HAND WRITTEN NOTES:

OF

ELECTRICAL ENGINEERING

SUBJECT:

MICROPROCESSOR
**Microprocessor**

\[
\text{IES} \rightarrow \text{obj} = 5-6 \& (0085) + 10 (8086)
\]

→ Conventional = 30 to 40 marks.

**Definition:** It is electronic device that fetch instruction from memory execute them & provide result.

* It is electronic device that have computing & decision making capability.

**Memory:**

- ROM = ROM
- RAM = Main Memory = Memory.

**Note:**

* A MP can't perform any task on its own.

\[
\text{MP} = \text{H/w MP} + \text{S/w}.
\]

**ROM:**

* All system related information store in it

→ It come in the picture at the time of power switch on condition.

**RAM:**

Programs or instructions always feed in RAM.

* It is also called Main Memory or Memory.

μcomputer

* If all the task of CPU performed by MP, then such type of device is known as μcomputer.
μ-computer = MP + Memory + I/P device + O/P device.

μ-computer on single platform is known as μ-controller (chip).

ASIC = Application specific integrated chip.
μ-controller is the example of ASIC design.

Machine Language: It instructions or command written in binary pattern, then such type of language is called w/e language.

It is platform dependent language or w/e specific language.

Assembly Language: It binary command replaced by English like word that is called "Mnemonic" such type of language is called Assembly language.

It is also platform dependent language.
Assembler is a S/W that converts the code into machine language. It is performed manually and is known as hand assembly.

**Low level language:**
- All platform dependent language known as low level language.
- Example: All assembly language.

**High level language:**
- All platform independent language known as high level language.
- Example: C, C++.

**Compiler:** It reads the whole program at once and produces its object code that is executed by the processor.
- Example: C, C++.

**Interpreter:** It reads one instruction at a time from the source code and produces its object code, that is executed by the processor, before reading the next instruction.
- Example: BASIC. It is also a S/W.
→ **8085** is commonly known as **8085 MPU**.
→ It is based upon **NMOS tech.**
→ It is improved version of **8080**.
→ It is a **8-bit MPU**.

**Bit of MPU.**

→ Data executed by MPU in one w/c cycle is **bit of MPU**.
→ Size of **ALU** (Arithmetic & Logic unit) is also known as **bit of MPU**.

**Bus→** It is **group of (parallel combination) metal wire** that is used for interfacing b/w two different device.

→ All instruction of **8080 MPU** is exactly common in **8085 MPU**. **upward compatibility**

<table>
<thead>
<tr>
<th>MPU</th>
<th>bit of MPU.</th>
<th>Used tech</th>
</tr>
</thead>
<tbody>
<tr>
<td>8080</td>
<td>8 bit</td>
<td>PMOS</td>
</tr>
<tr>
<td>8085</td>
<td>8 bit</td>
<td>NMOS</td>
</tr>
<tr>
<td>8086</td>
<td>8 bit</td>
<td>NMOS</td>
</tr>
<tr>
<td>8087</td>
<td>16 bit</td>
<td>NMOS (14 High density)</td>
</tr>
<tr>
<td>8088</td>
<td>8/16 bit</td>
<td>HEMOS</td>
</tr>
</tbody>
</table>

**8088** is the externally **8-bit** & internally **16 bit** MPU.
80186 → 16 bit
80286 → 16 bit
80386 → 32 bit
80486 → 32 bit
80586 → 64 bit MP = Pentium.

(i) Pentium
(ii) Pentium Pro
(iii) Pentium II
(iv) Pentium III
(v) Pentium IV

Note

\[
\text{MOV BC = 4T,} \\
1T = \frac{1}{4}.
\]

\[
\text{Exe. time = 4T = } \frac{4}{4}.
\]

\[
\text{Speed } \propto \frac{1}{\text{Exe. time }} \propto \frac{1}{\frac{4}{4}}.
\]

\[
\text{Speed } \propto \frac{1}{4}.
\]

\[
\propto \text{ bit of MP}.
\]

Architecture of 8085

\[\rightarrow \text{External Arch.} \rightarrow (\text{Pin diagram})\]

\[\rightarrow \text{Internal Arch.}\]

- External Arch.
  - Pin diagram
  - It is 40 pin I.C.
  - Pin no. is 20 is GND pin.
  - There is no chip enable pin in 8085
All 40 pins are grouped into six:

- Address Pins
- Data Pins
- Control & Status Pin
- Serial Data Transfer Pin
- Externally initiated signal Pin
- Power Supply & Frequency Pin

Note:
1. No. of pins active low: I/O, WR, RD, INTA, RESET2N.
2. Advantage of active low:
   - Power consumption is less
   - Effect of external noise is less
   - Interfacing will be better.
1. No. of pins direction outward = 27.
2. No. of pins direction inward = 21.
3. Address Pins.
   -> 16
   -> Unidirectional 4 outward.
   -> May be no. of memory location that can be interfaced = $2^{16}$ location.
   -> 8
   -> Bidirectional.
   -> Lower 8 pins of address pins can also be used as data pins.
   -> May be memory that can be interface with 8085 = $2^{16} \times 8$ bits.
      = $2^{16}$ Byte. = 65536 Byte.
   $2^{10}$ = kilo. = $2^{6}$ K. Byte.
   $2^{20}$ = mega. = 64 K. Byte.
   $2^{30}$ = Giga.
5. Memory word size -> May be no. of bits that can be stored at particular memory location that is known as word size.
6. $2^{16} \times 8$ bits memory. Find max no. of hardware pins. $\overline{V_{RAM}} = 16 + 8 + V_{cc} + V_{dd} + WR + RD$.
7. $2^n \times d$ = data. $2^n$ or $2^9$. May be may be.
→ Control & status pins direction mixed direction

* Si? By measuring logic on that we can find S out which w/c cycle going on inside the MP.

* \( 0 | \overline{m} = 1 \) = data transfer from \( IO \).
  \( \overline{E} = 0 \) = data transfer from memory.

* \( \overline{WR} = 0 \) = write operation can be performed.
  \( \overline{RD} = 0 \) = read operation can be performed.

\( ^{\#} \) Group of bit:
\( 1111 \) = nibble
\( 11 \) = byte
\( 11110111 \) = byte

\( ^{\circ} \) ALE (address latch enable) →

\( ALE = 1 \) \( A\overline{T_7} - A\overline{T_0} = adol. \)
\( ALE = 0 \) \( A\overline{T_7} - A\overline{T_0} = data. \)
A sequential ckt may be a latch if it is level trigger or follows its characteristic eqn in definite duration.

A seq. ckt may be a flip flop if it is edge triggered or follows its char. eqn in definite time instant.

\[ A \overline{N}_t - A \overline{D}_0 = \text{add.} \]

\[ A \overline{N}_t - A \overline{D}_0 = \text{data.} \]

By the use of ALE lower 8 pins of add. pins multiplexed/demultiplexed as data pins.

Multiplexing & Demultiplexing = Time division multiplexing = TDM.

Serial data transfer pin.

SIP = serial i/p data through this pin by use of RIM inst.

SOP = serial o/p data through this pin by use of SSM inst.

RIM = read interrupt mask.

SSM = set interrupt mask.

Power supply & freq. pins-

\[ V_{cc} = +5 \text{V.} \]

VGP = ground pin.

\[ f = \text{operating freq.} = 3.14 \text{MHz.} \]

To get 2.06MHz clock freq. a 6MHz crystal
Oscillator connected by \( \frac{5}{6} \times 1 \times x_1 \times x_2 \) and use a internally divide by two ckt to get 3 MHz. Clock freq.

+ Clock out by use of this pin clock freq. provide by up. to other interfaced peripheral for better synchronization.

+ Externally initiated signal Pins

- TRAP
- RST 7.5
- RST 6.5
- RST 5.5
- INT

\( S/6, 7, 8 \)

5 pin or 6 pin (Reset)

\( \downarrow \)

H/w interrupt.

HOLD. By use of this pin DMA (direct memory access) like peripheral make request to up for relinquish to its data pins.

- READY

Slower.

\[ \text{Microprocessor} \rightarrow \text{Peripheral} \]

\[ \text{Microprocessor} \rightarrow \text{READY} \]
interface with 8085 μP.

**RESET**: By use of this pin, Reset command gives to μP for reset.

**Note**: As soon as 8085 μP receives reset command, it generates reset out command for reset of other interfaced peripheral.

**INTERNAL ARCHITECTURE.**

→ **ALU** Arithmetic & logic unit.
  → Accumulator.
  → Timing & control ckt.
  → Registers.
  → Register array.

→ Interrupt control register.
→ Temporary Registers.
→ Latches.

**Registers.**

General Purpose Register (user Accessable).

→ B
→ C
→ A
→ E
→ H
→ L

(user accessible)

Special Purpose Register

(user not accessible)

→ Accumulator (A)
→ Flag Reg. or Status Reg (F)
→ Stack Pointer (SP)
→ Program Counter (PC).

**user accessible** = Used in programming.
Note: In 8085, 8, 8-bit register & 2, 16-bit register are used, that are user accessible.

(i) **Accumulator**
- It is a bit special type of Reg.
- In A & L (Arithmetic & logic) operation both the two no.s, then one no. always from accumulator & result will also store in Accumulator.

(ii) **Status or Flag Reg**
- It is a bit special type of Reg.
- It’s definite bits also works as flags.
- In 8085, 5 flags are definite at definite bit of Flag Reg.

\[ \begin{array}{cccccccc}
D_7 & D_6 & D_5 & D_4 & D_3 & D_2 & D_1 & D_0 \\
S & Z & X & AC & X & P & X & CY
\end{array} \]

- Carry Flag
- Parity Flag
- Auxiliary Carry Flag
- Zero Flag
- Sign Flag
- X = don’t care.

Note: Status of flag affect according to the condition generate in Arithmatic & logic operation.
@ Sign flag $S$.

If in the final result of arithmetic & logic operation, $D_7$ bit is 1, then sign flag is set i.e. no. is -ve.

$s = 1$.

→ If $D_7$ bit is 0 or msb = 0, then $s = 0$ or no. is +ve


If in the final result of A & L. operation all bits are zero, then $Z = set (1)$.

Otherwise, $Z = Reset (0)$.

$D_7 \ D_6 \ D_5 \ D_4 \ D_3 \ D_2 \ D_1 \ D_0$

d_7 \ d_6 \ d_5 \ d_4 \ d_3 \ d_2 \ d_1 \ d_0$

$s = 0$

$0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$

$Z = 1$.

