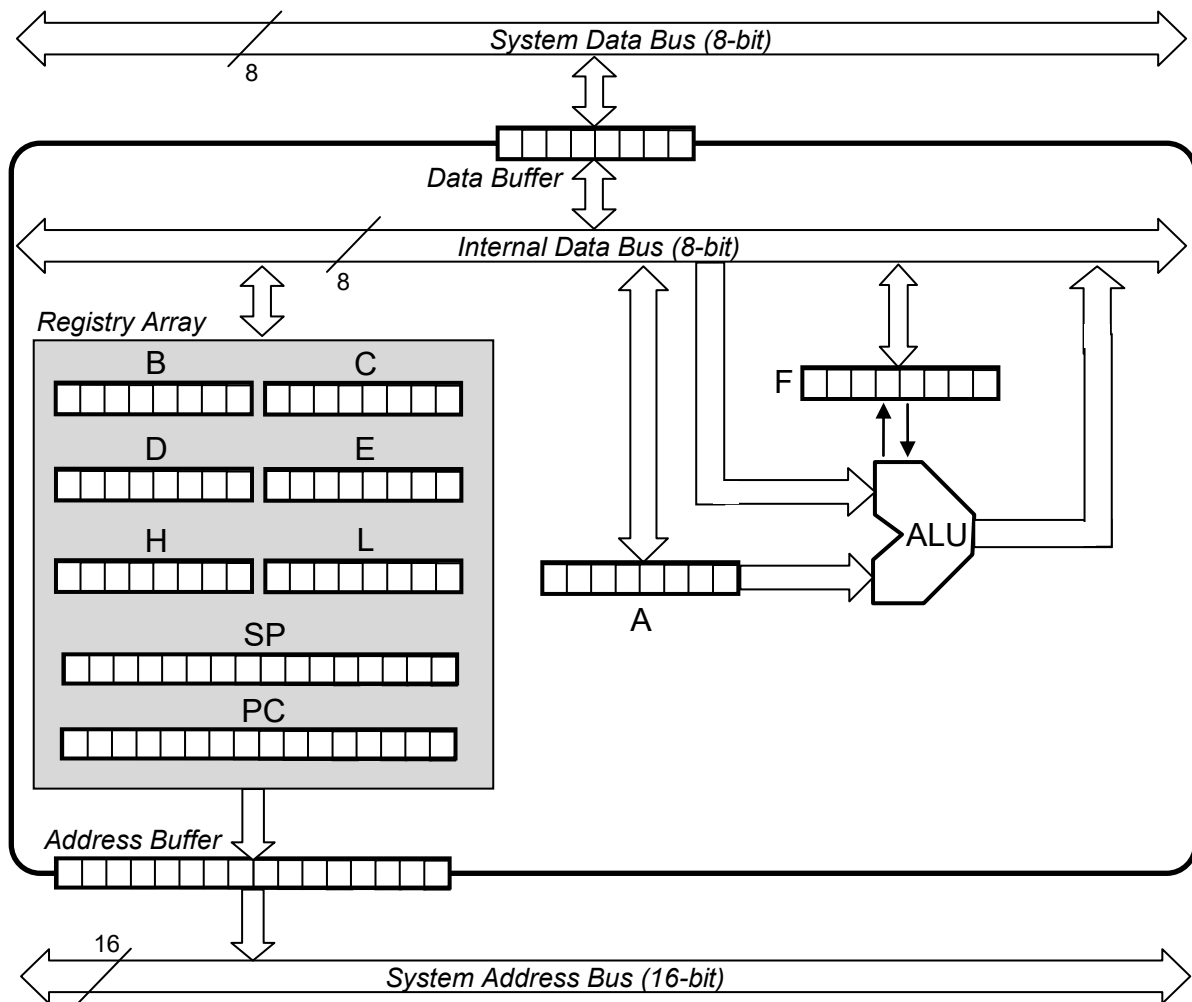


## Intel 8080 CPU block diagram



### Internal register addressing:

8-bit register (R)	3-bit address
A	111
B	000
C	001
D	010
E	011
H	100
L	101

16-bit register pair (P)	2-bit address
BC (B)	00
DE (D)	01
HL (H)	10
SP	11

### Flag register (F) structure:

S	Z	0	AC	0	P	1	CY
---	---	---	----	---	---	---	----

Where:

- **CY** – *carry flag*, set to 1 if the result of arithmetical or logical operation exceeds the 8-bit A register (Accumulator) or operation needs to borrow one bit – in other words it's carry/borrow from/to the bit 7;
- **P** – *parity flag*, set to 1 if the result of arithmetical operation has even number of bits equal to 1, set to 0 if this number is odd (in Z80 CPU this flag is also the *overflow* indicator for TC arithmetical operations);
- **AC** – *auxiliary carry flag*, set to 1 if there was carry from bit 3 to 4 in the result of arithmetical operation (useful in programming operations with packed BCD numbers);
- **Z** – *zero flag*, set to 1 if the result of arithmetical operation is zero;
- **S** – *sign flag*, equal to the most significant (oldest) bit of the result of arithmetical operation.

## Intel 8080 instruction set architecture

### Instruction code formats:

- One byte instructions:

i <sub>7</sub>	i <sub>6</sub>	i <sub>5</sub>	i <sub>4</sub>	i <sub>3</sub>	i <sub>2</sub>	i <sub>1</sub>	i <sub>0</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

All bits used to encode instruction, no operands.

i <sub>7</sub>	i <sub>6</sub>	i <sub>5</sub>	i <sub>4</sub>	i <sub>3</sub>	<b>r</b>	<b>r</b>	<b>r</b>
----------------	----------------	----------------	----------------	----------------	----------	----------	----------

One operand in 8-bit internal register A to L.

i <sub>7</sub>	i <sub>6</sub>	<b>r</b>	<b>r</b>	<b>r</b>	i <sub>2</sub>	i <sub>1</sub>	i <sub>0</sub>
----------------	----------------	----------	----------	----------	----------------	----------------	----------------

One operand in 8-bit internal register A to L.

i <sub>7</sub>	i <sub>6</sub>	<b>r1</b>	<b>r1</b>	<b>r1</b>	<b>r2</b>	<b>r2</b>	<b>r2</b>
----------------	----------------	-----------	-----------	-----------	-----------	-----------	-----------

Two operands in two 8-bit internal registers A to L.

i <sub>7</sub>	i <sub>6</sub>	<b>p</b>	<b>p</b>	i <sub>3</sub>	i <sub>2</sub>	i <sub>1</sub>	i <sub>0</sub>
----------------	----------------	----------	----------	----------------	----------------	----------------	----------------

One operand in 16-bit register pair BC, DE, HL or in SP.

- Two byte instructions:

i <sub>7</sub>	i <sub>6</sub>	i <sub>5</sub>	i <sub>4</sub>	i <sub>3</sub>	i <sub>2</sub>	i <sub>1</sub>	i <sub>0</sub>
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>

All bits of the first byte used to encode instruction, the second byte is the *immediate* operand (argument).

i <sub>7</sub>	i <sub>6</sub>	<b>r</b>	<b>r</b>	<b>r</b>	i <sub>2</sub>	i <sub>1</sub>	i <sub>0</sub>
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>

First operand in 8-bit internal register A to L, the second operand is *immediate*.

i <sub>7</sub>	i <sub>6</sub>	i <sub>5</sub>	i <sub>4</sub>	i <sub>3</sub>	i <sub>2</sub>	i <sub>1</sub>	i <sub>0</sub>
p <sub>7</sub>	p <sub>6</sub>	p <sub>5</sub>	p <sub>4</sub>	p <sub>3</sub>	p <sub>2</sub>	p <sub>1</sub>	p <sub>0</sub>

All bits of the first byte used to encode instruction, the second byte is the 8-bit address of I/O port.

- Three byte instructions with immediate memory addressing:

i <sub>7</sub>	i <sub>6</sub>	<b>p</b>	<b>p</b>	i <sub>3</sub>	i <sub>2</sub>	i <sub>1</sub>	i <sub>0</sub>
l <sub>7</sub>	l <sub>6</sub>	l <sub>5</sub>	l <sub>4</sub>	l <sub>3</sub>	l <sub>2</sub>	l <sub>1</sub>	l <sub>0</sub>
h <sub>7</sub>	h <sub>6</sub>	h <sub>5</sub>	h <sub>4</sub>	h <sub>3</sub>	h <sub>2</sub>	h <sub>1</sub>	h <sub>0</sub>

First operand in 16-bit register pair BC, DE, HL or in SP, the second byte is *lower* part of 16-bit second operand, the third byte is *higher* part of the 16-bit second operand.

