Laboratory work No 2

The instruction set and programming of the 8-bit microprocessor

1. The aim of the work

To get acquainted with the instruction set of the 8-bit microprocessor Intel® 8085; to acquire skills in programming by an assembler.

2. General knowledge

2.1. Characteristics of the microprocessor

The microprocessor Intel® 8085 was the follow-on microprocessor to the very successful Intel® 8080 microprocessor. The Intel® 8085 microprocessor got its name because it was Intel's first 5 volt microprocessor. The internal structure of the microprocessor Intel® 8085 is similar to the microprocessor Intel® 8080: analogous memory address space, length of data words, instruction set (supplemented only by two instructions) and addressing modes. The microprocessor Intel® 8085 is 100 % software compatible with the microprocessor Intel® 8080.

The main parameters of the microprocessor Intel® 8085:

- Clock speed – up to 6 MHz;
- Number of transistors – 6500;
- Manufacturing process – 3 μm;
- Addressable memory – up to 64 KB;
- Address bus – 16 bits;
- Data bus – 8 bits.

Intel® 8085 microprocessor was used in the various control devices, in personal computers, and even in NASA's space ships and satellites. Now due to its rather simple architecture and the instruction set the microprocessor Intel® 8085 is used for training purposes.

2.2. Unit of registers

The microprocessor Intel® 8085 has seven 8-bit registers for data storage. Register **A** called an accumulator and it is intended for information exchange with peripheral devices. When performing arithmetical, logical and displacement operations, one of operands is stored in a accumulator. The operation result is stored in it. Other six registers – **B, C, D, E, H** and **L** make up the so called general purpose register (GPR) unit and they can be used for storage of data and addresses. If 16-bit binary numbers have to be stored, these single 8-bit registers are joined into pairs **BC, DE, HL**. In the assembler language these pairs are identified as **B, D** and **H**.

The 16-bit stack pointer **SP** is intended for addressing the stack memory cells. A lower part **SPL** and a higher part **SPH** of the 8-bit register are accessible to the programmer separately.

The 16-bit program counter **PC** is intended for storing the instruction address. By selecting a routine instruction from the memory, the content of the counter is increased by a unity, i.e. another byte of the instruction is addressed (if there are no conditional or unconditional transitions in the program).

The 8-bit flags register **F** is intended for the fixation of certain flags of the operation result. The flag is fixed in a corresponding bit writing 1 or 0:

7	6	5	4	3	2	1	0
S	**Z**	0	**AC**	0	**P**	1	**CY**

- bit **S** – sign flag: 1 – if operation result is negative, 0 – if operation result is positive;

- bit **Z** – zero flag: 1 – if operation result is equal to zero, 0 – if operation result is not equal to zero;
- bit **AC** – auxiliary carry flag: 1 – when carry occurs from the third bit of binary number, 0 – there is no carry
- bit **P** – parity flag: 1 – if there is even number of units in the operation result binary code, 0 – if there is odd number of units in the operation result binary code;
- bit **CY** – carry flag: 1 – if the operation result does not fit in 8 bits (carry occurs from the highest bit or it was borrowed by performing the operation of subtraction).

These values of bits are incorporated in 16 and 32-bit flag registers of later generation Intel® architecture microprocessors as a lower byte of these registers.

2.3. Arithmetic logic unit

The 8-bit arithmetic logic unit (ALU) can perform arithmetical operations (addition and subtraction, with carry and without it), four logical and four displacement operations. By performing arithmetical and logical operations, one of operands is placed in the accumulator. The operation result remains in the accumulator. Only the cyclic displacement of the accumulator is performed.

The possibility of performing arithmetical operations with decimal numbers is also provided. For the decimal number storage the register bits are divided into two groups in fours and in each group one decimal number coded by 8421 code (BCD – Binary Coded Decimal) is stored.

ALU is directly related to the control unit consisting of the instruction register, where the first instruction byte is sent, and of the control signal formation device. In the latter there is the control memory where micro programs of various operations are stored. The control memory, thus the operation micro programs as well, are inaccessible to the user.

2.4. Data and instruction formats

Data are the processed binary information and results of its processing. Data are most frequently stored in the random access memory and in the microprocessor registers. The binary 8-bit number is called a byte (D8). Addresses are of 16 bits, thus, like 16-bit data, they are stored in parts: a lower byte – D16L or ADRL and a higher byte – D16H or ADRH.

