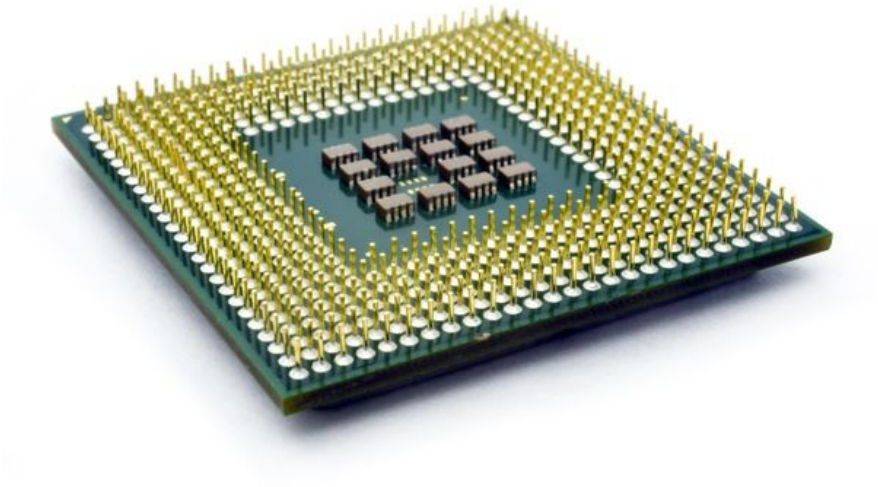


8085 Expermints

Practice and Experiments
Master version

Dr. Mustafa M. Shiple

Spring 2019

Contents

List of Figures

List of Tables

Experiment: 1

Installing 8085 IDE

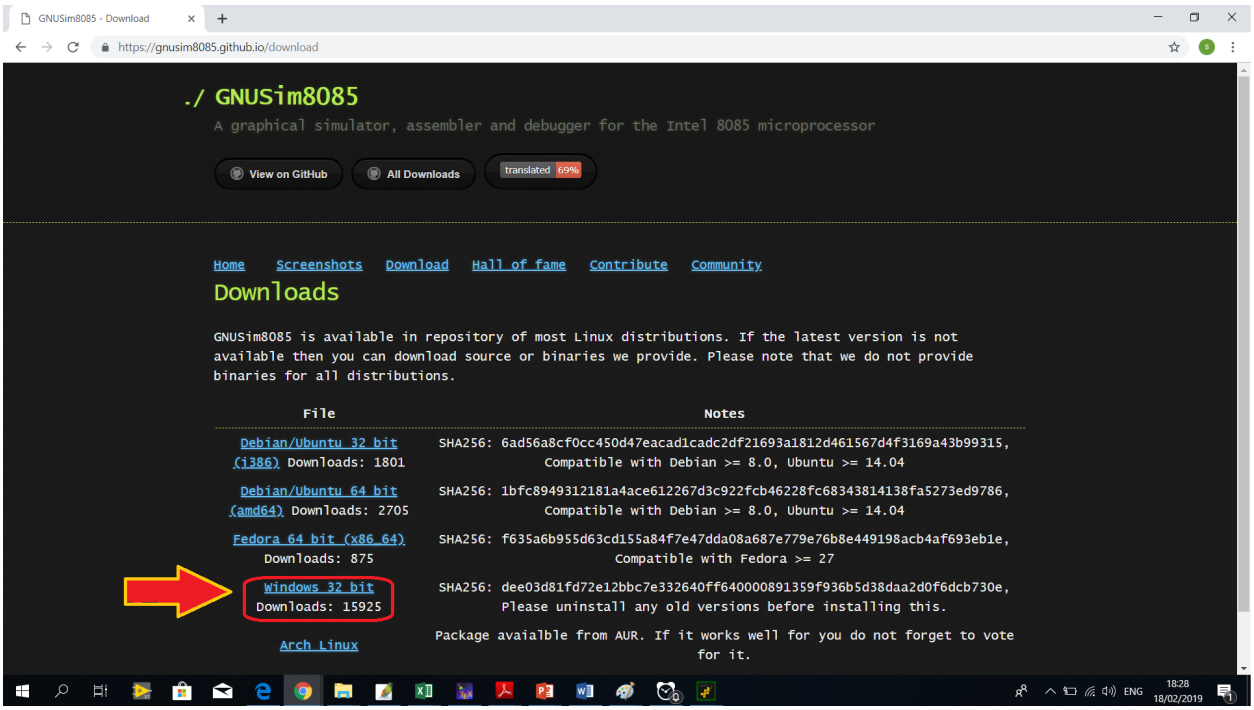
1.1 Experiment objectives:

- Learn how to install with GNUSIM8085 IDE.
- Practice assembly Design Flow.
- Debug a simple codes.
- Monitor the Registers and memory.