$s = 1$

$1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1$

→ There is no relation among the flags, they affect according to the result of A & L. operation respectively.

6. Parity flag $P$.

If in the final result of A & L. operation, total no. of 1's is even, then parity flag will get set $P = 1$.

If total no. of 1's is odd, $P = 0 - Reset$.

$1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ P = 1$

$1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ P = 0$

$0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ P = 1$
8085 µP based upon odd parity system.

\[ p \begin{align*}
&\begin{array}{c}
\downarrow \downarrow \\
D_7 & D_6 & D_5 & D_4 & D_3 & D_2 & D_1 & D_0
\end{array} \\
&\begin{array}{c}
\downarrow \downarrow \\
fe & he & od & odd
\end{array}
\end{align*}\]

\[ \text{carry flag (CY) set in the A + L operation carried generated/discard from D7 bit or from MSB then, } CY = \text{set (1)} , \]

\[ \text{otherwise, } CY = 0 \text{ (reset) .} \]

\[ D_7 \quad D_6 \quad D_5 \quad D_4 \quad D_3 \quad D_2 \quad D_1 \quad D_0 \]

\[ \begin{array}{c}
1 \\
1
\end{array} \]

Notes: In subtraction operation carry flag works as borrow flag.

\[ \text{set in the subtraction operation, borrow is taken at MSB or D7 bit, then borrow flag will get set, } CY = 1 , \]

\[ \text{otherwise, } CY = 0 . \]

\[ D_7 \quad D_6 \quad D_5 \quad D_4 \quad D_3 \quad D_2 \quad D_1 \quad D_0 \]

\[ \begin{array}{c}
0.01010001 \\
11110001
\end{array} \]

\[ CY = 1 . \]
During the A 4 L operation if carry passes from \( D_3 \) - \( D_4 \) or lower nibble to upper nibble then AC flag will set.

\[
\text{AC} = 1
\]

otherwise \( \text{AC} = 0 \).

\[\text{msb} \ D_7 \quad D_6 \quad D_5 \quad D_4 \quad D_3 \quad D_2 \quad D_1 \quad D_0\]

\[
d_7 \quad d_6 \quad d_5 \quad d_4 \quad d_3 \quad d_2 \quad d_1 \quad d_0
\]

**Note**
- Five flag divide in two category.
- (i) Status of flag that affect according final result of A 4 L operation. (S, Z, P)
- (ii) Status of flag that affect according condition generate during the operation, (CY, AC).

Out of five flag, status of u flag can be used by programmer during the program (C, Z, P, CY)

- Status of AC flag not available for program
- Status of AC flag used internally for the BCD adjustment of content of accumulator at the time of execution of DAA instruction

**DAA** → Decimal adjustment of content of accumulator.

BCD = Binary Coded decimal.
- It is not binary no, it is binary code decimal.
- Every digit of bin-decimal code in 4 bit of binary.
(iii) Stack Pointer (SP).

* It is 16 bit special type of reg.
* S.P. always hold the address of top of stack that define in main memory.

![Stack Diagram]

Note:
* S.P. stores data that data is the add. of top of stack.

(A) Program Counter (PC)

* It is 16 bit special type of Reg.
* P.C. will always hold the address of next execute instruction to be fetch.
MOV BC
MVI D 20H
LYI H 10 69H.

Note: If a 16 bit no, feed in memory then low order byte at lower order add. & higher order byte at always at higher order add.

* If a 16 bit no. store in register pair then 8 bit reg. coupled in specific manner.

BC ? = B
DE ? = D
HL ? = H

* Higher order byte of 16 bit no. will always store in higher order register.

B ?
D ?
H ?

& lower order byte always store in lower order register. C ?

(PSW) = Program status word.

→ PSW is 16 bit user defined register.

= A F
↓ accumulator
Flag Register

* PROGRAMMING.

Instruction: It is command that gives to computer to perform a specific task on specific date.

Format of instruction:

Label : operand operand ; description

(Operational code)
Flow Chart of Execution:

- Start
- 16 bit add. given by CPU to fetch instruction
- Fetch instruction from memory
- Decoding of instruction
- Execute

**ISA = Instruction Word Size.**

Total memory location required to feed instruction in memory is called ISA.

- On the basis of ISA inst. are of three types:
  1. 1 Byte inst.
  2. 2 Byte inst.
  3. 3 Byte inst.

**Condition:**

1. If in the instruction Register, Register pair or no operand.
   
   **ISA** = 1 Byte.

   ```
   MOV BC, 2 byte.
   LDAX B, 1 byte.
   NOP.
   ```

2. If in the instruction a bit no., either as an address or as address is given then

   **ISA** = 2 Byte.
MVI B 26H \{ \text{IWS} = 2 \text{ byte.} \}
IN 56H

(3) If in the instruction 16 bit no. either a
aad. or as data is given then
IWS = 3 byte.
LXI B 2016H ? \{ \text{IWS} = 3 \text{ byte.} \}
LDA 5926.

Instruction:

\[
\begin{array}{c}
\text{NAME.} \\
\text{FORMAT.} \\
\text{IWS.} \\
\text{OPERATION.} \\
\text{STATUS OF FLAG.} \\
\text{ADDRESSING MODE.} \\
\text{MACHINE CYCLE.} \\
\text{T-state.} \\
\end{array}
\]

Note: In 8085 74 opcodes (operational code) are available, by that 256 inst. can be def.
but only 256 inst. are available in 8085.

74 opcodes has 11 Groups:

1st Group: 8 bit data transfer inst.

(a) MOV Rd, Rs

Rd = destination Reg \{ A, B, C, D, E, H, L & M \}
Rs = source Reg

- MOV B,C
- MOV D,E

\{ \text{IWS} = 1 \text{ byte.} \}

- \text{operation: when this inst. will execu}
then content of source Reg. (Rs) will pass in Ra Reg. \[ Ra \leftarrow [Rs] \].

\[ \text{MOV A, C} \]
\[ [A] = 36 \text{H} \]
\[ [C] = 6F \text{H} \]
\[ [A] = F6 \text{H} \]
\[ [C] = F6 \text{H} \]

→ M: It is a bit user defined register in main memory, its address is the content of HL register pair.
\[ [H] = 16 \text{H} \]
\[ [L] = 9F \text{H} \]
\[ \text{MOV B} \text{ M.} \]
\[ [B] = 2C \text{H} \]

\[ \text{MVI R, 8 bit data} \]

→ R = A, B, C, D, E, H, L, M.

→ operation: When this instruction will execute 8 bit data given in the instruction will store in R.
\[ [R] \leftarrow \text{8 bit data} \]

\[ \text{MVI D, 69} \text{H}. \]
\[ [D] = 69 \text{H}. \]

→ 2's comp. = 8 R16.
IN 8 bit port add

→ IWS = 2 Byte
→ operation: When this instruction will execute
then 8 bit data available at the port add. is given
in instruction store Pass in Accumulator

Note: In 8085 I/P & O/P port add. is of 8 bit
so max. no. of I/P devices that can be connect = 2^8
& max. no. of O/P = 2^8 = 256
→ max. no. of I/P = 256 & O/P devices that can be connect
= 256 + 256 = 512.

IN A0H
→ [A] = 21H.

OUT 8 bit port add

→ IWS = 2 Byte
→ operation: When this instruction will execute
content of accumulator will available
8 bit port add. given in the instruction

Exit: [A] = F2H

OUT 52H
52 port add

01010001
= seven seg display
All above instructions are data transfer instructions, so status of flag will not affect.

2nd Group:

(i) Machine Control Instruction

(a) NOP no operand:

\[ \text{IWS} = 1 \text{ Byte} \]

\[ \text{operation} \] when this inst. will execute

\[ \text{up. will not perform any task} \]

\[ \text{It is used for creation of delay}. \]

\[ \text{NOP} \equiv 4T = \frac{4}{T} \text{ sec} \]

(b) HLT no operand:

\[ \text{IWS} = 1 \text{ Byte} \]

\[ \text{operation} \] when this inst. will execute

further increment of P.C. will stop.

it means after the execution of HLT inst., execution of program will terminate.
2000H MOV BC
2001H MOV D,E
2002H MVIS A 29H
2004H HLT
2005H PC=2005
PC = PC+1

Addressing Mode
Form of address of data given in the inst.
is known as addressing mode.

(i) Register add. mode -> It address of data
given in the form of Reg.
Ex: MOV BC
    MOV D,E.

(ii) Direct add. mode -> It add. of data
directly given in instruction.
Ex: IN 25H.

(iii) Immediate Add. mode -> It data itself
given in the inst.
Ex: MVIS B 20H.

Note -> It op-code has last chaara. i.e. the
it is immediate addressing mode &
vice versa is not true.

(iv) Register Indirect / Indirect Reg / Indirect add. mode
It add. of data given in the form of
content of Register.
Ex: MOV B M.
(i) Add instruction

(a) **ADD R**

- \( R = A, B, C, D, E, H, L, \& M \).
- **Operation:** When this inst. will execute, content of \( R \) will get added to \( [A] \) & final result will store in \( [A] \).

\[
[A] = [A] + [R].
\]

**Ex:**

\[
[A] = 29H, \quad [B] = 56H.
\]

**ADD B:**

\[
[A] = 0010 1001 \quad \text{[A] = 0111 1111 = 7FH}
\]

\[
[A] = 0111 1111 = 7FH
\]

\[
[R = [B] = 56H.]
\]

**Ex:**

\[
[H] = 30H, \quad [L] = F6H, \quad [A] = 10H.
\]

**ADD M**

\[
[A] = 0001 0000
\]

\[
[M] = 0001 0010
\]

\[
[R \# I = \text{Register add mode}.]
\]

\[
R = [A] = \text{Indirect} \quad \text{[A]} = 10H.
\]
\( ADR \): 8 bit data

\[ \Rightarrow 2{}^wS \equiv 2 \text{ Byte}. \]

\( \text{Operation}: \) When this instruction will execute, 8bit data will be added in content of acc. and result will store in \([A]\).

\[ [A] \leftarrow [A] + 8 \text{ bit data} \]

\( \Rightarrow \text{Add mode} = \text{Immediate} \).