The instruction formats can be of one, two and three bytes. The values of bytes are presented in Table 1. In one byte instructions the operation code byte (OCB) is the only byte.

Table 1. Formats of instructions

Format	1 byte	2 byte	3 byte
Of one byte	OCB	–	–
Of two bytes	OCB	D8 or port number (PORT)	–
Of three bytes	OCB	D16L or ADRL	D16H or ADRH

2.5. Operand addressing modes

To perform operation in the instruction, alongside with the operation type operands should also be indicated. The applied modes of the operand addressing will be discussed.

Register addressing. Operands during this addressing are in general purpose registers, the addresses of which are indicated in the only OCB. As there are few registers, for their addressing three binary bits are sufficient, as shown in Table 2.

Table 2. Addresses of registers

Register	Address_2	Register	Address_2
B	000	**H**	100
C	001	**L**	101
D	010	**M**	110
E	011	**A**	111

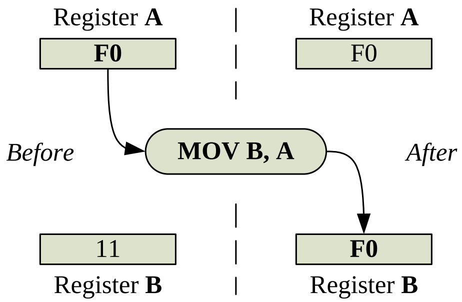
where **M** is the memory cell, the address of which is stored in the register pair **HL**.

For example, the format of instruction **MOV R1**, **R2** is the following:

7	6	5	4	3	2	1	0
0	1		**R1**			**R2**	

By entering specific register addresses instead of **R1** and **R2**, we obtain a binary code of the instruction.

Let's say that **R1** is a register **B**, and **R2** – a register **A**. By entering corresponding addresses 000 and 111 to the instruction format, we get a binary code of the instruction: $01000111_2 = 47_{16}$ – **MOV B**, **A**. **B** ← **A**:



This means that the content of register **A** is moved to the register **B**.

So as we see the instructions with register addressing is of one byte length:

1 byte

C7	C6	C5	C4	C3	C2	C1	C0	– operation code byte (OCB)

Operand addresses are indicated directly in the operation code byte.

Direct addressing. In case of this addressing, instruction formats of two or three bytes are applied. Here the port number (address) is the second instruction byte, and the operand address is the second and third bytes:

1 baitas

C7	C6	C5	C4	C3	C2	C1	C0	– operation code byte (OCB)

2 baitas

A7	A6	A5	A4	A3	A2	A1	A0	– port number (address) (PORT)

1 baitas

C7	C6	C5	C4	C3	C2	C1	C0	– operation code byte (OCB)

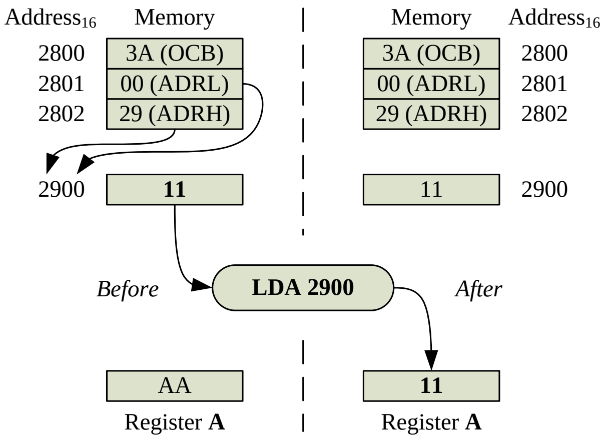
2 baitas

A7	A6	A5	A4	A3	A2	A1	A0	– lower byte of the address (ADRL)

3 baitas

A7	A6	A5	A4	A3	A2	A1	A0	– higher byte of the address (ADRH)

Addresses are of 16 bits, thus, like 16-bit data, they are stored in parts: ADRL and ADRH – lower and higher byte of address where is an operand. For example, instruction **LDA ADR**. **A** ← **M[ADR]**. Where **M[ADR]** is the content of the memory location, whose address is specified in second and third byte of the instruction.



This means that the content of the memory location, whose address is specified in second and third byte of the instruction, is moved to register **A**.