1.2 Experiment procedures:

1.2.1 Download GNUSIM8085 IDE

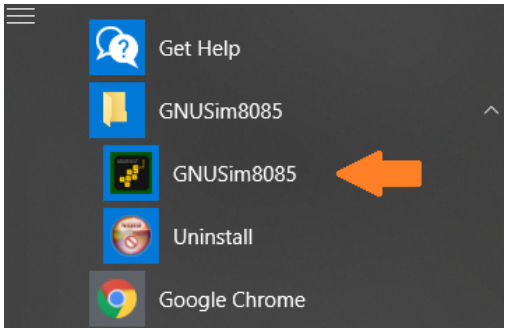
1. Open the webpage (https://gnusim8085.github.io/download) or click next link gnusim8085.



2. Select win 32 bit choice.

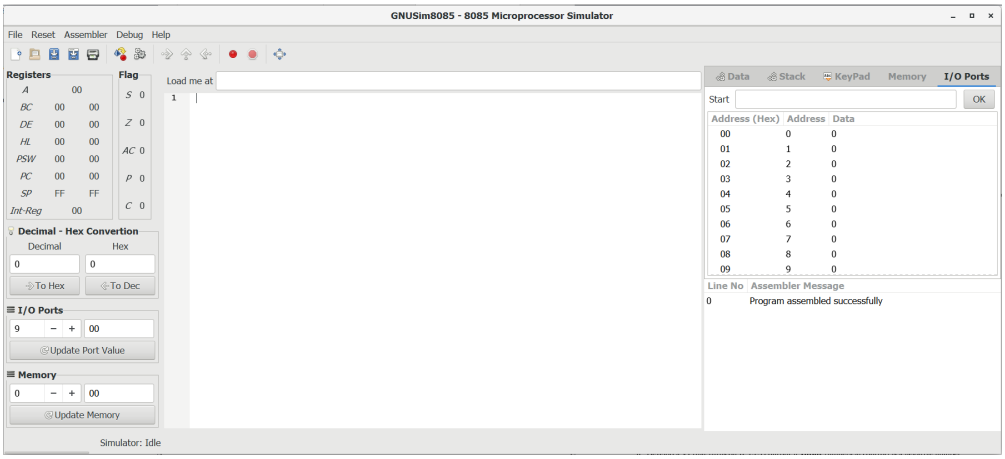
3. Install and follow instructions.

4. Check IDE in Windows program list.



1.2.2 Assembly Programming

5. Open GNUSim8085 .



6. Copy the next code and paste it in the IDE.

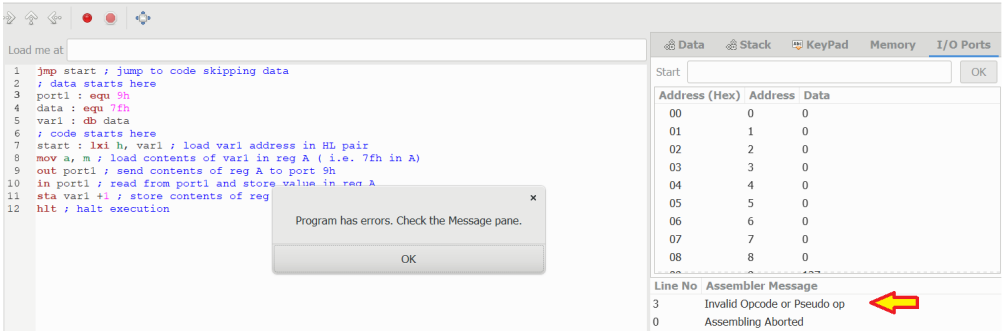
```
jmp start ; jump to code skipping data
; data starts here
  port1: equ 9h
  data: equ 7fh
  var1: db data
; code starts here

start:  lxi h, var1 ; load var1 address in HL pair
    mov a, m ; load contents of var1 in reg A ( i.e. 7fh in A)
    out port1 ; send contents of reg A to port 9h
    in port1 ; read from port1 and store value in reg A
    sta var1+1 ; store contents of reg A in memory address var+1
    hlt ; halt execution
```

