Microprocessor Architecture

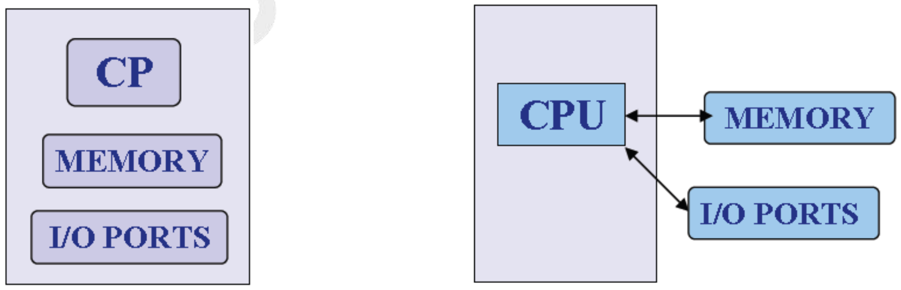
1.1 Lecture objectives: at the end of this lecture the student will able to:

- **Define the microprocessor and microcontroller and find differences between them.**
- **Define the microprocessor in general.**
- **Determine the application of microprocessor.**
- **Understand the architecture of microprocessor.**
- **Explain the applications of 8085 µP**

1.2 **A microprocessor** (µP) is a group of electronic circuits fabricated on a semiconductor chip using either **L**arge **S**cale **I**ntegrated circuit **(LSI)** or **V**ery **L**arge **S**cale **I**ntegrated circuit **(VLSI)** techniques that can read binary instructions written in a memory and process binary data according to those instructions. One or more microprocessors typically serve as a central processing unit (CPU) in a computer system or handheld device. Three basic characteristics differentiate microprocessors are instruction set, Bandwidth and Clock speed.

1.3 **A microcontroller**: A highly integrated chip that contains all the components comprising a controller. Typically this includes a CPU, RAM, some form of ROM, I/O ports, and timers. Unlike a general-purpose computer, which also includes all of these components, a microcontroller is designed for a very specific task - to control a particular system. A microcontroller is meant to be more self-contained and independent, and functions as a tiny, dedicated computer. The great advantage of microcontrollers, as opposed to using larger microprocessors, is that the parts-count and design costs of the item being controlled can be kept to a minimum.

Fig. (1.1) below simply show the difference between microprocessor and microcontroller. As shown in Fig. (1.1a), the microcontroller chip include memory block and I/P ports in additive to CPU, while the microprocessor chip include CPU only as shown in Fig(1.1b).



(a) microcontroller (b) microprocessor

Figure (1.1): The difference between microprocessor and microcomputer.

1.4 8085 Microprocessor features

The salient features of 8085 µp are:

- It is a 8 bit microprocessor.
- It is manufactured with N-MOS technology.
- It has 16-bit address bus and hence can address up to 2^{16} = 65536 bytes (64KB) memory locations through A^0-A^{15}.
- The first 8 lines of address bus and 8 lines of data bus are multiplexed $AD^0 – AD^7$.
- Data bus is a group of 8 lines $D^0 – D^7$.
- It supports external interrupt request.
- A 16 bit program counter (PC).
- A 16 bit stack pointer (SP).
- Six 8-bit general purpose register arranged in pairs: BC, DE, HL.
- It requires a signal +5V power supply and operates at 3.2 MHZ single phase clock.
- It is enclosed with 40 pins DIP (Dual in line package).

1.5 8085 Microprocessor architecture :

In general, the microprocessor system contains from two basic parts are hardware and software. The hardware of 8085 μp included three parts are registers group, ALU and control unit, see Fig. (1.2).

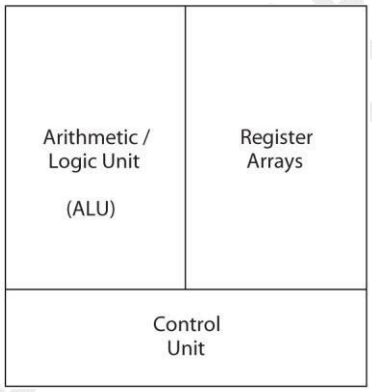


Figure 1.2: 8085 μp hardware parts.

1.5.1 Group of Register: there are two types of registers in 8085 microprocessor as shown below:

a- **General Purpose Register**: GPRs are six (8-bit) registers used to store data temporarily during execution of programs, these registers are (B, C, D, E, H, and L) .

a.1 8-bit B and 8-bit C registers can be used as one 16-bit BC register pair. When used as a pair the C register contains low-order byte. Some instructions may use BC register as a data pointer.

a.2 8-bit D and 8-bit E registers can be used as one 16-bit DE register pair. When used as pair the E register contains low-order byte. Some instructions may use DE register as a data pointer.

b- **Special Purpose Register**: SPRs are used to achieve special task as shown below:
b.1 Accumulator(A): Accumulator is an 8-bit register which used as part of the Arithmetic/Logic unit (ALU) to perform arithmetic and logic operations and the results of these operation stored in accumulator. Some times the accumulator used as general purpose register to temporarily storage.

b.2 Flag Register(F): Flag register is an 8-bit register used to indicate the status of five flags after execute the arithmetic or logic operation. The distribution of flags on flag register is shown in fig.2 below:

F7	F6	F5	F4	F3	F2	F1	F0
S	Z	X	AC	X	P	X	CY

Figure2: flags position on flag register

a) CY: carry flag where is one if found carry from MSB and zero if no carry from MSB of the arithmetic or logic operation.

b) P: parity flag where is one if the parity of result of arithmetic and logic operation is even and zero if parity of arithmetic and logic operation is odd.

c) AC: auxiliary carry flag where is one if carry found from fourth bit **F3** and zero if no carry from **F3** of the arithmetic or logic operation.

d) Z: zero flag where is one if the result of arithmetic and logic operation is zero and zero if any result found.

e) S: sign flag where is one if the seventh bit (**F7**) of arithmetic and logic operation is one and zero if the seventh bit (**F7**) of arithmetic and logic operation is zero.

b.3 Stack Pointer (SP): the stack pointer is 16-bit register used to point to the memory locations called **<u>stack</u>**. Some times this register used as general purpose register. The stack memory is part of overall R/W memory used with subroutine and push instruction and addressed by stack pointer.

b.4 Program counter (PC): PC is 16-bit register used to addressing the memory locations which loaded with codes of program, on the other hand used to sequence the execution of instructions.

b.5 Instruction Register/Decoder : Temporary store for the current instruction of a program. Latest instruction sent here from memory prior to execution. Decoder then takes instruction and „decodes" or interprets the instruction. Decoded instruction then passed to next stage.