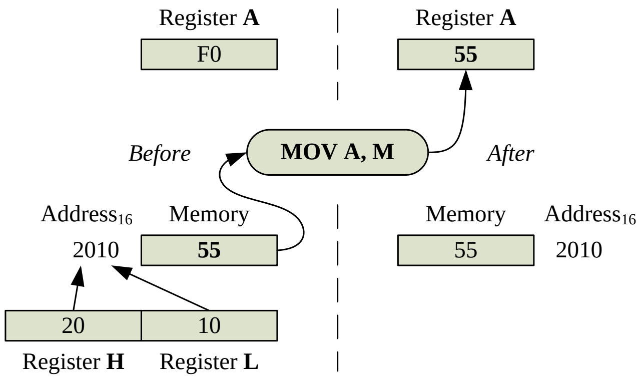
Register indirect addressing. This mode of addressing differs from the register mode in which not the operand itself is stored in registers, but the memory address of this operand. Since the address is of 16 bits, a pair of registers is needed for its storage. For example, the instruction **MOV R, M** means that the data from the memory location, whose address is indicated by the register pair **HL** (**M** in this case means a register pair **HL**) will be sent to register **R**.

The format of instruction **MOV R**, **M** is the following:

7	6	5	4	3	2	1	0
0	1		**R**			**M**	

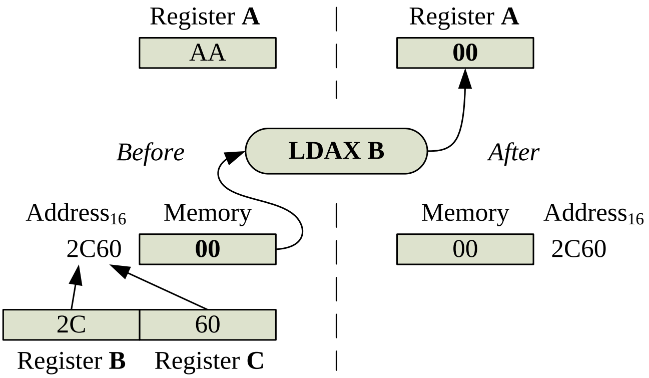
By entering specific addresses instead of **R** and **M**, we obtain a binary code of the instruction.

Let's say that **R** is a register **A**. By entering corresponding addresses 111 and 110 to the instruction format, we get a binary code of the instruction: $01111110_2 = 7E_{16}$ – **MOV A**, **M**. **A** ← **M[HL]**:



This means that the content of the memory location, whose address is in registers **H** and **L**, is moved to register **A**.

Besides for this purpose register pairs **BC** or **DE** can also be applied. For example, instruction **LDAX B**. **A** ← **M[BC]** or **STAX B**. **M[BC]** ← **A**. Where **M[BC]** is the content of the memory location, whose address is in register pair **BC**.



This means that the content of the memory location, whose address is in the register pair **BC**, is moved to register **A**.

Stack instructions also uses the register indirect addressing. The stack is a portion of read/write memory set aside by the user for the purpose of storing information temporarily. When the information is written on the stack, the operation is called **PUSH**. When the information is read from stack, the operation is called **POP**.

The microprocessor stores the information, much like stacking plates. Using this analogy of stacking plates it is easy to illustrate the stack operation (Fig. 1).

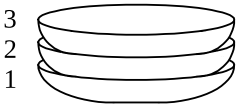


Fig. 1. Stacked plates

Fig. 1 shows the stacked plates. Here, we realize that if it is desired to take out the first stacked plate we will have to remove all plates above the first plate in the reverse order. This means that to remove first plate we will have to remove the third plate, then the second plate and finally the first plate. This means that, the first information pushed on to the stack is the last information popped off from the stack. This type of operation is known as a first in, last out (FILO). This stack is implemented with the help of special memory pointer register. The special pointer register is called the stack pointer **SP**. During **PUSH** and **POP** operation, stack pointer **SP** gives the address of memory where the information is to be stored or to be read. The stack pointer's **SP** contents are automatically manipulated to point to stack top. The memory location currently pointed by stack pointer **SP** is called top of stack.

Working principle of **PUSH** instruction is shown in Fig. 2.

