1			Z Z – – – –	5	r ←r ^ (HL+C)
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical exclusive-OR operation and write back the result into r.						
XOR r,(+SP)	1 1 1 0 0 1 1 0	0 0 r r r 1 0 1			Z Z – – – –	4	SP ← SP+1:r ← r^(SP)
	Combine the contents of 8-bit register r with that of the memory location at (SP+1) in a bitwise logical exclusive-OR operation and write back the result into r.						
XOR r,(PC+A)[Note]	0 1 0 0 1 1 1 1	0 0 r r r 1 0 1			Z Z – – – –	5	r ←r ^ (PC+A)
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Combine the contents of 8-bit register r with that of the memory location at the EA in a bitwise logical exclusive-OR operation and write back the result into r.						

Mnemonic	Instruction Code (Binary)				Flag J Z C H S V	Cycle	Operation
XOR (x),n	1 1 1 0 0 0 0 0 n n n n n n n n	x x x x x x x x	0 1 1 0 0 1 0 1		Z Z – – – –	6	(x) ← (x)^n
	Combine the contents of the memory location directly addressed by x with the immediate n in the instruction code in a bitwise logical exclusive-OR operation. Then write back the result into the same location. (0x0000 ≤ x ≤ 0x00FF)						
XOR (vw),n	1 1 1 0 0 0 0 1 0 1 1 0 0 1 0 1	w w w w w w w w n n n n n n n n	v v v v v v v v		Z Z – – – –	7	(vw) ← (vw) ^ n
	Combine the contents of the memory location directly addressed by vw with the immediate n in the instruction code in a bitwise logical exclusive-OR operation. Then write back the result into the same location. (0x0000 ≤ vw ≤ 0xFFFF)						
XOR (DE),n	1 1 1 0 0 0 1 0	0 1 1 0 0 1 0 1	n n n n n n n n		Z Z – – – –	5	(DE) ← (DE)^n
	Combine the contents of the memory location addressed by DE with the immediate n in the instruction code in a bitwise logical exclusive-OR operation. Then write back the result into the same location.						
XOR (HL),n	1 1 1 0 0 0 1 1	0 1 1 0 0 1 0 1	n n n n n n n n		Z Z – – – –	5	(HL) ← (HL)^n
	Combine the contents of the memory location addressed by HL with the immediate n in the instruction code in a bitwise logical exclusive-OR operation. Then write back the result into the same location.						
XOR (IX),n	1 1 1 0 0 1 0 0	0 1 1 0 0 1 0 1	n n n n n n n n		Z Z – – – –	5	(IX) ← (IX) ^ n
	Combine the contents of the memory location addressed by IX with the immediate n in the instruction code in a bitwise logical exclusive-OR operation. Then write back the result into the same location.						
XOR (IY),n	1 1 1 0 0 1 0 1	0 1 1 0 0 1 0 1	n n n n n n n n		Z Z – – – –	5	(IY) ← (IY)^n
	Combine the contents of the memory location addressed by IY with the immediate n in the instruction code in a bitwise logical exclusive-OR operation. Then write back the result into the same location.						
XOR (IX+d),n	1 1 0 1 0 1 0 0 n n n n n n n n	d d d d d d d d	0 1 1 0 0 1 0 1		Z Z – – – –	7	(IX+d) ← (IX+d) ^ n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical exclusive-OR operation. Then write back the result into the same location.						
XOR (IY+d),n	1 1 0 1 0 1 0 1 n n n n n n n n	d d d d d d d d	0 1 1 0 0 1 0 1		Z Z – – – –	7	(IY+d) ← (IY+d) ^ n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical exclusive-OR operation. Then write back the result into the same location.						
XOR (SP+d),n	1 1 0 1 0 1 1 0 n n n n n n n n	d d d d d d d d	0 1 1 0 0 1 0 1		Z Z – – – –	7	(SP+d) ← (SP+d)^n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical exclusive-OR operation. Then write back the result into the same location.						
XOR (HL+d),n	1 1 0 1 0 1 1 1 n n n n n n n n	d d d d d d d d	0 1 1 0 0 1 0 1		Z Z – – – –	7	(HL+d) ← (HL+d)^n
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical exclusive-OR operation. Then write back the result into the same location.						
XOR (HL+C),n	1 1 1 0 0 1 1 1	0 1 1 0 0 1 0 1	n n n n n n n n		Z Z – – – –	7	(HL+C) ← (HL+C)^n
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical exclusive-OR operation. Then write back the result into the same location.						
XOR (+SP),n	1 1 1 0 0 1 1 0	0 1 1 0 0 1 0 1	n n n n n n n n		Z Z – – – –	6	SP ← SP+1:(SP) ← (SP) ^ n
	Combine the contents of the memory location at (SP+1) with the immediate n in the instruction code in a bitwise logical exclusive-OR operation and write back the result into the same location.						
XOR (PC+A),n[Note]	0 1 0 0 1 1 1 1	0 1 1 0 0 1 0 1	n n n n n n n n		Z Z – – – –	7	(PC+A) ← (PC+A)^n
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Combine the contents of the memory location at the EA with the immediate n in the instruction code in a bitwise logical exclusive-OR operation. Then write back the result into the same location.						

Mnemonic	Instruction Code (Binary)			Flag J Z C H S V	Cycle	Operation
XOR rr,(x)	1 1 1 0 0 0 0 0	x x x x x x x x	1 0 r r r 1 0 1	Z Z – – – –	5	rr ← rr^(x)
	Combine the contents of 16-bit register rr with the 16-bit memory word directly addressed by x in a bitwise logical exclusive-OR operation. Then write back the result into rr. (0x0000 ≤ x ≤ 0x00FF)					
XOR rr,(vw)	1 1 1 0 0 0 0 1 1 0 r r r 1 0 1	w w w w w w w w	v v v v v v v v	Z Z – – – –	6	rr ← rr ^ (vw)
	Combine the contents of 16-bit register rr with the 16-bit memory word directly addressed by vw in a bitwise logical exclusive-OR operation. Then write back the result into rr. (0x0000 ≤ vw ≤ 0xFFFF)					
XOR rr,(DE)	1 1 1 0 0 0 1 0	1 0 r r r 1 0 1		Z Z – – – –	4	rr ← rr ^ (DE)
	Combine the contents of 16-bit register rr with the 16-bit memory word addressed by DE in a bitwise logical exclusive-OR operation. Then write back the result into rr.					
XOR rr,(HL)	1 1 1 0 0 0 1 1	1 0 r r r 1 0 1		Z Z – – – –	4	rr ← rr^(HL)
	Combine the contents of 16-bit register rr with the 16-bit memory word addressed by HL in a bitwise logical exclusive-OR operation. Then write back the result into rr.					
XOR rr,(IX)	1 1 1 0 0 1 0 0	1 0 r r r 1 0 1		Z Z – – – –	4	rr ← rr^(IX)
	Combine the contents of 16-bit register rr with the 16-bit memory word addressed by IX in a bitwise logical exclusive-OR operation. Then write back the result into rr.					
XOR rr,(IY)	1 1 1 0 0 1 0 1	1 0 r r r 1 0 1		Z Z – – – –	4	rr ← rr^(IY)
	Combine the contents of 16-bit register rr with the 16-bit memory word addressed by IY in a bitwise logical exclusive-OR operation. Then write back the result into rr.					
XOR rr,(IX+d)	1 1 0 1 0 1 0 0	d d d d d d d d	1 0 r r r 1 0 1	Z Z – – – –	6	rr ← rr^(IX+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical exclusive-OR operation. Then write back the result into rr.					
XOR rr,(IY+d)	1 1 0 1 0 1 0 1	d d d d d d d d	1 0 r r r 1 0 1	Z Z – – – –	6	rr ← rr^(IY+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical exclusive-OR operation. Then write back the result into rr.					
XOR rr,(SP+d)	1 1 0 1 0 1 1 0	d d d d d d d d	1 0 r r r 1 0 1	Z Z – – – –	6	rr ← rr ^ (SP+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical exclusive-OR operation. Then write back the result into rr.					
XOR rr,(HL+d)	1 1 0 1 0 1 1 1	d d d d d d d d	1 0 r r r 1 0 1	Z Z – – – –	6	rr ← rr^(HL+d)
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical exclusive-OR operation. Then write back the result into rr.					
XOR rr,(HL+C)	1 1 1 0 0 1 1 1	1 0 r r r 1 0 1		Z Z – – – –	6	rr ← rr ^ (HL+C)
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical exclusive-OR operation. Then write back the result into rr.					
XOR rr,(+SP)	1 1 1 0 0 1 1 0	1 0 r r r 1 0 1		Z Z – – – –	5	SP ← SP+1:rr ← rr^(SP)
	Combine the contents of 16-bit register rr with the 16-bit memory word addressed by (SP+1) in a bitwise logical exclusive-OR operation. Then write back the result into rr.					
XOR rr,(PC+A)[Note]	0 1 0 0 1 1 1 1	1 0 r r r 1 0 1		Z Z – – – –	6	rr ← rr^(PC+A)
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Combine the contents of 16-bit register rr with the 16-bit memory word addressed by the EA in a bitwise logical exclusive-OR operation. Then write back the result into rr.					
INC r	0 0 1 0 0 r r r			C Z – – – –	1	r ← r+1
	Increment the contents of 8-bit register r. The Jump Status and Zero flags are set to 1 when an overflow occurs. The Carry flag is not affected. Example: Assume L contains 0xFF. Then, the instruction “INC L” loads 0x00 into L and sets ZF and JF to 1.					
INC rr	0 0 1 1 0 r r r			C Z – – – –	2	rr ← rr+1
	Increment the contents of 16-bit register rr. The Jump Status and Zero flags are set to 1 when an overflow occurs. Example: Assume HL contains 0x1234. Then, the instruction “INC HL” loads 0x1235 into HL and clears ZF and JF to 0.					

Mnemonic	Instruction Code (Binary)						Flag J Z C H S V	Cycle	Operation
INC (x)	1 1 1 0 0 0 0 0	x x x x x x x x	1 1 1 1 0 0 0 0				C Z – – – –	5	(x) ← (x)+1
	Increment the contents of the memory location directly addressed by x. The Jump Status and Zero flags are set to 1 when an overflow occurs. (0x0000 ≤ x ≤ 0x00FF)								
INC (vw)	1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0	w w w w w w w w	v v v v v v v v				C Z – – – –	6	(vw) ← (vw)+1
	Increment the contents of the memory location directly addressed by vw. The Jump Status and Zero flags are set to 1 when an overflow occurs. (0x0000 ≤ vw ≤ 0xFFFF)								
INC (DE)	1 1 1 0 0 0 1 0	1 1 1 1 0 0 0 0					C Z – – – –	4	(DE) ← (DE)+1
	Increment the contents of the memory location addressed by DE. The Jump Status and Zero flags are set to 1 when an overflow occurs.								
INC (HL)	1 1 1 0 0 0 1 1	1 1 1 1 0 0 0 0					C Z – – – –	4	(HL) ← (HL)+1
	Increment the contents of the memory location addressed by HL. The Jump Status and Zero flags are set to 1 when an overflow occurs.								
INC (IX)	1 1 1 0 0 1 0 0	1 1 1 1 0 0 0 0					C Z – – – –	4	(IX) ← (IX)+1
	Increment the contents of the memory location addressed by IX. The Jump Status and Zero flags are set to 1 when an overflow occurs.								
INC (IY)	1 1 1 0 0 1 0 1	1 1 1 1 0 0 0 0					C Z – – – –	4	(IY) ← (IY)+1
	Increment the contents of the memory location addressed by IY. The Jump Status and Zero flags are set to 1 when an overflow occurs.								
INC (IX+d)	1 1 0 1 0 1 0 0	d d d d d d d d	1 1 1 1 0 0 0 0				C Z – – – –	6	(IX+d) ← (IX+d)+1
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Increment the contents of the memory location at the EA. The Jump Status and Zero flags are set to 1 when an overflow occurs.								
INC (IY+d)	1 1 0 1 0 1 0 1	d d d d d d d d	1 1 1 1 0 0 0 0				C Z – – – –	6	(IY+d) ← (IY+d)+1
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Increment the contents of the memory location at the EA. The Jump Status and Zero flags are set to 1 when an overflow occurs.								
INC (SP+d)	1 1 0 1 0 1 1 0	d d d d d d d d	1 1 1 1 0 0 0 0				C Z – – – –	6	(SP+d) ← (SP+d)+1
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Increment the contents of the memory location at the EA. The Jump Status and Zero flags are set to 1 when an overflow occurs.								
INC (HL+d)	1 1 0 1 0 1 1 1	d d d d d d d d	1 1 1 1 0 0 0 0				C Z – – – –	6	(HL+d) ← (HL+d)+1
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Increment the contents of the memory location at the EA. The Jump Status and Zero flags are set to 1 when an overflow occurs.								
INC (HL+C)	1 1 1 0 0 1 1 1	1 1 1 1 0 0 0 0					C Z – – – –	6	(HL+C) ← (HL+C)+1
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Increment the contents of the memory location at the EA. The Jump Status and Zero flags are set to 1 when an overflow occurs.								
INC (+SP)	1 1 1 0 0 1 1 0	1 1 1 1 0 0 0 0					C Z – – – –	5	SP ← SP+1:(SP) ← (SP)+1
	Increment the contents of the memory location at (SP+1). The Jump Status and Zero flags are set to 1 when an overflow occurs.								
INC (PC+A)[Note]	0 1 0 0 1 1 1 1	1 1 1 1 0 0 0 0					C Z – – – –	6	(PC+A) ← (PC+A)+1
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Increment the contents of the memory location at the EA. The Jump Status and Zero flags are set to 1 when an overflow occurs.								
DEC r	0 0 1 0 1 r r r						C Z – – – –	1	r ← r–1
	Decrement the contents of 8-bit register r. The Jump Status flag is set to 1 when an underflow occurs (i.e., when the result becomes 0xFF). The Carry flag is not affected. Example: Assume L contains 0x00. Then, the instruction “DEC L” loads 0xFF into L, clears ZF to 0 and sets JF to 1.								
DEC rr	0 0 1 1 1 r r r						C Z – – – –	2	rr ← rr–1
	Decrement the contents of 16-bit register rr. The Jump Status flag is set to 1 when an underflow occurs (i.e., when the result becomes 0xFFFF). Example: Assume HL contains 0x8765. Then, the instruction “DEC HL” loads 0x8764 into HL and clears ZF and JF to 0.								
DEC (x)	1 1 1 0 0 0 0 0	x x x x x x x x	1 1 1 1 1 0 0 0				C Z – – – –	5	(x) ← (x)–1
	Decrement the contents of the memory location directly addressed by x. The Jump Status flag is set to 1 when an underflow occurs. (0x0000 ≤ x ≤ 0x00FF)								
DEC (vw)	1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0	w w w w w w w w	v v v v v v v v				C Z – – – –	6	(vw) ← (vw)–1
	Decrement the contents of the memory location directly addressed by vw. The Jump Status flag is set to 1 when an underflow occurs. (0x0000 ≤ vw ≤ 0xFFFF)								
DEC (DE)	1 1 1 0 0 0 1 0	1 1 1 1 1 0 0 0					C Z – – – –	4	(DE) ← (DE)–1
	Decrement the contents of the memory location addressed by DE. The Jump Status flag is set to 1 when an underflow occurs.								
DEC (HL)	1 1 1 0 0 0 1 1	1 1 1 1 1 0 0 0					C Z – – – –	4	(HL) ← (HL)–1
	Decrement the contents of the memory location addressed by HL. The Jump Status flag is set to 1 when an underflow occurs.								
DEC (IX)	1 1 1 0 0 1 0 0	1 1 1 1 1 0 0 0					C Z – – – –	4	(IX) ← (IX)–1
	Decrement the contents of the memory location addressed by IX. The Jump Status flag is set to 1 when an underflow occurs.								