1.5.2 Arithmetic/Logic Unit (ALU): this unit is part of microprocessor contain from digital circuits which used to execute the arithmetic and logic operation of data that stored in general purpose register or directly data. The result of most arithmetic and logic operation are saved in accumulator.

1.5.3 Control Unit: this part of microprocessor is responsible of provides the necessary timing and control signals which organize flow of data between the microprocessor and memory or peripherals devices. Several signals are provide by this unit as shown below:

a) Reset out: to reset all units coupled to microprocessor.
b) Reset In: to reset all parts of microprocessor.
c) Hold and HLDA: to arrange transfer of data in case of DMA mode.

d) IO/M : to determine if the communicate operation is between microprocessor and memory or IO devices.
e) ALE : to activate the (AD0 AD7)pins as address lines .
f) $\overline{\text{WR}}$: to write data on memory or IO devices.
g) $\overline{\text{RD}}$: to read data from memory or IO devices.
h) Ready: indicate that the microprocessor is available to transfer data or not.
i) Clock Out: to synchronize all units coupled to microprocessor.

Fig. 1.3 show the details of 8085 microprocessor hardware parts.

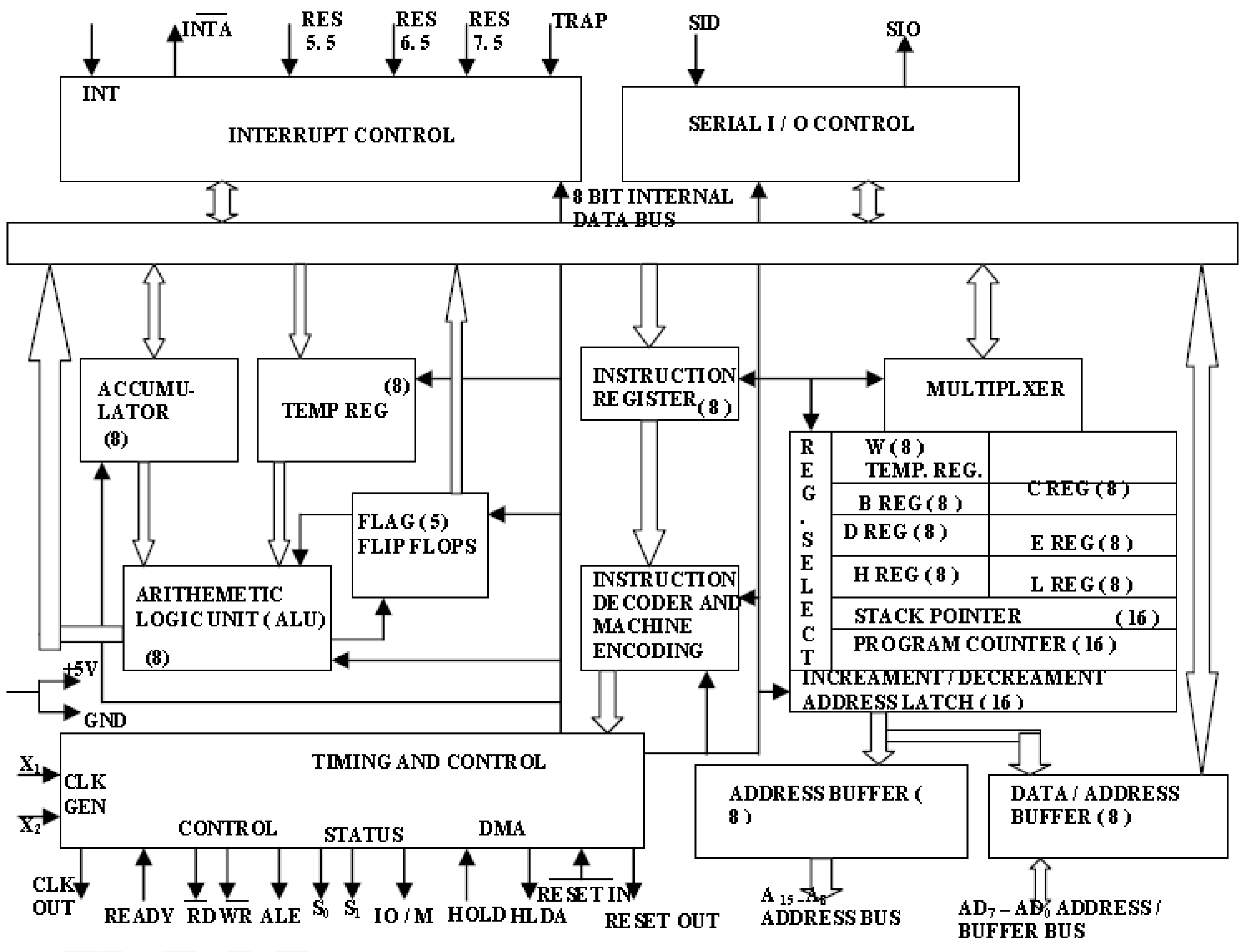


Figure 1.3: The block diagram of 8085 microprocessor hardware details.

