UNIT I - INTEL 8085

1.1 INTRODUCTION TO MICROPROCESSOR BASED SYSTEM

The microprocessor is a semiconductor device (Integrated Circuit) manufactured by the VLSI (Very Large Scale Integration) technique. It includes the ALU, register arrays and control circuit on a single chip. To perform a function or useful task we have to form a system by using microprocessor as a CPU and interfacing memory, input and output devices to it. A system designed using a microprocessor as its CPU is called a microcomputer.

The Microprocessor based system (single board microcomputer) consists of microprocessor as CPU, semiconductor memories like EPROM and RAM, input device, output device and interfacing devices. The memories, input device, output device and interfacing devices are called peripherals. The popular input devices are keyboard and floppy disk and the output devices are printer, LED/LCD displays, CRT monitor, etc.

The above block diagram shows the organization of a microprocessor based system. In this system, the microprocessor is the master and all other peripherals are slaves. The master controls all the peripherals and initiates all operations.

The work done by the processor can be classified into the following three groups.

1. Work done internal to the processor
2. Work done external to the processor
3. Operations initiated by the slaves or peripherals.

The work done internal to the processors are addition, subtraction, logical operations, data transfer operations, etc. The work done external to the processor are reading/writing the memory and reading/writing the J/O devices or the peripherals. If the peripheral requires the attention of the master then it can interrupt the master and initiate an operation.
The microprocessor is the master, which controls all the activities of the system. To perform a specific job or task, the microprocessor has to execute a program stored in memory. The program consists of a set of instructions. It issues address and control signals and fetches the instruction and data from memory. The instruction is executed one by one internal to the processor and based on the result it takes appropriate action.

**BUSES:**
The buses are group of lines that carries data, address or control signals.

- *The CPU Bus has multiplexed lines, i.e., same line is used to carry different signals.*
  The CPU interface is provided to demultiplex the multiplexed lines, to generate chip select signals and additional control signals.

- *The system bus has separate lines for each signal.*
  All the slaves in the system are connected to the same system bus. At any time instant communication takes place between the master and one of the slaves. All the slaves have tri-state logic and hence normally remain in high impedance state. Only when the slave is selected it comes to the normal logic.

**PERIPHERAL DEVICES:**
- The EPROM memory is used to store permanent programs and data.
- The RAM memory is used to store temporary programs and data.
- The input device is used to enter the program, data and to operate the system.
- The output device is used for examining the results.
  Since the speed of I/O devices does not match with the speed of microprocessor, an interface device is provided between system bus and I/O devices. *Generally I/O devices are slow devices.*

**Advantages of Microprocessor based system**
1. Computational/processing speed is high.
2. Intelligence has been brought to systems.
3. Automation of industrial processes and office administration.
4. Since the devices are programmable, there is flexibility to alter the system by changing the software alone.
5. Less number of components, compact in size and cost less. Also it is more reliable.
6. Operation and maintenance are easier.

**Disadvantages of Microprocessor based System**
1. It has limitations on the size of data.
2. The applications are limited by the physical address space.
3. The analog signals cannot be processed directly and digitizing the analog signals introduces errors.
4. The speed of execution is slow and so real time applications are not possible.
5. Most of the microprocessors does not support floating point operations.
INTEL 8085 – Pin Diagram & Description

The INTEL 8085 is a 8-bit microprocessor.
It operates on 8-bit data and uses 16-bit address to access the memory.
With the help of 16-bit address, 8085 can access $2^{16} = 65536 = 64K$ memory locations.

- It is a 40-pin DIP chip designed using NMOS.
- It operates with a power supply of +5 volts and GND.
- 8085 generates the clock signal internally by dividing the external supplied clock signal by two.