7. Save the file  , name it "lab1.asm".

8. Compile the file  , this step to check the syntax and any typo errors.

1.2.3 Correcting the errors

9. An error window is displayed. Note: on the right pane, the first error is defined and its line is mentioned too.



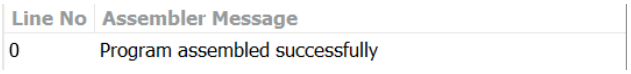
10. Delete space between "port1" and ":". (Hint: port1 is a label the correct syntax is port1:).

11. Compile the file  again, the color of port1 should be changed to pink. Moreover, the next error pointing to line number 4.

12. Repeat step number [10] for labels {data, var1,and start}.

13. Last error, delete the space between "var1" and "+" in line 11.

14. After compilation, the right pane should be as follows:



1.2.4 Program Memory

15. Check what will be written in program memory and the assigned physical addresses. from pull-down menus select (Assembler¿¿ Show listing)

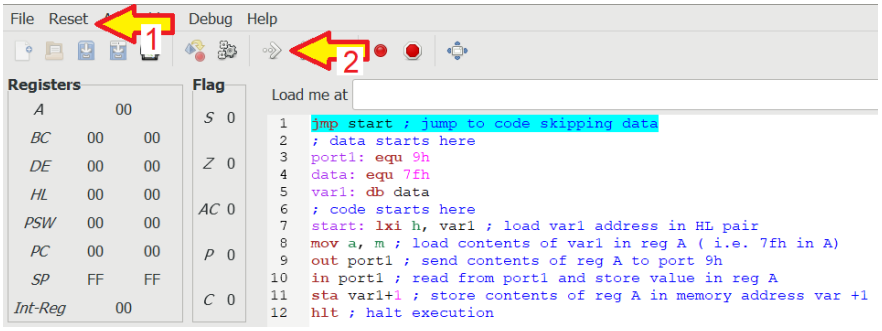
```
; Assembler generated listing; Not editable.
; Generated by GNUSim8085: http://www.gnusim8085.org/
4200 C3 04 042      jmp start ; jump to code skipping data
                    ; data starts here
           port1: equ 9h
           data: equ 7fh
4203    7F (1 bytes)      var1: db data
                    ; code starts here
```

```
4204 21 03 042     start: lxi h, var1 ; load var1 address in HL
   pair
4207 7E          mov a, m ; load contents of var1 in reg A ( i.e. 7
   fh in A)
4208 D3 09         out port1 ; send contents of reg A to port 9h
420A DB 09         in port1 ; read from port1 and store value in
   reg A
420C 32 04 042     sta var1+1 ; store contents of reg A in memory
    address var +1
420F 76            hlt ; halt execution
```

- What is the address of mnemonics mov a,m? 4207.
- What is the opcode of mnemonics mov a,m? 7E.
- How many bytes mnemonics mov a,m will occupy? 1 byte.
- How many bytes mnemonics in port1 will occupy? 2 bytes.
- How many bytes mnemonics sta var1+1 will occupy? 3 bytes.
- How compiler state the address of label "start" in the opcode of jmp start? start with LSB 04 then MSB 042.

1.2.5 Execution step by step

16. For the sake of certainty, Reset All. Then click "Step in the code" 



17. observe the register value changes in left pane.

Experiment: 2

Examine Logic manipulation

2.1 Experiment objectives:

- Learn how to use assembly logic instructions.
- Debug a simple codes.
- Monitor the Registers and memory.