- Three byte instructions with direct memory addressing:

i <sub>7</sub>	i <sub>6</sub>	i <sub>5</sub>	i <sub>4</sub>	i <sub>3</sub>	i <sub>2</sub>	i <sub>1</sub>	i <sub>0</sub>
l <sub>7</sub>	l <sub>6</sub>	l <sub>5</sub>	l <sub>4</sub>	l <sub>3</sub>	l <sub>2</sub>	l <sub>1</sub>	l <sub>0</sub>
h <sub>7</sub>	h <sub>6</sub>	h <sub>5</sub>	h <sub>4</sub>	h <sub>3</sub>	h <sub>2</sub>	h <sub>1</sub>	h <sub>0</sub>

All bits of the first byte used to encode instruction, the second byte is *lower* part of 16-bit address in memory, the third byte is *higher* part of address of the operand.

i <sub>7</sub>	i <sub>6</sub>	i <sub>5</sub>	i <sub>4</sub>	i <sub>3</sub>	i <sub>2</sub>	i <sub>1</sub>	i <sub>0</sub>
l <sub>7</sub>	l <sub>6</sub>	l <sub>5</sub>	l <sub>4</sub>	l <sub>3</sub>	l <sub>2</sub>	l <sub>1</sub>	l <sub>0</sub>
h <sub>7</sub>	h <sub>6</sub>	h <sub>5</sub>	h <sub>4</sub>	h <sub>3</sub>	h <sub>2</sub>	h <sub>1</sub>	h <sub>0</sub>

All bits of the first byte used to encode instruction, the second byte is *lower* part of 16-bit *address*, the third byte is *higher* part of *address* of jump or call.

Internal registers A, B, C, D, E, H, L (**rrr**), pairs of registers BC, DE, HL and SP register (**pp**) are addressed according to rules shown on first page.

Intel's processors always store data longer than one byte in the *lower-to-higher* byte order – *little endian* convention.

Mnemonics used for instructions are copyrighted, so other processors which instruction lists are compatible with 8080 (Z80 for example) have different names and mnemonics for the same instructions.

## Instruction list:

### Data transfer instructions

**MOV R1, R2** (Move register)

0	1	<i>r1</i>	<i>r1</i>	<i>r1</i>	<i>r2</i>	<i>r2</i>	<i>r2</i>
---	---	-----------	-----------	-----------	-----------	-----------	-----------

$R1 \leftarrow R2$  data from R2 is copied to R1

**MOV R, M** (Move from memory, address in HL)

0	1	<i>r</i>	<i>r</i>	<i>r</i>	1	1	0
---	---	----------	----------	----------	---	---	---

$R \leftarrow [HL]$  data from memory (address in HL) copied to R

**MOV M, R** (Move to memory, address in HL)

0	1	1	1	0	<i>r</i>	<i>r</i>	<i>r</i>
---	---	---	---	---	----------	----------	----------

$[HL] \leftarrow R$  data from R copied to memory (address in HL)

**MVI R, data8** (Move to register immediate)

0	0	<i>r</i>	<i>r</i>	<i>r</i>	1	1	0
$d_7$	$d_6$	$d_5$	$d_4$	$d_3$	$d_2$	$d_1$	$d_0$

$R \leftarrow \text{data8}$  1 byte (next to instruction) copied to R

**MVI M, data8** (Move to memory immediate)

0	0	1	1	0	1	1	0
$d_7$	$d_6$	$d_5$	$d_4$	$d_3$	$d_2$	$d_1$	$d_0$

$[HL] \leftarrow \text{data8}$  1 byte copied to memory (address in HL)

**LXI P, data16** (Load register pair immediate)

0	0	<i>p</i>	<i>p</i>	0	0	0	1
$l_7$	$l_6$	$l_5$	$l_4$	$l_3$	$l_2$	$l_1$	$l_0$
$h_7$	$h_6$	$h_5$	$h_4$	$h_3$	$h_2$	$h_1$	$h_0$

$P \leftarrow \text{data16}$  2 bytes copied to register pair  
"lower" register – 2<sup>nd</sup> byte  
"higher" register – 3<sup>rd</sup> byte

**LDA addr16** (Load accumulator direct)

0	0	1	1	1	0	1	0
$l_7$	$l_6$	$l_5$	$l_4$	$l_3$	$l_2$	$l_1$	$l_0$
$h_7$	$h_6$	$h_5$	$h_4$	$h_3$	$h_2$	$h_1$	$h_0$