1.6 Application of Microprocessor

Microprocessor has applications in several fields as shown below:
1-Home
2-Automotive
3-Communication/ Telecom
4-Medical
5-Office

6- Transport
7-Railways
8-Retail
9-Energy / Power
10-Industrial
11-Defence, Space, Aerospace

<u>Home work:</u>

Q1/ Give at least one example to each one of application of microprocessor.

Q2/ Write the contents of flag register after execute the following operation $(AC)_{16}$ + $(3B)_{16}$.

Instruction Set

Lecture objectives: at the end of this lecture the student will able to:

1- **Define the instruction and program.**

2- **Classify the instructions set of 8085 microprocessor.**

3- **Understand the assembly and machine languages**

4- **Explain the operation of data transfer instructions in 8085 microprocessor.**

2.1 Definition of instruction and program in 8085 microprocessor:

The second part of microprocessor system is the software. Software is the programs that can be executed by the system.

An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions that a microprocessor supports is called ***Instruction Set***. A 8085 microprocessor has ***246*** instructions. Each instruction is represented by an ***8-bit*** binary value. These 8-bits of binary value is called ***Op-Code*** or ***Instruction Byte***.

A program is some of instructions written in certain method to achieve the certain task or function. The program is the communication method between person and system. Therefore, to write and execute any program by using 8085 microprocessor must studying the instructions that used in this processor.

Any instruction or program in 8085 microprocessor is written in two methods, the first is assembly language and the second is machine language. In assembly language the instruction is written as words such as (**MOV, LXI, ADD C, JMP, PUSH PSW**, etc). The other form of instruction is a binary number which called the machine language, where each instruction has special code consist of two digits written by hexadecimal system called the Opcode such as (**76** which is Opcode of **HLT** instruction, **7E** which is Opcode of **MVI M** instruction, etc).

2.2 Instruction Set Classification

The 8085 microprocessor instructions are classified into five groups as in below:

1- Data transfer group.
2- Arithmetic group.
3- Logic group.
4- Branch group.
5- Stack, I/O, and machine control group.

2.3 Data Transfer Group Instructions:

These instructions move data between registers, or between memory and registers. These instructions copy data from source to destination, where the contents of source are not modified through copying.

Opcode	Operand	Description
MOV	Rd, Rs M, Rs Rd, M	Copy from source to destination.

A- **MOV (destination), (source) [one byte instruction]**

Transferring the contents of source to destination where the contents of source is 8-bit number or two digits in hexadecimal system.

Where Rd (register destination) and Rs (register source) are one of registers (**A, B, C, D, E, H**, and **L**) and $M_{(HL)}$ is the memory location which addressing by contents of registers pair (**HL**).

Examples

MOV A,B transferring the contents of register **B** to register **A**.

MOV C,M transferring the contents of memory location which addressing by contents of registers pair **HL** to register **C.**

MOV M,D transferring the contents of register **D** to memory location which addressing by contents of register pair **HL**.

Opcode	Operand	Description
MVI	Rd, Data M, Data	Move immediate 8-bit

B- **MVI destination, data** **[two bytes instruction]**

Loading the destination with 8-bit data immediately. Where Rd is one of registers (**A, B, C, D, E, H**, and **L**) and $\mathbf{M_{HL}}$ is the memory location addressing by contents of register pair (**HL**).

Examples

MVI H,34 Loading the register **H** with data **34** immediately.

MVI M,34 Loading the Mem. location which addressing by (**HL**) with data **34** immediately.

Opcode	Operand	Description
LXI	Reg. pair, 16-bit data	Load register pair immediate

C- **LXI destination,(16-bit)Data** **[three bytes instruction]**

Loading the destination with data 16-bit immediately, where the destination is Registers Pair (**BC, DE, HL**) or 16 bit register **SP.**

LXI B,3456 Loading registers pair **BC** with data **3456** immediately where **34** loaded in register **B** and **56** loaded in register **C.**

Opcode	Operand	Description
LDA	16-bit address	Load Accumulator

D- **LDA 16-bit number** **[three bytes instruction]**

Transferring the contents of memory location which addressed by the operand to accumulator. Operand is (16-bit) number used as address to memory location.

Example

LDA AD21 transferring the contents of memory location which has address **AD21** to **A**

Opcode	Operand	Description
STA	16-bit address	Store accumulator direct

E- **STA 16-bit number** **[three bytes instruction]**

Transferring the contents of accumulator to memory location which addressed by operand. Operand is (16-bit) number used as address to memory location.

Example

STA AD21 transferring (**A**) to memory location which has address **AD21**.

Opcode	Operand	Description
LDAX	B/D Register Pair	Load accumulator indirect

F- **LDAX R.P.** **[one byte instruction]**

Transferring the contents of memory location which addressed by Rp to accumulator. Rp is on of the register pair (**BC, DE**)

Example

LDAX D transferring the contents of memory location which addressing by (**DE**) to **A**

Opcode	Operand	Description
STAX	Reg. pair	Store accumulator indirect

G- **STAX R.P.** **[one byte instruction]**

Transferring the contents accumulator to memory location which addressed by R.P. R.P. is on of the register pair (**BC, DE**)

Example

STAX B transferring (**A**) to memory location which addressing by (**BC**) .

Opcode	Operand	Description
LHLD	16-bit address	Load H-L registers direct

H- **LHLD 16-bit number** **[three bytes instruction]**

Transferring the ($M_{16\text{-bit number}}$) to register **L** and transfer the ($M_{16\text{-bit number}+1}$) to register **H**.

Example

LHLD 2000 transferring (M_{2000}) to register **L** and (M_{2001}) to register **H.**

Opcode	Operand	Description
SHLD	16-bit address	Store H-L registers direct

I- **SHLD 16-bit number** **[three bytes instruction]**

Transferring the (**L**) to (**$M_{16\text{-bit number}}$**) and transfer the (**H**) to (**$M_{16\text{-bit number}+1}$**).