\( SUB \ R \)

\[ \Rightarrow 2{}^wS \equiv 1 \text{ Byte} \]

\( R \equiv A, B, C, D, E, H, L, M. \)

\( \text{Operation}: \) \([R]\) will get subtracted from content of accumulator and result will store in Accumulator.

\[ [A] \leftarrow [A] - [R] \]

\( \Rightarrow \text{Add mode} \rightarrow R \neq M \text{ Register add} \)

\[ R = M \text{ Indirect Reg. add} \]

\( SUI...8 \text{ bit data} \)

\[ \Rightarrow 2{}^wS \equiv 2 \text{ Byte} \]

\( \text{Operation}: \) 8 bit data given in this instruction will get subtracted from content of \([A]\) and result will store in \([A]\).

\[ [A] \leftarrow [A] - 8 \text{ bit data} \]

\( \Rightarrow \text{Add. mode} \rightarrow \text{Immediate} \).

\( SFR \ R \)

\[ \Rightarrow 2{}^wS \equiv 1 \text{ Byte}. \]

\[ R = A, B, C, D, E, H, L, M. \]
Operation: content of R increased by 1

\[ [R] \leftarrow [R] + \text{1LSB} \]

→ Add. mode

\( R \neq M \rightarrow \text{Register add. mode} \)
\( R = M \rightarrow \text{Indirect reg. add.} \)

Ext: \( \text{MVI B 2Ah} \)

\( [B] = 00101001 \)

\( [B] = 00101001 \)

\( \overline{00101010} \)

\( [8] = 9Ah \)

⑦ \( \text{DCR R} \)

→ 2US & 1 byte

→ \( R = A, B, C, D, E, H, L, M \)

Operation: content of \( R \) decreased by 1

Result will store in \( R' \)

\[ [R'] \leftarrow [R] - \text{1LSB} \]

→ Add. mode

\( R \neq M \rightarrow \text{Register add. mode} \)
\( R = M \rightarrow \text{Indirect add. mode} \)

Note:
① ADD, ADD, SUB, SUI will affect status of all flag.
② INR & DCR will affect only status of four flags \([S, Z, AC, P]\).

→ INR & DCR will not affect status of carry flag

\( [B] = FF \quad E = 11111111 \)

\( [B] = 00000000 \)

\( [B] = 00000000 \)
(a) \text{ANA R}

\rightarrow \text{2ws = 1 byte.}

\rightarrow \text{R = A, B, C, D, E, H, L, M.}

\rightarrow \text{operation: content of R will get AND operation with content of \text{[A]}, bit by bit 4 result will store in A, C.}

\begin{align*}
\text{Ext} & \quad \text{MVI A 56H} \rightarrow \text{[A] = 56H} \\
\text{MVI D 29H} & \rightarrow \text{[D] = 29H} \\
\text{ANA D} & \\
\text{[A]} & = 01010110 \\
\text{[D]} & = 00101001 \\
\text{[A]} & = 00000000
\end{align*}

\rightarrow \text{Add mode: } A \neq M \rightarrow \text{Reg. Add. mode.}

\rightarrow A = M \rightarrow \text{Indirect Reg. add.}

(b) \text{ANI 8 bit data}

\rightarrow \text{1ws = 2 byte.}

\rightarrow \text{operation: 8 bit data given in inst. will get AND operation with content of \text{[A]} bit by bit 4 result will store in \text{[A]}.}

\rightarrow \text{Add. mode: Immediate add. mode.}
(a) **OR operation**

\[
\begin{array}{c}
A \\
B \\
\rightarrow Y = A + B.
\end{array}
\]

\[
\begin{array}{c|c|c}
A & B & Y \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0
\end{array}
\]

(a) ORA R

\[\text{Iws} = 1 \text{ Byte}.\]

\[R = A, B, C, D, E, H, L \leq M.\]

Operation: [R] will get OR operation with content of [A]. Bit by bit 4 result will store in [A].

Add. mode: \[R \neq M \rightarrow \text{Register Add. Mode}\]

\[R = M \rightarrow \text{Indirect Reg. Mode},\]

(b) **ORI 8 bit data**

\[\text{Iws} = 2 \text{ Byte}.\]

Operation: 8 bit data will get OR operation with content of [A]. Bit by bit 4 result will store in [A].

Add. mode: Immediate Addressing.

(c) **EXOR operation**

\[
\begin{array}{c}
A \\
B \\
\rightarrow Y = A \oplus B = \overline{A}B + A\overline{B}.
\end{array}
\]

\[
\begin{array}{c|c|c|c}
A & B & Y \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0
\end{array}
\]
XRA R
→ 2ws = 1 Byte
→ R = A, B, C, D, E, H, L, M
→ Operation: Content of R will get EX-OR operation with [A]. Bit by bit 4 result will store in [A].
→ Add. mode: R ≠ M → Reg. add. mode.
   R = M → Indirect add. mode.

Exp:
[A] = 26H
[D] = 71H

XRA D.

XRA 8 bit data
→ 2ws = 2 Byte.
→ Operation: 8 bit data in instruction will get EX-OR operation with [A]. Bit by bit 4 result will store in [A].
→ Add. mode: Immediate Add. mode.

CMA no operand
→ Complement of Accumulator.
Note: \( \frac{\text{0}}{\text{1}} - \frac{\text{1}}{\text{0}} = \overline{A} \).
→ 2ws = 1 Byte.
→ Operation: Content of [A] will get complement bit by bit 4 result will store in [A].
→ Add. mode: Implicit.
\[ \begin{bmatrix} A \end{bmatrix} = 00101000 \]
\[ \begin{bmatrix} A \end{bmatrix} = 11010111 = \text{H} \]

**Note:**

<table>
<thead>
<tr>
<th>S</th>
<th>Z</th>
<th>AC</th>
<th>P.</th>
<th>CY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANA</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>0</td>
</tr>
<tr>
<td>ANI</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>0</td>
</tr>
<tr>
<td>ORA</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>0</td>
</tr>
<tr>
<td>ORI</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>0</td>
</tr>
<tr>
<td>XRA</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>0</td>
</tr>
<tr>
<td>XRI</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>0</td>
</tr>
<tr>
<td>CMA</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

\(\checkmark\) = According to the result.

\(\times\) = Not affect.

1 = set.

0 = reset.

1. CMA will not affect status of any flag.
2. AND, OR, EX-OR will always reset the Carry flag.
3. AND operation always set the AC flag.
4. OR & EX-OR operation will always reset the AC flag.
5. AND, OR, EX-OR affect other flag as per result.

**Question:**

If \(0001\) MOV A, 23H

\(0002\) MOV B, 32H

\(0004\) XRA B

\(0005\) ADD 88H

\(0007\) HLT

After the execution of HLT inst. value of PC = 0, B = 9, PSW = 2.

**Sol.** PC = FFF0H.
\[ [A] = 23H \]

\[ [B] = 32H \]

\[ [A] = 00100011 \\
[B] = 00110010 \]

\[ [A] = 00010001, [B] = 32H \]

\[ [A] = 88H = 10001000 \]

\[ [A] = 10011001 \]

\[ PSW = A \rightarrow \text{flag reg.} \]

\[ 9984H \]

\[ \text{MVI A 2AH.} \]

\[ \text{ADH A} \]

\[ \text{ORI AF} \]

\[ \text{INR A} \]

\[ \text{CMA} \]

\[ \text{A} \]

\[ \text{After the execution of program} \]

\[ \text{status of flags} \]

\[ \text{flags:} \]

\[ 011110 \]
Write the one instruction that make content of ACC 00H regardless of its previous status.

Solt
1. MVIA A 00H.
2. SUB A
3. ANI 00H.
4. XRA A → [A] = \begin{array}{c}
\text{00000000} \\
\text{10111100}
\end{array}

After the arithmetic operation b/w the two no., status of flag reg = B3H. then after the arithmetic operation content of [A] maybe

- 75H
- 65H
- 98H
- 86H

INSTRUCTION CYCLE, M/C CYCLE & T-STATE

i) Instruction Cycle: Total time required for execution to complete execution of one instruction is known as instruction cycle.

→ Every inst. cycle is the combination of one or more than one w/c cycle.
My M/C cycle: During the execution of instructions of different types of tasks performed, it is known as M/C cycle.

- In 8085 CPU, six types of M/C cycle are defined.

2. Opcode fetch M/C cycle (F/S) or machine code fetch M/C cycle

Time required for execution to fetch op-code (M/C code) regarding a instruction from memory is known as opcode fetch M/C cycle.

- MOV BC → 1 byte = M/C code.
- CMP → 1 byte = op-code.

Note:
- It is first or only first M/C cycle of every instruction.
- Opcode fetch M/C cycle is the special case of memory read operation.

- F = 4T
- S = 6T

\[ T_{-\text{state}} = \frac{1}{f_{\text{oper.spec}}} \]

- M/C code regarding, op-code fetch M/C that req. 6T state for M/C code →

\{ CALL, RET
  RESTART
  INX, DEX
  SPHL, PCHL
  PUSH \}

"CRISP"
6. **Memory Read M-Cy (R)**
   
   Total time required for execution to read 8-bit data from memory.
   
   \[ R = 3T \]

7. **Memory Write M-Cy (w)**
   
   Total time required for execution to store 8-bit data in memory.
   
   \[ W = 3T \]

8. **Input Read M-Cy (I)**
   
   Total time required for execution to read 8-bit data from I/P port.
   
   \[ I = 3T \]

9. **O/I Write M-Cy (O)**
   
   Total time required for execution to make available 8-bit data at O/I port.
   
   \[ O = 3T \]

10. **Bus Idle M-Cy (B)**
    
    Duration for which buses will be in idle condition, during the execution of some specific inst.
    
    \[ B = 3T \]

**Note:** This w/c cycle is required only indad instruction.
<table>
<thead>
<tr>
<th>op code fetch m.c.y.</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory read m.c.y. (R)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Memory write m.c.y. (W)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Input read m.c.y. (I)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>out write m.c.y. (O)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

\[ S_1, S_0 \]

- 1 1  
- 1 0  
- 0 1  

\[ S_1, S_0 \]

- 0 0  

\[ \text{bus idle m.c.y.} \]

**Branch operation**

- Loop is the special case of branch operation.