$A \leftarrow [\text{addr16}]$  1 byte copied to register A from memory  
lower byte of address – 2<sup>nd</sup> byte  
higher byte of address – 3<sup>rd</sup> byte

**STA addr16** (Store accumulator direct)

0	0	1	1	0	0	1	0
$l_7$	$l_6$	$l_5$	$l_4$	$l_3$	$l_2$	$l_1$	$l_0$
$h_7$	$h_6$	$h_5$	$h_4$	$h_3$	$h_2$	$h_1$	$h_0$

$[\text{addr16}] \leftarrow A$  1 byte copied to memory from register A  
lower byte of address – 2<sup>nd</sup> byte  
higher byte of address – 3<sup>rd</sup> byte

**LHLD addr16** (Load H and L direct)

0	0	1	1	0	0	1	0
$l_7$	$l_6$	$l_5$	$l_4$	$l_3$	$l_2$	$l_1$	$l_0$
$h_7$	$h_6$	$h_5$	$h_4$	$h_3$	$h_2$	$h_1$	$h_0$

$L \leftarrow [\text{addr16}]$  2 bytes copied from memory to HL  
 $H \leftarrow [\text{addr16} + 1]$  lower byte of address – 2<sup>nd</sup> byte  
higher byte of address – 3<sup>rd</sup> byte

**SHLD addr16** (Store H and L direct)

0	0	1	0	0	0	1	0
$l_7$	$l_6$	$l_5$	$l_4$	$l_3$	$l_2$	$l_1$	$l_0$
$h_7$	$h_6$	$h_5$	$h_4$	$h_3$	$h_2$	$h_1$	$h_0$

$[\text{addr16}] \leftarrow L$  2 bytes copied from HL to memory  
 $[\text{addr16} + 1] \leftarrow H$  lower byte of address – 2<sup>nd</sup> byte  
higher byte of address – 3<sup>rd</sup> byte

**LDAX P** (Load accumulator indirect, address in BC or DE)

0	0	<i>p</i>	<i>p</i>	1	0	1	0
---	---	----------	----------	---	---	---	---

$A \leftarrow [P]$  data from memory (address in P) copied to A

**STAX P** (Store accumulator indirect, address in BC or DE)

0	0	<i>p</i>	<i>p</i>	0	0	1	0
---	---	----------	----------	---	---	---	---

$[P] \leftarrow A$  data from A copied to memory (address in P)

**XCHG** (Exchange H and L with D and E)

1	1	1	0	1	0	1	1
---	---	---	---	---	---	---	---

$H \leftrightarrow D$   $L \leftrightarrow E$  data in HL and DE is switched

## Arithmetical instructions

**ADD R** (Add register)

1	0	0	0	0	<i>r</i>	<i>r</i>	<i>r</i>
---	---	---	---	---	----------	----------	----------

$A \leftarrow A + R$  data from R is added to data in A  
flags affected: Z, S, P, CY, AC

**ADD M** (Add memory, address in HL)

1	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

$A \leftarrow A + [HL]$  data from memory is added to A  
flags affected: Z, S, P, CY, AC

**ADI data8** (Add immediate)

1	1	0	0	0	1	1	0
$d_7$	$d_6$	$d_5$	$d_4$	$d_3$	$d_2$	$d_1$	$d_0$

$A \leftarrow A + \text{data8}$  one byte (next to instruction) added to A  
flags affected: Z, S, P, CY, AC

**ADC R** (Add register with carry)

1	0	0	0	1	<i>r</i>	<i>r</i>	<i>r</i>
---	---	---	---	---	----------	----------	----------

$A \leftarrow A + R + CY$  data from R and CY flag are added to A  
flags affected: Z, S, P, CY, AC

**ADC M** (Add memory with carry, address in HL)

1	0	0	0	1	1	1	0
---	---	---	---	---	---	---	---

$A \leftarrow A + [HL] + CY$  data from memory and CY added to A  
flags affected: Z, S, P, CY, AC

**ACI data8** (Add immediate with carry)

1	1	0	0	1	1	1	0
$d_7$	$d_6$	$d_5$	$d_4$	$d_3$	$d_2$	$d_1$	$d_0$

$A \leftarrow A + \text{data8} + CY$  one byte and CY added to A  
flags affected: Z, S, P, CY, AC

**SUB R** (Subtract register)

1	0	0	1	0	<i>r</i>	<i>r</i>	<i>r</i>
---	---	---	---	---	----------	----------	----------

$A \leftarrow A - R$  data from R is subtracted from A  
flags affected: Z, S, P, CY, AC