Example

SHLD 2000 transferring (**L**) to (**M_{2000}**) and transfer the (**H**) to (**M_{2001}**).

Opcode	Operand	Description
XCHG	None	Exchange H-L with D-E

J- XCHG **[one byte instruction]**

Replacement the (**D**) with (**H**) and (**E**) with (**L**).

Opcode	Operand	Description
SPHL	None	Copy H-L pair to the Stack Pointer (SP)

K- SPHL **[one byte instruction]**

This instruction loads the contents of **HL** pair into **SP**.

Opcode	Operand	Description
XTHL	None	Exchange H–L with top of stack

L-XTHL **[one byte instruction]**

The contents of reg. **L** is exchanged with the stack location pointed out by the contents of the SP and the contents of reg. **H** are exchanged with the stack location pointed out by the contents of the (SP + 1).

Opcode	Operand	Description
PCHL	None	Load program counter with H-L contents

M- PCHL **[one byte instruction]**

The contents of registers **H** and **L** are copied into the program counter (**PC**). The contents of **H** are placed as the high-order byte and the contents of **L** as the low-order byte.

Homework:

Transfer the contents of stack memory locations (4000, 4001, 4002, 4003) to the memory locations (2030, 2031, 2032, 2033) [***Note:*** the second location not stack memory].

Arithmetic and Logic Instructions Groups

Lecture objectives: at the end of this lecture the student will able to:

1- **Define the arithmetic and logic operations that can be executed in 8085 microprocessor.**

2- **Explain the operation of arithmetic instructions in 8085 microprocessor.**

3- **Explain the operation of logic instructions in 8085 microprocessor.**

4- **Write programs to execute the arithmetic and logic operation.**

3.1 The arithmetic and logic operations:

3.1.1 Addition

Any 8-bit number, or the contents of register, or the contents of memory location can be added to the contents of accumulator. The result of summation operation is stored in the accumulator. Where, no two other 8-bit registers can be added directly.

3.1.2 Subtraction

Any 8-bit number, or the contents of register, or the contents of memory location can be subtracted from the contents of accumulator. The result is stored in the accumulator, where subtraction operation is performed in 2's complement form. If the result is negative, it is stored in 2's complement form. No two other 8-bit registers can be subtracted directly.

3.1.3 Increment / Decrement

The 8-bit contents of a register or a memory location can be incremented or decremented by 1. Also, the 16-bit contents of a register pair can be incremented or decremented by 1. Increment or decrement can be performed on any register or a memory location.

3.2 Arithmetic instructions

Opcode	Operand	Description
ADD	R M	Add register or memory to accumulator

A- **ADD operand** **[one byte instruction]**

Add (**A**) to operand and save the result in accumulator. There are two types of **ADD** instruction according to type operand. Operand is one of registers (**A, B, C, D, E, H**, and **L**) or memory location which addressed by (**HL**).

ADD C	Add (**A**) to (**C**) and save the result in accumulator.
ADD M	Add (**$M_{(HL)}$**) to (**A**) and save the result in accumulator.

Opcode	Operand	Description
ADC	R M	Add register or memory to accumulator with carry

B- **ADC operand** **[one byte instruction]**

Add (**A**) and operand and CY and save the result in accumulator. There are two types from **ADC** instruction according to type operand. Operand is one of registers (**A, B, C, D, E, H**, and **L**) or memory location which addressed by (**HL**).

ADC B	Add (**B**) with (**A**) with CY and save the result in accumulator.
ADC M	Add (**M$_{(HL)}$**) with (**A**) with CY and save the result in accumulator.

Opcode	Operand	Description
ADI	8-bit data	Add immediate to accumulator

C- **ADI, 8-bit data** **[two bytes instruction]**

Add 8-bit number with (**A**) and save the result in accumulator.

ADI 37	Add **37** to (**A**) and save the result in accumulator.

Opcode	Operand	Description
ACI	8-bit data	Add immediate to accumulator with carry

D- **ACI, 8-bit data** **[two bytes instruction]**

Add 8-bit number with (**A**) with CY and save the result in accumulator.

ACI 37	add **37** with (**A**) with CY and save the result in accumulator.

Opcode	Operand	Description
SUB	R M	Subtract register or memory from accumulator

E- **SUB operand** **[one byte instruction]**

Subtract operand from (**A**) and save the result in accumulator. There are two types from **SUB** instruction according to type operand. Operand is one of registers (**A, B, C, D, E, H**, and **L**) or memory location which addressed by (**HL**).

SUB C	subtract (**C**) from (**A**) and save the result in accumulator.
SUB M	subtract (**M$_{(HL)}$**) from (**A**) and save the result in accumulator.

Opcode	Operand	Description
SBB	R M	Subtract register or memory from accumulator with borrow

F- **SBB operand** **[one byte instruction]**

Subtract operand and borrow from (**A**) and save the result in accumulator. There are two types of **SBB** instruction according to type operand. Operand is one of registers (**A, B, C, D, E, H,** and **L**) or memory location which addressed by (**HL**).

SBB B	Subtract (**B**) and borrow from (**A**) and save the result in accumulator.
SBB M	Subtract (**M**$_{\textbf{(HL)}}$) and borrow from (**A**) and save the result in accumulator.

Opcode	Operand	Description
SUI	8-bit data	Subtract immediate from accumulator

G- **SUI, 8-bit data** **[two bytes instruction]**

subtract 8-bit data from (**A**)and save the result in accumulator.

SUI 37	Subtract **37** from (**A**) and save the result in accumulator.