![8085 Pin Diagram]

*Fig 1.3: 8085 Microprocessor Signals and Pin Assignment*
### TABLE 1.1: 8085 SIGNAL DESCRIPTION SUMMARY

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD₀ - AD₇</td>
<td>Address/Data Bus</td>
<td>Bidirectional, Tristate</td>
</tr>
<tr>
<td>A₀ - A₁₅</td>
<td>Address Bus</td>
<td>Output, Tristate</td>
</tr>
<tr>
<td>ALE</td>
<td>Address Latch Enable</td>
<td>Output, Tristate</td>
</tr>
<tr>
<td>RD</td>
<td>Read Control</td>
<td>Output, Tristate</td>
</tr>
<tr>
<td>WR</td>
<td>Write Control</td>
<td>Output, Tristate</td>
</tr>
<tr>
<td>IO/M</td>
<td>I/O or memory indicator</td>
<td>Output, Tristate</td>
</tr>
<tr>
<td>S₀, S₁</td>
<td>Bus State Indicator</td>
<td>Output</td>
</tr>
<tr>
<td>READY</td>
<td>Wait state request</td>
<td>Input</td>
</tr>
<tr>
<td>SID</td>
<td>Serial Input Data</td>
<td>Input</td>
</tr>
<tr>
<td>SOD</td>
<td>Serial Output Data</td>
<td>Output</td>
</tr>
<tr>
<td>HOLD</td>
<td>HOLD request</td>
<td>Input</td>
</tr>
<tr>
<td>HLDA</td>
<td>HOLD acknowledge</td>
<td>Output</td>
</tr>
<tr>
<td>INTR</td>
<td>Interrupt request</td>
<td>Input</td>
</tr>
<tr>
<td>TRAP</td>
<td>Nonmaskable interrupt request</td>
<td>Input</td>
</tr>
<tr>
<td>RST 5.5</td>
<td>Hardware vectored interrupt request</td>
<td>Input</td>
</tr>
<tr>
<td>RST 6.5</td>
<td>Hardware vectored interrupt request</td>
<td>Input</td>
</tr>
<tr>
<td>RST 7.5</td>
<td>Hardware vectored interrupt request</td>
<td>Input</td>
</tr>
<tr>
<td>INTA</td>
<td>Interrupt acknowledge</td>
<td>Output</td>
</tr>
<tr>
<td>RESET IN</td>
<td>System reset</td>
<td>Input</td>
</tr>
<tr>
<td>RESET OUT</td>
<td>Peripherals reset</td>
<td>Output</td>
</tr>
<tr>
<td>X₁, X₂</td>
<td>Crystal or RC Connection</td>
<td>Input</td>
</tr>
<tr>
<td>CLK (OUT)</td>
<td>Clock Signal</td>
<td>Output</td>
</tr>
<tr>
<td>Vₛₑ, Vₛₙ</td>
<td>Power, ground</td>
<td></td>
</tr>
</tbody>
</table>

**Note**: A overbar on the signal indicates that it is active low. (i.e., the signal is normally high and when the signal is activated it is low).

### IO/M, S₁, S₀, Operation performed by the 8085

<table>
<thead>
<tr>
<th>IO/M</th>
<th>S₁</th>
<th>S₀</th>
<th>Operation performed by the 8085</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Memory WRITE</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Memory READ</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>I/O WRITE</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>I/O READ</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Opcode fetch</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Interrupt acknowledge</td>
</tr>
</tbody>
</table>
INTEL 8085 ARCHITECTURE

The architecture of 8085 is shown in figure given below. The internal architecture of 8085 includes the ALU, timing and control unit, instruction register and decoder, register array, interrupt control and serial I/O control.

OPERATIONS PERFORMED BY 8085

The ALU performs the arithmetic and logical operations.

The operations performed by ALU of 8085 are addition, subtraction, increment, decrement, logical AND, OR, EXCL U8IVE -OR, compare, complement and left / right shift. The accumulator and temporary register are used to hold the data during an arithmetic / logical operation. After an operation the result is stored in the accumulator and the flags are set or reset according to the result of the operation.

FLAG REGISTER:

There are five flags in 8085, which are sign flag (8), zero flag (Z), auxiliary carry flag (AC), parity flag (P) and carry flag (CY). The bit positions reserved for these flags in the flag register are shown in figure below.

```
D_7  D_6  D_5  D_4  D_3  D_2  D_1  D_0
  S    Z    AC   P    CY
```

Fig 1.7 : Bit positions of various flags in the flag register of 8085
After an ALU operation, if the most significant bit of the result is 1, then sign flag is set. The zero flag is set, if the ALU operation results in zero and it is reset if the result is non-zero. In an arithmetic operation, when a carry is generated by the lower nibble, the auxiliary carry flag is set. After an arithmetic or logical operation, if the result has an even number of 1’s the parity flag is set, otherwise it is reset.