- In 8085, there exist 3 instrs. defined for branch operation:
  1. **JMP**
  2. **CALL**
  3. **RESTART**

- **JMP**→ It is of two types:
  1. unconditional jmp inst
  2. conditional jmp inst

**Date**

0/12/11
(i) unconditional JMP

* JMP 16 bit add.
  \[\text{Iws} = 3 \text{ Byte}\]

\[\text{operation: when \ inst. \ will \ execute, \ control/execute}\]
\[\text{action \ will \ JMP \ at \ 16 \ bit \ add \ (vector/location)}\]
\[\text{given \ in \ the \ instruction.}\]

\[
\begin{array}{c}
\text{Ex}^1 : 1000H \ \text{MOV BC} \\
1001 : \text{MVI} \ P, 27H \\
1003 : \text{JMP} \ 2089H \\
1005 : \text{HIT} \\
1006 : \text{RES} \\
1007 : \text{HIR} \\
1008 : \text{RET} - 2089
\end{array}
\]

\[\Rightarrow \text{Add. Mode} \rightarrow \text{Immediate Add. Mode.}\]

\[\Rightarrow \text{it \ is \ data \ transfer \ instr., \ so \ status}\]
\[\text{of \ flag \ will \ not \ affect.}\]

(ii) conditional JMP instruction

* \text{JC} 16 \ bit \ add., \text{Jmp} \ inst. \ will \ execute \ if \ CY flag = 0

\text{JNC} \quad 16 \ bit \ add., \quad \text{Z} = 1

\text{JZ} \quad 16 \ bit \ add., \quad \text{Z} = 0

\text{JNZ} \quad 16 \ bit \ add., \quad \text{S} = 0

\text{JP} \quad 16 \ bit \ add., \quad \text{P} = 1

\text{JM} \quad 16 \ bit \ add., \quad \text{P} = 0

\text{JPE} \quad 16 \ bit \ add., \quad \text{P} = 0

\text{JPO} \quad 16 \ bit \ add., \quad \text{P} = 0

\[\Rightarrow \text{Iws} = 3 \text{ Byte.}\]
If condition satisfied then instruction will execute.

Otherwise, skip it.

1005 JPE 16 bit add.

→ Add. Mode → Immediate Add. mode.

→ status of flag will not affect.

* Inst.*

<table>
<thead>
<tr>
<th>(i) MOV BC</th>
<th>M.Cy.</th>
<th>T-state</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ii) MVI D, 23H</td>
<td>F</td>
<td>4T</td>
</tr>
<tr>
<td>(iii) MOV B M</td>
<td>FR</td>
<td>T T</td>
</tr>
<tr>
<td>(iv) MOV M, D</td>
<td>FW</td>
<td>T T</td>
</tr>
<tr>
<td>(v) NOP</td>
<td>F</td>
<td>4T</td>
</tr>
<tr>
<td>(vi) IN 2FH</td>
<td>FR1</td>
<td>10T</td>
</tr>
<tr>
<td>(vii) OUT 25H</td>
<td></td>
<td>10T</td>
</tr>
</tbody>
</table>

1000; 5N
F, 1001 2F Port add.

Controller—goes to this E.

(FR2)

3000 OUT.
3001 25

(viii) ADD B
1000H ADD B

F, 4T.

© Wiki Engineering
(i) 16 bit data transfer instruction

\[ \text{LXI } Rp, \ 16 \text{ bit data}. \]

\[ \text{IWs = 3 Byte.} \]

\[ Rp = BC \iff B \]

\[ NE \iff D. \]

\[ HL \iff H. \]

\[ \text{operation: 16 bit data will store in Rp.} \]

\[ [Rp] \leftarrow 16 \text{ bit data}. \]

\[ \text{Ext } \text{LXI} \ D, \ 2053H \]

\[ [D] = 20H \]

\[ [E] = 59H. \]

\[ \text{Add. mode: Immediate Add. mode.} \]

\[ \text{m cy} \iff FRR \iff 107. \]

(b) \[ \text{LDA, 16 bit address} \]

\[ \text{Note: store memory} \]

\[ \text{load.} \]

\[ \text{IWs = 3 Byte.} \]

\[ \text{operation: } [A] \leftarrow [16 \text{ bit add}] \]

\[ \text{16 bit data available in the memory} \]

\[ \text{a bit data available at 16 bit add, that is given in instruction will load in accumulator,} \]

\[ \text{memory.} \]

\[ \text{Ext } \text{LDA} \ 5029H \]

\[ [A] = [5029] \]

\[ [A] = 2DH \]

\[ 2DH \iff 5029 \]
0) **STA 16 bit add**

- **Operation**: \([A] \rightarrow [16\text{ bit add}]\)
- Content of \([A]\) will store at the 16 bit add in memory that is given in the instruction.

1) **Add. mode = Direct Addl. Mode**

\[ mcy = FRRW \rightarrow 13T \]

- \([A] = 2015\]
- \([x] = 3083\]
- \(LDA 2015\)
- \(STA 3083\)

2) **LDA X Rp**

- **Operation**: 8 bit data available at the 16 bit add. in memory, that is given in the form of content of Rp, will load in ACC.

\([BC] = 500FH\]

- **LDA X R**

\([A] = 32H\]

\([mcy] = FR\]
(C) \text{STAX Rp}. 

\[ \text{Swc} = 1 \text{ Byte}. \]

\text{operation: content of accumulator will store at}
\text{16 bit addr. in memory, that is given in the form of content of Rp.}

\[ [A] \rightarrow [R_p] \]

\[ [A] = 12_{16} \]
\[ L_1 50 80_{16} \]
\[ \text{STA X A} \]

\[ \rightarrow \text{PC} - 1 \rightarrow \text{FW} \]

\text{Note: for LIA X, STAX,}
\text{Rp = BC? DE?}

\text{Note: data transfer inst., so status of flag will not affect.}

6th Group.
(i) 16 bit arithmetic instruction

\( \text{(a) INX Rp.} \)

\[ \rightarrow Rp = EC \quad Z_p \]
\[ ME \quad H_l = H. \]

\[ \rightarrow \text{Swc} = 1 \text{ Byte}. \]

\text{operation: content of Rp will increase by one if}
\text{result will store in Rp.}
\text{Ro} \leftarrow Rp + 1_{16} 8_{8}.

\text{Accel. Mode: Registry addl.}
\text{M-OU} \rightarrow S.
b) $\text{DEC } \text{R}_p$

$\rightarrow \text{I ws} = 1 \text{ Byte}$

$\rightarrow \text{R}_p = \text{BC} \text{ DE} \text{ HL}$

$\rightarrow$ operation: content of $\text{R}_p$ decrease by one & result will store in $\text{R}_p$.

$[\text{R}_p] \leftarrow [\text{R}_p] + 1$

$\rightarrow$ Add. mode $= \text{Reg.}$

$\rightarrow ncy = 0$ $s$.

\textit{Note!}: INX $\&$ DEC will not affect status of any flag.

7th Group: \[ \text{V. 2up}^3 \]

(i) 8 bit logical rotational inst.

$\rightarrow$ These inst. execute on the basis of content of accumulator.

$\rightarrow$ When these inst. execute content of $[A]$ will shift by 1 bit, either left or right as per inst.

\begin{itemize}
  \item \textit{RLC} no operand, content accumulator left by 1 bit without CY.
  \item \textit{RAL} no operand, " " " " " with CY.
  \item \textit{RRC} no operand, " " " " " right without CY.
  \item \textit{RAR} no operand, " " " " " with CY.
\end{itemize}

$\rightarrow$ I ws $= 1$ Byte.

$\rightarrow$ operation: content of $[A]$ rotate left or right by 1 bit as per instruction.

$\rightarrow$ Add. mode $= \text{Implicit Add. Mode}$.
(i) RLC

(iii) RAL

(iii) RRC

(iv) RAR

→ mi-cy: Fetch (F).

Note: Rotational inst. affect only one flag that is carry flag.
LXI H, 2000H
LDA 2001H
XRA M
MOV E, A
MVI D, 20
LDAX D
OUT 01.

After the execution of this program display
at I/O port.

\[ \text{Sol' } \]

\[
\begin{align*}
[H] &= 20 \text{H}, [L] = 00 \text{H} \\
[A] &= 02 \text{H} \\
[M] &= 0000000000000000 \\
[A] &= 0000000010 \\
[E] &= 02 \text{H}, [A] = 02 \text{H}. \\
[D] &= 20 \text{H}.
\end{align*}
\]

\[
\begin{align*}
\text{display at I/O} &= [A] = 02 \text{H}.
\end{align*}
\]

\[ \text{Sol' } \]

\[
\begin{align*}
\text{1000H: MVI A A1H} & \\
1002 \text{H: LXH 1007H} & \\
1005 \text{H: SUB M} & \\
1006 \text{H: OUT 05H} & \\
1008 \text{H: HLT}. & \\
\end{align*}
\]

After the execution of above program display
at I/O port.

(a) A1H \( \checkmark \) true
(b) 05H \( \checkmark \) can't determine
Q.\textsuperscript{3} \quad MVI A, FOH
ORA A
Loop: INR A
JNC loop
HLT.

\textbf{Solution:} \quad [A] = FO = \begin{array}{c} 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \end{array} 
\quad \rightarrow \quad \text{ORA} \quad \begin{array}{c} 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \end{array} 
\quad \rightarrow \quad \text{INRA} \quad [A] = \begin{array}{c} 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \end{array}
\quad \rightarrow \quad \text{2 infinite times loop will execute.}

\textbf{8th Group:} 
\begin{enumerate}
\item A bit logical compare instruction.
\item CMP R.
\rightarrow \text{Word size} = A, B, C, D, E, H, L + M.
\rightarrow \text{Word size} = 1 \text{ Byte.}
\rightarrow \text{Operation} = \text{content of R will compare with content of accumulator 4 status of flag -- after accordingly.}
\rightarrow \text{Note:} \quad \text{it is nothing but SUB} (A-R) \quad \text{status of flag will affect according the result of SUB} (A-R), \quad \text{but content of A \& R will not change.}
\rightarrow \text{Add. mode:} \quad R \oplus M = \text{Reg. add. mode} = F
\rightarrow \text{R} \oplus M = \text{Indirect add. mode} = F \text{R.
0. **CPI**: 8-bit data.

- **3ws**: 2 Byte.
- **Operation**: 8-bit data compare with content of [A] & according to the result, status of [A] will affect but content of [A] will not change.
- **Add. mode**: Immediate Add.

- m, Cy, R.

- All flag will affect.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cy</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A] &gt; 8 bit data</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>[A] &lt; 8 bit data</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>[A] = 8 bit data</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

⇒ Rest of flag according to the result of SUB(A - 8-bit data).