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
DEC (IY)	1 1 1 0 0 1 0 1 | 1 1 1 1 1 0 0 0	C Z – – – –	4	(IY) ← (IY)–1
	Decrement the contents of the memory location addressed by IY. The Jump Status flag is set to 1 when an underflow occurs.			
DEC (IX+d)	1 1 0 1 0 1 0 0 | d d d d d d d d | 1 1 1 1 1 0 0 0	C Z – – – –	6	(IX+d) ← (IX+d)–1
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Decrement the contents of the memory location at the EA. The Jump Status flag is set to 1 when an underflow occurs.			
DEC (IY+d)	1 1 0 1 0 1 0 1 | d d d d d d d d | 1 1 1 1 1 0 0 0	C Z – – – –	6	(IY+d) ← (IY+d)–1
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Decrement the contents of the memory location at the EA. The Jump Status flag is set to 1 when an underflow occurs.			
DEC (SP+d)	1 1 0 1 0 1 1 0 | d d d d d d d d | 1 1 1 1 1 0 0 0	C Z – – – –	6	(SP+d) ← (SP+d)–1
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Decrement the contents of the memory location at the EA. The Jump Status flag is set to 1 when an underflow occurs.			
DEC (HL+d)	1 1 0 1 0 1 1 1 | d d d d d d d d | 1 1 1 1 1 0 0 0	C Z – – – –	6	(HL+d) ← (HL+d)–1
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Decrement the contents of the memory location at the EA. The Jump Status flag is set to 1 when an underflow occurs.			
DEC (HL+C)	1 1 1 0 0 1 1 1 | 1 1 1 1 1 0 0 0	C Z – – – –	6	(HL+C) ← (HL+C)–1
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Decrement the contents of the memory location at the EA. The Jump Status flag is set to 1 when an underflow occurs.			
DEC (+SP)	1 1 1 0 0 1 1 0 | 1 1 1 1 1 0 0 0	C Z – – – –	5	SP ← SP+1:(SP) ← (SP)–1
	Decrement the contents of the memory location at (SP-1). The Jump Status flag is set to 1 when an underflow occurs.			
DEC (PC+A)[Note]	0 1 0 0 1 1 1 1 | 1 1 1 1 1 0 0 0	C Z – – – –	6	(PC+A) ← (PC+A)–1
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Decrement the contents of the memory location at the EA. The Jump Status flag is set to 1 when an underflow occurs.			
DAA g	1 1 1 0 1 g g g | 1 1 0 1 1 0 1 0	C Z C H – –	2	Decimal adjustment against g (after addition)

When performing an add operation on an 8-bit packed BCD number, the contents of register g is decimal adjusted after the execution of an add instruction (ADD/ADDC).
For multi-digit BCD number, add and adjust operations must be performed on each byte, beginning with the lowest-order digit.

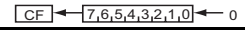
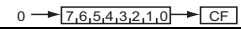
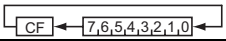
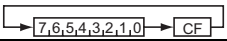
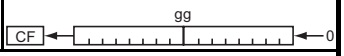
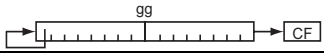
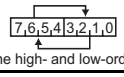
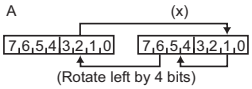
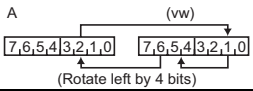
Carry Flag Before Adjustment	High-order 4 Bits of Register g Before Adjusting	Half Carry Flag Before Adjustment	Low-order 4 Bits of Register g Before Adjusting	Value Added for Adjustment	Carry Flag After Adjustment
0	0 to 9	0	0 to 9	00	0
0	0 to 8	0	A to F	06	0
0	0 to 9	1	0 to 3	06	0
0	A to F	0	0 to 9	60	1
0	9 to F	0	A to F	66	1
0	A to F	1	0 to 3	66	1
1	0 to 2	0	0 to 9	60	1
1	0 to 2	0	A to F	66	1
1	0 to 3	1	0 to 3	66	1

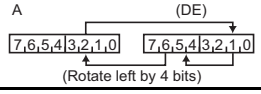
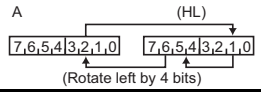
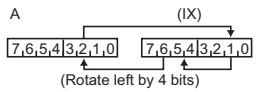
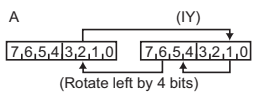
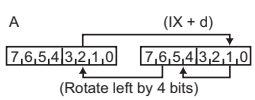
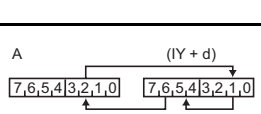
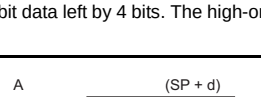
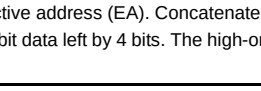
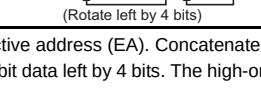
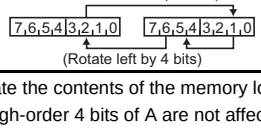
Example: Assume A and B contain 0x26 and 0x57 respectively. The instruction “ADD A, B” loads 0x7D into A and clears CF and HF to 0. Then, the instruction “DAA A” loads 0x83 into A while CF is not affected.

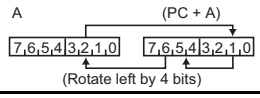
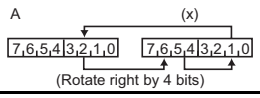
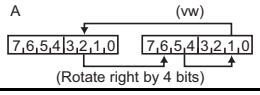
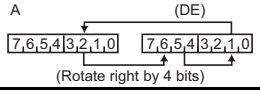
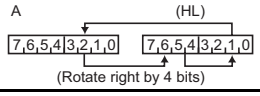
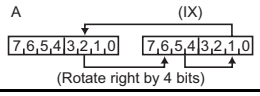
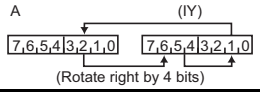
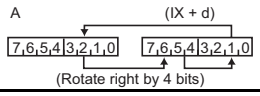
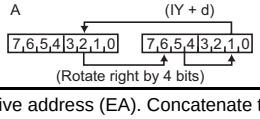
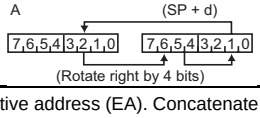
<table>
<tr><th>Mnemonic</th><th>Instruction Code (Binary)</th><th>Flag
J Z C H S V</th><th>Cycle</th><th>Operation</th></tr>
<tr><td rowspan="2">DAS g</td><td>1 1 1 0 1 g g g | 1 1 0 1 1 0 1 1</td><td>C Z C H – –</td><td>2</td><td>Decimal adjustment against g (after subtraction)</td></tr>
<tr><td colspan="4">When performing a subtract operation on an 8-bit packed BCD number, the contents of register g is decimal adjusted after the execution of a subtract instruction (SUB/SUBB).
For multi-digit BCD number, subtract and adjust operations must be performed on each byte, beginning with the lowest-order digit.
<table>
<tr><th>Carry Flag Before Adjustment</th><th>High-order 4 Bits of Register g Before Adjusting</th><th>Half Carry Flag Before Adjustment</th><th>Low-order 4 Bits of Register g Before Adjusting</th><th>Value Added for Adjustment</th><th>Carry Flag After Adjustment</th></tr>
<tr><td>0</td><td>0 to 9</td><td>0</td><td>0 to 9</td><td>00</td><td>0</td></tr>
<tr><td>0</td><td>0 to 9</td><td>1</td><td>6 to F</td><td>FA</td><td>0</td></tr>
<tr><td>1</td><td>7 to F</td><td>0</td><td>0 to 9</td><td>A0</td><td>1</td></tr>
<tr><td>1</td><td>6 to F</td><td>1</td><td>6 to F</td><td>9A</td><td>1</td></tr>
</table>
Example: Assume A and B contain 0x87 and 0x39 respectively. The instruction “SUB A, B” loads 0x4E into A, clears CF to 0 and sets HF to 1. Then, the instruction “DAS A” loads 0x48 into A while CF is not affected.</td></tr>
<tr><td rowspan="2">MUL W,A</td><td>1 1 1 0 1 0 0 0 | 1 1 1 1 0 0 1 0</td><td>Z Z – – – –</td><td>13</td><td>WA ← W×A</td></tr>
<tr><td colspan="4">Multiply the contents of W (unsigned integer) and that of A (unsigned integer) and places the product into WA. The Zero flag is set to 1 when the high-order 8 bits of the result (the value loaded into W) is 0x00; otherwise, it is cleared to 0.
Example 1: Assume W and A contain 0x87 and 0xF2 respectively. Then, this instruction loads 0x7F9E into WA and clears ZF to 0.
Example 2: Assume W and A contain 0x16 and 0x05 respectively. Then, this instruction loads 0x006E into WA and sets ZF to 1.</td></tr>
<tr><td rowspan="2">MUL B,C</td><td>1 1 1 0 1 0 0 1 | 1 1 1 1 0 0 1 0</td><td>Z Z – – – –</td><td>13</td><td>BC ← B×C</td></tr>
<tr><td colspan="4">Multiply the contents of B (unsigned integer) and that of C (unsigned integer) and places the product into BC. The Zero flag is set to 1 when the high-order 8 bits of the result (the value loaded into B) is 0x00; otherwise, it is cleared to 0.</td></tr>
<tr><td rowspan="2">MUL D,E</td><td>1 1 1 0 1 0 1 0 | 1 1 1 1 0 0 1 0</td><td>Z Z – – – –</td><td>13</td><td>DE ← D×E</td></tr>
<tr><td colspan="4">Multiply the contents of D (unsigned integer) and that of E (unsigned integer) and places the product into DE. The Zero flag is set to 1 when the high-order 8 bits of the result (the value loaded into D) is 0x00; otherwise, it is cleared to 0.</td></tr>
<tr><td rowspan="2">MUL H,L</td><td>1 1 1 0 1 0 1 1 | 1 1 1 1 0 0 1 0</td><td>Z Z – – – –</td><td>13</td><td>HL ← H×L</td></tr>
<tr><td colspan="4">Multiply the contents of H (unsigned integer) and that of L (unsigned integer) and places the product into HL. The Zero flag is set to 1 when the high-order 8 bits of the result (the value loaded into H) is 0x00; otherwise, it is cleared to 0.</td></tr>
<tr><td rowspan="2">DIV WA,C</td><td>1 1 1 0 1 0 0 0 | 1 1 1 1 0 0 1 1</td><td>Z Z C – – –</td><td>13</td><td>A ← WA÷C, W ← Remainder</td></tr>
<tr><td colspan="4">Divide the contents of WA (unsigned integer) by that of C (unsigned integer) and places the quotient and the remainder into A and W, respectively. The Carry flag is set to 1 when the divisor (contents of C) is 0x00 or when the quotient is greater than 0x100 (at the same time, the contents of A and W become undefined); otherwise, it is cleared to 0. The Zero flag is set to 1 when the remainder is 0x00; otherwise it is cleared to 0. The contents of C is not affected.
Example 1: Assume WA and C contain 0x1234 and 0x56 respectively.
Then, this instruction loads 0x36 and 0x10 into A and W respectively, and clears CF and ZF to 0.
Example 2: Assume WA and C contain 0x4830 and 0x9A respectively.
Then, this instruction loads 0x78 and 0x00 into A and W respectively, clears CF to 0 and sets ZF to 1.
Example 3: Assume WA and C contain 0x3210 and 0x27 respectively.
Then, this instruction loads 0x48 and 0x18 into A and W respectively, sets CF to 1 and clea ZF to 0.</td></tr>
<tr><td rowspan="2">DIV DE,C</td><td>1 1 1 0 1 0 1 0 | 1 1 1 1 0 0 1 1</td><td>Z Z C – – –</td><td>13</td><td>E ← DE÷C, D ← Remainder</td></tr>
<tr><td colspan="4">Divide the contents of DE (unsigned integer) by that of C (unsigned integer) and places the low-order 8 bits of the quotient and the remainder into E and D, respectively. The Carry flag is set to 1 when the divisor (contents of C) is 0x00 or when the quotient is greater than 0x100 (at the same time, the contents of DE becomes undefined). The Zero flag is set to 1 when the remainder is 0x00. The contents of C is not affected.</td></tr>
<tr><td rowspan="2">DIV HL,C</td><td>1 1 1 0 1 0 1 1 | 1 1 1 1 0 0 1 1</td><td>Z Z C – – –</td><td>13</td><td>L ← HL÷C, H ← Remainder</td></tr>
<tr><td colspan="4">Divide the contents of HL (unsigned integer) by that of C (unsigned integer) and places the low-order 8 bits of the quotient and the remainder into L and H, respectively. The Carry flag is set to 1 when the divisor (contents of C) is 0x00 or when the quotient is greater than 0x100 (at the same time, the contents of DE becomes undefined). The Zero flag is set to 1 when the remainder is 0x00. The contents of C is not affected.</td></tr>
<tr><td rowspan="2">NEG CS,gg</td><td>1 1 1 0 1 g g g | 1 1 1 1 1 0 1 0</td><td>1 – – – – –</td><td>3</td><td>if CF=1 then gg ← 0–gg else null</td></tr>
<tr><td colspan="4">When the Carry flag is set, replace the contents of 16-bit register gg with its 2’s complement and set the Jump Status flag to 1.
When the Carry flag is cleared, set the Jump Status flag to 1 and skip to the next instruction (while the contents of register gg is not affected).
Example 1: Assume HL contains 0x5678 and CF is set.
Then, the instruction “NEG CS, HL” replaces the contents of HL with 0xA988 and sets CF to 1.
Example 2: Assume DE contains 0x89AB and CF is cleared.
Then, the instruction “NEG CS, DE” replaces the contents of DE with 0x89AB and clears CF to 0.</td></tr>
</table>

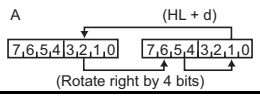
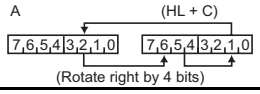
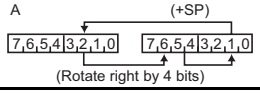
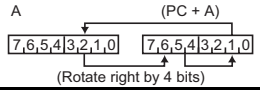
Note: There are restrictions on instructions that uses the operand (PC + A). For more details, see "1.4 Addressing Mode".