**SUB M** (Subtract memory, address in HL)

1	0	0	1	0	1	1	0
---	---	---	---	---	---	---	---

$A \leftarrow A - [HL]$  data from memory is subtracted from A  
flags affected: Z, S, P, CY, AC

**SUI data8** (Subtract immediate)

1	1	0	1	0	1	1	0
$d_7$	$d_6$	$d_5$	$d_4$	$d_3$	$d_2$	$d_1$	$d_0$

$A \leftarrow A - \text{data8}$  one byte subtracted from A  
flags affected: Z, S, P, CY, AC

**SBB R** (Subtract register with borrow)

1	0	0	1	1	<i>r</i>	<i>r</i>	<i>r</i>
---	---	---	---	---	----------	----------	----------

$A \leftarrow A - R - CY$  *R and CY are subtracted from A*  
*flags affected: Z, S, P, CY, AC*

**SBB M** (Subtract memory with borrow)

1	0	0	1	1	1	1	0
---	---	---	---	---	---	---	---

$A \leftarrow A - [HL] - CY$  *data from memory and CY subtracted*  
*from A, flags affected: Z, S, P, CY, AC*

**SBI data8** (Subtract immediate with borrow)

1	1	0	1	1	1	1	0
<i>d</i> <sub>7</sub>	<i>d</i> <sub>6</sub>	<i>d</i> <sub>5</sub>	<i>d</i> <sub>4</sub>	<i>d</i> <sub>3</sub>	<i>d</i> <sub>2</sub>	<i>d</i> <sub>1</sub>	<i>d</i> <sub>0</sub>

$A \leftarrow A - \text{data8} - CY$  *one byte and CY subtracted from A*  
*flags affected: Z, S, P, CY, AC*

**INR R** (Increment register)

0	0	<i>r</i>	<i>r</i>	<i>r</i>	1	0	0
---	---	----------	----------	----------	---	---	---

$R \leftarrow R + 1$  *Data in R is incremented by 1*  
*flags affected: Z, S, P, AC*

**INR M** (Increment memory, address in HL)

0	0	1	1	0	1	0	0
---	---	---	---	---	---	---	---

$[HL] \leftarrow [HL] + 1$  *Data in memory is incremented by 1*  
*flags affected: Z, S, P, AC*

**DCR R** (Decrement register)

0	0	<i>r</i>	<i>r</i>	<i>r</i>	1	0	1
---	---	----------	----------	----------	---	---	---

$R \leftarrow R - 1$  *Data in R is decremented by 1*  
*flags affected: Z, S, P, AC*

**DCR M** (Decrement memory, address in HL)

0	0	1	1	0	1	0	1
---	---	---	---	---	---	---	---

$[HL] \leftarrow [HL] - 1$  *Data in memory is decremented by 1*  
*flags affected: Z, S, P, AC*

**INX P** (Increment register pair)

0	0	<i>p</i>	<i>p</i>	0	0	1	1
---	---	----------	----------	---	---	---	---

$P \leftarrow P + 1$  *Data in register pair P is incremented by 1*  
*flags affected: none*

**DCX P** (Decrement register pair)

0	0	<i>p</i>	<i>p</i>	1	0	1	1
---	---	----------	----------	---	---	---	---

$P \leftarrow P - 1$  *Data in register pair P is decremented by 1*  
*flags affected: none*

**DAD P** (Decrement register pair)

0	0	<i>p</i>	<i>p</i>	1	0	0	1
---	---	----------	----------	---	---	---	---

$HL \leftarrow HL + P$  *Data in register pair P is added to HL*  
*flags affected: CY (from higher byte)*

**DAA** (Decimal adjust Accumulator)

0	0	1	0	0	1	1	1
---	---	---	---	---	---	---	---

$A \leftarrow \text{adjust}_{\text{BCD}}(A)$  *Data in A is adjusted as packed BCD:*  
*if  $(a_3 \dots a_0) > 9$  or  $AC=1$  then  $(a_3 \dots a_0) \leftarrow (a_3 \dots a_0) + 6$*   
*if  $(a_7 \dots a_4) > 9$  or  $CY=1$  then  $(a_7 \dots a_4) \leftarrow (a_7 \dots a_4) + 6$*   
*explanation: 6 is the 4-bit U2 code of -10*  
*flags affected: Z, S, P, CY, AC*