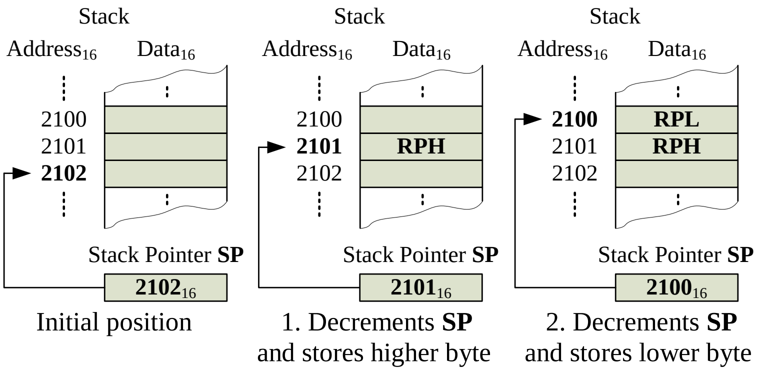
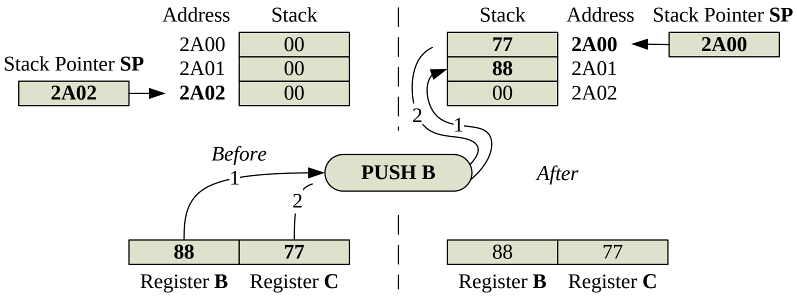


Fig. 2. Steps involved in **PUSH** instruction

The content of the high order register **RPH** of register pair **RP** is moved to the memory location whose address is one less than the content of register **SP**. The content of the low order register **RPL** of register pair **RP** is moved to the memory location whose address is two less than the content of register **SP**. The content of register **SP** is decremented by 2. The **RP** is 16-bit register pair such as **BC**, **DE**, **HL** and **PSW** (contents of register **A** and flags register **F**). Only higher order register is to be specified within the instruction. Note: register pair **RP** = **SP** may not be specified.

For example, the instruction **PUSH B** works as follows:



Working principle of **POP** instruction is shown in Fig. 3.

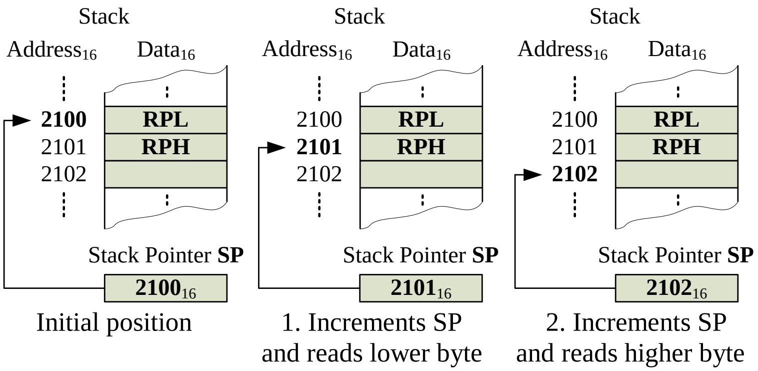
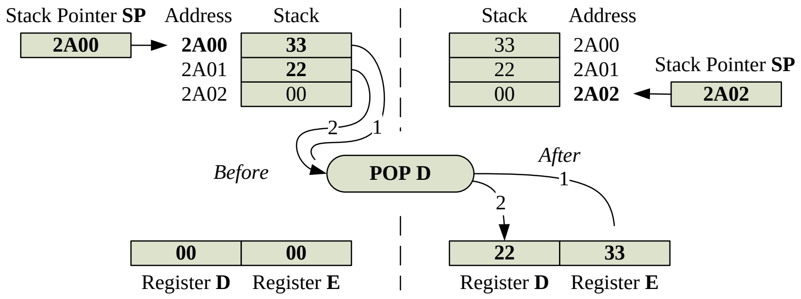


Fig. 3. Steps involved in **POP** instruction

The content of the memory location, whose address is specified by the content of register **SP**, is moved to the low order register **RPL** of register pair **RP**. The content of the memory location, whose address is one more than the content of register **SP**, is moved to the high order register **RPH** of register pair **RP**. The content of register **SP** is incremented by 2. Note: Register pair **RP** = **SP** may not be specified. The **RP** is 16-bit register pair such as **BC**, **DE**, **HL** and **PSW** (contents of register **A** and flags register **F**). Only higher order register is to be specified within the instruction. Note: register pair **RP** = **SP** may not be specified.