2.2 Experiment procedures:

2.2.1 Assembly Programming

1. Open GNUSim8085 .
2. Copy the next code and paste it in the IDE.

```
;<Program title>
    jmp start
;data

;code
start:  MVI A,56h
    CMA
    STC
    MVI B,0FH
    ANA B
    ANI 1
    ORA B
    ORI 80h
    XRA B
    XRA A
    XRI 32h
    STC
    hlt
```

3. Save the file  , name it "lab2.asm".

4. Compile the file  , this step to check the syntax and any typo errors.

2.2.2 Correcting the errors

5. Correct errors if exist.

2.2.3 Program Memory

6. Check what will be written in program memory and the assigned physical addresses. from pull-down menus select (Assembler¿¿ Show listing)

```
4200 C3 03 042            jmp start
;data


;code
4203 3E 56    start:   MVI A,56h
4205 2F         CMA
4206 37         STC
4207 06 0F      MVI B,0FH
4209 A0         ANA B
420A E6 01      ANI 1
420C B0         ORA B
420D F6 80      ORI 80h
420F A8         XRA B
4210 AF         XRA A
4211 EE 32      XRI 32h
4213 37         STC
4214 76           hlt
```

- Calculate clock cycles?.....71....Clock cycles.
- How many bytes this code occupy?.....21..Bytes.

2.2.4 Execution step by step

7. For the sake of certainty, Reset All. Then click "Step in the code"
   - what is the effect of instruction in address 4206h on carry flag.
   - what is the effect of instruction in address 4209h on carry flag.
8. observe the register value changes in left pane.

Experiment: 3

Assembly functions

3.1 Experiment objectives:

- Learn how to use assembly functions.
- Debug a simple codes.
- Monitor the Registers and memory.

3.2 Experiment procedures:

3.2.1 Assembly Programming

1. Open GNUSim8085 .
2. Copy the next code and paste it in the IDE.

```
;<Program title>

jmp start

;data


;code
start:     MVI A,56h
           CMA
           STC
           MVI B,0FH
           CC Function1
           XRA B
           XRA A
           XRI 32h
           STC
           hlt
Function1:  ANA B
           ANI 1
           ORA B
           ORI 80h
```

```
        RET
```

3. Save the file  , name it "lab3.asm".

4. Compile the file  , this step to check the syntax and any typo errors.

3.2.2 Correcting the errors

5. Correct errors if exist.

3.2.3 Program Memory

6. Check what will be written in program memory and the assigned physical addresses. from pull-down menus select (Assembler¿¿ Show listing)

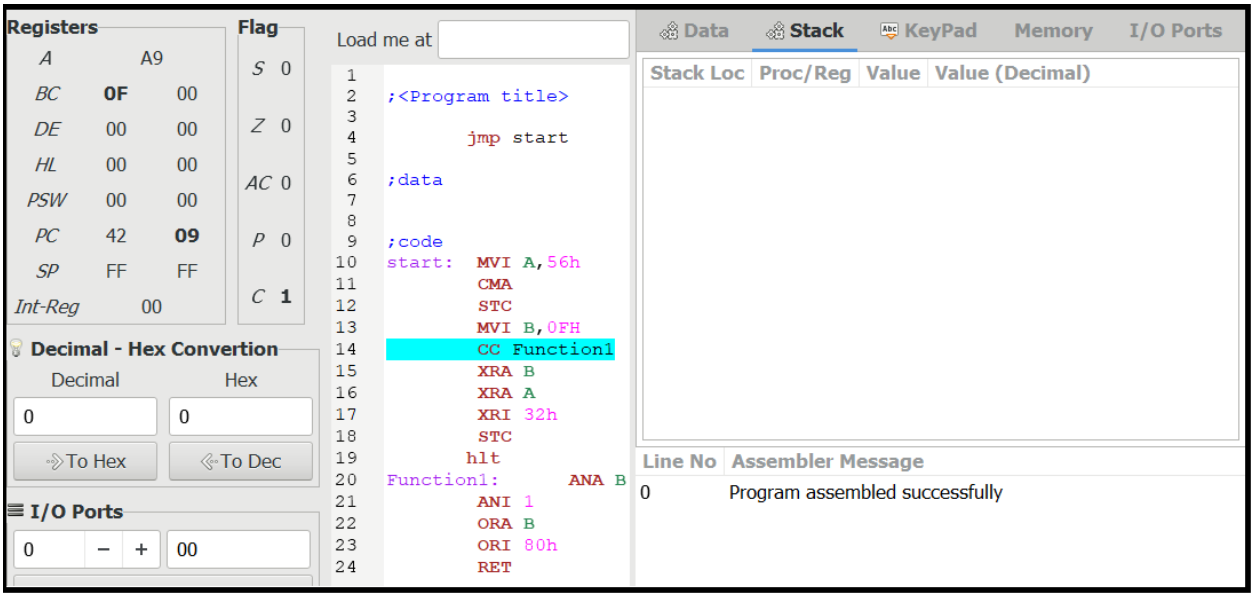
```
4200 C3 03 042          jmp start

;data


;code
4203 3E 56   start:   MVI A,56h
4205 2F               CMA
4206 37               STC
4207 06 0F            MVI B,0FH
4209 DC 12 42         CC Function1
420C A8               XRA B
420D AF               XRA A
420E EE 32            XRI 32h
4210 37               STC
4211 76                 hlt
4212 A0    Function1:     ANA B
4213 E6 01            ANI 1
4215 B0               ORA B
4216 F6 80            ORI 80h
4218 C9               RET
```