2.3 Shift, Rotate and Nibble Manipulation Instructions

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
SHLC g	1 1 1 0 1 g g g | 1 1 1 1 0 1 0 0	C Z * – – –	2	
	Logically shift the contents of 8-bit register g left by 1 bit. (At the same time, insert a 0 into the least-significant bit of g. The Carry flag holds the most-significant bit shifted out of g.) The Zero flag is set to 1 when g is 0x00 as a result of a shift. Example: Assume A contains 0y00111011 and CF is set. After executing this instruction, A has 0y01110110; and CF and ZF are cleared to 0.			
SHRC g	1 1 1 0 1 g g g | 1 1 1 1 0 1 0 1	C Z * – – –	2	
	Logically shift the contents of 8-bit register g right by 1 bit. (At the same time, insert a 0 into the most-significant bit of g. The Carry flag holds the least-significant bit shifted out of g.) The Zero flag is set to 1 when the g is 0x00 as a result of a shift. Example: Assume A contains 0y01011101 and CF is cleared. After executing this instruction, A has 0y00101110; CF is set to 1; and ZF is cleared to 0.			
ROLC g	1 1 1 0 1 g g g | 1 1 1 1 0 1 1 0	C Z * – – –	2	
	Rotate the contents of 8-bit register g and the Carry flag to left by 1 bit altogether. The Zero flag is set to 1 when g is 0x00 as a result of a rotation. Example 1: Assume A contains 0y10010110 and CF is cleared. After executing this instruction, A has 0y00101100, CF and JF are set to 1 and ZF is cleared to 0. Example 2: Assume A contains 0y10000000 and CF is cleared. After executing this instruction, A has 0y00000000; and CF, JF and ZF are set to 1.			
RORC g	1 1 1 0 1 g g g | 1 1 1 1 0 1 1 1	C Z * – – –	2	
	Rotate the contents of 8-bit register g and the Carry flag to right by 1 bit altogether. The Zero flag is set to 1 when g is 0x00 as a result of a rotation. Example 1: Assume A contains 0y01101101 and CF is set. After executing this instruction, A has 0y10110110; CF is set to 1; and ZF is cleared to 0.			
SHLCA gg	1 1 1 0 1 g g g | 1 1 1 1 0 0 0 0	C Z * – S V	3	
	Arithmetically shift the contents of 16-bit register gg left by 1 bit. (At the same time, insert a 0 into the least-significant bit of gg. The Carry flag holds the most-significant bit shifted out of gg.) The Zero flag is set to 1 when gg is 0x0000 as a result of a shift. The Overflow flag is set to 1 when the most-significant bit of gg changes as a result of a shift. Example: Assume HL contains 0x3456 and CF is set. After executing the instruction “SHLCA HL”, HL has 0x68AC; and CF, JF, ZF, SF and VF are cleared to 0.			
SHRCA gg	1 1 1 0 1 g g g | 1 1 1 1 0 0 0 1	C Z * – S 0	3	
	Arithmetically shift the contents of 16-bit register gg right by 1 bit. (The most-significant bit of gg is not affected. The Carry flag holds the least-significant bit shifted out of gg.) The Zero flag is set to 1 when gg is 0x0000 as a result of a shift. Example: Assume DE contains 0x89AB and CF is cleared. After executing the instruction “SHLCA DE”, HL has 0xC4D5; CF and SF are set to 1; and JF, ZF and VF are cleared to 0.			
SWAP g	1 1 1 0 1 g g g | 1 1 1 1 1 1 1 1	1 – – – – –	7	 (Swap the high- and low-order 4 bits)
	Swap the high-order 4 bits of 8-bit register g with the remaining low-order 4 bits. Example: Assume A contains 0x25. After executing this instruction, A has 0x52.			
ROLD A,(x)	1 1 1 0 0 0 0 0 | x x x x x x x x | 1 1 1 1 0 1 1 0	1 – – – – –	9	 (Rotate left by 4 bits)
	Concatenate the contents of the memory location directly addressed by x with the low-order 4 bits of A. Rotate the concatenated 12-bit data left by 4 bits. The high-order 4 bits of A are not affected. (0x0000 ≤ x ≤ 0x00FF) Example: Assume A and the memory location at address 0x0087 contain 0x12 and 0x56 respectively. After executing the instruction “ROLD A, (0x87)”, A has 0x15 and the memory location at address 0x0087 has 0x62.			
ROLD A,(vw)	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v | 1 1 1 1 0 1 1 0	1 – – – – –	10	 (Rotate left by 4 bits)
	Concatenate the contents of the memory location directly addressed by vw with the low-order 4 bits of A. Rotate the concatenated 12-bit data left by 4 bits. The high-order 4 bits of A are not affected. (0x0000 ≤ vw ≤ 0xFFFF)			

Mnemonic	Instruction Code (Binary)			Flag J Z C H S V	Cycle	Operation
ROLD A,(DE)	1 1 1 0 0 0 1 0	1 1 1 1 0 1 1 0		1 – – – – –	8	
	Concatenate the contents of the memory location addressed by DE with the low-order 4 bits of A. Rotate the concatenated 12-bit data left by 4 bits. The high-order 4 bits of A are not affected.					
ROLD A,(HL)	1 1 1 0 0 0 1 1	1 1 1 1 0 1 1 0		1 – – – – –	8	
	Concatenate the contents of the memory location addressed by HL with the low-order 4 bits of A. Rotate the concatenated 12-bit data left by 4 bits. The high-order 4 bits of A are not affected.					
ROLD A,(IX)	1 1 1 0 0 1 0 0	1 1 1 1 0 1 1 0		1 – – – – –	8	
	Concatenate the contents of the memory location addressed by IX with the low-order 4 bits of A. Rotate the concatenated 12-bit data left by 4 bits. The high-order 4 bits of A are not affected.					
ROLD A,(IY)	1 1 1 0 0 1 0 1	1 1 1 1 0 1 1 0		1 – – – – –	8	
	Concatenate the contents of the memory location addressed by IY with the low-order 4 bits of A. Rotate the concatenated 12-bit data left by 4 bits. The high-order 4 bits of A are not affected.					
ROLD A,(IX+d)	1 1 0 1 0 1 0 0	d d d d d d d d	1 1 1 1 0 1 1 0	1 – – – – –	10	
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Concatenate the contents of the memory location at the EA with the low-order 4 bits of A. Rotate the concatenated 12-bit data left by 4 bits. The high-order 4 bits of A are not affected.					
ROLD A,(IY+d)	1 1 0 1 0 1 0 1	d d d d d d d d	1 1 1 1 0 1 1 0	1 – – – – –	10	
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Concatenate the contents of the memory location at the EA with the low-order 4 bits of A. Rotate the concatenated 12-bit data left by 4 bits. The high-order 4 bits of A are not affected.					
ROLD A,(SP+d)	1 1 0 1 0 1 1 0	d d d d d d d d	1 1 1 1 0 1 1 0	1 – – – – –	10	
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Concatenate the contents of the memory location at the EA with the low-order 4 bits of A. Rotate the concatenated 12-bit data left by 4 bits. The high-order 4 bits of A are not affected.					
ROLD A,(HL+d)	1 1 0 1 0 1 1 1	d d d d d d d d	1 1 1 1 0 1 1 0	1 – – – – –	10	
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Concatenate the contents of the memory location at the EA with the low-order 4 bits of A. Rotate the concatenated 12-bit data left by 4 bits. The high-order 4 bits of A are not affected.					
ROLD A,(HL+C)	1 1 1 0 0 1 1 1	1 1 1 1 0 1 1 0		1 – – – – –	10	
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Concatenate the contents of the memory location at the EA with the low-order 4 bits of A. Rotate the concatenated 12-bit data left by 4 bits. The high-order 4 bits of A are not affected.					
ROLD A,(+SP)	1 1 1 0 0 1 1 0	1 1 1 1 0 1 1 0		1 – – – – –	9	
	Concatenate the contents of the memory location at (SP+1) with the low-order 4 bits of A. Rotate the concatenated 12-bit data left by 4 bits. The high-order 4 bits of A are not affected.					

Mnemonic	Instruction Code (Binary)			Flag J Z C H S V	Cycle	Operation
ROLD A,(PC+A)[Note]	0 1 0 0 1 1 1 1	1 1 1 1 0 1 1 0		1 – – – – –	10	
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Concatenate the contents of the memory location at the EA with the low-order 4 bits of A. Rotate the concatenated 12-bit data left by 4 bits. The high-order 4 bits of A are not affected.					
RORD A,(x)	1 1 1 0 0 0 0 0	x x x x x x x x	1 1 1 1 0 1 1 1	1 – – – – –	9	
	Concatenate the contents of the memory location directly addressed by x with the low-order 4 bits of A. Rotate the concatenated 12-bit data right by 4 bits. The high-order 4 bits of A are not affected. (0x0000 ≤ x ≤ 0x00FF) Example: Assume A and the memory location at address 0x0087 contain 0x12 and 0x56 respectively. After executing the instruction "ROLD A, (0x87)", A has 0x16 and the memory location at address 0x0087 has 0x25.					
RORD A,(vw)	1 1 1 0 0 0 0 1 1 1 1 1 0 1 1 1	w w w w w w w w	v v v v v v v v	1 – – – – –	10	
	Concatenate the contents of the memory location directly addressed by vw with the low-order 4 bits of A. Rotate the concatenated 12-bit data right by 4 bits. The high-order 4 bits of A are not affected. (0x0000 ≤ vw ≤ 0xFFFF)					
RORD A,(DE)	1 1 1 0 0 0 1 0	1 1 1 1 0 1 1 1		1 – – – – –	8	
	Concatenate the contents of the memory location addressed by DE with the low-order 4 bits of A. Rotate the concatenated 12-bit data right by 4 bits. The high-order 4 bits of A are not affected.					
RORD A,(HL)	1 1 1 0 0 0 1 1	1 1 1 1 0 1 1 1		1 – – – – –	8	
	Concatenate the contents of the memory location addressed by HL with the low-order 4 bits of A. Rotate the concatenated 12-bit data right by 4 bits. The high-order 4 bits of A are not affected.					
RORD A,(IX)	1 1 1 0 0 1 0 0	1 1 1 1 0 1 1 1		1 – – – – –	8	
	Concatenate the contents of the memory location addressed by IX with the low-order 4 bits of A. Rotate the concatenated 12-bit data right by 4 bits. The high-order 4 bits of A are not affected.					
RORD A,(IY)	1 1 1 0 0 1 0 1	1 1 1 1 0 1 1 1		1 – – – – –	8	
	Concatenate the contents of the memory location addressed by IY with the low-order 4 bits of A. Rotate the concatenated 12-bit data right by 4 bits. The high-order 4 bits of A are not affected.					
RORD A,(IX+d)	1 1 0 1 0 1 0 0	d d d d d d d d	1 1 1 1 0 1 1 1	1 – – – – –	10	
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Concatenate the contents of the memory location at the EA with the low-order 4 bits of A. Rotate the concatenated 12-bit data right by 4 bits. The high-order 4 bits of A are not affected.					
RORD A,(IY+d)	1 1 0 1 0 1 0 1	d d d d d d d d	1 1 1 1 0 1 1 1	1 – – – – –	10	
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Concatenate the contents of the memory location at the EA with the low-order 4 bits of A. Rotate the concatenated 12-bit data right by 4 bits. The high-order 4 bits of A are not affected.					
RORD A,(SP+d)	1 1 0 1 0 1 1 0	d d d d d d d d	1 1 1 1 0 1 1 1	1 – – – – –	10	
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Concatenate the contents of the memory location at the EA with the low-order 4 bits of A. Rotate the concatenated 12-bit data right by 4 bits. The high-order 4 bits of A are not affected.					