Opcode	Operand	Description
SBI	8-bit data	Subtract immediate from accumulator with borrow

H- **SBI, 8-bit data** **[two bytes instruction]**

subtract 8-bit data and borrow from (**A**) and save the result in accumulator.

SBI 37	Subtract **37** and borrow from (**A**) and save the result in accumulator.

The previous instructions has the following basics:

1- Implicitly the accumulator is one of the registers.
2- These instructions are modify all flags according to data conditions of results.
3- The result of arithmetic operation stored in the accumulator.
4- Don’t affect the contents of the other register.

Opcode	Operand	Description
INR	R M	Increment register or memory by 1

I- INR operand **[one byte instruction]**

Increment the contents of operand by 1. There are two types of **INR** instruction according to operand type. Operand is one of registers (**A, B, C, D, E, H**, and **L**) or memory location which addressed by (**HL**).

INR E	increment by one the contents of register E.
INR M	increment by one the (**$M_{(HL)}$**).

Opcode	Operand	Description
DCR	R M	Decrement register or memory by 1

J- **DCR operand** **[one byte instruction]**

Decrement one from contents of operand. There are two types of **DCR** instruction according to operand type. Operand is one of registers (**A, B, C, D, E, H**, and **L**) or memory location which addressed by (**HL**).

DCR E	decrement one from contents of register E.
DCR M	decrement one from contents (**$M_{(HL)}$**).

The previous two instructions has the following basics:

1- This instruction affect the contents of specified register with instruction only.

2- This instruction affect all flags except CY flag.

Opcode	Operand	Description
INX	R	Increment register pair by 1

K- **INX R_P** **[one byte instruction]**

Increment one to contents of register pair. There is one types of **INX** instruction. Register pair is one of the (**BC, DE, HL**) and 16-bit register **SP**

INX D	increment one to contents of register pair **DE**.

Opcode	Operand	Description
DCX	R	Decrement register pair by 1

L- **DCX R_P** **[one byte instruction]**

Decrement one from contents of register pair. There is one types of **DCX** instruction. Register pair is one of the (**BC, DE, HL**) and 16-bit register **SP**

DCX B decrement one from contents of register pair **BC**.

Note : **INX** and **DCX** instruction don't affect any flags.

Opcode	Operand	Description
DAD	Reg. pair	Add register pair to H-L pair

M- **DAD, R.P.** **[one byte instruction]**

Add contents of register pair to contents of **HL** and save the result in **HL.**

DAD B Add (BC) to (HL) and save the result in HL.

Note: this instruction affect CY flag only.

3.3 Logical group instructions:

These instructions perform logical operations on data stored in registers, memory and status flags. The logical operations are (AND, OR, XOR, Rotate, Compare, Complement).

3.3.1 AND, OR, XOR logical operation:

Any 8-bit data, or the contents of register, or memory location can logically have (AND or OR or XOR) operations with the contents of accumulator where, the result is stored in accumulator.

Opcode	Operand	Description
ANA	R M	Logical AND register or memory with accumulator

A- **ANA operand** **[one byte instruction]**

This instruction execute AND logical operation between (A) and contents of operand and save the result in accumulator. Operand is one of registers (**A, B, C, D, E, H**, and **L**) or memory location which addressed by (**HL**) as shown below:

ANA D AND logical operation between (**D**) and (**A**) and saving the result in accumulator.

Opcode	Operand	Description
ANI	8-bit data	Logical AND immediate with accumulator

B- **ANI, 8-bit data** **[two bytes instruction]**

This instruction execute AND logical operation between (A) and 8-bit number and save the result in accumulator.

ANI 6A AND logical operation between **6A** and (**A**) and saving the result in accumulator.

Opcode	Operand	Description
ORA	R M	Logical OR register or memory with accumulator

C- **OR operand** **[one byte instruction]**

This instruction execute OR logical operation between (A) and contents of operand and save the result in accumulator. Operand is one of registers (**A, B, C, D, E, H**, and **L**) or memory location which addressing by (**HL**) as shown below:

OR C OR logical operation between (**C**) and (**A**) and saving the result in accumulator.

Opcode	Operand	Description
ORI	8-bit data	Logical OR immediate with accumulator

D- **OR, 8-bit data** **[two bytes instruction]**

This instruction execute the OR logical operation between (**A**) and 8-bit number and save the result in accumulator.

ORI 6A OR logical operation between **6A** and (**A**) and saving result in accumulator.

Opcode	Operand	Description
XRA	R M	Logical XOR register or memory with accumulator

E- **XRA operand** **[one byte instruction]**

this instruction execute Ex-OR logical operation between (**A**) and contents of operand and save the result in accumulator. The operand is one of registers (**A, B, C, D, E, H**, and **L**) or memory location which addressed by (**HL**) as shown below:

XRA E Ex-OR logical operation between (**E**) and (**A**) and saving the result in accumulator.

Opcode	Operand	Description
XRI	8-bit data	XOR immediate with accumulator

F- **XRI, 8-bit data** **[two bytes instruction]**

This instruction execute Ex- OR logical operation between (**A**) and 8-bit number and save the result in accumulator.

XRI 6A Ex-OR operation between **6A** and (**A**) and saving result in accumulator.

The previous instructions have following features:

- The result is placed in the accumulator.
- S, Z, P are modified to reflect the result.
- CY and AC are reset in OR and EX-OR operations, while in AND operation CY is reset and AC is set.

3.3.2 Rotate operation:

Each bit in the accumulator can be shifted either left or right to the next position.

Opcode	Operand	Description
RAR	None	Rotate accumulator right through carry

A- **RAR** **[one byte instruction]**

Rotate the content of **A**ccumulator to **R**ight with carry. Fig. 3.1 shows the rotate operation.