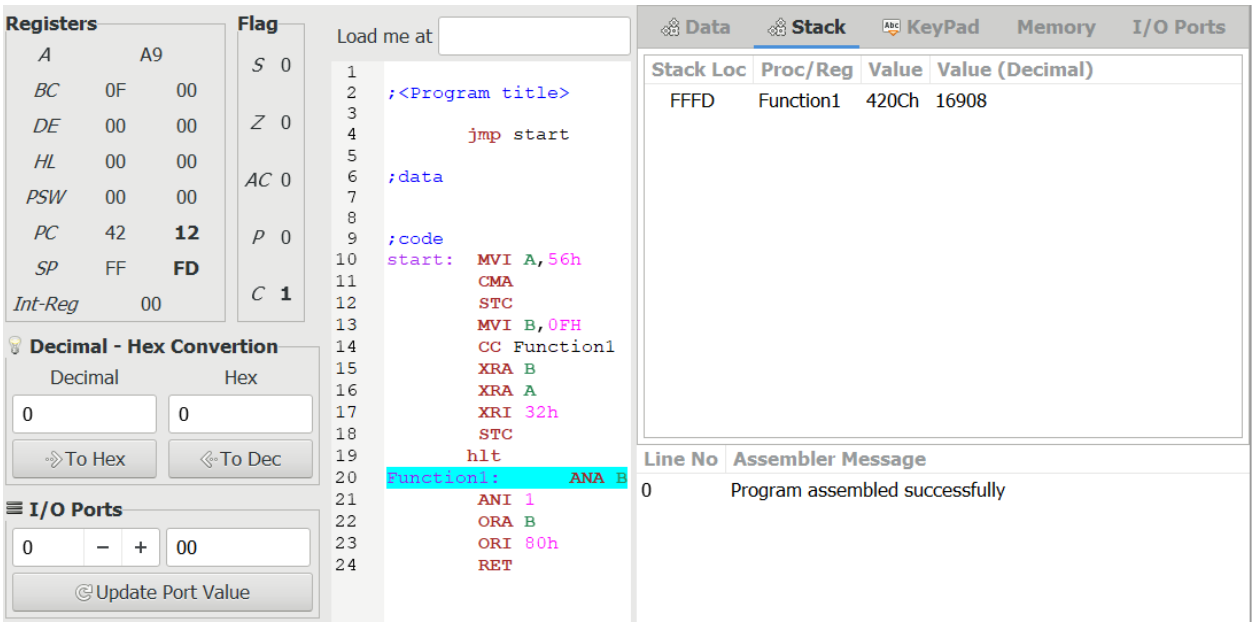
3.2.4 Execution step by step

7. For the sake of certainty, Reset All. Then click "Step in the code"

8. observe the register value changes in left pane. and stack in in the right pane.

9. check the stack memory before and after executing "CC Function1" instruction.





10.
- Guess what is stored in stack memory?.
- Which instruction the address is pointing for?

11. check the stack memory after executing "RET" instruction.