If an arithmetic operation results in a carry, the carry flag is set otherwise it is reset. Among the five flags, the AC flag is used internally for BCD arithmetic and other four flags can be used by the programmer to check the conditions of the result of an operation.

**TIMING & CONTROL UNIT:**

The timing and control unit synchronizes all the microprocessor operations with the clock and generates the control signals necessary for communication between the microprocessor and peripherals.

**INSTRUCTION REGISTER & DECODER:**

When an instruction is fetched from memory it is placed in instruction register. Then it is decoded and encoded into various machine cycles.

**REGISTER ARRAY:**

- Apart from Accumulator (A-register), there are six general-purpose programmable registers B, C, D, E, H and L.

- They can be used as 8-bit registers or paired to store 16-bit data. The allowed pairs are B-C, D-E and H-L.

- The temporary registers W and Z are intended for internal use of the processor and it cannot be used by the programmer.

- **STACK POINTER (SP):**
  The stack pointer SP, holds the address of the stack top. The stack is a sequence of RAM memory locations defined by the programmer. The stack is used to save the content of registers during the execution of a program.

- **PROGRAM COUNTER (PC):**
  The program counter (PC) keeps track of program execution. To execute a program the starting address of the program is loaded in program counter. The PC sends out an address to fetch a byte of instruction from memory and increment its content automatically. Hence, when a byte of instruction is fetched, the PC holds the address of the next byte of the instruction or next instruction.
INSTRUCTION EXECUTION AND DATA FLOW in 8085

The program instructions are stored in memory, which is an external device. To execute a program in 8085, the starting address of the program should be loaded in program counter. The 8085 output the content of program counter in address bus and asserts read control signal low. Also, the program counter is incremented.

The address and the read control signal enable the memory to output the content of memory location on the data bus. Now the content of data bus is the opcode of an instruction. The read control signal is made high by timing and control unit after a specified time. At the rising edge of read control signals, the opcode is latched into microprocessor internal bus and placed in instruction register.

The instruction-decoding unit, decodes the instructions and provides information to timing and control unit to take further actions.

INSTRUCTION FORMAT OF 8085

The 8085 have 74 basic instructions and 246 total instructions. The instruction set of 8085 is defined by the manufacturer Intel Corporation. Each instruction of 8085 has 1 byte opcode. With 8 bit binary code, we can generate 256 different binary codes. In this, 246 codes have been used for opcodes.

<table>
<thead>
<tr>
<th>Instruction Size</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-byte instruction</td>
<td>OPCODE</td>
</tr>
<tr>
<td>Two-bytes instruction</td>
<td>OPCODE 8-bit data/address</td>
</tr>
<tr>
<td>Three-bytes instruction</td>
<td>OPCODE Low byte data/address</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>High byte data/address</td>
</tr>
</tbody>
</table>

The size of 8085 instructions can be 1 byte, 2 bytes or 3 bytes.

- The 1-byte instruction has an opcode alone.
- The 2 bytes instruction has an opcode followed by an eight-bit address or data.
- The 3 bytes instruction has an opcode followed by 16 bit address or data. While storing the 3 bytes instruction in memory, the sequence of storage is, opcode first followed by low byte of address or data and then high byte of address or data.
ADDRESSING MODES

Every instruction of a program has to operate on a data. The method of specifying the data to be operated by the instruction is called *Addressing*. The 8085 has the following 5 different types of addressing.

1. Immediate Addressing
2. Direct Addressing
3. Register Addressing
4. Register Indirect Addressing
5. Implied Addressing

**Immediate Addressing**
In immediate addressing mode, the data is specified in the instruction itself. The data will be apart of the program instruction. All instructions that have ‘I’ in their mnemonics are of Immediate addressing type.

_Eg._ MVI B, 3E\(^{H}\) - Move the data 3E\(^{H}\) given in the instruction to B register.

**Direct Addressing**
In direct addressing mode, the address of the data is specified in the instruction. The data will be in memory. In this addressing mode, the program instructions and data can be stored in different memory blocks. This type of addressing can be identified by 16-bit address present in the instruction.