For example, the instruction **POP D** works as follows:



So as we see the instructions with register indirect addressing is of one byte length:

1 byte

C7	C6	C5	C4	C3	C2	C1	C0	– operation code byte (OCB)

Immediate addressing or direct operand. These are data of one or two bytes connected to operation code byte. The instruction format is of two or three bytes and is similar to the instruction format with direct addressing. The only difference is that in case of the immediate addressing after operation code byte not address bytes, but data bytes follow:

1 baitas

C7	C6	C5	C4	C3	C2	C1	C0	– operation code byte (OCB)

2 baitas

D7	D6	D5	D4	D3	D2	D1	D0	– 8-bit data (D8)

1 baitas

C7	C6	C5	C4	C3	C2	C1	C0	– operation code byte (OCB)

2 baitas

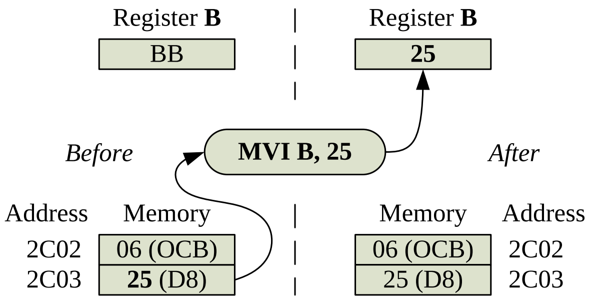
D7	D6	D5	D4	D3	D2	D1	D0	– lower byte of 16-bit data (D16L)

3 baitas

D7	D6	D5	D4	D3	D2	D1	D0	– higher byte of 16-bit data (D16H)

16-bit data, they are stored in parts: a lower byte – D16L or ADRL and a higher byte – D16H or ADRH.

For example, instruction **MVI R**, **D8**. **R** ← **D8**. Where **D8** is a data byte following the operation code byte (second byte of instruction).



This means that the content of second byte of the instruction (immediate data or direct operand) is moved to register **B**.

3. Task

1. To switch on the microprocessor training system M85-01 and prepare for work.

2. To program and execute given instructions for each mode of the operand addressing:

a) Register addressing mode

ADD B; **SUB L**; **MOV D, H**; **INR E**; **DCR C**; **RLC**; **RRC**.

b) Direct addressing mode

LDA 2200; **STA 203A**; **LHLD 2015**; **SHLD 200B**.

c) Register indirect addressing mode

ADD M; **SUB M**; **MOV M, C**; **LDAX D**; **STAX B**; **PUSH D**; **POP B**.

d) Immediate addressing mode or direct operand

ADI 0C; **SUI 01**; **MVI L, 05**; **MVI M, 55**; **LXI H, 2010**; **LXI SP, 2012**.

Write the results of experiments in Table 3.

Table 3. Operand addressing modes

Address$_{16}$	Instruction code		Initial data	Results	Notes
	hexadecimal	mnemonic			
a) Register addressing mode					
2000 2001	80 EF	**ADD B** **RST 5**	**A** = 05 **B** = 06	**A** = 0B	**A** ← **A** + **B**
...	...	...	...	...	...
b) Direct addressing mode					
...	...	...	...	...	...

Note: descriptions and hexadecimal codes of all instructions are presented in appendix.

3. To create and execute a program which would sum up two 8-bit operands from different memory locations and would write the result into the third memory location. For the operand addressing to select different modes of operand addressing. When writing the program and its execution results refer to Table 3.

4. To create and execute a program which would calculate a given expression and would write the result *w* into the freely chosen memory location. 8-bit operands *x*, *y* and *z* are in different freely chosen memory locations. For the operand addressing to select

different modes of operand addressing. When writing the program and its execution results refer to Table 3.

$$w = 2x + (y - z) / 4.$$

5. To create and execute a program which would sum up two 8-bit operands in the stack and would write back the result to the stack. When writing the program and its execution results refer to Table 3.