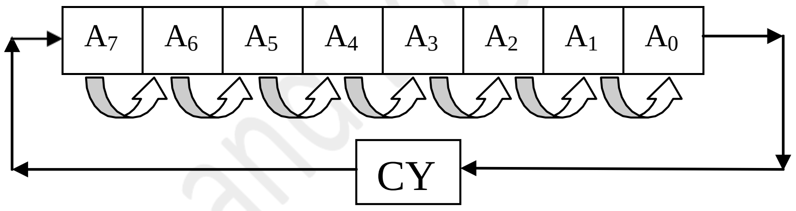


Figure 3.1: RAR instruction

Note: CY is modified according to bit D0. S, Z, P, AC are not affected.

Opcode	Operand	Description
RAL	None	Rotate accumulator left through carry

B- **RAL** **[one byte instruction]**

Rotate the content of **A**ccumulator to **L**eft with carry. Fig. 3.2 shows the rotate operation.

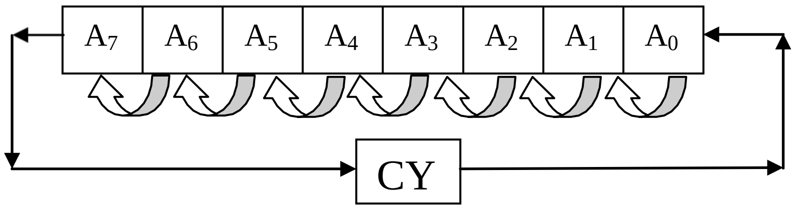


Figure 3.2: RAL instruction

Note: CY is modified according to bit D7, while, S, Z, P, AC are not affected.

Opcode	Operand	Description
RRC	None	Rotate accumulator right

C- **RRC** **[one byte instruction]**

Rotate the content of **A**ccumulator to **R**ight without carry. Fig. 3.3 shows the rotate operation.

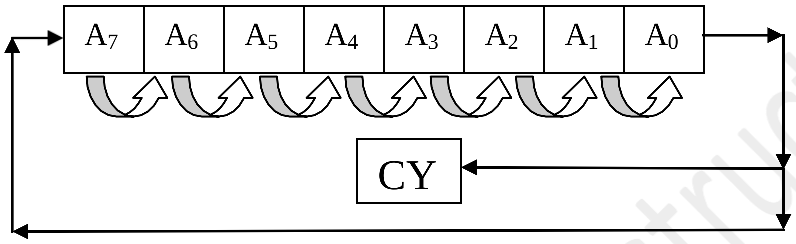


Figure 3.3: RRC instruction.

Note: CY is modified according to bit D0. S, Z, P, AC are not affected.

Opcode	Operand	Description
RLC	None	Rotate accumulator left

D- **RLC** **[one byte instruction]**
Rotate the content of **A**ccumulator to **L**eft without carry. Figure 3.4 below shows the rotate operation.

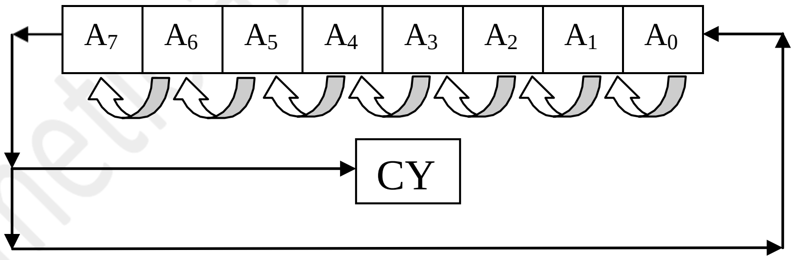


Figure 3.4: RLC instruction.

Note: CY is modified according to bit D7, S, Z, P, AC are not affected.

3.3.3 Complement:

The contents of accumulator can be complemented where, each 0 is replaced by 1 and each 1 is replaced by 0.

Opcode	Operand	Description
CMA	None	Complement accumulator

A- **CMA** **[one byte instruction]**

NOT logical operation(complement) to contents of accumulator.

Note: No flags are affected by this instruction.

Opcode	Operand	Description
STC	None	Set carry

B- **STC** **[one byte instruction]**

Set CY flag. (CY=1)

Note: No other flags are affected by this instruction.

Opcode	Operand	Description
CMC	None	Complement carry

C- **CMC** **[one byte instruction]**

NOT logical operation (complement) to CY flag.

Note: No other flags are affected by this instruction.

3.3. 4 Compare:

Any 8-bit data, or the contents of register, or memory location can be compares for:

- Equality
- Greater Than
- Less Than

with the contents of accumulator.

The result is reflected in status flags.

Opcode	Operand	Description
CMP	R M	Compare register or memory with accumulator

A- **CMP operand** **[one byte instruction]**

The contents of the operand (register or memory) are compared with the contents of the accumulator. Both contents are preserved. The flags are modified according to cases below:

If (**A**)> **operand**: [CY=0,Z=0]

If (**A**)< **operand:** [CY=1,Z=0]

If (**A**)= **operand:** [CY=0,Z=1]

Operand is one of register (**A, B, C, D, E, H,** and **L**) or (M_{HL}).

CMP E compare between (**E**) and (**A**).

Opcode	Operand	Description
CPI	8-bit data	Compare immediate with accumulator

B- **CPI operand** **[two bytes instruction]**

The 8-bit data is compared with the contents of accumulator. The values being compared remain unchanged. The flags are modified according to cases below:

If (**A**)> **data:** [CY=0,Z=0]

If (**A**)< **data:** [CY=1,Z=0]

If (**A**)= **data:** [CY=0,Z=1]

Branch Instructions Group

Lecture objectives: at the end of this lecture the student will able to:

1. **Explain the working of jumping instructions.**
2. **Explain the working of call and return instructions.**
3. **Write programs including jump or calling instructions.**

4.1 Branch Group Instructions:

The branch instructions allow the microprocessor to change the sequence of program execution. A 8085 microprocessor has two types of jumping are unconditional and conditional.