**Note**: **MIP, CPI** will affect all flag.

- **Sign no. system**:
  - Direct sign mag repr.
  - 1's complement
  - 2's complement

⇒ In all above one concept is common.
MSB always show the sign of no.

\[ \text{MSB} = 1 \equiv -\text{ve}. \]
\[ \text{MSB} = 0 \equiv +\text{ve}. \]

1. Direct sign mag. repr.

- Only for sign. \( (n-1) \text{ bit} \)

\[ \text{Example: } +5 \quad 0101 \]
\[ -5 \quad 1111 \]

2. 1's compliment reprs.

3. 2's compliment reprs.

In that two concept is common.

(i) The no. always written as straight binary

with \( \text{MSB} = 0 \), either in 1's comp. or 2's comp.

Ex's: Given no. in 1's comp. Find it's decimal eq.

\[ 0100 = +4 \]
\[ 0110 = +6 \]

Given no. in 2's comp. Find it's decimal eq.

\[ 0100 = +4 \]
\[ 0110 = +6 \]

(ii) But if we take 1's compliment or 2's compliment of any binary no. then its sign will be change but magnitude will be same.

Ex: Write -6 in 1's compliment.

\[ +6 = 0110 \quad 1's \text{ comp.} \quad 1001 \equiv -6. \]

In 2's complement.

\[ +6 = 0110 \quad 2's \text{ comp.} \quad 1010 \equiv -6. \]
1. Given no. is in 1's comp. Find its decimal eq.
   \[ 1010 = -(+5) = -5. \]

2. Given no. in 2's comp. Find its decimal eq.
   \[ 1010 \text{ 2's comp.} \xrightarrow{\text{add 1}} 0110 = -(6). \]

Experiment

\[ [A] = 37 \quad H \quad \Rightarrow \quad [A] = 00110111 \]
\[ [B] = 25 \quad \Rightarrow \quad [B] = 00100101 \]
\[ [A] - [B]. \]

\[ A + (-B) \]
\[ A + [2's \ B]. \]

\[ [A] = 00110111 \quad \Rightarrow \quad [A] = 00110 \quad \Rightarrow \quad S = 0, Z = 0, P = 1, AC = 0. \]

2's comp. of \[ B \]

\[ 2's [B] = +11010011 \]
\[ \times 0001 \quad 0010 \]

\[ [A] = 25 \quad H \]
\[ [B] = 37 \quad H \]

\[ 2's \text{ comp.} \]

\[ [A] = 00100101 \]
\[ [B] = 00110111 \]
\[ \times 1110 \quad 1110 \]

\[ [A] - [B]. \]

\[ A + (-B) \]
\[ A + [2's \ B]. \]

\[ [A] = 00100101 \quad \Rightarrow \quad [A] = 0010 \quad \Rightarrow \quad S = 1, Z = 0, P = 1, AC = 0. \]
1st method

\[ (A-R) \text{ direct. } \frac{CY}{SP} \text{ use this. } \]

\[ A + (2^5 \text{ or } R) \rightarrow AC. \text{ Use this.} \]

2nd method

\[ A - B = A + 2^5B. \]

\[ \frac{CY}{SP} \]

Example:

\[ \text{MVI A 29} \]
\[ \text{ORA A.} \]
\[ \text{RLC} \]
\[ \text{RAL} \]
\[ \text{HLT.} \]

\[ \text{Solution: } \]

\[ [A] = 29 = 00101001 \]

\[ \rightarrow \text{ OR A.} \]

\[ \begin{array}{c}
0010 & 1001 \\
\hline
0010 & 0100 \\
\end{array} \rightarrow CY = 0. \]

\[ [A] = 00101001 \]

\[ \text{RLC } [A] \rightarrow CY = 0 \text{ without carry.} \]

\[ [A] = \text{52H, } CY = 0. \]

\[ \text{RAL } [A] \rightarrow CY = 0 \text{ with carry.} \]

\[ [A] = 01010010 \]

\[ [A] = 10100100 \]

\[ \rightarrow [A] = A4, \text{ CY} = 0. \]
→ Stack is the group of continuous memory location in main memory that is used for temporary storage of information during the execution of main program.

→ At a time two byte data can store at the top of stack, or two byte data can retrieve from the top of stack.

→ If two byte data stored at the top of stack, it goes upward with numerically decreasing order of its address.

→ If two byte retrieve from the top of stack, it goes downward with numerically increasing of its address.

→ In 8085, two instructions are destined to store or retrieve the two byte data from the top of stack, i.e., `PUSH`, `POP`.

  - `PUSH` = store
  - `POP` = Retrieve.

→ Stack can be initialize in main memory by inst.

  - `LXI SP 16 bit data`
  - `SP = 16 bit data`
During the execution of CALL subroutine, address of next instruction will store at the top of stack.

→ Stack works on the principle of LIFO = Last in first out. (3)

*(1)* PUSH Rp.

→ Rp = BC ≠ B
→ DE ≠ D
→ HL ≠ H

→ 2ws ⇒ 1 byte.

→ Operation: When this inst. will execute, the content of register pair will store at the top of stack:

```
LXI SP, 2019H → SP = 2019H
LXI B, 5686H → [BC] = 5686H
PUSH B.
ADD A, C.
```

→ After the execution of PUSH instruction, content of S.P. will decrease by two.

→ Add. mode = Reg. add. mode ← source data PUSH

→ M - CY. = 1 0 0 5 0 ← 100s PUSH A

→ SWW.

Note: ① It is data transfer instruction. So status of flag will not affect.

② There is no conditional PUSH inst.
(ii) \( \text{POP Rp} \)

\[ \rightarrow \text{Rp} = \text{BC} O \text{DE} \]

\[ \text{HL} \quad \text{PSW} \rightarrow \text{Acc. flag} \]

\[ \rightarrow \text{2ws} = 1 \text{ byte} \]

\[ \rightarrow \text{operation: when this instr. will execute, content of two byte data at register will retrieve from the top of stack & store in Rp}. \]

\[ \text{Eg:} \]

\[ \text{POP D} \]

\[ \begin{array}{c|c|c}
\text{D} & \text{E} & \text{F} \\
29 \text{H} & 56 \text{H} & 15 \text{H} \\
2018 \text{H} & 2017 \text{H} & \text{SP - 2016 H} \\
\end{array} \]

\[ p = 56 \text{H} \quad \text{?} \quad e = 15 \text{H} \quad \text{?} \quad \text{SP = 2018 H} \]

\[ \rightarrow \text{After the execution of POP instr., content of s.p. will increase by two.} \]

\[ \rightarrow \text{Add. mode: POP Rp} \quad \text{source data = Indirect + Reg. add. work} \]

\[ \text{destination data = Register + add. mode} \]

\[ \rightarrow m-cy. = \text{F RR} \]

**Note:** \( \text{POP} \) is data transfer inst., so status of flag will not affect, but in case of \( \text{POP PSW} \), status of flag may be affected. Indirectly.

\[ \text{PSW, } \text{EAF} = 16 \text{H} \]

\[ \text{F1} = 2 \times 2 \times 2 \times 2 \times \text{F} \times \text{F} \]

\[ = 01010101 \]

\[ \text{SP = 2016H} \]

\[ \text{SP + 2} \]

\[ \text{KSP = 2018H} \]
Subroutine:

- It is a set of instructions written separately from the main program regarding the task that occur frequently in the main program.

- To develop subroutine in 8085 two instructions are defined:
  (i) CALL
  (ii) RET

(i) CALL inst:

- It is of two types:
  a) Unconditional CALL inst.
  b) Conditional CALL inst.

a) Unconditional CALL inst:

- CALL, 16 bit add

- I ws = 3 Byte.

- Operation: When this instruction will execute, control/execution will transfer at 16 bit vector location given in the instruction but before transfer, address of next instruction will store at the top of stack.

```
1006 MOV BC
1007 CALL 4219
1008 INVI 25H
CALL = PUSH + JP
```
Add. mode : Immediate add. mode

→ M-Cy : 100F \[ \text{CAL} \]
1008 \[ \frac{15}{15} \]
1009 \[ \frac{42}{42} \]
100A
PC

Example → 1000H : LXI SP 27 FFH

After the execution value of

SP & HL Pair,

SP \[ \frac{27}{27} \] HL
① 27 F7 1006
② 27 F6 1006
③ 27 FF 1006
④ 27 FF 1006

→ SP = 27 FFH

[Hl] = 1006

Conditional CAL inst.:

CC 16 bit add.; CALL inst. will execute if Cy = 1.
CNE \[ \frac{\text{not equal}}{\neq} \]
CZ \[ \frac{\text{Z}}{=} \]
CMZ \[ \frac{\text{not equal}}{\neq} \]
CP
CM
CPE
CPO

→ 2wcy = 3 byte.

Operation: if condition satisfy then CALL inst.

will execute otherwise skip it.

Add. mode : Immediate Add. mode.
Note: CALL inst. is data transfer inst., no status of flag will not affect.

(c)th group (Regarding)

16 bit arithmetic inst.

(iii) DAD Rp.
   \[ 2ws = 1 \text{ Byte.} \]
   \[ Rp = \{BCD\} \]
   \[ DEF \]
   \[ HL \]

- Operation: content of Rp will get added in HL pair.
  & result will stored in HL.
  \[ [HL] \leftarrow [HL] + [Rp] \]

- Add. mode = Reg. add. mode.

\[ \boxed{\text{M-cycle} = \text{FBF}} \]

(i) RET.

It is of two type:
- (i) Unconditional RET.
- (i) Conditional RET.

Note: It is the last inst. of every subroutine

Unconditional RET
RETI no operand

- SW = 1 Byte
- Operation: When this inst. will execute, two byte (latest stack) data retrieve from the top of stack, store in P.C. & next inst. will fetch from there.

main program: 2015 subroutine.

PC = SP
PC = [1009] 2. Pop

1006 CALL 2015

1009 <--- JMP

RET = POP + JMP.

  - Reg. indirect Add. mode, [S: P]

- M-CY: SRR

Note: This is data transfer inst. So status of flag will not affect.

After, the execution of RET inst. content of S: P. will increase by two.

Example:
LXI SP, 2715
CALL 3000H

3000H
LXI H, 1015H
PUSH B
PUSH PSW
PUSH H
LXI SP, 3C F4H
POP H
POP PSW
POP B
RET.