## Logical instructions

### ANA R (AND with register)

1	0	1	0	0	<i>r</i>	<i>r</i>	<i>r</i>
---	---	---	---	---	----------	----------	----------

$A \leftarrow A \wedge R$  Bits in A logically multiplied with bits from R  
flags affected: Z, S, P, CY=0, AC=0

### ANA M (AND with memory – address in HL)

1	0	1	0	0	1	1	0
---	---	---	---	---	---	---	---

$A \leftarrow A \wedge [HL]$  Bits in A logically multiplied with bits from memory, flags affected: Z, S, P, CY=0, AC=0

### ANI data8 (AND immediate)

1	1	1	0	0	1	1	0
$d_7$	$d_6$	$d_5$	$d_4$	$d_3$	$d_2$	$d_1$	$d_0$

$A \leftarrow A \wedge \text{data8}$  Bits in A logically multiplied with bits from 2<sup>nd</sup> byte of instruction, flags affected: Z, S, P, CY=0, AC=0

### XRA R (XOR with register)

1	0	1	0	1	<i>r</i>	<i>r</i>	<i>r</i>
---	---	---	---	---	----------	----------	----------

$A \leftarrow A \otimes R$  Bits in A logically xor-ed with bits from R  
flags affected: Z, S, P, CY=0, AC=0

### XRA M (XOR with memory – address in HL)

1	0	1	0	1	1	1	0
---	---	---	---	---	---	---	---

$A \leftarrow A \otimes [HL]$  Bits in A logically xor-ed with bits from memory, flags affected: Z, S, P, CY=0, AC=0

### XRI data8 (XOR immediate)

1	1	1	0	1	1	1	0
$d_7$	$d_6$	$d_5$	$d_4$	$d_3$	$d_2$	$d_1$	$d_0$

$A \leftarrow A \otimes \text{data8}$  Bits in A logically xor-ed with bits from 2<sup>nd</sup> byte of instruction, flags affected: Z, S, P, CY=0, AC=0

### ORA R (OR with register)

1	0	1	1	0	<i>r</i>	<i>r</i>	<i>r</i>
---	---	---	---	---	----------	----------	----------

$A \leftarrow A \vee R$  Bits in A logically added with bits from R  
flags affected: Z, S, P, CY=0, AC=0

### ORA M (OR with memory – address in HL)

1	0	1	1	0	1	1	0
---	---	---	---	---	---	---	---

$A \leftarrow A \vee [HL]$  Bits in A logically added with bits from memory, flags affected: Z, S, P, CY=0, AC=0

### ORI data8 (OR immediate)

1	1	1	1	0	1	1	0
$d_7$	$d_6$	$d_5$	$d_4$	$d_3$	$d_2$	$d_1$	$d_0$

$A \leftarrow A \vee \text{data8}$  Bits in A logically added with bits from 2<sup>nd</sup> byte of instruction, flags affected: Z, S, P, CY=0, AC=0

### CMP R (Compare with register)

1	0	1	1	1	<i>r</i>	<i>r</i>	<i>r</i>
---	---	---	---	---	----------	----------	----------

$A - R$  Data in R is subtracted from data in A, no result is stored, only flags are affected: Z, S, P, CY, AC

### CMP M (Compare with memory – address in HL)

1	0	1	1	1	1	1	0
---	---	---	---	---	---	---	---

$A - [HL]$  Data in memory is subtracted from data in A, no result is stored, only flags are affected: Z, S, P, CY, AC

**CPI data8** (Compare immediate)

1	1	1	1	1	1	1	0
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>

A - data8 Data in 2<sup>nd</sup> byte of instruction is subtracted from data in A, only flags are affected: Z, S, P, CY, AC

Comment:

Interpretation of “compare” operations is possible by checking Z and CY flags after execution:

if Z=1 then values compared are equal,  
 else (if Z=0)  
   if CY=0 then A > compared value,  
   else (if CY=1) A < compared value.