Mnemonic	Instruction Code (Binary)			Flag J Z C H S V	Cycle	Operation
RORD A,(HL+d)	1 1 0 1 0 1 1 1	d d d d d d d d	1 1 1 1 0 1 1 1	1 – – – – –	10	
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Concatenate the contents of the memory location at the EA with the low-order 4 bits of A. Rotate the concatenated 12-bit data right by 4 bits. The high-order 4 bits of A are not affected.					
RORD A,(HL+C)	1 1 1 0 0 1 1 1	1 1 1 1 0 1 1 1		1 – – – – –	10	
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Concatenate the contents of the memory location at the EA with the low-order 4 bits of A. Rotate the concatenated 12-bit data right by 4 bits. The high-order 4 bits of A are not affected.					
RORD A,(+SP)	1 1 1 0 0 1 1 0	1 1 1 1 0 1 1 1		1 – – – – –	9	
	Concatenate the contents of the memory location at (SP+1) with the low-order 4 bits of A. Rotate the concatenated 12-bit data right by 4 bits. The high-order 4 bits of A are not affected.					
RORD A,(PC+A)[Note]	0 1 0 0 1 1 1 1	1 1 1 1 0 1 1 1		1 – – – – –	10	
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Concatenate the contents of the memory location at the EA with the low-order 4 bits of A. Rotate the concatenated 12-bit data right by 4 bits. The high-order 4 bits of A are not affected.					

Note: There are restrictions on instructions that uses the operand (PC + A). For more details, see "1.4 Addressing Mode".