8085 Instruction Set

Instructions		Operation	Cycles	Bytes	Flag	Description
Mnemonics	Arguments					
ACI	n	A=A+n+CY	7	2	**SZAPC**	Add with Carry Immediate
ADC	r	A=A+r+CY(21X)	4	1	**SZAPC**	Add with Carry
ADC	M	A=A+[HL]+CY	7	1	**SZAPC**	Add with Carry to Memory
ADD	r	A=A+r (20X)	4	1	**SZAPC**	Add
ADD	M	A=A+[HL]	7	1	**SZAPC**	Add to Memory
ADI	n	A=A+n	7	2	**SZAPC**	Add Immediate
DAD	B	HL=HL+BC	10	1	SZAP**C**	Double Add BC to HL
DAD	D	HL=HL+DE	10	1	SZAP**C**	Double Add DE to HL
DAD	H	HL=HL+HL	10	1	SZAP**C**	Double Add HL to HL
DAD	SP	HL=HL+SP	10	1	SZAP**C**	Double Add SP to HL
DCR	r	r=r-1	4	1	**SZAP**C	Decrement
DCR	M	[HL]=[HL]-1	4	1	**SZAP**C	Decrement Memory
DCX	B	BC=BC-1	6	1		Decrement BC
DCX	D	DE=DE-1	6	1		Decrement DE
DCX	H	HL=HL-1	6	1		Decrement HL
DCX	SP	SP=SP-1	6	1		Decrement Stack Pointer
INR	r	r=r+1 (0X4)	4	1	**SZAP**C	Increment
INR	M	[HL]=[HL]+1	4	1	**SZAP**C	Increment Memory
INX	B	BC=BC+1	6	1		Increment BC
INX	D	DE=DE+1	6	1		Increment DE
INX	H	HL=HL+1	6	1		Increment HL
INX	SP	SP=SP+1	6	1		Increment Stack Pointer
SBB	r	A=A-r-CY	4	1	**SZAPC**	Subtract with Borrow
SBB	M	A=A-[HL]-CY	7	1	**SZAPC**	Subtract with Borrow
SBI	n	A=A-n-CY	7	2	**SZAPC**	Subtract with Borrow Immed
SUB	r	A=A-r (22X)	4	1	**SZAPC**	Subtract
SUB	M	A=A-[HL]	7	1	**SZAPC**	Subtract Memory
SUI	n	A=A-n	7	2	**SZAPC**	Subtract Immediate

Instructions		Operation	Cycles	Bytes	Flag	Description
Mnemonics	Arguments					
ANA	r	A=A & r	4	1	**SZAP**0	AND Accumulator
ANA	M	A=A & [HL]	4	1	**SZAP**0	AND Accumulator and Memory
ANI	n	A=A & n	7	2	**SZ**0**P**0	AND Immediate
CMA		A=~A	4	1	SZAPC	Complement Accumulator
ORA	r	A=A | r	4	1	**SZ**0**P**0	Inclusive OR Accumulator
ORA	M	A=A | [HL]	7	1	**SZ**0**P**0	Inclusive OR Accumulator
ORI	n	A=A | n	7	2	**SZ**0**P**0	Inclusive OR Immediate
XRA	r	A=A⊕r	4	1	**SZ**0**P**0	Exclusive OR Accumulator
XRA	M	A=A⊕ [HL]	7	1	**SZ**0**P**0	Exclusive OR Accumulator
XRI	n	A=A⊕n	7	2	**SZ**0**P**0	Exclusive OR Immediate
RAL		A={CY,A}<-	4	1	SZAP**C**	Rotate Accumulator Left
RAR		A=->{CY,A}	4	1	SZAP**C**	Rotate Accumulator Right
RLC		A=A<-	4	1	SZAP**C**	Rotate Left Circular
RRC		A=->A	4	1	SZAP**C**	Rotate Right Circular
CMP	r	A-r	4	1	**SZAPC**	Compare
CMP	M	A-[HL]	7	1	**SZAPC**	Compare with Memory
CPI	n	A-n	7	2	**SZAPC**	Compare Immediate

Instructions		Operation	Cycles	Bytes	Flag	Description
Mnemonics	Arguments					
MOV	r1,r2	r1=r2	4	1	-----	Move register to register
MOV	M,r	[HL]=r	7	1	-----	Move register to Memory
MOV	r,M	r=[HL]	7	1	-----	Move Memory to register
MVI	r,n	r=n	7	2	-----	Move Immediate
MVI	M,n	[HL]=n	10	2	-----	Move Immediate to Memory

LDA	a	A=[a]	13	3	-----	Load Accumulator direct
LDAX	B	A=[BC]	7	1	-----	Load Accumulator indirect
LDAX	D	A=[DE]	7	1	-----	Load Accumulator indirect
LHLD	a	HL=[a]	16	3	-----	Load HL Direct
LXI	B,nn	BC=nn	10	3	-----	Load Immediate BC
LXI	D,nn	DE=nn	10	3	-----	Load Immediate DE
LXI	H,nn	HL=nn	10	3	-----	Load Immediate HL
LXI	SP,nn	SP=nn	10	3	-----	Load Immediate Stack Ptr