**RLC** (Rotate left / rotate logically left)

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

$A_{i+1} \leftarrow A_i$ ,  $A_0 \leftarrow A_7$ ,  $CY \leftarrow A_7$  Bits in A shifted left, oldest bit copied to youngest bit and CY, flags affected: CY

**RRC** (Rotate right / rotate logically right)

0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---

$A_i \leftarrow A_{i+1}$ ,  $A_7 \leftarrow A_0$ ,  $CY \leftarrow A_0$  Bits in A shifted right, youngest bit copied to oldest bit and CY, flags affected: CY

**RAL** (Rotate left through carry / rotate arithmetically left)

0	0	0	1	0	1	1	1
---	---	---	---	---	---	---	---

$A_{i+1} \leftarrow A_i$ ,  $A_0 \leftarrow CY$ ,  $CY \leftarrow A_7$  Bits in A shifted left, CY copied to youngest bit, oldest bit copied to CY, flags affected: CY

**RAR** (Rotate right through carry / rotate arithmetically right)

0	0	0	1	0	1	1	1
---	---	---	---	---	---	---	---

$A_i \leftarrow A_{i+1}$ ,  $A_7 \leftarrow CY$ ,  $CY \leftarrow A_0$  Bits in A shifted right, CY copied to oldest bit, youngest bit copied to CY, flags affected: CY

**CMA** (Complement Accumulator)

0	0	1	0	1	1	1	1
---	---	---	---	---	---	---	---

$A \leftarrow \neg A$  Bitwise negation of A (one's complement) flags affected: none

**CMC** (Complement carry)

0	0	1	1	1	1	1	1
---	---	---	---	---	---	---	---

$CY \leftarrow \neg CY$  Negation (inversion) of CY flag flags affected: CY

**Branch instructions**Comment:

These instructions are passing control to the new address in program (not just to the address of next instruction). There are two basic types of branches:

- *unconditional* – just go to new address,
- *conditional* – check if particular condition (detected by status of one of the flags in F register) occurs and jump if so, continue with next instruction if not.

The conditions are encoded (inside the codes of instructions) according to this table:

Condition	Mnemonic (CND)	Code (ccc)
Not zero (Z = 0)	NZ	000
Zero (Z = 1)	Z	001
No carry (CY = 0)	NC	010
Carry (CY = 1)	C	011
Parity odd (P = 0)	PO	100
Parity even (P = 1)	PE	101
Plus (S = 0)	P	110
Minus (S = 1)	M	111

#### **JMP addr16** (Jump)

1	1	0	0	0	0	1	1
$l_7$	$l_6$	$l_5$	$l_4$	$l_3$	$l_2$	$l_1$	$l_0$
$h_7$	$h_6$	$h_5$	$h_4$	$h_3$	$h_2$	$h_1$	$h_0$

$PC \leftarrow [addr16]$  *Unconditional jump to direct address*  
*lower byte of address – 2<sup>nd</sup> byte*  
*higher byte of address – 3<sup>rd</sup> byte*

#### **JCND addr16** (Conditional jump)

1	1	<b>c</b>	<b>c</b>	<b>c</b>	0	1	0
$l_7$	$l_6$	$l_5$	$l_4$	$l_3$	$l_2$	$l_1$	$l_0$
$h_7$	$h_6$	$h_5$	$h_4$	$h_3$	$h_2$	$h_1$	$h_0$

if (CND) then  $PC \leftarrow [addr16]$   
*lower byte of address – 2<sup>nd</sup> byte*  
*higher byte of address – 3<sup>rd</sup> byte*

#### **CALL addr16** (Call procedure)

1	1	0	0	1	1	0	1
$l_7$	$l_6$	$l_5$	$l_4$	$l_3$	$l_2$	$l_1$	$l_0$
$h_7$	$h_6$	$h_5$	$h_4$	$h_3$	$h_2$	$h_1$	$h_0$

$[SP-1] \leftarrow PC_H$ ,  $[SP-2] \leftarrow PC_L$ ,  $SP \leftarrow SP-2$ ,  
 $PC \leftarrow [addr16]$  *lower byte of address – 2<sup>nd</sup> byte*  
*higher byte of address – 3<sup>rd</sup> byte*

#### **CCND addr16** (Conditional call)

1	1	<b>c</b>	<b>c</b>	<b>c</b>	0	1	0
$l_7$	$l_6$	$l_5$	$l_4$	$l_3$	$l_2$	$l_1$	$l_0$
$h_7$	$h_6$	$h_5$	$h_4$	$h_3$	$h_2$	$h_1$	$h_0$

if (CND) then  $[SP-1] \leftarrow PC_H$ ,  $[SP-2] \leftarrow PC_L$ ,  
 $SP \leftarrow SP-2$ , *lower byte of address – 2<sup>nd</sup> byte*  
 $PC \leftarrow [addr16]$  *higher byte of address – 3<sup>rd</sup> byte*