2.4 Bit and Flag Manipulation Instructions

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
SET g.b	1 1 1 0 1 g g g | 1 1 0 0 0 b b b	Z * – – – –	3	ZF ← $\overline{\text{g.b}}$:g.b ← 1
	Invert bit b of 8-bit register g and place the result into the Zero flag. Then set the specified register bit of g to 1. Example: Assume A contains 0x3C. After executing the instruction “SET A. 7”, A has 0xBC; and ZF is set to 1.			
SET (x).b	1 1 0 0 0 b b b | x x x x x x x x	Z * – – – –	4	ZF ← $\overline{\text{(x).b}}$:(x).b ← 1
	Invert bit b in the memory location directly addressed by x and place the result into the Zero flag. Then set the specified memory bit to 1. (0x0000 ≤ x ≤ 0x00FF)			
SET (vw).b	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v 1 1 0 0 0 b b b	Z * – – – –	6	ZF ← $\overline{\text{(vw).b}}$:(vw).b ← 1
	Invert bit b in the memory location directly addressed by vw and place the result into the Zero flag. Then set the specified memory bit to 1. (0x0000 ≤ vw ≤ 0xFFFF)			
SET (DE).b	1 1 1 0 0 0 1 0 | 1 1 0 0 0 b b b	Z * – – – –	4	ZF ← $\overline{\text{(DE).b}}$:(DE).b ← 1
	Invert bit b in the memory location addressed by DE and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (HL).b	1 1 1 0 0 0 1 1 | 1 1 0 0 0 b b b	Z * – – – –	4	ZF ← $\overline{\text{(HL).b}}$:(HL).b ← 1
	Invert bit b in the memory location addressed by HL and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (IX).b	1 1 1 0 0 1 0 0 | 1 1 0 0 0 b b b	Z * – – – –	4	ZF ← $\overline{\text{(IX).b}}$:(IX).b ← 1
	Invert bit b in the memory location addressed by IX and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (IY).b	1 1 1 0 0 1 0 1 | 1 1 0 0 0 b b b	Z * – – – –	4	ZF ← $\overline{\text{(IY).b}}$:(IY).b ← 1
	Invert bit b in the memory location addressed by IY and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (IX+d).b	1 1 0 1 0 1 0 0 | d d d d d d d d | 1 1 0 0 0 b b b	Z * – – – –	6	ZF ← $\overline{\text{(IX+d).b}}$:(IX+d).b ← 1
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (IY+d).b	1 1 0 1 0 1 0 1 | d d d d d d d d | 1 1 0 0 0 b b b	Z * – – – –	6	ZF ← $\overline{\text{(IY+d).b}}$:(IY+d).b ← 1
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (SP+d).b	1 1 0 1 0 1 1 0 | d d d d d d d d | 1 1 0 0 0 b b b	Z * – – – –	6	ZF ← $\overline{\text{(SP+d).b}}$:(SP+d).b ← 1
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (HL+d).b	1 1 0 1 0 1 1 1 | d d d d d d d d | 1 1 0 0 0 b b b	Z * – – – –	6	ZF ← $\overline{\text{(HL+d).b}}$:(HL+d).b ← 1
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (HL+C).b	1 1 1 0 0 1 1 1 | 1 1 0 0 0 b b b	Z * – – – –	6	ZF ← $\overline{\text{(HL+C).b}}$:(HL+C).b ← 1
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (+SP).b	1 1 1 0 0 1 1 0 | 1 1 0 0 0 b b b	Z * – – – –	5	SP ← SP+1:ZF ← $\overline{\text{(SP).b}}$:(SP).b ← 1
	Invert bit b in the memory location at (SP+1) and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (PC+A).b[Note]	0 1 0 0 1 1 1 1 | 1 1 0 0 0 b b b	Z * – – – –	6	ZF ← $\overline{\text{(PC+A).b}}$:(PC+A).b ← 1
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (x).A	1 1 1 0 0 0 0 0 | x x x x x x x x | 1 1 1 1 0 0 1 0	Z * – – – –	5	ZF ← $\overline{\text{(x).A}}$:(x).A ← 1
	Invert the bit specified by the low-order 3 bits of A in the memory location directly addressed by x and place the result into the Zero flag. Then set the specified memory bit to 1. (0x0000 ≤ x ≤ 0x00FF)			
SET (vw).A	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v 1 1 1 1 0 0 1 0	Z * – – – –	6	ZF ← $\overline{\text{(vw).A}}$:(vw).A ← 1
	Invert the bit specified by the low-order 3 bits of A in the memory location directly addressed by vw and place the result into the Zero flag. Then set the specified memory bit to 1. (0x0000 ≤ vw ≤ 0xFFFF)			
SET (DE).A	1 1 1 0 0 0 1 0 | 1 1 1 1 0 0 1 0	Z * – – – –	4	ZF ← $\overline{\text{(DE).A}}$:(DE).A ← 1
	Invert the bit specified by the low-order 3 bits of A in the memory location addressed by DE and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (HL).A	1 1 1 0 0 0 1 1 | 1 1 1 1 0 0 1 0	Z * – – – –	4	ZF ← $\overline{\text{(HL).A}}$:(HL).A ← 1
	Invert the bit specified by the low-order 3 bits of A in the memory location addressed by HL and place the result into the Zero flag. Then set the specified memory bit to 1.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
SET (IX).A	1 1 1 0 0 1 0 0 | 1 1 1 1 0 0 1 0	Z * – – – –	4	ZF ← $\overline{(IX).A}$:(IX).A ← 1
	Invert the bit specified by the low-order 3 bits of A in the memory location addressed by IX and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (IY).A	1 1 1 0 0 1 0 1 | 1 1 1 1 0 0 1 0	Z * – – – –	4	ZF ← $\overline{(IY).A}$:(IY).A ← 1
	Invert the bit specified by the low-order 3 bits of A in the memory location addressed by IY and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (IX+d).A	1 1 0 1 0 1 0 0 | d d d d d d d d | 1 1 1 1 0 0 1 0	Z * – – – –	6	ZF ← $\overline{(IX+d).A}$:(IX+d).A ← 1
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (IY+d).A	1 1 0 1 0 1 0 1 | d d d d d d d d | 1 1 1 1 0 0 1 0	Z * – – – –	6	ZF ← $\overline{(IY+d).A}$:(IY+d).A ← 1
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (SP+d).A	1 1 0 1 0 1 1 0 | d d d d d d d d | 1 1 1 1 0 0 1 0	Z * – – – –	6	ZF ← $\overline{(SP+d).A}$:(SP+d).A ← 1
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (HL+d).A	1 1 0 1 0 1 1 1 | d d d d d d d d | 1 1 1 1 0 0 1 0	Z * – – – –	6	ZF ← $\overline{(HL+d).A}$:(HL+d).A ← 1
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (HL+C).A	1 1 1 0 0 1 1 1 | 1 1 1 1 0 0 1 0	Z * – – – –	6	ZF ← $\overline{(HL+C).A}$:(HL+C).A ← 1
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (+SP).A	1 1 1 0 0 1 1 0 | 1 1 1 1 0 0 1 0	Z * – – – –	5	SP ← SP+1:ZF ← $\overline{(SP).A}$:(SP).A ← 1
	Invert the bit specified by the low-order 3 bits of A in the memory location at (SP+1) and place the result into the Zero flag. Then set the specified memory bit to 1.			
SET (PC+A).A[Note]	0 1 0 0 1 1 1 1 | 1 1 1 1 0 0 1 0	Z * – – – –	6	ZF ← $\overline{(PC+A).A}$:(PC+A).A ← 1
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Zero flag. Then set the specified memory bit to 1.			
CLR g.b	1 1 1 0 1 g g g | 1 1 0 0 1 b b b	Z * – – – –	3	ZF ← $\overline{g.b}$:g.b ← 0
	Invert bit b of 8-bit register g and place the result into the Zero flag. Then clear the specified register bit of g to 0. Example: Assume A contains 0x3C. After executing the instruction “CLR A. 2”, A has 0x38; and ZF is cleared to 0.			
CLR (x).b	1 1 0 0 1 b b b | x x x x x x x x	Z * – – – –	4	ZF ← $\overline{(x).b}$:(x).b ← 0
	Invert bit b in the memory location directly addressed by x and place the result into the Zero flag. Then clear the specified memory bit to 0. (0x0000 ≤ x ≤ 0x00FF)			
CLR (vw).b	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v 1 1 0 0 1 b b b	Z * – – – –	6	ZF ← $\overline{(vw).b}$:(vw).b ← 0
	Invert bit b in the memory location directly addressed by vw and place the result into the Zero flag. Then clear the specified memory bit to 0. (0x0000 ≤ vw ≤ 0xFFFF)			
CLR (DE).b	1 1 1 0 0 0 1 0 | 1 1 0 0 1 b b b	Z * – – – –	4	ZF ← $\overline{(DE).b}$:(DE).b ← 0
	Invert bit b in the memory location addressed by DE and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (HL).b	1 1 1 0 0 0 1 1 | 1 1 0 0 1 b b b	Z * – – – –	4	ZF ← $\overline{(HL).b}$:(HL).b ← 0
	Invert bit b in the memory location addressed by HL and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (IX).b	1 1 1 0 0 1 0 0 | 1 1 0 0 1 b b b	Z * – – – –	4	ZF ← $\overline{(IX).b}$:(IX).b ← 0
	Invert bit b in the memory location addressed by IX and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (IY).b	1 1 1 0 0 1 0 1 | 1 1 0 0 1 b b b	Z * – – – –	4	ZF ← $\overline{(IY).b}$:(IY).b ← 0
	Invert bit b in the memory location addressed by IY and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (IX+d).b	1 1 0 1 0 1 0 0 | d d d d d d d d | 1 1 0 0 1 b b b	Z * – – – –	6	ZF ← $\overline{(IX+d).b}$:(IX+d).b ← 0
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (IY+d).b	1 1 0 1 0 1 0 1 | d d d d d d d d | 1 1 0 0 1 b b b	Z * – – – –	6	ZF ← $\overline{(IY+d).b}$:(IY+d).b ← 0
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (SP+d).b	1 1 0 1 0 1 1 0 | d d d d d d d d | 1 1 0 0 1 b b b	Z * – – – –	6	ZF ← $\overline{(SP+d).b}$:(SP+d).b ← 0
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Then clear the specified memory bit to 0.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
CLR (HL+d).b	1 1 0 1 0 1 1 1 d d d d d d d d 1 1 0 0 1 b b b	Z * – – – –	6	ZF ← $\overline{(HL+d).b}$:(HL+d).b ← 0
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (HL+C).b	1 1 1 0 0 1 1 1 1 1 0 0 1 b b b	Z * – – – –	6	ZF ← $\overline{(HL+C).b}$:(HL+C).b ← 0
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (+SP).b	1 1 1 0 0 1 1 0 1 1 0 0 1 b b b	Z * – – – –	5	SP ← SP+1:ZF ← $\overline{(SP).b}$:(SP).b ← 0
	Invert bit b in the memory location at (SP+1) and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (PC+A).b[Note]	0 1 0 0 1 1 1 1 1 1 0 0 1 b b b	Z * – – – –	6	ZF ← $\overline{(PC+A).b}$:(PC+A).b ← 0
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (x).A	1 1 1 0 0 0 0 0 x x x x x x x x 1 1 1 1 1 0 1 0	Z * – – – –	5	ZF ← $\overline{(x).A}$:(x).A ← 0
	Invert the bit specified by the low-order 3 bits of A in the memory location directly addressed by x and place the result into the Zero flag. Then clear the specified memory bit to 0. (0x0000 ≤ x ≤ 0x00FF)			
CLR (vw).A	1 1 1 0 0 0 0 1 w w w w w w w w v v v v v v v v 1 1 1 1 1 0 1 0	Z * – – – –	6	ZF ← $\overline{(vw).A}$:(vw).A ← 0
	Invert the bit specified by the low-order 3 bits of A in the memory location directly addressed by vw and place the result into the Zero flag. Then clear the specified memory bit to 0. (0x0000 ≤ vw ≤ 0xFFFF)			
CLR (DE).A	1 1 1 0 0 0 1 0 1 1 1 1 1 0 1 0	Z * – – – –	4	ZF ← $\overline{(DE).A}$:(DE).A ← 0
	Invert the bit specified by the low-order 3 bits of A in the memory location addressed by DE and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (HL).A	1 1 1 0 0 0 1 1 1 1 1 1 1 0 1 0	Z * – – – –	4	ZF ← $\overline{(HL).A}$:(HL).A ← 0
	Invert the bit specified by the low-order 3 bits of A in the memory location addressed by HL and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (IX).A	1 1 1 0 0 1 0 0 1 1 1 1 1 0 1 0	Z * – – – –	4	ZF ← $\overline{(IX).A}$:(IX).A ← 0
	Invert the bit specified by the low-order 3 bits of A in the memory location addressed by IX and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (IY).A	1 1 1 0 0 1 0 1 1 1 1 1 1 0 1 0	Z * – – – –	4	ZF ← $\overline{(IY).A}$:(IY).A ← 0
	Invert the bit specified by the low-order 3 bits of A in the memory location addressed by IY and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (IX+d).A	1 1 0 1 0 1 0 0 d d d d d d d d 1 1 1 1 1 0 1 0	Z * – – – –	6	ZF ← $\overline{(IX+d).A}$:(IX+d).A ← 0
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (IY+d).A	1 1 0 1 0 1 0 1 d d d d d d d d 1 1 1 1 1 0 1 0	Z * – – – –	6	ZF ← $\overline{(IY+d).A}$:(IY+d).A ← 0
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (SP+d).A	1 1 0 1 0 1 1 0 d d d d d d d d 1 1 1 1 1 0 1 0	Z * – – – –	6	ZF ← $\overline{(SP+d).A}$:(SP+d).A ← 0
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (HL+d).A	1 1 0 1 0 1 1 1 d d d d d d d d 1 1 1 1 1 0 1 0	Z * – – – –	6	ZF ← $\overline{(HL+d).A}$:(HL+d).A ← 0
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (HL+C).A	1 1 1 0 0 1 1 1 1 1 1 1 1 0 1 0	Z * – – – –	6	ZF ← $\overline{(HL+C).A}$:(HL+C).A ← 0
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (+SP).A	1 1 1 0 0 1 1 0 1 1 1 1 1 0 1 0	Z * – – – –	5	SP ← SP+1:ZF ← $\overline{(SP).A}$:(SP).A ← 0
	Invert the bit specified by the low-order 3 bits of A in the memory location at (SP+1) and place the result into the Zero flag. Then clear the specified memory bit to 0.			
CLR (PC+A).A[Note]	0 1 0 0 1 1 1 1 1 1 1 1 1 0 1 0	Z * – – – –	6	ZF ← $\overline{(PC+A).A}$:(PC+A).A ← 0
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Zero flag. Then clear the specified memory bit to 0.			
LD CF,g.b	1 1 1 0 1 g g g 0 1 0 1 1 b b b	$\overline{C}$ – * – – –	2	CF ← g.b
	Load the value of bit b of 8-bit register g into the Carry flag. Example: Assume A contains 0y01101101. Then, the instruction “LD CF, A. 4” clears CF to 0 and sets JF to 1.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
LD CF,(x).b	0 1 0 1 1 b b b | x x x x x x x x	$\overline{C}$ – * – – –	3	CF ← (x).b
	Load the value of bit b in the memory location directly addressed by x into the Carry flag. (0x0000 ≤ x ≤ 0x00FF)			
LD CF,(vw).b	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v 0 1 0 1 1 b b b	$\overline{C}$ – * – – –	5	CF ← (vw).b
	Load the value of bit b in the memory location directly addressed by vw into the Carry flag. (0x0000 ≤ vw ≤ 0xFFFF)			
LD CF,(DE).b	1 1 1 0 0 0 1 0 | 0 1 0 1 1 b b b	$\overline{C}$ – * – – –	3	CF ← (DE).b
	Load the value of bit b in the memory location addressed by DE into the Carry flag.			
LD CF,(HL).b	1 1 1 0 0 0 1 1 | 0 1 0 1 1 b b b	$\overline{C}$ – * – – –	3	CF ← (HL).b
	Load the value of bit b in the memory location addressed by HL into the Carry flag.			
LD CF,(IX).b	1 1 1 0 0 1 0 0 | 0 1 0 1 1 b b b	$\overline{C}$ – * – – –	3	CF ← (IX).b
	Load the value of bit b in the memory location addressed by IX into the Carry flag.			
LD CF,(IY).b	1 1 1 0 0 1 0 1 | 0 1 0 1 1 b b b	$\overline{C}$ – * – – –	3	CF ← (IY).b
	Load the value of bit b in the memory location addressed by IY into the Carry flag.			
LD CF,(IX+d).b	1 1 0 1 0 1 0 0 | d d d d d d d d | 0 1 0 1 1 b b b	$\overline{C}$ – * – – –	5	CF ← (IX+d).b
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Load the value of bit b in the memory location at the EA into the Carry flag.			
LD CF,(IY+d).b	1 1 0 1 0 1 0 1 | d d d d d d d d | 0 1 0 1 1 b b b	$\overline{C}$ – * – – –	5	CF ← (IY+d).b
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Load the value of bit b in the memory location at the EA into the Carry flag.			
LD CF,(SP+d).b	1 1 0 1 0 1 1 0 | d d d d d d d d | 0 1 0 1 1 b b b	$\overline{C}$ – * – – –	5	CF ← (SP+d).b
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Load the value of bit b in the memory location at the EA into the Carry flag.			
LD CF,(HL+d).b	1 1 0 1 0 1 1 1 | d d d d d d d d | 0 1 0 1 1 b b b	$\overline{C}$ – * – – –	5	CF ← (HL+d).b
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Load the value of bit b in the memory location at the EA into the Carry flag.			
LD CF,(HL+C).b	1 1 1 0 0 1 1 1 | 0 1 0 1 1 b b b	$\overline{C}$ – * – – –	5	CF ← (HL+C).b
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Load the value of bit b in the memory location at the EA into the Carry flag.			
LD CF,(+SP).b	1 1 1 0 0 1 1 0 | 0 1 0 1 1 b b b	$\overline{C}$ – * – – –	4	SP ← SP+1:CF ← (SP).b
	Load the value of bit b in the memory location at (SP+1) into the Carry flag.			
LD CF,(PC+A).b[Note]	0 1 0 0 1 1 1 1 | 0 1 0 1 1 b b b	$\overline{C}$ – * – – –	5	CF ← (PC+A).b
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Load the value of bit b in the memory location at the EA into the Carry flag.			
LD CF,(x).A	1 1 1 0 0 0 0 0 | x x x x x x x x | 1 1 1 1 1 1 0 0	$\overline{C}$ – * – – –	4	CF ← (x).A
	Load the value of the bit specified by the low-order 3 bits of A in the memory location directly addressed by x into the Carry flag. (0x0000 ≤ x ≤ 0x00FF)			
LD CF,(vw).A	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v 1 1 1 1 1 1 0 0	$\overline{C}$ – * – – –	5	CF ← (vw).A
	Load the value of the bit specified by the low-order 3 bits of A in the memory location directly addressed by vw into the Carry flag. (0x0000 ≤ vw ≤ 0xFFFF)			
LD CF,(DE).A	1 1 1 0 0 0 1 0 | 1 1 1 1 1 1 0 0	$\overline{C}$ – * – – –	3	CF ← (DE).A
	Load the value of the bit specified by the low-order 3 bits of A in the memory location addressed by DE into the Carry flag.			
LD CF,(HL).A	1 1 1 0 0 0 1 1 | 1 1 1 1 1 1 0 0	$\overline{C}$ – * – – –	3	CF ← (HL).A
	Load the value of the bit specified by the low-order 3 bits of A in the memory location addressed by HL into the Carry flag.			
LD CF,(IX).A	1 1 1 0 0 1 0 0 | 1 1 1 1 1 1 0 0	$\overline{C}$ – * – – –	3	CF ← (IX).A
	Load the value of the bit specified by the low-order 3 bits of A in the memory location addressed by IX into the Carry flag.			
LD CF,(IY).A	1 1 1 0 0 1 0 1 | 1 1 1 1 1 1 0 0	$\overline{C}$ – * – – –	3	CF ← (IY).A
	Load the value of the bit specified by the low-order 3 bits of A in the memory location addressed by IY into the Carry flag.			
LD CF,(IX+d).A	1 1 0 1 0 1 0 0 | d d d d d d d d | 1 1 1 1 1 1 0 0	$\overline{C}$ – * – – –	5	CF ← (IX+d).A
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Load the value of the bit specified by the low-order 3 bits of A in the memory location at the EA into the Carry flag.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
LD CF,(IY+d).A	1 1 0 1 0 1 0 1 d d d d d d d d 1 1 1 1 1 1 0 0	$\overline{C}$ – * – – –	5	CF ← (IY+d).A
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Load the value of the bit specified by the low-order 3 bits of A in the memory location at the EA into the Carry flag.			
LD CF,(SP+d).A	1 1 0 1 0 1 1 0 d d d d d d d d 1 1 1 1 1 1 0 0	$\overline{C}$ – * – – –	5	CF ← (SP+d).A
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Load the value of the bit specified by the low-order 3 bits of A in the memory location at the EA into the Carry flag.			
LD CF,(HL+d).A	1 1 0 1 0 1 1 1 d d d d d d d d 1 1 1 1 1 1 0 0	$\overline{C}$ – * – – –	5	CF ← (HL+d).A
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Load the value of the bit specified by the low-order 3 bits of A in the memory location at the EA into the Carry flag.			
LD CF,(HL+C).A	1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 0	$\overline{C}$ – * – – –	5	CF ← (HL+C).A
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Load the value of the bit specified by the low-order 3 bits of A in the memory location at the EA into the Carry flag.			
LD CF,(+SP).A	1 1 1 0 0 1 1 0 1 1 1 1 1 1 0 0	$\overline{C}$ – * – – –	4	SP ← SP+1:CF ← (SP).A
	Load the value of the bit specified by the low-order 3 bits of A in the memory location at (SP+1) into the Carry flag.			
LD CF,(PC+A).A Note	0 1 0 0 1 1 1 1 1 1 1 1 1 1 0 0	$\overline{C}$ – * – – –	5	CF ← (PC+A).A
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Load the value of the bit specified by the low-order 3 bits of A in the memory location at the EA into the Carry flag.			
TEST g.b#1	1 1 1 0 1 g g g 0 1 0 1 1 b b b	* – $\overline{J}$ – – –	2	JF ← $\overline{g.b}$
	Invert bit b of 8-bit register g and place the result into the Jump Status flag. Example: Assume A contains 0y01011100. Then, the instruction “TEST A. 5” sets ZF to 1 and clears CF to 0.			
TEST (x).b#1	0 1 0 1 1 b b b x x x x x x x x	* – $\overline{J}$ – – –	3	JF ← $\overline{(x).b}$
	Invert bit b in the memory location directly addressed by x and place the result into the Jump Status flag. (0x0000 ≤ x ≤ 0x00FF)			
TEST (vw).b#1	1 1 1 0 0 0 0 1 w w w w w w w w v v v v v v v v 0 1 0 1 1 b b b	* – $\overline{J}$ – – –	5	JF ← $\overline{(vw).b}$
	Invert bit b in the memory location directly addressed by vw and place the result into the Jump Status flag. (0x0000 ≤ vw ≤ 0xFFFF)			
TEST (DE).b#1	1 1 1 0 0 0 1 0 0 1 0 1 1 b b b	* – $\overline{J}$ – – –	3	JF ← $\overline{(DE).b}$
	Invert bit b in the memory location addressed by DE and place the result into the Jump Status flag.			
TEST (HL).b#1	1 1 1 0 0 0 1 1 0 1 0 1 1 b b b	* – $\overline{J}$ – – –	3	JF ← $\overline{(HL).b}$
	Invert bit b in the memory location addressed by HL and place the result into the Jump Status flag.			
TEST (IX).b#1	1 1 1 0 0 1 0 0 0 1 0 1 1 b b b	* – $\overline{J}$ – – –	3	JF ← $\overline{(IX).b}$
	Invert bit b in the memory location addressed by IX and place the result into the Jump Status flag.			
TEST (IY).b#1	1 1 1 0 0 1 0 1 0 1 0 1 1 b b b	* – $\overline{J}$ – – –	3	JF ← $\overline{(IY).b}$
	Invert bit b in the memory location addressed by IY and place the result into the Jump Status flag.			
TEST (IX+d).b#1	1 1 0 1 0 1 0 0 d d d d d d d d 0 1 0 1 1 b b b	* – $\overline{J}$ – – –	5	JF ← $\overline{(IX+d).b}$
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Jump Status flag.			
TEST (IY+d).b#1	1 1 0 1 0 1 0 1 d d d d d d d d 0 1 0 1 1 b b b	* – $\overline{J}$ – – –	5	JF ← $\overline{(IY+d).b}$
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Jump Status flag.			
TEST (SP+d).b#1	1 1 0 1 0 1 1 0 d d d d d d d d 0 1 0 1 1 b b b	* – $\overline{J}$ – – –	5	JF ← $\overline{(SP+d).b}$
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Jump Status flag.			
TEST (HL+d).b#1	1 1 0 1 0 1 1 1 d d d d d d d d 0 1 0 1 1 b b b	* – $\overline{J}$ – – –	5	JF ← $\overline{(HL+d).b}$
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Jump Status flag.			
TEST (HL+C).b#1	1 1 1 0 0 1 1 1 0 1 0 1 1 b b b	* – $\overline{J}$ – – –	5	JF ← $\overline{(HL+C).b}$
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Jump Status flag.			
TEST (+SP).b#1	1 1 1 0 0 1 1 0 0 1 0 1 1 b b b	* – $\overline{J}$ – – –	4	SP ← SP+1:JF ← $\overline{(SP).b}$
	Invert bit b in the memory location at (SP+1) and place the result into the Jump Status flag.			
TEST (PC+A).b#1Note	0 1 0 0 1 1 1 1 0 1 0 1 1 b b b	* – $\overline{J}$ – – –	5	JF ← $\overline{(PC+A).b}$
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Jump Status flag.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
TEST (x).A#1	1 1 1 0 0 0 0 0 | x x x x x x x x | 1 1 1 1 1 1 0 0	* – J – – –	4	JF ← $\overline{(x).A}$
	Invert the bit specified by the low-order 3 bits of A in the memory location directly addressed by x and place the result into the Jump Status flag. (0x0000 ≤ x ≤ 0x00FF)			
TEST (vw).A#1	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v 1 1 1 1 1 1 0 0	* – J – – –	5	JF ← $\overline{(vw).A}$
	Invert the bit specified by the low-order 3 bits of A in the memory location directly addressed by vw and place the result into the Jump Status flag. (0x0000 ≤ vw ≤ 0xFFFF)			
TEST (DE).A#1	1 1 1 0 0 0 1 0 | 1 1 1 1 1 1 0 0	* – J – – –	3	JF ← $\overline{(DE).A}$
	Invert the bit specified by the low-order 3 bits of A in the memory location addressed by DE and place the result into the Jump Status flag.			
TEST (HL).A#1	1 1 1 0 0 0 1 1 | 1 1 1 1 1 1 0 0	* – J – – –	3	JF ← $\overline{(HL).A}$
	Invert the bit specified by the low-order 3 bits of A in the memory location addressed by HL and place the result into the Jump Status flag.			
TEST (IX).A#1	1 1 1 0 0 1 0 0 | 1 1 1 1 1 1 0 0	* – J – – –	3	JF ← $\overline{(IX).A}$
	Invert the bit specified by the low-order 3 bits of A in the memory location addressed by IX and place the result into the Jump Status flag.			
TEST (IY).A#1	1 1 1 0 0 1 0 1 | 1 1 1 1 1 1 0 0	* – J – – –	3	JF ← $\overline{(IY).A}$
	Invert the bit specified by the low-order 3 bits of A in the memory location addressed by IY and place the result into the Jump Status flag.			
TEST (IX+d).A#1	1 1 0 1 0 1 0 0 | d d d d d d d d | 1 1 1 1 1 1 0 0	* – J – – –	5	JF ← $\overline{(IX+d).A}$
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Jump Status flag.			
TEST (IY+d).A#1	1 1 0 1 0 1 0 1 | d d d d d d d d | 1 1 1 1 1 1 0 0	* – J – – –	5	JF ← $\overline{(IY+d).A}$
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Jump Status flag.			
TEST (SP+d).A#1	1 1 0 1 0 1 1 0 | d d d d d d d d | 1 1 1 1 1 1 0 0	* – J – – –	5	JF ← $\overline{(SP+d).A}$
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Jump Status flag.			
TEST (HL+d).A#1	1 1 0 1 0 1 1 1 | d d d d d d d d | 1 1 1 1 1 1 0 0	* – J – – –	5	JF ← $\overline{(HL+d).A}$
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Jump Status flag.			
TEST (HL+C).A#1	1 1 1 0 0 1 1 1 | 1 1 1 1 1 1 0 0	* – J – – –	5	JF ← $\overline{(HL+C).A}$
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Jump Status flag.			
TEST (+SP).A#1	1 1 1 0 0 1 1 0 | 1 1 1 1 1 1 0 0	* – J – – –	4	SP ← SP+1:JF ← $\overline{(SP).A}$
	Invert the bit specified by the low-order 3 bits of A in the memory location at (SP+1) and place the result into the Jump Status flag.			
TEST (PC+A).A#1Note	0 1 0 0 1 1 1 1 | 1 1 1 1 1 1 0 0	* – J – – –	5	JF ← $\overline{(PC+A).A}$
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA and place the result into the Jump Status flag.			
LD g.b,CF	1 1 1 0 1 g g g | 1 1 1 0 1 b b b	1 – – – – –	2	g.b ← CF
	Store the contents of the Carry flag into bit b of 8-bit register g. Example: Assume A contains 0x15 and CF is set. After executing the instruction “LD A. 5, CF”, A has 0x35; CF is not affected.			
LD (x).b,CF	1 1 1 0 0 0 0 0 | x x x x x x x x | 1 1 1 0 1 b b b	1 – – – – –	5	(x).b ← CF
	Store the contents of the Carry flag into bit b in the memory location directly addressed by x. (0x0000 ≤ x ≤ 0x00FF)			
LD (vw).b,CF	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v 1 1 1 0 1 b b b	1 – – – – –	6	(vw).b ← CF
	Store the contents of the Carry flag into bit b in the memory location directly addressed by vw. (0x0000 ≤ vw ≤ 0xFFFF)			
LD (DE).b,CF	1 1 1 0 0 0 1 0 | 1 1 1 0 1 b b b	1 – – – – –	4	(DE).b ← CF
	Store the contents of the Carry flag into bit b in the memory location addressed by DE.			
LD (HL).b,CF	1 1 1 0 0 0 1 1 | 1 1 1 0 1 b b b	1 – – – – –	4	(HL).b ← CF
	Store the contents of the Carry flag into bit b in the memory location addressed by HL			
LD (IX).b,CF	1 1 1 0 0 1 0 0 | 1 1 1 0 1 b b b	1 – – – – –	4	(IX).b ← CF
	Store the contents of the Carry flag into bit b in the memory location addressed by IX.			
LD (IY).b,CF	1 1 1 0 0 1 0 1 | 1 1 1 0 1 b b b	1 – – – – –	4	(IY).b ← CF
	Store the contents of the Carry flag into bit b in the memory location addressed by IY.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
LD (IX+d).b,CF	1 1 0 1 0 1 0 0 d d d d d d d d 1 1 1 0 1 b b b	1 – – – – –	6	(IX+d).b ← CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Store the contents of the Carry flag into bit b in the memory location at the EA.			
LD (IY+d).b,CF	1 1 0 1 0 1 0 1 d d d d d d d d 1 1 1 0 1 b b b	1 – – – – –	6	(IY+d).b ← CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Store the contents of the Carry flag into bit b in the memory location at the EA.			
LD (SP+d).b,CF	1 1 0 1 0 1 1 0 d d d d d d d d 1 1 1 0 1 b b b	1 – – – – –	6	(SP+d).b ← CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Store the contents of the Carry flag into bit b in the memory location at the EA.			
LD (HL+d).b,CF	1 1 0 1 0 1 1 1 d d d d d d d d 1 1 1 0 1 b b b	1 – – – – –	6	(HL+d).b ← CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Store the contents of the Carry flag into bit b in the memory location at the EA.			
LD (HL+C).b,CF	1 1 1 0 0 1 1 1 1 1 1 0 1 b b b	1 – – – – –	6	(HL+C).b ← CF
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Store the contents of the Carry flag into bit b in the memory location at the EA.			
LD (+SP).b,CF	1 1 1 0 0 1 1 0 1 1 1 0 1 b b b	1 – – – – –	5	SP ← SP+1:(SP).b ← CF
	Store the contents of the Carry flag into bit b in the memory location at (SP+1)			
LD (PC+A).b,CF[Note]	0 1 0 0 1 1 1 1 1 1 1 0 1 b b b	1 – – – – –	6	(PC+A).b ← CF
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Store the contents of the Carry flag into bit b in the memory location at the EA.			
LD (x).A,CF	1 1 1 0 0 0 0 0 x x x x x x x x 1 1 1 1 0 0 1 1	1 – – – – –	5	(x).A ← CF
	Store the contents of the Carry flag into the bit specified by the low-order 3 bits of A in the memory location directly addressed by x. (0x0000 ≤ x ≤ 0x00FF)			
LD (vw).A,CF	1 1 1 0 0 0 0 1 w w w w w w w w v v v v v v v v 1 1 1 1 0 0 1 1	1 – – – – –	6	(vw).A ← CF
	Store the contents of the Carry flag into the bit specified by the low-order 3 bits of A in the memory location directly addressed by vw. (0x0000 ≤ vw ≤ 0xFFFF)			
LD (DE).A,CF	1 1 1 0 0 0 1 0 1 1 1 1 0 0 1 1	1 – – – – –	4	(DE).A ← CF
	Store the contents of the Carry flag into the bit specified by the low-order 3 bits of A in the memory location addressed by DE.			
LD (HL).A,CF	1 1 1 0 0 0 1 1 1 1 1 1 0 0 1 1	1 – – – – –	4	(HL).A ← CF
	Store the contents of the Carry flag into the bit specified by the low-order 3 bits of A in the memory location addressed by HL.			
LD (IX).A,CF	1 1 1 0 0 1 0 0 1 1 1 1 0 0 1 1	1 – – – – –	4	(IX).A ← CF
	Store the contents of the Carry flag into the bit specified by the low-order 3 bits of A in the memory location addressed by IX.			
LD (IY).A,CF	1 1 1 0 0 1 0 1 1 1 1 1 0 0 1 1	1 – – – – –	4	(IY).A ← CF
	Store the contents of the Carry flag into the bit specified by the low-order 3 bits of A in the memory location addressed by IY.			
LD (IX+d).A,CF	1 1 0 1 0 1 0 0 d d d d d d d d 1 1 1 1 0 0 1 1	1 – – – – –	6	(IX+d).A ← CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Store the contents of the Carry flag into the bit specified by the low-order 3 bits of A in the memory location at the EA.			
LD (IY+d).A,CF	1 1 0 1 0 1 0 1 d d d d d d d d 1 1 1 1 0 0 1 1	1 – – – – –	6	(IY+d).A ← CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Store the contents of the Carry flag into the bit specified by the low-order 3 bits of A in the memory location at the EA.			
LD (SP+d).A,CF	1 1 0 1 0 1 1 0 d d d d d d d d 1 1 1 1 0 0 1 1	1 – – – – –	6	(SP+d).A ← CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Store the contents of the Carry flag into the bit specified by the low-order 3 bits of A in the memory location at the EA.			
LD (HL+d).A,CF	1 1 0 1 0 1 1 1 d d d d d d d d 1 1 1 1 0 0 1 1	1 – – – – –	6	(HL+d).A ← CF
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Store the contents of the Carry flag into the bit specified by the low-order 3 bits of A in the memory location at the EA.			
LD (HL+C).A,CF	1 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1	1 – – – – –	6	(HL+C).A ← CF
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Store the contents of the Carry flag into the bit specified by the low-order 3 bits of A in the memory location at the EA.			
LD (+SP).A,CF	1 1 1 0 0 1 1 0 1 1 1 1 0 0 1 1	1 – – – – –	5	SP ← SP+1:(SP).A ← CF
	Store the contents of the Carry flag into the bit specified by the low-order 3 bits of A in the memory location at (SP+1).			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
LD (PC+A).A,CF[Note]	0 1 0 0 1 1 1 1 | 1 1 1 1 0 0 1 1	1 – – – – –	6	(PC+A).A ← CF
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Store the contents of the Carry flag into the bit specified by the low-order 3 bits of A in the memory location at the EA.			
CPL g.b	1 1 1 0 1 g g g | 1 1 1 0 0 b b b	Z * – – – –	3	ZF ← $\overline{g.b}$:g.b ← $\overline{g.b}$
	Invert bit b of 8-bit register g and place the result into the Zero flag. Also write back the result into the specified register bit of g. Example: Assume A contains 0x3C. After executing the instruction “CPL A. 3”, A has 0x34; and ZF is cleared to 1.			
CPL (x).b	1 1 1 0 0 0 0 0 | x x x x x x x x | 1 1 1 0 0 b b b	Z * – – – –	5	ZF ← $\overline{(x).b}$:(x).b ← $\overline{(x).b}$
	Invert bit b in the memory location directly addressed by x and place the result into the Zero flag. Also write back the result into the specified memory bit. (0x0000 ≤ x ≤ 0x00FF)			
CPL (vw).b	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v 1 1 1 0 0 b b b	Z * – – – –	6	ZF ← $\overline{(vw).b}$:(vw).b ← $\overline{(vw).b}$
	Invert bit b in the memory location directly addressed by vw and place the result into the Zero flag. Also write back the result into the specified memory bit. (0x0000 ≤ vw ≤ 0xFFFF)			
CPL (DE).b	1 1 1 0 0 0 1 0 | 1 1 1 0 0 b b b	Z * – – – –	4	ZF ← $\overline{(DE).b}$:(DE).b ← $\overline{(DE).b}$
	Invert bit b in the memory location addressed by DE and place the result into the Zero flag. Also write back the result into the specified memory bit.			
CPL (HL).b	1 1 1 0 0 0 1 1 | 1 1 1 0 0 b b b	Z * – – – –	4	ZF ← $\overline{(HL).b}$:(HL).b ← $\overline{(HL).b}$
	Invert bit b in the memory location addressed by HL and place the result into the Zero flag. Also write back the result into the specified memory bit.			
CPL (IX).b	1 1 1 0 0 1 0 0 | 1 1 1 0 0 b b b	Z * – – – –	4	ZF ← $\overline{(IX).b}$:(IX).b ← $\overline{(IX).b}$
	Invert bit b in the memory location addressed by IX and place the result into the Zero flag. Also write back the result into the specified memory bit.			
CPL (IY).b	1 1 1 0 0 1 0 1 | 1 1 1 0 0 b b b	Z * – – – –	4	ZF ← $\overline{(IY).b}$:(IY).b ← $\overline{(IY).b}$
	Invert bit b in the memory location addressed by IY and place the result into the Zero flag. Also write back the result into the specified memory bit.			
CPL (IX+d).b	1 1 0 1 0 1 0 0 | d d d d d d d d | 1 1 1 0 0 b b b	Z * – – – –	6	ZF ← $\overline{(IX+d).b}$:(IX+d).b ← $\overline{(IX+d).b}$
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Also write back the result into the specified memory bit.			
CPL (IY+d).b	1 1 0 1 0 1 0 1 | d d d d d d d d | 1 1 1 0 0 b b b	Z * – – – –	6	ZF ← $\overline{(IY+d).b}$:(IY+d).b ← $\overline{(IY+d).b}$
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Also write back the result into the specified memory bit.			
CPL (SP+d).b	1 1 0 1 0 1 1 0 | d d d d d d d d | 1 1 1 0 0 b b b	Z * – – – –	6	ZF ← $\overline{(SP+d).b}$:(SP+d).b ← $\overline{(SP+d).b}$
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Also write back the result into the specified memory bit.			
CPL (HL+d).b	1 1 0 1 0 1 1 1 | d d d d d d d d | 1 1 1 0 0 b b b	Z * – – – –	6	ZF ← $\overline{(HL+d).b}$:(HL+d).b ← $\overline{(HL+d).b}$
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Also write back the result into the specified memory bit.			
CPL (HL+C).b	1 1 1 0 0 1 1 1 | 1 1 1 0 0 b b b	Z * – – – –	6	ZF ← $\overline{(HL+C).b}$:(HL+C).b ← $\overline{(HL+C).b}$
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Also write back the result into the specified memory bit.			
CPL (+SP).b	1 1 1 0 0 1 1 0 | 1 1 1 0 0 b b b	Z * – – – –	5	SP ← SP+1:ZF ← $\overline{(SP).b}$:(SP).b ← $\overline{(SP).b}$
	Invert bit b in the memory location at (SP+1) and place the result into the Zero flag. Also write back the result into the specified memory bit.			
CPL (PC+A).b[Note]	0 1 0 0 1 1 1 1 | 1 1 1 0 0 b b b	Z * – – – –	6	ZF ← $\overline{(PC+A).b}$:(PC+A).b ← $\overline{(PC+A).b}$
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Invert bit b in the memory location at the EA and place the result into the Zero flag. Also write back the result into the specified memory bit.			
CPL (x).A	1 1 1 0 0 0 0 0 | x x x x x x x x | 1 1 1 1 1 0 1 1	Z * – – – –	5	ZF ← $\overline{(x).A}$:(x).A ← $\overline{(x).A}$
	Invert the bit specified by the low-order 3 bits of A in the memory location directly addressed by x, place the result into the Zero flag, and also write back the result into the specified memory bit. (0x0000 ≤ x ≤ 0x00FF)			
CPL (vw).A	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v 1 1 1 1 1 0 1 1	Z * – – – –	6	ZF ← $\overline{(vw).A}$:(vw).A ← $\overline{(vw).A}$
	Invert the bit specified by the low-order 3 bits of A in the memory location directly addressed by vw, place the result into the Zero flag, and also write back the result into the specified memory bit. (0x0000 ≤ vw ≤ 0xFFFF)			
CPL (DE).A	1 1 1 0 0 0 1 0 | 1 1 1 1 1 0 1 1	Z * – – – –	4	ZF ← $\overline{(DE).A}$:(DE).A ← $\overline{(DE).A}$
	Invert the bit specified by the low-order 3 bits of A in the memory location addressed by DE, place the result into the Zero flag, and also write back the result into the specified memory bit.			