After the execution of this program content of S: P will be.
Conditional RET inst #7,

RC  no operand, RET instr will execute if CY=1.
RNC  
RZ  
RNL  
RP  
RIM  
RPE  
RPO  

→ 2's = 1 byte.
→ Operation: If condition satisfied then RET will execute, otherwise skip it.

→ Addr. mode: → Indirect Reg. mode → Implicit Addr. mode.

→ M-CY → cond. satisfied → SRR
cond. not satisfied → S/R

Note: #8. It is data transfer inst, so status of flag will not affect.
It is just like one byte CALL inst.
This inst. is used when execution trap among the interrupts.
It is like an s/w interrupt.

- RSTN no operand
- 8WS = 1 byte.
- \( N = 0, 1, 2, 3, 4, 5, 6, 7 \).

Operation: When this inst. will execute, control (execution) will jump at the specific vector location of memory page no: 000H.

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Vector location</th>
</tr>
</thead>
<tbody>
<tr>
<td>RST 0</td>
<td>00 00 00 0H</td>
</tr>
<tr>
<td>RST 1</td>
<td>00 00 00 1H</td>
</tr>
<tr>
<td>RST 2</td>
<td>00 00 00 2H</td>
</tr>
<tr>
<td>RST 3</td>
<td>00 00 00 3H</td>
</tr>
<tr>
<td>RST 4</td>
<td>00 00 00 4H</td>
</tr>
<tr>
<td>RST 5</td>
<td>00 00 00 5H</td>
</tr>
<tr>
<td>RST 6</td>
<td>00 00 00 6H</td>
</tr>
<tr>
<td>RST 7</td>
<td>00 00 00 7H</td>
</tr>
</tbody>
</table>

Note: \( xx \) is the hexadecimal conv. of \( XY \times 8 \).

\[
\begin{array}{c|c|c}
 16 & 16 & 16 \\
 1 & \hline
 \end{array}
\]

\( (16)_{16} = (16)_{10} \).

Add. mode = Implicit add. mode.
Add. \( c_y = 5 \).
After the completion of the execution of subroutine when control come back in main program, at this time value of \([A]=9\).

\[
\text{sol}^1c: \quad [A] = 00H = 01H.
\]

RET will pop from top of stack.

\[
[A] = 01 + 1 = 02H.
\]

*Instruction Related to HL Pair.*

(1) LHL $16$ bit add.

\[ \rightarrow 2\text{ws} = 3 \text{ Byte} \]

- Operation's when this inst. will execute, 8 bit data available at the memory location th is given in the inst. load in L & next 8 bit data at next memory location will load in H.

\[
[L] \leftarrow [16 \text{ bit add}] \\
[H] \leftarrow [16 \text{ bit add}+1]
\]

\[\text{Ex}^1: \quad \text{LHL D 2019} \]

\[
[L] \leftarrow [2019] \Rightarrow [1] = 2H \]

\[
[H] \leftarrow [E0H] \Rightarrow [11] = 60H.
\]
→ Add. Mode = Direct add. mode.

→ \textbf{w-c cycle:} F R R R R \rightarrow 16 T.

100H \text{ LHLD} 2019.

\text{\textit{f}} \rightarrow 1000 \text{ LHLD}.
1001 \rightarrow \text{FCH}.
1002 \rightarrow \text{20H}.

\text{(ii) LHLD} 16 \text{ bit add}

→ 2ws \equiv 3 \text{ byte}.

→ Operation: When this inst. will execute content of 'L' will store at the 16 bit memory address that is given in the inst. & content of 'H' will store at next memory address.

\begin{align*}
[L] & \rightarrow [16 \text{ bit add.}] \\
[H] & \rightarrow [16 \text{ bit add+1}].
\end{align*}

→ \textbf{w-c cycle:} F R R W W \rightarrow 16 T.

1006 \text{ LHLD} 2015

\text{\textit{f}} \rightarrow 1000 \text{ LHLD}.
1007 \rightarrow \text{FCH}.
1008 \rightarrow \text{20H}.

\rightarrow \text{Add. Mode = Direct add. mode.}

\text{(iii) XCHG no operand.}

→ 2ws \equiv 1 \text{ byte}.

→ Operation: When this inst. will execute content of HL pair exchange with content of DE pair.
(iv) XTHL no operand.

\[ \text{XTHL } \rightarrow 2\text{WS } = 1 \text{Byte}. \]

Operation: When this inst. will execute content of HL pair will exchange from top of stack.

For intermediate storage, memory locations are used

\[ \text{XTHL } \equiv \text{PUSH} + \text{POP}. \]

\[ M-\text{cy} \rightarrow \text{FHLHlRR}. \]

\[ \text{Add. Mode} \rightarrow \text{Implicit} \rightarrow \text{Register.} \rightarrow \text{Indirect Reg.} \]

(v) SPHL no operand.

\[ 2\text{WS } = 1 \text{Byte}. \]

Operation: When this inst. will execute content of HL pair will copy in S.P.

\[ [\text{SP}] \leftarrow [\text{HL}] \]

Note: It is indirect method to initialize the stack in main memory.

\[ \text{Add mode } \rightarrow \text{Implicit} \rightarrow \text{Regis. add.} \]

\[ M-\text{cy} \rightarrow S \]

\[ 2\text{OP} = \text{SPHL} \]

\[ \text{SP} = \text{HL} \]

\[ \text{LX} \leftarrow \text{SP} = 2015 \]

\[ \text{CALL} \ 3019 \text{H} \]

\[ 004 \]
(vi) PCHL no operand

→ $3w5 \equiv 1$ Byte.
→ Operation: After the execution of this instruction, the content of the HL pair will copy in P.C.
→ $M \leftarrow y = s$
→ Add: mode 'r' $\rightarrow$ implicit. $\rightarrow$ Reg Add.

Example:

1001 LXI H, 3125H
1004 MV3 A, 29H
1006 PCHL.
1007 .

Split
→ $[HL] = 3125$.
→ $[A] = 29$
→ $[PC] = 3125$.

Note: All above instructions are data transfer instructions, so status of flags will not affect.

10th Group

8 bit advance arithmetic instruction.

(ii) ADC R
→ $R \equiv A, B, C, D, E, H, L, M$.
→ $2w5 \equiv 1$ Byte.
→ Operation: When this instruction will execute, the content of $R$ will get added in $[A]$ with carry flag status, and the result will store in $[A]$.

$[A] \leftarrow [A] + [R] + C \cdot LS B$. 

After the exp. of
PCHL ins. next will fetch from
\[ MV_2 \rightarrow 25H \]
\[ MV_2 \rightarrow 1719H \]
\[ ORA \rightarrow A \]
\[ ADC \rightarrow 17 \]

\[ [A] = 25H \]
\[ [D] = 19H \]
\[ [A] = 25H \quad CY = 0 \]
\[ [A] = 0010 \ 0101 \]
\[ [D] = 0001 \ 1000 \]

\[ [A] = 0D11 \ 1110 \rightarrow CY = 0 \]
\[ AC = 0 \]
\[ Z = 0 \]
\[ N = 0 \]
\[ P = 0 \]

\[
\rightarrow \text{Add. Mode?} \quad R \neq M = \text{Reg. add.} = F. \\
R = M = \text{Immediate add. model.}
\]

(ii) \( AC \) is 8 bit data.

\[ \rightarrow \text{ws} = 2 \text{ Byte.} \]

Operation: a bit data will get added with \( [A] \) with carry flag status & result will store in \( [A] \).

\[ [A] \leftarrow [A] + 8 \text{ bit data} + CY_{LSB} \]

\[ \rightarrow \text{Add. mode?} \quad \text{Immediate add. mode.} \]

\[ R - CY = F.R. \]

(iii) SBBR

\[ \rightarrow \text{ws} = 1 \text{ Byte; } R = A, B, C, D, E, H, L, 4M. \]

Operation: content of \( R \) will get subtract from \( [A] \) with carry flag status & result will store in \( [A] \).

\[ [A] \leftarrow [A] - [R] - CY_{LSB} \]

\[ \rightarrow \text{Add. mode?} \quad \text{Reg. add. mode.} \]

\[ m - CY = F.R. \]
Operation \[ [A] \Rightarrow [A] - \text{8 bit data} - [C] \]

\[ \Rightarrow 2 \text{ws} = 2 \text{Byte} \]

\[ \Rightarrow \text{Add. mode:} \text{ Immediate} \]

\[ \Rightarrow M-Cy. \Rightarrow F.R. \]

Note: All above four inst. ADC, AC2, SBB, SBD is arithmetic inst. so status of all flag will affect.

11th Group

Some advance inst.

(i) STC no operand

\[ \Rightarrow \text{Iws} = 1 \text{Byte} \]

\[ \Rightarrow \text{Operation:} \text{ After the execution of this inst, status of carry flag will get set, regardless of previous status.} \]

\[ \Rightarrow \text{Add. mode:} \text{ Implicit add. mode.} \]

\[ \Rightarrow M-Cy. = F. \]

(ii) CMC no operand

\[ \Rightarrow \text{Iws} = 1 \text{Byte} \]

\[ \Rightarrow \text{Operation:} \text{ After the execution of this inst, status of carry flag will get complement, } \]

\[ \text{CY} \leftarrow \overline{C} \text{Y} \]

\[ \Rightarrow \text{Add. mode:} \text{ Implicit add. mode.} \]

\[ \Rightarrow M-Cycle = F. \]

Note: STC & CMC affect only one flag that is carry flag.
(iii) DAA no. operand.
→ Decimal adjustment of content of Acc.
→ This is 1 Byte.
→ Operation: When this instr. will execute content of [A] will adjust in BCD format by assuming earlier operation was BCD addition.

→ Add. mode = Implicit add. mode.
→ m-y+ + F.

Note: DAA Inst. affect status of all flag.

Note: 1. If lower nibble of [A] is greater than 1001, 0110 will get added in it.
2. If lower nibble of [A] is 1001 or less than 1001, but ACC = set, then 0110 will get added in it.
3. If lower nibble of [A] is greater than 1001, 0110 will get added in it.
4. If lower nibble of [A] is equal to 1001 or less than 1001, then 0110 will get added in it.

Ex: LXI H 8A79H
    MOV A, L
    ADD H
    DAA
    MOV, H, A
    PCHL.

Solv. [H3 = 8A79H
[A] = +1
ADD H = 7 →
[A] = 0111 1001

PCHL:

→ [H] = 6979H
→ PC = CHL
→ PC = 6979H
In 8085 five H.W. interrupts are present.