The branch group instructions is classified into three categories:

1. Jump instructions.
2. Call instructions.
3. Return instructions.

The branching instruction alter the normal sequential flow. These instructions alter either unconditionally or conditionally.

4.1.1 Jumping instructions

A- **JMP operand** **[three bytes instruction]**

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand, where, this lead to change the sequence of instructions execution. Operand is 16-bit number.

JMP 2300 changing the execution of program starting from address (**2300**).

B- **JC operand** **[three bytes instruction]**

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand if the (**CY=1**), where, this lead to change the sequence of instructions execution. Operand is 16-bit number.

JC 2300 changing the execution of program starting from address (**2300**) if (**CY=1**).

C- **JNC operand** **[three bytes instruction]**

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand if the (**CY=0**), where, this lead to change the sequence of instructions execution. Operand is 16-bit number.

JNC 2300 changing the execution of program starting from address (**2300**) <u>if</u> (**CY=0**).

D- **JPO operand** **[three bytes instruction]**

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand if the (**P=0, odd parity**), where, this lead to change the sequence of instructions execution. Operand is 16-bit number.

JPO 2300 changing the execution of program starting from address (**2300**) <u>if</u> (**P=0**).

E- **JPE operand** **[three bytes instruction]**

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand if the (**P=1, even parity**), where, this lead to change the sequence of instructions execution. Operand is 16-bit number.

JPE 2300 changing the execution of program starting from address (**2300**) <u>if</u> (**P=1**).

F- **JZ operand** **[three bytes instruction]**

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand if the (**Z=1**), where, this lead to change the sequence of instructions execution. Operand is 16-bit number.

JZ 2300 changing the execution of program starting from address (**2300**) <u>if</u> (Z=**1**).

G- **JNZ operand** **[three bytes instruction]**

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand if the (**Z=0**), where, this lead to change the sequence of instructions execution. Operand is 16-bit number.

JNZ 2300 changing the execution of program starting from address (**2300**) <u>if</u> (Z=**0**).

H- **JP operand** **[three bytes instruction]**

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand if the (S=**0**), where, this lead to change the sequence of instructions execution. Operand is 16-bit number.

JP 2300 changing the execution of program starting from address (**2300**) <u>if</u> (S=**0**).

I- **JM operand** **[three bytes instruction]**

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand if the (S=**1**), where, this lead to change the sequence of instructions execution. Operand is 16-bit number.

JM 2300 changing the execution of program starting from address (**2300**) <u>if</u> (S=**1**).

4.1.2 Call instructions

A- Call operand [three bytes instruction]

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand unconditionally, where, this lead to execute the subroutine that stored in memory starting from operand. Operand is 16-bit number.

Call 2300 calling the subroutine which stored in memory starting from (**2300**).

B- CC operand [three bytes instruction]

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand if (**CY=1**), where, this lead to execute the subroutine that stored in memory starting from operand. Operand is 16-bit number.

CC 2300 calling the subroutine which stored in memory starting from (**2300**) if **CY**=1

C- CNC operand [three bytes instruction]

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand if (**CY=0**), where, this lead to execute the subroutine that stored in memory starting from operand. Operand is 16-bit number.

CNC 2300 calling the subroutine which stored in memory starting from (**2300**) if **CY=0**

D- CPO operand [three bytes instruction]

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand if (**P=0**), where, this lead to execute the subroutine that stored in memory starting from operand. Operand is 16-bit number.

CPO 2300 calling the subroutine which stored in memory starting from (**2300**) if **P=0**.

E- CPE operand [three bytes instruction]

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand if (**P=1**), where, this lead to execute the subroutine that stored in memory starting from operand. Operand is 16-bit number.

CPE 2300 calling the subroutine which stored in memory starting from (**2300**) if **P=1**.

F- CZ operand [three bytes instruction]

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand if (**Z=1**), where, this lead to execute the subroutine that stored in memory starting from operand. Operand is 16-bit number.

CZ 2300 calling the subroutine which stored in memory starting from (**2300**) if **Z=1**.

G- CNZ operand [three bytes instruction]

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand if (**Z=0**), where, this lead to execute the subroutine that stored in memory starting from operand. Operand is 16-bit number.

CNZ 2300 calling the subroutine which stored in memory starting from (**2300**) if **Z=0**.

H- CP operand [three bytes instruction]

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand if (S=**0**), where, this lead to execute the subroutine that stored in memory starting from operand. Operand is 16-bit number.

CP 2300 calling the subroutine which stored in memory starting from (**2300**) if S=**0**.

I- CM operand

This instruction is loading the program counter with operand where the fetch and execute cycle of program starting from memory location that addressed by operand if (S=**1**), where, this lead to execute the subroutine that stored in memory starting from operand. Operand is 16-bit number..

CM 2300 calling the subroutine which stored in memory starting from (**2300**) if **S=1**.

4.1.3 Return instructions:

Return to the main program from subroutine.

A- **RET** changing the execution of program from subroutine to the main program unconditionally

---

B- RC changing the execution of program from subroutine to the main program if **CY=1**

C- RNC changing the execution of program from subroutine to the main program if **CY=0**

D- RZ changing the execution of program from subroutine to the main program if **Z=1**

E- RNZ changing the execution of program from subroutine to the main program if **Z=0**

F- RPE changing the execution of program from subroutine to the main program if **P=1**

G- RPO changing the execution of program from subroutine to the main program if **P=1**

H- RP changing the execution of program from subroutine to the main program if **S=0**

I- RM changing the execution of program from subroutine to the main program if **S=1**

Home work:

Write the content of accumulator after execute the following program:

```
LXI H,2000
MVI C,67
MOV M,C
MVI A,FF
CMP M
JNC A1
ANA C
A1: HLT
```

Stack and Subroutine

Lecture objectives: at the end of this lecture the student will able to:

1- **Define the stack memory.**

2- **Study the operation of PUSH and POP instructions.**

3- **Define the subroutine.**

4- **Write the subroutine.**

5.1 Stack Memory

5.1.1 Definition of Stack Memory

A stack memory is part from R/W memory used for temporary storage of data in subroutine or with **PUSH** / **POP** instructions.