6. To create and execute a program which would compare two 8-bit operands, would subtract a smaller operand from the larger one and to write difference in the stack at the selected address. When writing the program and its execution results refer to Table 3.

4. Contents of the report

1. The aim of the work.
2. Results of instructions execution with different operand addressing modes (Table 3).
3. Programs of tasks 3, 4, 5 and 6, and results of their execution (Table 3).
5. Conclusions.

5. Test questions

1. What are the main characteristics of the microprocessor Intel® 8085?
2. What are general purpose registers of the microprocessor Intel® 8085?
3. What are bit values of the microprocessor Intel® 8085 flags register?
4. What is the purpose of the program counter?
5. What is the stack and what is its purpose?
6. What operations can the arithmetic logic unit of the microprocessor Intel® 8085 perform?

7. Enumerate and explain the instruction formats of the microprocessor Intel® 8085.

8. What operand addressing is called register addressing?

9. What operand addressing is called register indirect addressing?

10. What operand addressing is called direct addressing?

11. What operand addressing is called immediate addressing?

References

GRAŽULEVIČIUS, G. 2008. *Mikroprocesorinė technika*: mokomoji knyga. I dalis. Vilnius: Technika, 224 p. ISBN 978-9955-28-280-8.

ROUTT, W. A. 2007. *Microprocessor Architecture, Programming, and Systems Featuring the 8085*. USA, New York: Thomson Delmar Learning. 271 p. ISBN 1-4180-3241-7.

COFFRON, J. W. *Microprocessor Programming, Troubleshooting and Interfacing*: The Z80, 8080 and 8085. USA, New Jersey: Pearson Prentice Hall; 1st Edition. 1997, October 28. 528 p. ISBN 0-13-581976-8.

UFFENBECK, J. *Microcomputers and Microprocessors*: The 8080, 8085, and Z-80 Programming, Interfacing and Troubleshooting. USA, New Jersey: Pearson Prentice Hall; 3rd Edition. 1999, June 18. 729 p. ISBN 0-13-209198-4.

Appendix

The instruction set of Intel® 8080 / 8085 microprocessors

One byte transfers

MOV R1, R2.	R1 ← R2.
MOV R, M.	R ← M[HL].
MOV M, R.	M[HL] ← R.
MVI R, D8.	R ← D8.
MVI M, D8.	M[HL] ← D8.
STAX RP.	M[RP] ← A.
LDAX RP.	A ← M[RP].
STA ADR.	M[ADR] ← A.
LDA ADR.	A ← M[ADR].

Input and output instructions

IN PORT.	A ← PORT.
OUT PORT.	PORT ← A.

Two bytes transfers

LXI RP, D16.	RPL ← D16L, RPH ← D16H.
SHLD ADR.	M[ADR] ← L, M[ADR+1] ← H.
LHLD ADR.	L ← M[ADR], H ← M[ADR+1].
PUSH RP.	M[SP – 1] ← RPH, M[SP – 2] ← RPL, SP ← SP – 2.
PUSH PSW.	M[SP – 1] ← A, M[SP – 2] ← F, SP ← SP – 2.
POP RP.	RPL ← M[SP], RPH ← M[SP + 1], SP ← SP + 2.
POP PSW.*	F ← M[SP], A ← M[SP + 1], SP ← SP + 2.
SPHL.	SP ← HL.

Exchange of bytes

XCHG.	H ↔ D, L ↔ E.
XTHL.	L ↔ M[SP], H ↔ M[SP + 1].

Arithmetic and logic instructions with one operand
8-bit instructions

CMC.**	CY ← ¬CY.	INR R.***	R ← R + 1.	Decimal adjust	
STC.**	CY ← 1.	DCR R.***	R ← R – 1.		
CMA.	A ← ¬A.	INR M.***	M[HL] ← M[HL] + 1.	DAA.*	If $A_{3–0}$ > 9 or AC = 1,
		DCR M.***	M[HL] ← M[HL] – 1.		then $A_{3–0}$ ← $A_{3–0}$ + 6;
		INX RP.	RP ← RP + 1.		if $A_{7–4}$ > 9 or CY = 1,
		DCX RP.	RP ← RP – 1.		then $A_{7–4}$ ← $A_{7–4}$ + 6.