- TRAP  Non-maskable.
- RST 7.5
- RST 6.5
- RST 5.5
- INTR.

Note: TRAP also known as RST 4.5.

→ on the basis of different characteristics interrupts are classified in different groups

1) Maskable & Non-maskable interrupt

Interrupts that can make disable = Maskable int.
Interrupts that can not make disable = Non-maskable

Note: To control interrupt process in 8085 a interrupt enable flip-flop is present.

→ If interrupt enable flag is set = interrupt process enable.

→ If int enable flag is reset = interrupt process is disable.

Note: Masking & Non-masking concept valid only when interrupt process is enable.

→ To set or reset of interrupt enable flag two inst. are defined

(a) EI no operand.

IWS ∈ 1 byte

operation: → interrupt enable flag will get set.

→ interrupt process will enable.

Addl mode = 'implicit' addl.

m - cy. = f.
(b) DI no operand.
   → 2H1S = 1
   → Operation + Interrupt enable 71 will get reset
       → Interrupt process will disable.
   → Add, move: Implicit.

m → Cy ½ F.

Note 7 * EI & DI are the w/ of control inst
   so status of flag will not affect.

*Note 7 TRAP is independent of EI & DI.

(ii) Vectored & Non-vectored Interrupt

When vectored interrupt acknowledge control execution will jump at fixed vector location
of memory page 00H.

<table>
<thead>
<tr>
<th>TRAP</th>
<th>RST 7.5</th>
<th>vectored</th>
</tr>
</thead>
<tbody>
<tr>
<td>RST 6.5</td>
<td>RST 5.5</td>
<td></td>
</tr>
</tbody>
</table>

INTR → Non-vectored.

# vector location to non vectored interrupt provided by externally.

<table>
<thead>
<tr>
<th>Vectored Interrupt</th>
<th>Vector location</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRAP</td>
<td>0024H</td>
</tr>
<tr>
<td>RST 7.5</td>
<td>003CH</td>
</tr>
<tr>
<td>RST 6.5</td>
<td>0034H</td>
</tr>
<tr>
<td>RST 5.5</td>
<td>003CH</td>
</tr>
</tbody>
</table>
Triggering

Edge = TRAP (Edge trigger).
Level = both RST 7.5, RST 6.5, RST 5.5 (Level trigger).

Priority:
TRAP (highest)
RST 7.5
RST 6.5
RST 5.5
INTR (lowest)

Note: When an interrupt is acknowledged following steps execute automatically.

(i) Execution of current inst. will complete first.
(ii) Add. of next inst. will store at the top of stack.
(iii) P2 inst. will execute automatically.
(iv) Execution will transfer at interrupt sub routine.
Note 1: The programmer should write EI inst. at the last of interrupt service routine.

Note 2: In externally initiated signal HOLD has the highest priority.

Note 3: Minimum duration for which an interrupt should occur to get definite execution.

\[
\begin{align*}
18T \\
17.5T
\end{align*}
\]

* SIM (Set interrupt mask)

→ This interrupt is used for:
  - to mask the interrupt.
  - to serial transfer of data through SOI pin.

Note: This inst. execute on the basis of content of ACC.

→ SIM no operand

→ AWS = 1 byte

→ Operation: On the basis of accumulator the inst. execute as per their property.

→ Add: Mode: Implicit Add.

→ \[ m-Cy \sim f \]

\[
\begin{array}{cccccccc}
\alpha & P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0 \\
\hline
SOP & SOE & X & R7:5 & SE & M7:5 & M6:5 & M5:5 & M5:0
\end{array}
\]

Serial data transfer through SOI pin.

1. Serial data transfer process will enable.
1. RST 7.5 to be set. Enable it to be set.
1. RST 5.5 to be masked.
1. RST 7.5 to be masked.
MV $2$, A, $OEH$. $[A] = 00001110$

**R3M (Read Interrupt Mask)**

- This interrupt inst. is used for:
  - To know status of pending interrupts.
  - To know status of masked inter.
  - Used for serial data transfer through S2Y pin.

R3M no operand

- 2ws = 1 Byte.
- Operation: When this inst. execute according its feature information will load in [A].
- Addr. Mode: Implicit
- M-Cy: P.

Diagram:

- Serial data transfer through S2Y pin.

Note: R3M & RSM is uncle control inst. so status of flag will not affect.
What kind of task is performed by above set of inst.

(i) How many times up exceed memory read w/c cycle = ?
(ii) How many times up perform memory read operation or how many times data read from memory.
(iii) How many times up read operation execute or how many time Rp signal will go active low.

Total T-state read

Total time duration of execution or time elapsed in the execution of program

\[ T = 3 \text{MHz} \]

Q1:

\[ \frac{Q1f}{10T} = \text{FRR} \]

\[ \text{LXI H, 20H1H} \]

\[ \text{MVZ A, D6H} \]

\[ \text{CMP M} \]

\[ \text{MOV B, A} \]

\[ \text{IN 20H} \]

\[ \text{SUB B} \]

\[ \text{OUT 52H} \]

\[ \text{2NX H} \]

\[ \text{MOV M, A} \]

\[ \text{HLT} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{OUT 52H} \]

\[ \text{SUB B} \]

\[ \text{OUT 52H} \]

\[ \text{2NX H} \]

\[ \text{MOV M, A} \]

\[ \text{HLT} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR3} \]

\[ \text{UT} = \text{F} \]

\[ \text{10T = FR0} \]
\[ 28H = (40)_{10} \]

Support, \( B = 0.1 \)

\[ z = 0.1 \]

\[ z = 0.1 \quad 2 + \text{tms} \]

\[ z = 0.1 \quad 2 + \text{tms} \]

\[ \Rightarrow \text{Total } T = 7T + 33[18T] + 1[15T] + 4T \]

\[ = 728T. \]

\[ 2T \]

\[ MV2B \]

\[ 38H \equiv FR = 7T \]

\[ FFH \equiv FR = 7T \]

\[ = F \equiv 4T \]

\[ \Rightarrow \text{Total } T \text{ for inner comp. 255 times.} \]

\[ = 254(1UT) + 1(11T) = 3567T. \]

\[ \text{Total } T \text{ for program.} \]

\[ = 7T + 33[7T + 3567T + 4T + 10T] + 1[7T + 3567T + 4T + 4T] + 4T. \]

\[ = 200936T. \]
INTERFACING

- Memory interfacing
- I/O interfacing
- Memory mapped I/O interfacing
- I/O mapped I/O interfacing or Peripheral mapped I/O interfacing

Common steps of interfacing:

- All command signal connect directly.
- Data pins connect directly as:
  \[ (\text{MSB})_{\text{device}} \leftrightarrow (\text{MSB})_{\text{MP}} \]
  \[ (\text{LSB})_{\text{device}} \leftrightarrow (\text{LSB})_{\text{MP}} \]

- Address pins of devices connect directly as:
  \[ (\text{LSB})_{\text{device}} \leftrightarrow (\text{LSB})_{\text{MP}} \]
  \[ (\text{MSB})_{\text{device}} \leftrightarrow (\text{---})_{\text{MP}} \]

- Remaining Address buses used for development of chip selection logic.

Note: Chip selection logic can be developed by two ways:

1. By use of logic gates or buffer ckt.
2. By use of decoder ckt.
Memory Interfacing

Let

memory size = 2\(^{12}\) x 8.

2\(^n\) x d.

n = 12 = Add. pins

d = 8 = Data pins

A_{15} A_{14} A_{13} A_{12} A_9 A_8 A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0

0 0 0 0 0 0 0 0 0 0 0 1 = 0001\(_{16}\)

1 1 1 1 1 1 1 1 1 1 1 1 = 1111\(_{16}\).

Note:
First find address & data pin ordering of memory

\(\bar{A}_1\)

Find out memory add. range interfaced with 8085 MP

\(y = \bar{A}_1 \cdot B\)

\(y = \bar{A}_1 \cdot B\)

Solution:
2K Byte.

2K x 8 = 2\(^{12}\) x 8

Add = 11

Data = 8

for chip

A_{15} A_{14} A_{13} A_{12} A_9 A_8 A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0

\(\text{min add:} \bar{A}_{15} \bar{A}_{14} \bar{A}_{13} \bar{A}_{12} \bar{A}_9 \bar{A}_8 \bar{A}_7 \bar{A}_6 \bar{A}_5 \bar{A}_4 \bar{A}_3 \bar{A}_2 \bar{A}_1 \bar{A}_0\)
min \ add = F800H \ 2 \ \text{ That means b/w this if may add = FFFFH } \ \text{ proc ws gives value, memo is selected.} \ 777

Q:\ 4K Byte RAM, interfaced with 8085 MPU with chip selection logic, CS = \overline{A_{15}} \cdot \overline{A_{14}} \cdot \overline{A_{13}}.

Then find its memory interfaced range.

(a) 5000H to 5FFFH
(b) 6000H to 6FFFH
(c) 6000H to 6FFFH + 7000H to 7FFFH
(d) 5000H to 5FFFH + 6000H to 6FFFH.

Solt\ 4K Byte = 2^{12} \times 8.
Add. = 12.
Data = 8.

A_{15} \ A_{14} \ A_{13} \ A_{12} \ A_{11} \ A_{10} \ A_9 \ A_8 \ A_7 \ A_6 \ A_5 \ A_4 \ A_3 \ A_2 \ A_1 \ A_0

x=0
\min\ add = 0110000000000000
\max\ add = 0110111111111111

x=1
\min\ add = 0111000000000000
\max\ add = 0111111111111111

CS = \overline{A_{15}} \cdot \overline{A_{14}} \cdot \overline{A_{13}}
chip selection logic by decoder kit

decoder

\[
\begin{array}{c}
A \\
B \\
\end{array}
\rightarrow
\begin{array}{c}
2 \times 4 \\
D_3 \\
D_2 \\
D_1 \\
D_0 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
A B & D_3 & D_2 & D_1 & D_0 \\
\hline
0 0 & 0 & 0 & 0 & 1 \\
0 1 & 0 & 0 & 1 & 0 \\
1 0 & 0 & 1 & 0 & 0 \\
1 1 & 1 & 0 & 0 & 0 \\
\end{array}
\]

\[
f(AB) = \overline{A} \overline{B} D_0 + \overline{A} B D_1 + A \overline{B} D_2 + A B D_3.
\]

\[
\begin{array}{c}
1 = A_{1S} \\
0 = A_{1H} \\
\end{array}
\]

\[
\begin{array}{c}
L = A_{13} \\
0 = A_{12} \\
0 = A_{11} \\
\end{array}
\]

\[
1000
\]

Find out memory add. range of interfaced memory

Sol: 2 K Byte = \(2^n \times 8\)

Add = 111, Data = 0

\[
\begin{array}{c}
A_{15} \ A_{14} \ A_{13} \ A_{12} \ A_{11} \ A_{10} \ A_9 \ A_8 \ A_7 \ A_6 \ A_5 \ A_4 \ A_3 \ A_2 \ A_1 \ A_0 \\
1 \ 0 \ 1 \ 0 \ 0 \ A_{10} \ A_9 \ A_8 \ A_7 \ A_6 \ A_5 \ A_4 \ A_3 \ A_2 \ A_1 \ A_0 \\
\end{array}
\]

Add = 100000

A \text{FF} \text{FF}
Case 1 - At decoder $t_n$ is not given.