Mnemonic	Instruction Code (Binary)				Flag J Z C H S V	Cycle	Operation
CPL (HL).A	1 1 1 0 0 0 1 1	1 1 1 1 1 0 1 1			Z * – – – –	4	ZF ← $\overline{(HL).A}$:(HL).A ← $\overline{(HL).A}$
	Invert the bit specified by the low-order 3 bits of A in the memory location addressed by HL, place the result into the Zero flag, and also write back the result into the specified memory bit.						
CPL (IX).A	1 1 1 0 0 1 0 0	1 1 1 1 1 0 1 1			Z * – – – –	4	ZF ← $\overline{(IX).A}$:(IX).A ← $\overline{(IX).A}$
	Invert the bit specified by the low-order 3 bits of A in the memory location addressed by IX, place the result into the Zero flag, and also write back the result into the specified memory bit.						
CPL (IY).A	1 1 1 0 0 1 0 1	1 1 1 1 1 0 1 1			Z * – – – –	4	ZF ← $\overline{(IY).A}$:(IY).A ← $\overline{(IY).A}$
	Invert the bit specified by the low-order 3 bits of A in the memory location addressed by IY, place the result into the Zero flag, and also write back the result into the specified memory bit.						
CPL (IX+d).A	1 1 0 1 0 1 0 0	d d d d d d d d	1 1 1 1 1 0 1 1		Z * – – – –	6	ZF ← $\overline{(IX+d).A}$:(IX+d).A ← $\overline{(IX+d).A}$
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA, place the result into the Zero flag, and also write back the result into the specified memory bit.						
CPL (IY+d).A	1 1 0 1 0 1 0 1	d d d d d d d d	1 1 1 1 1 0 1 1		Z * – – – –	6	ZF ← $\overline{(IY+d).A}$:(IY+d).A ← $\overline{(IY+d).A}$
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA, place the result into the Zero flag, and also write back the result into the specified memory bit.						
CPL (SP+d).A	1 1 0 1 0 1 1 0	d d d d d d d d	1 1 1 1 1 0 1 1		Z * – – – –	6	ZF ← $\overline{(SP+d).A}$:(SP+d).A ← $\overline{(SP+d).A}$
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA, place the result into the Zero flag, and also write back the result into the specified memory bit.						
CPL (HL+d).A	1 1 0 1 0 1 1 1	d d d d d d d d	1 1 1 1 1 0 1 1		Z * – – – –	6	ZF ← $\overline{(HL+d).A}$:(HL+d).A ← $\overline{(HL+d).A}$
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA, place the result into the Zero flag, and also write back the result into the specified memory bit.						
CPL (HL+C).A	1 1 1 0 0 1 1 1	1 1 1 1 1 0 1 1			Z * – – – –	6	ZF ← $\overline{(HL+C).A}$:(HL+C).A ← $\overline{(HL+C).A}$
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA, place the result into the Zero flag, and also write back the result into the specified memory bit.						
CPL (+SP).A	1 1 1 0 0 1 1 0	1 1 1 1 1 0 1 1			Z * – – – –	5	SP ← SP+1:ZF ← $\overline{(SP).A}$:(SP).A ← $\overline{(SP).A}$
	Invert the bit specified by the low-order 3 bits of A in the memory location at (SP+1), place the result into the Zero flag, and also write back the result into the specified memory bit.						
CPL (PC+A).A[Note]	0 1 0 0 1 1 1 1	1 1 1 1 1 0 1 1			Z * – – – –	6	ZF ← $\overline{(PC+A).A}$:(PC+A).A ← $\overline{(PC+A).A}$
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Invert the bit specified by the low-order 3 bits of A in the memory location at the EA, place the result into the Zero flag, and also write back the result into the specified memory bit.						
XOR CF,g.b	1 1 1 0 1 g g g	0 1 0 1 0 b b b			$\overline{C}$ – * – – –	2	CF ← CF ^ g.b
	Combine the contents of bit b of 8-bit register g with that of the Carry flag in an exclusive-OR operation and place the result into the Carry flag. Example: Assume A contains 0x58 and CF is set. After executing the instruction “XOR CF, A. 3”, A remains unchanged; CF is cleared to 0; and JF is set to 1.						
XOR CF,(x).b	1 1 1 0 0 0 0 0	x x x x x x x x	0 1 0 1 0 b b b		$\overline{C}$ – * – – –	4	CF ← CF ^ (x).b
	Combine the contents of bit b in the memory location directly addressed by x with that of the Carry flag in an exclusive-OR operation and place the result into the Carry flag. (0x0000 ≤ x ≤ 0x00FF)						
XOR CF,(vw).b	1 1 1 0 0 0 0 1 0 1 0 1 0 b b b	w w w w w w w w	v v v v v v v v		$\overline{C}$ – * – – –	5	CF ← CF ^ (vw).b
	Combine the contents of bit b in the memory location directly addressed by vw with that of the Carry flag in an exclusive-OR operation and place the result into the Carry flag. (0x0000 ≤ vw ≤ 0xFFFF)						
XOR CF,(DE).b	1 1 1 0 0 0 1 0	0 1 0 1 0 b b b			$\overline{C}$ – * – – –	3	CF ← CF ^ (DE).b
	Combine the contents of bit b in the memory location addressed by DE with that of the Carry flag in an exclusive-OR operation and place the result into the Carry flag.						