_Eg._ LDA 1050\(^{H}\) - Load the data available in memory location 1050\(^{H}\) in accumulator.

**Register Addressing**
In register addressing mode, the instruction specifies the name of the register in which the data is available. This type of addressing can be identified by register names (such as ‘A’, ‘B’, … ) in the instruction.

_Eg._ MOV A, B - Move the content of B register to A register.

**Register Indirect Addressing**
In register indirect addressing mode, the instruction specifies the name of the register in which the address of the data is available. Here the data will be in memory and the address will be in the register pair. This type of addressing can be identified by letter ‘M’ present in the instruction.

_Eg._ MOV A, M - The memory data addressed by HL pair is moved to A register.

**Implied Addressing**
In implied addressing mode, the instruction itself specifies the type of operation and location of data to be operated. This type of instruction does not have any address, register name, immediate data specified along with it.

_Eg._ CMA - Complement the content of accumulator.
INSTRUCTION SET

The 8085 instruction set can be classified into the following five functional headings.

**Group I - DATA TRANSFER INSTRUCTIONS:**
Includes the instructions that moves (copies) data between registers or between memory locations and registers. In all data transfer operations the content of source register is not altered. Hence the data transfer is copying operation.

Ex: i) MOV A,B    ii) LDA 4600    iii) LHLD 4200

**Group II - ARITHMETIC INSTRUCTIONS:**
Includes the instructions which performs the addition, subtraction, increment or decrement operations. The flag conditions are altered after execution of an instruction in this group.

Ex: i) ADD B    ii) SUB C    iii) INR D    iv) INX H

**Group III - LOGICAL INSTRUCTIONS:**
The instructions which performs the logical operations like AND, OR, Exclusive-OR, complement, compare and rotate instructions are grouped under this heading. The flag conditions are altered after execution of an instruction in this group.

Ex: i) ORA B    ii) XRA A    iii) RAR

**Group IV - BRANCHING INSTRUCTIONS:**
The instructions that are used to transfer the program control from one memory location to another memory location are grouped under this heading.

Ex: i) JZ 4200    ii) RST 7    iii) CALL 4300

**Group V - MACHINE CONTROL INSTRUCTIONS:**
Includes the instructions related to interrupts and the instruction used to halt program execution.

Ex: i) SIM    ii) RIM    iii) HLT

The 74 basic instructions of 8085 are listed in Table-2.1. The opcode of each instruction, size, machine cycles, number of T-state and the total number of instructions in each type are also shown in table in next page. The instructions affecting the status flag are listed in table followed.
### Table-2.1 : Summary of 8085 instruction set