Higher order bus $q_t = \text{MSB of decoder } t_n$.
Lower order bus $q_t = \text{LSB of decoder } t_n$.

Case 2: 0/1 $p$ of decoder $\rightarrow$ i/p $q$ of decoder $\rightarrow$ Add. bus value.

$\mathbf{1} + (A\overline{B}C) \rightarrow A = \overline{A_{13}} = 1$
$B = A_{12} = 0$
$C = A_{11} = 0.$

$\mathbf{2} \rightarrow (CBA) \rightarrow 0_0$
$100$

$A = A_{13} = 0.$
$B = A_{12} = 0.$
$C = A_{11} = 1.$

I/O Interfacing

I/O mapped I/O Interface

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Memory mapped I/O Interface</th>
<th>I/O mapped I/O Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command sig.</td>
<td>MEMRD / MEMWR</td>
<td>IORP / IOWR</td>
</tr>
<tr>
<td>Instructions.</td>
<td>All memory related inst. are valid.</td>
<td>IN &amp; OUT only two inst are valid.</td>
</tr>
<tr>
<td>Execution</td>
<td>Data transfer b/w any memory reg. &amp; L, opera. perform directly with any reg.</td>
<td>Data transfer b/w Acc &amp; I/O is possible only.</td>
</tr>
<tr>
<td>No. of device interface</td>
<td>64 K Byte memory shared b/w system &amp; memory &amp; 2/I/O device</td>
<td>Max 256 2/I/O &amp; 256 0/p dev. can be interface.</td>
</tr>
<tr>
<td>H/W requirement</td>
<td>More H/W required</td>
<td>Less H/W required.</td>
</tr>
<tr>
<td>Speed</td>
<td>Slower</td>
<td>Faster</td>
</tr>
<tr>
<td>Application</td>
<td>Smaller system</td>
<td>For longer system</td>
</tr>
</tbody>
</table>

Date: 10/12/11
Some important peripherals

8255 = Programmable peripheral interface.
8237 = DMA controller.
8279 = Programmable keyboard + display interface.
8259 = Programmable interrupt controller ckt.
8155 = Programmable 2-0 port + timer ckt.
8254/8253 = Programmable interval timer.

Some important digital IC

74182 = Look ahead carry generator.
74180 = 8 bit parity generator & checker ckt.
7493 = 4 bit binary counter
7474 = Seven segment decoder.
7400 = Decade counter.
7400 = Quad 2 3-line NAND gate.
7402 = " " " " NOR " .
7408 = " " " " AND " .
7432 = " " " " OR " .
7486 = " " " " EX-OR " .

8255 (PP2)

It is 40 pin IC.

It has three ports each having port A, port B, and port C.

© Wiki Engineering
Every port individually can act as I/O port.

Port 'c' can also work as two port as port 'c' upper & port 'c' lower having port add. of 4 bit (lower nibble & upper nibble).

Port 'c' upper & port 'c' lower generate control signal for port 'a' & port 'b' respectively.

It works in two mode.

(i) Bit Set Reset mode (BSR)
(ii) I/O mode.

I/O mode

\[ \downarrow \]

Mode '0'

(General mode)

\[ \downarrow \]

Mode '1'

(Handshaking mode)

Mode '2'

(Bidirectional mode)

(i) Mode '0'

In this mode port 'a', port 'b' port 'c' works as I/O mode separately.

(ii) Mode '1'

In this mode Port 'a', Port 'b' works in handshaking mode & Port 'c' upper & Port 'c' lower generate control signal respectively.

(iii) Mode '2'

In this mode of operation Port 'a' works in bidirectional mode only & Port 'c' upper generate control signal for it.

w.b. Page 93.

Quest No. 22.
Case 3: $X=0$
\[ \begin{array}{cccccccc}
A_7 & A_6 & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
\end{array} \]

$Y=0$
\[ \begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
\end{array} \]

$Y=1$
\[ \begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
\end{array} \]

$Y=1$
\[ \begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array} \]

\[ F8 \]
\[ \frac{+1}{FC} \]
\[ \Rightarrow F8H \to \text{FFH} \]

This is in continuation.

8086 (Introduction)

- Comparison b/w 8085 & 8086
- Internal Architecture
- Memory segmentation & Physical address
- Flag Reg.
- Address Mode.

8085 CPU
- It is 40 pin SC
- Based on NMOS tech.
- $V_{cc} = +5V$
- Operating freq. = 3MHz
- It is 8 bit MP.
- Add Pin = 16, Data Pin = 8

8086
- It is 40 Pin SC
- Based on NMOS tech.
- $V_{cc} = +5V$
- Operating freq. = 5MHz
- It is 16 bit MP.
- Add. Pin = 20, Data Pin 16
Max memory that can interface = 6.4 k byte → Max memory that can interface = 1 M byte.

→ Flag Reg of 8 bit

→ No. of flags = 5

→ Flag Reg 16 bit

→ No. of flags = 9

Internal Arch. ⇒

Internal Arch. of 8086 divide in two parts.

(i) Bus interface unit (B2U).

(ii) Execution unit.

(i) BUS interface unit (B2U)

→ All type of data & add. transfer over buses managed by this part.

→ In this part there is a Queue of six location that is used for storage of instruction the fetch during the execution of current inst.

Queue works on the principle of FIFO.

(ii) EU (Execution unit)

→ All type of data execution occur in this part.

→ ALU of 8086 present in execution part.

Memory Segmentation

Note: 8086 μp, 16 bit data can be read from memory (continuous location) in one cycle. If 1st byte at even address
It requires 2 m/c cycle 16 bit data from memory.
If 1st byte at odd address.

But in the case of 8088 it requires two m/c cycle to read 16 bit data from two continuous memory location (only 8 bit in one m/c cycle), but process 16 bit in one m/c, so it is externally 8 bit internally 16 bit m/c.

Memory Segmentation

1 M Byte memory divide in four segment.
- Code segment = 64 K Byte = Instructions feed in this segment.
- Stack segment = 64 K Byte = Temporary information during the execution of main program.
- Data segment = 64 K Byte = Data storage.
- Extra segment = 64 K Byte

1 M Byte = 16 → But we use only 4

Segment can be defined any where in 1 M Byte memory.

Bottom add. of each segment select in such a way that lower 4 bit should be zero.

Upper 16 bit of bottom add. store in a specific type of reg. dedicated to segment.
code segment = cs (16 bit).
stack segment = ss (16 bit).
data segment = ds (16 bit).
extra segment = es (16 bit).

Regarding each segment there is another 16 bit reg, that holds the 16 bit addr. in range of 64 KByte (0000H to FFFFH).

code segment = Instruction Pointer (IP)
stack segment = Stack pointer (SP) { 16 bit
data segment = Source index (SI)
extra segment = Destination index (DI)

* Physical Address

\[
\text{Actual Memory Addr} = \text{Memory Location Addr}
\]

Instruction Pointer (IP) is similar as program counter (P.C.) of 8085 mp.

\[
\begin{align*}
\text{CS} : & \text{ IP} \\
& \text{offset add.}
\end{align*}
\]

\[
\begin{align*}
\text{CS} &= 2134H \\
\text{IP} &= 1269H \\
\text{Physical} &= 225A9H \\
\text{Add.}
\end{align*}
\]
Flag Reg.

- Size of flag reg is 16 bit.
- All five flags of 8085 common in 8086 up.
- Total no. of flags in 8086 is nine.
- Nine flags can be divide in two categories:
  1. Conditional flag: status of these flags affect according to condition generate in CPU. No. of conditional flag = 6.

    - Carry flag (CY)
    - Parity flag (P)
    - Auxiliary carry flag (AC)
    - Zero flag (Z)
    - Sign flag (S)
    - Overflow flag (O)

  2. Process control flag:

    These flags (status of these flags) used for machine control or process controlling.
    - Trap flag (T)
    - Interrupt flag (I)
    - String direction flag (D)
Addressing mode

Form of effective addr. present in the instr. is known as add. mode.

1. **Immediate Addr. Mode**
   - If data itself given in the instr. then immediate addr. mode (in the operand).

2. **Direct Addr. Mode**
   - If effective add. itself part of instr. then it is known as direct addr. mode.

3. **Indirect Reg. Addr. Mode**
   - If EA is given in the form of content of reg

4. **Register relative Addr. Mode**
   - Relative = 8 or 16 bit displacement no.
   - \( EA = [BX] + 8 \) or 16 bit displacement no.

   **Note:**
   - \( BH \ BL = BX \).
   - \( AH \ AL = AX \).

5. **Base Index addr. mode**
   - \( VA = [BX] + [SI] \) or
     \( [DI] \).
LXI B, 2001H
LXI D, 3001H
MVI H, 0AH

loop : LDA Y B
STA X D
INC X B
INC X D
DEC R H
JNZ loop
HLT

Q:n ORG 7000H
BEGIN : LXI H, 7000H
MOV A, R
ADD H
JPE END
RST 0

END : PC HL
HLT

Infinite times.

w.b. Page 100.
Q: 20

MVI 13 XXH
L2: MVI C, FFH
L1: DEC C
JNZ L1
DEC R B
JNZ L2
HLT

Inner loop completed. exe = 3567 T

100 x 10^3 = T + (N-1) [T + 3567T + 4T + 107]
N = 10^{-10} + 1 [T + 3567T + 4T + T] + 4