Arithmetic and logic instructions with two operands
8-bit instructions

ADD R.*	A ← A + R.	ADI D8.*	A ← A + D8.	ADD M.*	A ← A + M[HL].
ADC R.*	A ← A + R + CY.	ACI D8.*	A ← A + D8 + CY.	ADC M.*	A ← A + M[HL] + CY.
SUB R.*	A ← A – R.	SUI D8.*	A ← A – D8.	SUB M.*	A ← A – M[HL].
SBB R.*	A ← A – (R + CY).	SBI D8.*	A ← A – (D8 + CY).	SBB M.*	A ← A – (M[HL] + CY).
ANA R.*	A ← A ∧ R.	ANI D8.*	A ← A ∧ D8.	ANA M.*	A ← A ∧ M[HL].
ORA R.*	A ← A ∨ R.	ORI D8.*	A ← A ∨ D8.	ORA M.*	A ← A ∨ M[HL].
XRA R.*	A ← A ∀R.	XRI D8.*	A ← A ∀D8.	XRA M.*	A ← A ∀M[HL].

CMP R.*	A – R.
CMP M.*	A – M[HL].
CPI D8.*	A – D8.

16-bit instructions

DAD RP.**	HL ← HL + RP.

Accumulator content rotation instructions

RLC.**	A_{n+1} ← A_n, n = 0–6, A_0 ← A_7, CY ← A_7. Rotate to left.
RRC.**	A_n ← A_{n+1}, n = 0–6, A_7 ← A_0, CY ← A_0. Rotate to right.
RAL.**	A_{n+1} ← A_n, n = 0–6, A_0 ← CY, CY ← A_7. Rotate left through carry flag CY.
RAR.**	A_n ← A_{n+1}, n = 0–6, A_7 ← CY, CY ← A_0. Rotate right through carry flag CY.

Branch instructions

PCHL.	PC ← HL.
JMP ADR.	PC ← M[ADR].
Jcc ADR.	PC ← M[ADR].

Call to subroutine and return from subroutine instructions

CALL ADR.	M[SP] ← PC + 3, SP ← SP – 2, PC ← ADR.
Ccc ADR.	M[SP] ← PC + 3, SP ← SP – 2, PC ← ADR.
RST N.	PC ← 8 × N (N = 0, 1, …, 7). 8 × N = ADR.
	ADR = 0_{16}, 8_{16}, 10_{16}, 18_{16}, 20_{16}, 28_{16}, 30_{16}, 38_{16}.
RET.	PC ← M[SP], SP ← SP + 2.
Rcc.	PC ← M[SP], SP ← SP + 2.

Microprocessor control instructions

EI	Enable interrupts.	
DI	Disable interrupts.	
HLT	Halt.	
NOP	PC ← PC + 1.	No operation.

Format of flags register F

D7	D6	D5	D4	D3	D2	D1	D0
S	Z	0	AC	0	P	1	CY

Notes:
***** – instruction affects all flags.
****** – instruction affects only flag **CY**.

******* – instruction affects all flags except **CY**.
R, **R1**, **R2** – the contents of registers **A**, **B**, **C**, **D**, **E**, **H** or **L** (8-bits);
M – memory location, whose address is in register pair **HL**;
M[HL] – content of the memory location, whose address is in register pair **HL** (8-bits);
D8 – 8-bit immediate data or direct operand (second byte of the instruction);
D16 – 16-bit immediate data or direct operand (second and third byte of the instruction);
D16L and **D16H** – lower and higher byte of 16-bit immediate data or direct operand;
ADR – 16-bit address of memory location (second and third byte of the instruction);
M[ADR] – content of the memory location, whose address is specified in second and third byte of the instruction (8-bits);
RP – register pair **BC**, **DE**, **HL** or content of stack pointer **SP** (16-bits);
RPL and **RPH** – low and high order register of register pair;
M[RP] – content of the memory location, whose address is in register pair **BC** or **DE** (8-bits);
M[SP] – stack;
SP – 16-bit stack pointer;
PC – 16-bit program counter;
PSW – 16-bit processor status word (contents of register **A** and flags register **F**);
PORT – 8-bit number (address) of input or output port (second byte of the instruction);
N – interrupt maintenance subroutine number;
n – bit number (bits are numbered from right to left from 0 to 7);
cc – branch condition (must be replaced by **NZ**, **Z**, **NC**, **C**, **PO**, **PE**, **P** or **M**);
CY – carry bit of flags register **F**;
P – parity bit of flags register **F**;
AC – auxiliary carry bit of flags register **F**;
Z – zero bit of flags register **F**;
S – sign bit of flags register **F**;
$\wedge$ – logical AND;
$\vee$ – logical inclusive OR;
$\forall$ – logical exclusive OR;
$\neg$ – complement;
$\leftarrow$ – transfer;
$\leftrightarrow$ – exchange.