5.1.2 Addressing of Stack Memory:

A stack memory is addressed by using stack pointer (SP) where the contents of SP used to determine the address of first memory location of stack memory (beginning of stack memory). Stack memory addressed using concept **FILO** (First In Last Out) or **LIFO** (Last IN First Out). The data saving in stack memory start with location that previous the first location which addressed by contents of SP.

5.2 **PUSH** and **POP** instructions: two instructions are used to data storage in stack memory which addressed by stack pointer.

5.2.1 PUSH operand (R_p)

Transferring the contents of operand (register pair) to the stack memory which beginning addressed by contents of stack pointer. Operand is one of register pair (**BC, DE, HL, AF**).

A- **PUSH B**	Transferring the (BC) to stack memory.	(**one byte instruction**)
B- **PUSH D**	Transferring the (DE) to stack memory.	(**one byte instruction**)
C- **PUSH H**	Transferring the (HL) to stack memory.	(**one byte instruction**)
D- **PUSH PSW**	Transferring the (AF) to stack memory.	(**one byte instruction**)

Note: PSW is stand for (Program Status Word) and AF is the accumulator and flag register

5.2.2 POP operand

Transferring the contents of stack memory locations which beginning addressed by contents of stack pointer to operand. Operand is one of register pair (**BC, DE, HL, AF**).

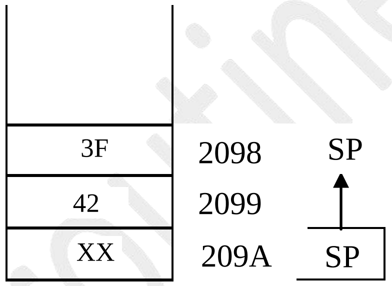
A- **POP B**	Transferring the contents of latest two stack memory locations to BC.
B- **POP D**	Transferring the contents of latest two stack memory locations to DE.
C- **POP H**	Transferring the contents of latest two stack memory locations to HL.
D- **POP PSW**	Transferring the contents of latest two stack memory locations to AF.

Example: write the contents of HL register and (SP) after execute the following instructions

```
LXI SP,209A
LXI H,423F
PUSH H
HLT
```

Solution:

First (**SP**)=209A (H)=42 (L)=3F



There for (SP)=2098 (H)=42 L=3F

As shown in Fig. (5.1)

Figure 5.1: Saving (HL) in stack memory

Example: write the contents of AF register and (SP) after execute the following instructions

```
LXI SP,3000
LXI D,200F
PUSH D
POP PSW
HLT
```

Solution:

First the (DE) transferred to stack memory locations (2FFF, 2FFE) by PUSH D instruction as shown in Fig. (5.2):

Then by POP PSW instruction, the content of memory location 2FFE transferring to flag register and contents of 2FFF to accumulator that mean

(SP)= 3000 (A)= 20 (F)=0F, see Fig. (5.3)

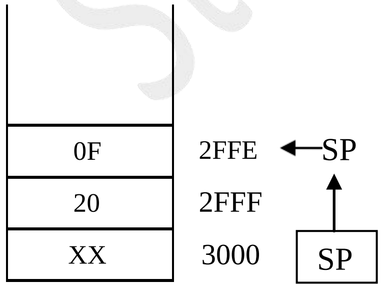


Figure 5.2

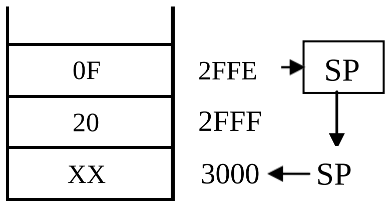


Figure 5.3

5.3 Subroutine

5.3.1 Subroutine definition: A subroutine is a group of instructions written separately from the main program to perform a function that occurs repeatedly in the main program. A large software project is usually divided into sub tasks called modules. These modules are developed independently as subroutines by different programmers. Each programmer can use all the microprocessor registers to write a subroutine without affect other parts of the program.

The subroutine can be calling from main program by using Calling instructions (condition or unconditional calling according to application). The RET instruction used to return to main program (there are condition and unconditional return).

Some operations occur when microprocessor execute the subroutine as shown in points below:

1- The contents of the program counter are saved in stack memory, where (PC) represent the address of memory location that loaded with code of instruction which follow the CALL instruction.

2- The operand of calling instruction will loaded in program counter, this means the execution sequence will transferred to subroutine.

3- The microprocessor is loading the contents of latest two locations of stack memory in program counter when it execute the return instruction in subroutine.

Example: Explain the steps of execution the following program.

Address of memory locations	Instruction	operand
1000	**LXI SP**	2300
	Some of Ins.	
100E	**MVI B**	34
1010	**CALL**	2070
1013	.**MOV A,C**	-------
	Some of Ins.	
1030	**HLT**	------
2070	**First ins. in subroutine**	
	Some of Ins.	-------
2070	**RET**	-------

Solution:

As shown in above table in example, the program stored in R/W memory starting with address (1000 H), so, when starting to execute the program the contents of PC and SP become:

PC=1000
SP=2300

Now, when the microprocessor arrive to execute the CALL instruction, it will add two to (PC) that means the (PC) equal to (1013 H) then save this number stack memory that have addresses (22FF 22FE). After that, the operand of CALL instruction [2070] will be loaded in PC to execute the program starting from instruction saved in memory location 2070 which it is represent the subroutine. At the end of the subroutine (when execute the RET instruction), the microprocessor will loading the data in two top locations of stack memory that means return to main program (exactly to MOV A,C instruction).