#### **RET** (Return from procedure)

1	1	0	0	1	0	0	1
---	---	---	---	---	---	---	---

$PC_L \leftarrow [SP]$ ,  $PC_H \leftarrow [SP+1]$ ,  $SP \leftarrow SP+2$

#### **RCND** (Conditional return from procedure)

1	1	<b>c</b>	<b>c</b>	<b>c</b>	0	0	0
---	---	----------	----------	----------	---	---	---

if (CND) then  $PC_L \leftarrow [SP]$ ,  $PC_H \leftarrow [SP+1]$ ,  $SP \leftarrow SP+2$

#### **RST N** (Restart procedure / interrupt routine No. N)

1	1	<b>n</b>	<b>n</b>	<b>n</b>	1	1	1
---	---	----------	----------	----------	---	---	---

$[SP-1] \leftarrow PC_H$ ,  $[SP-2] \leftarrow PC_L$ ,  $SP \leftarrow SP-2$ ,  $PC \leftarrow N \times 8$

#### **PCHL** (Move HL to PC)

1	1	1	0	1	0	0	1
---	---	---	---	---	---	---	---

$PC_H \leftarrow H$ ,  $PC_L \leftarrow L$



## Stack manipulations

**PUSH P** (Push register pair B, D or H on stack)

1	1	<i>p</i>	<i>p</i>	0	1	0	1	$[SP-1] \leftarrow P_H, [SP-2] \leftarrow P_L, SP \leftarrow SP-2$
---	---	----------	----------	---	---	---	---	--

**PUSH PSW** (Push Processor Status Word on stack)

1	1	1	1	0	1	0	1	$[SP-1] \leftarrow A, [SP-2] \leftarrow F, SP \leftarrow SP-2$
---	---	---	---	---	---	---	---	--

**POP P** (Pop register pair B, D or H from stack)

1	1	<i>p</i>	<i>p</i>	0	0	0	1	$P_L \leftarrow [SP], P_H \leftarrow [SP+1], SP \leftarrow SP+2$
---	---	----------	----------	---	---	---	---	--

**POP PSW** (Pop Processor Status Word from stack)

1	1	1	1	0	0	0	1	$F \leftarrow [SP], A \leftarrow [SP+1], SP \leftarrow SP+2$ flags affected: Z, S, P, CY, AC
---	---	---	---	---	---	---	---	---

**XTHL** (Exchange stack top with HL)

1	1	1	0	0	0	1	1	$L \leftrightarrow [SP], H \leftrightarrow [SP+1]$
---	---	---	---	---	---	---	---	--

**SPHL** (Move HL to SP)

1	1	1	1	1	0	0	1	$SP \leftarrow HL$
---	---	---	---	---	---	---	---	--------------------

## Input / Output instructions

**IN *adr8*** (Input from port)

1	1	0	1	1	0	1	1	$A \leftarrow \text{Port}[\text{adr8}]$	One byte of data from port stored in A Notice: port address is 8-bit long
<i>p</i> <sub>7</sub>	<i>p</i> <sub>6</sub>	<i>p</i> <sub>5</sub>	<i>p</i> <sub>4</sub>	<i>p</i> <sub>3</sub>	<i>p</i> <sub>2</sub>	<i>p</i> <sub>1</sub>	<i>p</i> <sub>0</sub>		

**OUT *adr8*** (Output to port)

1	1	0	1	0	0	1	1	$\text{Port}[\text{adr8}] \leftarrow A$	One byte of data from A stored in port Notice: port address is 8-bit long
<i>p</i> <sub>7</sub>	<i>p</i> <sub>6</sub>	<i>p</i> <sub>5</sub>	<i>p</i> <sub>4</sub>	<i>p</i> <sub>3</sub>	<i>p</i> <sub>2</sub>	<i>p</i> <sub>1</sub>	<i>p</i> <sub>0</sub>		

## Other instructions

**EI** (Enable interrupt)

1	1	1	1	1	0	1	1	<i>INT</i> input (hardware interrupt signal) is enabled
---	---	---	---	---	---	---	---	---

**DI** (Disable interrupt)

1	1	1	1	0	0	1	1	<i>INT</i> input (hardware interrupt signal) is disabled (blocked)
---	---	---	---	---	---	---	---	--

**HLT** (Halt)

0	1	1	1	0	1	1	0	Processor is stopped until hardware interrupt occurs
---	---	---	---	---	---	---	---	--

**NOP** (No operation)

0	0	0	0	0	0	0	0	Processor doesn't perform any operation
---	---	---	---	---	---	---	---	---