The instructions hexadecimal codes of Intel® 8080 / 8085 microprocessors

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**	
0	NOP	LXI B,D16	STAX B	INX B	INR B	DCR B	MVI B,D8	RLC	–	DAD B	LDAX B	DCX B	INR C	DCR C	MVI C,D8	RRC	**0**
1	–	LXI D,D16	STAX D	INX D	INR D	DCR D	MVI D,D8	RAL	–	DAD D	LDAX D	DCX D	INR E	DCR E	MVI E,D8	RAR	**1**
2	RIM[1]	LXI H,D16	SHLD ADR	INX H	INR H	DCR H	MVI H,D8	DAA	–	DAD H	LHLD ADR	DCX H	INR L	DCR L	MVI L,D8	CMA	**2**
3	SIM[1]	LXI SP, D16	STA ADR	INX SP	INR M	DCR M	MVI M,D8	STC	–	DAD SP	LDA ADR	DCX SP	INR A	DCR A	MVI A,D8	CMC	**3**
4	MOV B,B	MOV B,C	MOV B,D	MOV B,E	MOV B,H	MOV B,L	MOV B,M	MOV B,A	MOV C,B	MOV C,C	MOV C,D	MOV C,E	MOV C,H	MOV C,L	MOV C,M	MOV C,A	**4**
5	MOV D,B	MOV D,C	MOV D,D	MOV D,E	MOV D,H	MOV D,L	MOV D,M	MOV D,A	MOV E,B	MOV E,C	MOV E,D	MOV E,E	MOV E,H	MOV E,L	MOV E,M	MOV E,A	**5**
6	MOV H,B	MOV H,C	MOV H,D	MOV H,E	MOV H,H	MOV H,L	MOV H,M	MOV H,A	MOV L,B	MOV L,C	MOV L,D	MOV L,E	MOV L,H	MOV L,L	MOV L,M	MOV L,A	**6**
7	MOV M,B	MOV M,C	MOV M,D	MOV M,E	MOV M,H	MOV M,L	HLT	MOV M,A	MOV A,B	MOV A,C	MOV A,D	MOV A,E	MOV A,H	MOV A,L	MOV A,M	MOV A,A	**7**
8	ADD B	ADD C	ADD D	ADD E	ADD H	ADD L	ADD M	ADD A	ADC B	ADC C	ADC D	ADC E	ADC H	ADC L	ADC M	ADC A	**8**
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB M	SUB A	SBB B	SBB C	SBB D	SBB E	SBB H	SBB L	SBB M	SBB A	**9**
A	ANA B	ANA C	ANA D	ANA E	ANA H	ANA L	ANA M	ANA A	XRA B	XRA C	XRA D	XRA E	XRA H	XRA L	XRA M	XRA A	**A**
B	ORA B	ORA C	ORA D	ORA E	ORA H	ORA L	ORA M	ORA A	CMP B	CMP C	CMP D	CMP E	CMP H	CMP L	CMP M	CMP A	**B**
C	RNZ	POP B	JNZ ADR	JMP ADR	CNZ ADR	PUSH B	ADI D8	RST 0	RZ	RET	JZ ADR	–	CZ ADR	CALL ADR	ACI D8	RST 1	**C**
D	RNC	POP D	JNC ADR	OUT PORT	CNC ADR	PUSH D	SUI D8	RST 2	RC	–	JC ADR	IN PORT	CC ADR	–	SBI D8	RST 3	**D**
E	RPO	POP H	JPO ADR	XTHL	CPO ADR	PUSH H	ANI D8	RST 4	RPE	PCHL	JPE ADR	XCHG	CPE ADR	–	XRI D8	RST 5	**E**
F	RP	POP PSW	JP ADR	DI	CP ADR	PUSH PSW	ORI D8	RST 6	RM	SPHL	JM ADR	EI	CM ADR	–	CPI D8	RST 7	**F**
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**	

[1] Instruction is only supported on the Intel® 8085 microprocessor.