STA	a	[a]=A	13	3	-----	Store Accumulator
STAX	B	[BC]=A	7	1	-----	Store Accumulator indirect
STAX	D	[DE]=A	7	1	-----	Store Accumulator indirect

SHLD	a	[a]=HL	16	3	-----	Store HL Direct
SPHL		SP=HL	6	1	-----	Move HL to SP
XCHG		HL<->DE	4	1	-----	Exchange HL with DE
XTHL		[SP]<->HL	16	1	-----	Exchange stack Top with HL

IN	p	A=[p]	10	2	-----	Input
OUT	p	[p]=A	10	2	-----	Output

Instructions		Operation	Cycles	Bytes	Flag	Description
Mnemonics	Arguments					
JMP	a	PC=a	7	3	-----	Jump unconditional
JM	a	If S=1	7/(10~s)	3	-----	Jump on Minus
JP	a	If S=0	7/(10~s)	3	-----	Jump on Plus
JC	a	If CY=1	7/(10~s)	3	-----	Jump on Carry
JNC	a	If CY=0	7/(10~s)	3	-----	Jump on No Carry
JZ	a	If Z=1	7/(10~s)	3	-----	Jump on Zero
JNZ	a	If Z=0	7/(10~s)	3	-----	Jump on No Zero
JPE	a	If P=1	7/(10~s)	3	-----	Jump on Parity Even
JPO	a	If P=0	7/(10~s)	3	-----	Jump on Parity Odd
PCHL		PC=[HL]	6	1	-----	Jump HL indirect
CALL	a	-[SP]=PC,PC=a	18	3	-----	Call unconditional
CM	a	If S=1	9/(18~s)	3	-----	Call on Minus
CP	a	If S=0	9/(18~s)	3	-----	Call on Plus
CC	a	If CY=1	9/(18~s)	3	-----	Call on Carry
CNC	a	If CY=0	9/(18~s)	3	-----	Call on No Carry
CZ	a	If Z=1	9/(18~s)	3	-----	Call on Zero
CNZ	a	If Z=0	9/(18~s)	3	-----	Call on No Zero
CPE	a	If P=1	9/(18~s)	3	-----	Call on Parity Even
CPO	a	If P=0	9/(18~s)	3	-----	Call on Parity Odd
RET		PC=[SP]+	10	1	-----	Return
RM		If S=1	6/(12~s)	1	-----	Return on Minus
RP		If S=0	6/(12~s)	1	-----	Return on Plus
RC		If CY=1	6/(12~s)	1	-----	Return on Carry
RNC		If CY=0	6/(12~s)	1	-----	Return on No Carry
RZ		If Z=1	6/(12~s)	1	-----	Return on Zero
RNZ		If Z=0	6/(12~s)	1	-----	Return on No Zero
RPE		If P=1	6/(12~s)	1	-----	Return on Parity Even
RPO		If P=0	6/(12~s)	1	-----	Return on Parity Odd

Instructions		Operation	Cycles	Bytes	Flag	Description
Mnemonics	Arguments					
CMC		CY=~CY	4	1	SZAPC	Complement Carry
STC		CY=1	4	1	1	Set Carry
NOP			4	1	-----	No Operation
RST	z	-[SP]=PC,PC=z	12	1	-----	Restart (3X7)
HLT			5	1		Halt
DAA		A=BCD format	4	1	**SZAPC**	Decimal Adjust Accumulator
SIM		mask=A	4	1	-----	Set Interrupt Mask
RIM		A=mask	4	1	-----	Read Interrupt Mask
DI			4	1	-----	Disable Interrupts
EI			4	1	-----	Enable Interrupts
POP	B	BC=[SP]+	10	1	-----	Pop BC
POP	D	DE=[SP]+	10	1	-----	Pop DE
POP	H	HL=[SP]+	10	1	-----	Pop HL
POP	PSW	{PSW,A}=[SP]+	10	1	-----	Pop Processor Status Word
PUSH	B	-[SP]=BC	12	1	-----	Push BC
PUSH	D	-[SP]=DE	12	1	-----	Push DE
PUSH	H	-[SP]=HL	12	1	-----	Push HL
PUSH	PSW	-[SP]={PSW,A}	12	1	-----	Push Processor Status Word