Mnemonic	Instruction Code (Binary)				Flag J Z C H S V	Cycle	Operation
XOR CF,(HL).b	1 1 1 0 0 0 1 1	0 1 0 1 0 b b b			$\overline{C}$ – * – – –	3	CF ← CF ^ (HL).b
	Combine the contents of bit b in the memory location addressed by HL with that of the Carry flag in an exclusive-OR operation and place the result into the Carry flag.						
XOR CF,(IX).b	1 1 1 0 0 1 0 0	0 1 0 1 0 b b b			$\overline{C}$ – * – – –	3	CF ← CF ^ (IX).b
	Combine the contents of bit b in the memory location addressed by IX with that of the Carry flag in an exclusive-OR operation and place the result into the Carry flag.						
XOR CF,(IY).b	1 1 1 0 0 1 0 1	0 1 0 1 0 b b b			$\overline{C}$ – * – – –	3	CF ← CF ^ (IY).b
	Combine the contents of bit b in the memory location addressed by IY with that of the Carry flag in an exclusive-OR operation and place the result into the Carry flag.						
XOR CF,(IX+d).b	1 1 0 1 0 1 0 0	d d d d d d d d	0 1 0 1 0 b b b		$\overline{C}$ – * – – –	5	CF ← CF ^ (IX+d).b
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IX to form an effective address (EA). Combine the contents of bit b in the memory location at the EA with that of the Carry flag in an exclusive-OR operation and place the result into the Carry flag.						
XOR CF,(IY+d).b	1 1 0 1 0 1 0 1	d d d d d d d d	0 1 0 1 0 b b b		$\overline{C}$ – * – – –	5	CF ← CF ^ (IY+d).b
	Sign-extend the 8-bit displacement d in the instruction code and add the result to IY to form an effective address (EA). Combine the contents of bit b in the memory location at the EA with that of the Carry flag in an exclusive-OR operation and place the result into the Carry flag.						
XOR CF,(SP+d).b	1 1 0 1 0 1 1 0	d d d d d d d d	0 1 0 1 0 b b b		$\overline{C}$ – * – – –	5	CF ← CF ^ (SP+d).b
	Sign-extend the 8-bit displacement d in the instruction code and add the result to SP to form an effective address (EA). Combine the contents of bit b in the memory location at the EA with that of the Carry flag in an exclusive-OR operation and place the result into the Carry flag.						
XOR CF,(HL+d).b	1 1 0 1 0 1 1 1	d d d d d d d d	0 1 0 1 0 b b b		$\overline{C}$ – * – – –	5	CF ← CF ^ (HL+d).b
	Sign-extend the 8-bit displacement d in the instruction code and add the result to HL to form an effective address (EA). Combine the contents of bit b in the memory location at the EA with that of the Carry flag in an exclusive-OR operation and place the result into the Carry flag.						
XOR CF,(HL+C).b	1 1 1 0 0 1 1 1	0 1 0 1 0 b b b			$\overline{C}$ – * – – –	5	CF ← CF ^ (HL+C).b
	Sign-extend the contents of C and add the result to HL to form an effective address (EA). Combine the contents of bit b in the memory location at the EA with that of the Carry flag in an exclusive-OR operation and place the result into the Carry flag.						
XOR CF,(+SP).b	1 1 1 0 0 1 1 0	0 1 0 1 0 b b b			$\overline{C}$ – * – – –	4	SP ← SP+1:CF ← CF ^ (SP).b
	Combine the contents of bit b in the memory location at (SP+1) with that of the Carry flag in an exclusive-OR operation and place the result into the Carry flag.						
XOR CF,(PC+A).b[Note]	0 1 0 0 1 1 1 1	0 1 0 1 0 b b b			$\overline{C}$ – * – – –	5	CF ← CF ^ (PC+A).b
	Sign-extend the contents of A and add the result to PC to form an effective address (EA). Combine the contents of bit b in the memory location at the EA with that of the Carry flag in an exclusive-OR operation and place the result into the Carry flag.						
SET CF	0 0 0 0 0 1 0 1				0 – 1 – – –	1	CF ← 1
	Set the Carry flag to 1.						
CLR CF	0 0 0 0 0 1 0 0				1 – 0 – – –	1	CF ← 0
	Clear the Carry flag to 0.						
CPL CF	0 0 0 0 0 1 1 0				* – * – – –	1	JF ← CF:CF ← $\overline{CF}$
	Place the value of the Carry flag into the Jump Status flag. Then invert the Carry flag bit. Example: Assume CF is cleared. Then, this instruction clears JF to 0 and sets CF to 1.						

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
DI[#1]	1 1 0 0 1 0 0 0 | 0 0 1 1 1 0 1 0	Z * – – – –	4	ZF ← $\overline{\text{IMF}}$:IMF ← 0
	Invert the Interrupt Master Enable flag bit and place the result into the Zero flag. Then clear the Interrupt Master Enable flag to 0 (which disables the maskable interrupts).			
EI[#1]	1 1 0 0 0 0 0 0 | 0 0 1 1 1 0 1 0	Z * – – – –	4	ZF ← $\overline{\text{IMF}}$:IMF ← 1
	Invert the Interrupt Master Enable flag bit and place the result into the Zero flag. Then set the Interrupt Master Enable flag to 1 (which enables the maskable interrupts).			

#1 Assembler instructions

Note: There are restrictions on instructions that uses the operand (PC + A). For more details, see "1.4 Addressing Mode".

2.5 Jump Instructions

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
JRS T,$+2+d	1 0 0 d d d d d	1 – – – – –	4/2(t/f)	if JF=1 then PC ← PC+d else null
	If JF is set, sign-extend the 5-bit displacement d in the instruction code (-16 to +15) and add the result to PC (2 plus the address of the current JRS instruction) to form an effective address (EA). Then, jump to the EA. (Latency = 4 clock cycles) If JF is cleared, set JF to 1 and skip to the next instruction. (Latency = 2 clock cycles). Example: Assume JF is set. Then, the instruction “JRS T, $+9” at address 0xC134 causes a jump to the address 0xC13D.			
JRS F,$+2+d	1 0 1 d d d d d	1 – – – – –	4/2(t/f)	if JF=0 then PC ← PC+d else null
	If JF is cleared, sign-extend the 5-bit displacement d in the instruction code (-16 to +15) and add the result to PC (2 plus the address of the current JRS instruction) to form an effective address (EA). Then, jump to the EA. (Latency = 4 clock cycles) If JF is set, just skip to the next instruction. (Latency = 2 clock cycles)			
JR T,$+2+d	1 1 0 1 1 1 1 0 d d d d d d d d	1 – – – – –	4/2(t/f)	if JF=1 then PC ← PC+d else null
	If JF is set, sign-extend the 8-bit displacement d in the instruction code (-128 to +127) and add the result to PC (2 plus the address of the current JR instruction) to form an effective address (EA). Then, jump to the EA. (Latency = 4 clock cycles) If JF is cleared, set JF to 1 and skip to the next instruction. (Latency = 2 clock cycles) Example: Assume JF is set. Then, the instruction “JR T, $+0xF6” at address 0xC134 causes a jump to the address 0xC12A.			
JR F,$+2+d	1 1 0 1 1 1 1 1 d d d d d d d d	1 – – – – –	4/2(t/f)	if JF=0 then PC ← PC+d else null
	If JF is cleared, sign-extend the 8-bit displacement d in the instruction code (-128 to +127) and add the result to PC to form an effective address (EA). Then, jump to the EA. (Latency = 4 clock cycles) If JF is set, just skip to the next instruction. (Latency = 2 clock cycles)			
JR EQ,$+2+d/ JR Z,$+2+d	1 1 0 1 1 0 0 0 d d d d d d d d	1 – – – – –	4/2(t/f)	if ZF=1 then PC ← PC+d else null
	If ZF is set, sign-extend the 8-bit displacement d in the instruction code (-128 to +127) and add the result to PC to form an effective address (EA). Then, jump to the EA. (Latency = 4 clock cycles) If ZF is cleared, set JF to 1 and skip to the next instruction. (Latency = 2 clock cycles)			
JR NE,$+2+d/ JR NZ,$+2+d	1 1 0 1 1 0 0 1 d d d d d d d d	1 – – – – –	4/2(t/f)	if ZF=0 then PC ← PC+d else null
	If ZF is cleared, sign-extend the 8-bit displacement d in the instruction code (-128 to +127) and add the result to PC to form an effective address (EA). Then, jump to the EA. (Latency = 4 clock cycles) If ZF is set, set JF to 1 and skip to the next instruction. (Latency = 2 clock cycles)			
JR CS,$+2+d/ JR LT,$+2+d	1 1 0 1 1 0 1 0 d d d d d d d d	1 – – – – –	4/2(t/f)	if CF=1 then PC ← PC+d else null
	If CF is set, sign-extend the 8-bit displacement d in the instruction code (-128 to +127) and add the result to PC to form an effective address (EA). Then, jump to the EA. (Latency = 4 clock cycles) If CF is cleared, set JF to 1 and skip to the next instruction. (Latency = 2 clock cycles)			
JR CC,$+2+d/ JR GE,$+2+d	1 1 0 1 1 0 1 1 d d d d d d d d	1 – – – – –	4/2(t/f)	if CF=0 then PC ← PC+d else null
	If CF is cleared, sign-extend the 8-bit displacement d in the instruction code (-128 to +127) and add the result to PC to form an effective address (EA). Then, jump to the EA. (Latency = 4 clock cycles) If CF is set, set JF to 1 and skip to the next instruction. (Latency = 2 clock cycles)			
JR LE,$+2+d	1 1 0 1 1 1 0 0 d d d d d d d d	1 – – – – –	4/2(t/f)	if (CF ∣ ZF)=1 then PC ← PC+d else null
	If CF or ZF or both are set, sign-extend the 8-bit displacement d in the instruction code (-128 to +127) and add the result to PC to form an effective address (EA). Then, jump to the EA. (Latency = 4 clock cycles) If both CF and ZF are cleared, set JF to 1 and skip to the next instruction. (Latency = 2 clock cycles)			
JR GT,$+2+d	1 1 0 1 1 1 0 1 d d d d d d d d	1 – – – – –	4/2(t/f)	if (CF ∣ ZF)=0 then PC ← PC+d else null
	If both CF and ZF are cleared, sign-extend the 8-bit displacement d in the instruction code (-128 to +127) and add the result to PC to form an effective address (EA). Then, jump to the EA. (Latency = 4 clock cycles) If CF or ZF or both are set, set JF to 1 and skip to the next instruction. (Latency = 2 clock cycles)			
JR M,$+3+d	1 1 1 0 1 0 0 0 1 1 0 1 0 0 0 0 d d d d d d d d	1 – – – – –	5/3(t/f)	if SF=1 then PC ← PC+d else null
	If SF is set, sign-extend the 8-bit displacement d in the instruction code (-128 to +128) and add the result to PC (3 plus the address of the current JR instruction) to form an effective address (EA). Then, jump to the EA. (Latency = 5 clock cycles) If SF is cleared, set JF to 1 and skip to the next instruction. (Latency = 3 clock cycles).			
JR P,$+3+d	1 1 1 0 1 0 0 0 1 1 0 1 0 0 0 1 d d d d d d d d	1 – – – – –	5/3(t/f)	if SF=0 then PC ← PC+d else null
	If SF is cleared, sign-extend the 8-bit displacement d in the instruction code (-128 to +128) and add the result to PC to form an effective address (EA). Then, jump to the EA. (Latency = 5 clock cycles) If SF is set, set JF to 1 and skip to the next instruction. (Latency = 3 clock cycles).			
JR SLT,$+3+d	1 1 1 0 1 0 0 0 1 1 0 1 0 0 1 0 d d d d d d d d	1 – – – – –	5/3(t/f)	if (SF ^ VF)=1 then PC ← PC+d else null
	Combine the Sign and Overflow flags in an exclusive-OR operation. If the result is 1, sign-extend the 8-bit displacement d in the instruction code (-128 to +128) and add the result to PC to form an effective address (EA). Then, jump to the EA. (Latency = 5 clock cycles) If the result is 0, set JF to 1 and skip to the next instruction. (Latency = 3 clock cycles).			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
JR SGE,$+3+d	1 1 1 0 1 0 0 0 | 1 1 0 1 0 0 1 1 | d d d d d d d d	1 – – – – –	5/3(t/f)	if (SF ^ VF)=0 then PC ← PC+d else null
	Combine the Sign and Overflow flags in an exclusive-OR operation. If the result is 0, sign-extend the 8-bit displacement d in the instruction code (-128 to +128) and add the result to PC to form an effective address (EA). Then, jump to the EA. (Latency = 5 clock cycles) If the result is 1, set JF to 1 and skip to the next instruction. (Latency = 3 clock cycles).			
JR SLE,$+3+d	1 1 1 0 1 0 0 0 | 1 1 0 1 0 1 0 0 | d d d d d d d d	1 – – – – –	5/3(t/f)	if ZF ∣ (SF ^ VF)=1 then PC ← PC+d else null
	Combine the Sign and Overflow flags in an exclusive-OR operation. If the result or ZF or both are 1, sign-extend the 8-bit displacement d in the instruction code (-128 to +128) and add the result to PC to form an effective address (EA). Then, jump to the EA. (Latency = 5 clock cycles) If both the result and ZF are 0, set JF to 1 and skip to the next instruction. (Latency = 3 clock cycles).			
JR SGT,$+3+d	1 1 1 0 1 0 0 0 | 1 1 0 1 0 1 0 1 | d d d d d d d d	1 – – – – –	5/3(t/f)	if ZF∣ (SF ^ VF)=0 then PC ← PC+d else null
	Combine the Sign and Overflow flags in an exclusive-OR operation. If both the result and ZF are 0, sign-extend the 8-bit displacement d in the instruction code (-128 to +128) and add the result to PC to form an effective address (EA). Then, jump to the EA. (Latency = 5 clock cycles) If the result or ZF or both are 1, set JF to 1 and skip to the next instruction. (Latency = 3 clock cycles).			
JR VS,$+3+d	1 1 1 0 1 0 0 0 | 1 1 0 1 0 1 1 0 | d d d d d d d d	1 – – – – –	5/3(t/f)	if VF=1 then PC ← PC+d else null
	If VF is set, sign-extend the 8-bit displacement d in the instruction code (-128 to +127) and add the result to PC to form an effective address (EA). Then, jump to the EA. (Latency = 5 clock cycles) If VF is cleared, set JF to 1 and skip to the next instruction. (Latency = 2 clock cycles)			
JR VC,$+3+d	1 1 1 0 1 0 0 0 | 1 1 0 1 0 1 1 1 | d d d d d d d d	1 – – – – –	5/3(t/f)	if VF=0 then PC ← PC+d else null
	If VF is cleared, sign-extend the 8-bit displacement d in the instruction code (-128 to +127) and add the result to PC to form an effective address (EA). Then, jump to the EA. (Latency = 5 clock cycles) If VF is set, set JF to 1 and skip to the next instruction. (Latency = 2 clock cycles)			
JR $+2+d	1 1 1 1 1 1 0 0 | d d d d d d d d	1 – – – – –	4	PC ← PC+d
	Sign-extend the 8-bit displacement d in the instruction code (-128 to +128) and add the result to PC (2 plus the address of the current JR instruction) to form an effective address (EA). Then, unconditionally jump to the EA. Example: The instruction “JR $+0x73” at address 0xD5A7 causes a jump to the address 0xD61C.			
JP mn	1 1 1 1 1 1 1 0 | n n n n n n n n | m m m m m m m m	– – – – – –	4	PC ← mn
	Unconditionally jump to the memory location directly addressed by 16-bit immediate mn.			
JP gg	1 1 1 0 1 g g g | 1 1 1 1 1 1 1 0	– – – – – –	3	PC ← gg
	Unconditionally jump to the memory location addressed by 16-bit register gg. Example: Assume HL contains 0xE325. Then, the instruction “JP HL” causes a jump to the address 0xE325.			
JP (x)	1 1 1 0 0 0 0 0 | x x x x x x x x | 1 1 1 1 1 1 1 0	– – – – – –	6	PC ← (x+1,x)
	Form an effective address (EA) with 2 consecutive bytes from the memory location directly addressed by x. Then, unconditionally jump to the EA. (0x0000 ≤ x ≤ 0x00FF) Example: Assume memory locations at addresses 0x0085 and 0x0086 contain 0x27 and 0xC3 respectively. Then, the instruction “JP (0x85)” causes a jump to the address 0xC327.			
JP (vw)	1 1 1 0 0 0 0 1 | w w w w w w w w | v v v v v v v v 1 1 1 1 1 1 1 0	– – – – – –	7	PC ← (vw+1,vw)
	Form an effective address (EA) with 2 consecutive bytes from the memory location directly addressed by vw. Then, unconditionally jump to the EA. (0x0000 ≤ vw ≤ 0xFFFF)			
JP (DE)	1 1 1 0 0 0 1 0 | 1 1 1 1 1 1 1 0	– – – – – –	5	PC ← (DE+1,DE)
	Form an effective address (EA) with 2 consecutive bytes from the memory location addressed by DE. Then, unconditionally jump to the EA. Example: Assume DE and the memory locations at addresses 0x0125 and 0x0126 contain 0x0125, 0x87 and 0xE5 respectively. Then, the instruction “JP (DE)” causes a jump to the address 0xE587.			
JP (HL)	1 1 1 0 0 0 1 1 | 1 1 1 1 1 1 1 0	– – – – – –	5	PC ← (HL+1,HL)
	Form an effective address (EA) with 2 consecutive bytes from the memory location addressed by HL. Then, unconditionally jump to the EA. Example: Assume HL and the memory locations at addresses 0x0125 and 0x0126 contain 0x0125, 0x87 and 0xE5 respectively. Then, the instruction “JP (HL)” causes a jump to the address 0xE587.			
JP (IX)	1 1 1 0 0 1 0 0 | 1 1 1 1 1 1 1 0	– – – – – –	5	PC ← (IX+1,IX)
	Form an effective address (EA) with 2 consecutive bytes from the memory location addressed by IX. Then, unconditionally jump to the EA. Example: Assume IX and the memory locations at addresses 0x0125 and 0x0126 contain 0x0125, 0x87 and 0xE5 respectively. Then, the instruction “JP (IX)” causes a jump to the address 0xE587.			
JP (IY)	1 1 1 0 0 1 0 1 | 1 1 1 1 1 1 1 0	– – – – – –	5	PC ← (IY+1,IY)
	Form an effective address (EA) with 2 consecutive bytes from the memory location addressed by IY. Then, unconditionally jump to the EA. Example: Assume IY and the memory locations at addresses 0x0125 and 0x0126 contain 0x0125, 0x87 and 0xE5 respectively. Then, the instruction “JP (IY)” causes a jump to the address 0xE587.			
JP (IX+d)	1 1 0 1 0 1 0 0 | d d d d d d d d | 1 1 1 1 1 1 1 0	– – – – – –	7	PC ← (IX+d+1,IX+d)
	Form an effective address (EA) with 2 consecutive bytes from the memory location addressed by sign-extending the 8-bit displacement d in the instruction code and adding the result to IX. Then, unconditionally jump to the EA.			

Mnemonic	Instruction Code (Binary)	Flag J Z C H S V	Cycle	Operation
JP (IY+d)	1 1 0 1 0 1 0 1 | d d d d d d d d | 1 1 1 1 1 1 1 0	– – – – – –	7	PC ← (IY+d+1,IY+d)
	Form an effective address (EA) with 2 consecutive bytes from the memory location addressed by sign-extending the 8-bit displacement d in the instruction code and adding the result to IY. Then, unconditionally jump to the EA.			
JP (SP+d)	1 1 0 1 0 1 1 0 | d d d d d d d d | 1 1 1 1 1 1 1 0	– – – – – –	7	PC ← (SP+d+1,SP+d)
	Form an effective address (EA) with 2 consecutive bytes from the memory location addressed by sign-extending the 8-bit displacement d in the instruction code and adding the result to SP. Then, unconditionally jump to the EA.			
JP (HL+d)	1 1 0 1 0 1 1 1 | d d d d d d d d | 1 1 1 1 1 1 1 0	– – – – – –	7	PC ← (HL+d+1,HL+d)
	Form an effective address (EA) with 2 consecutive bytes from the memory location addressed by sign-extending the 8-bit displacement d in the instruction code and adding the result to HL. Then, unconditionally jump to the EA.			
JP (HL+C)	1 1 1 0 0 1 1 1 | 1 1 1 1 1 1 1 0	– – – – – –	7	PC ← (HL+C+1,HL+C)
	Form an effective address (EA) with 2 consecutive bytes from the memory location addressed by sign-extending the contents of C and adding the result to HL. Then, unconditionally jump to the EA.			
JP (+SP)	1 1 1 0 0 1 1 0 | 1 1 1 1 1 1 1 0	– – – – – –	6	SP ← SP+1:PC ← (SP+1,SP)
	Form an effective address (EA) with 2 consecutive bytes from the memory location addressed by (SP+1). Then, unconditionally jump to the EA.			
JP (PC+A)[Note]	0 1 0 0 1 1 1 1 | 1 1 1 1 1 1 1 0	– – – – – –	7	PC ← (PC+A+1,PC+A)
	Form an effective address (EA) with 2 consecutive bytes from the memory location addressed by sign-extending the contents of A and adding the result to PC (2 plus the address of the current JP instruction). Then, unconditionally jump to the EA. This instruction is useful for multi-way branches.			

Note: There are restrictions on instructions that uses the operand (PC + A). For more details, see "1.4 Addressing Mode".