<table>
<thead>
<tr>
<th>No.</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>No. of bytes</th>
<th>Machine cycles</th>
<th>No. of T-states</th>
<th>Total No. of instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>MOV Rd, Rs</td>
<td>01DDSSS</td>
<td>1</td>
<td>F</td>
<td>4T</td>
<td>49</td>
</tr>
<tr>
<td>2.</td>
<td>MOV Rd, M</td>
<td>01DD110</td>
<td>1</td>
<td>F, R</td>
<td>7T</td>
<td>7</td>
</tr>
<tr>
<td>3.</td>
<td>MOV M, Rs</td>
<td>0110SSS</td>
<td>1</td>
<td>F, W</td>
<td>7T</td>
<td>7</td>
</tr>
<tr>
<td>4.</td>
<td>MVI Rd, d8</td>
<td>00DD110</td>
<td>2</td>
<td>F, R</td>
<td>7T</td>
<td>7</td>
</tr>
<tr>
<td>5.</td>
<td>MVI M, d8</td>
<td>0010110</td>
<td>2</td>
<td>F, R, W</td>
<td>10T</td>
<td>1</td>
</tr>
<tr>
<td>6.</td>
<td>LDA addr16</td>
<td>00111010</td>
<td>3</td>
<td>F, R, R, R</td>
<td>13T</td>
<td>1</td>
</tr>
<tr>
<td>7.</td>
<td>LDAX rp</td>
<td>00RP1110</td>
<td>1</td>
<td>F, R</td>
<td>7T</td>
<td>2</td>
</tr>
<tr>
<td>8.</td>
<td>LXI rp, d16</td>
<td>00RP001</td>
<td>3</td>
<td>F, R, R</td>
<td>10T</td>
<td>4</td>
</tr>
<tr>
<td>9.</td>
<td>LHLD addr16</td>
<td>00101010</td>
<td>3</td>
<td>F,R,R,R</td>
<td>16T</td>
<td>1</td>
</tr>
<tr>
<td>10.</td>
<td>STA addr16</td>
<td>0011010</td>
<td>3</td>
<td>F, R, R,W</td>
<td>13T</td>
<td>1</td>
</tr>
<tr>
<td>11.</td>
<td>STAX rp</td>
<td>00RP010</td>
<td>1</td>
<td>F, W</td>
<td>7T</td>
<td>2</td>
</tr>
<tr>
<td>12.</td>
<td>SHLD addr16</td>
<td>00100010</td>
<td>3</td>
<td>F,R,R,W,W</td>
<td>16T</td>
<td>1</td>
</tr>
<tr>
<td>13.</td>
<td>SPHL</td>
<td>1111001</td>
<td>1</td>
<td>S</td>
<td>6T</td>
<td>1</td>
</tr>
<tr>
<td>14.</td>
<td>XCHG</td>
<td>1110111</td>
<td>1</td>
<td>F</td>
<td>4T</td>
<td>1</td>
</tr>
<tr>
<td>15.</td>
<td>XTHL</td>
<td>1100011</td>
<td>1</td>
<td>F,R,R,W,W</td>
<td>16T</td>
<td>1</td>
</tr>
<tr>
<td>16.</td>
<td>PUSH rp</td>
<td>11RP010</td>
<td>1</td>
<td>S, W, W</td>
<td>12T</td>
<td>3</td>
</tr>
<tr>
<td>17.</td>
<td>PUSH PSW</td>
<td>1111010</td>
<td>1</td>
<td>S, W, W</td>
<td>12T</td>
<td>1</td>
</tr>
<tr>
<td>18.</td>
<td>POP rp</td>
<td>11RP0001</td>
<td>1</td>
<td>F, R, R</td>
<td>10T</td>
<td>3</td>
</tr>
<tr>
<td>19.</td>
<td>POP PSW</td>
<td>1111001</td>
<td>1</td>
<td>F,R,R</td>
<td>10T</td>
<td>1</td>
</tr>
<tr>
<td>20.</td>
<td>IN addr8</td>
<td>11011011</td>
<td>2</td>
<td>F,R,I</td>
<td>10T</td>
<td>1</td>
</tr>
<tr>
<td>21.</td>
<td>OUT addr8</td>
<td>1101011</td>
<td>2</td>
<td>F, R, O</td>
<td>10T</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>22.</td>
<td>ADD reg</td>
<td>10000SSS</td>
<td>1</td>
<td>F</td>
<td>4T</td>
<td>7</td>
</tr>
<tr>
<td>23.</td>
<td>ADD M</td>
<td>10000110</td>
<td>1</td>
<td>F,R</td>
<td>7T</td>
<td>1</td>
</tr>
<tr>
<td>24.</td>
<td>ADI d8</td>
<td>11000110</td>
<td>2</td>
<td>F,R</td>
<td>7T</td>
<td>1</td>
</tr>
<tr>
<td>25.</td>
<td>ADC reg</td>
<td>10001SSS</td>
<td>1</td>
<td>F</td>
<td>4T</td>
<td>7</td>
</tr>
<tr>
<td>26.</td>
<td>ADC M</td>
<td>10001110</td>
<td>1</td>
<td>F,R</td>
<td>7T</td>
<td>1</td>
</tr>
<tr>
<td>27.</td>
<td>ACI d8</td>
<td>11001110</td>
<td>2</td>
<td>F,R</td>
<td>7T</td>
<td>1</td>
</tr>
<tr>
<td>28.</td>
<td>DAA</td>
<td>00100111</td>
<td>1</td>
<td>F</td>
<td>4T</td>
<td>1</td>
</tr>
<tr>
<td>29.</td>
<td>DAD rp</td>
<td>00RP1001</td>
<td>1</td>
<td>F,B,B,</td>
<td>10T</td>
<td>4</td>
</tr>
<tr>
<td>30.</td>
<td>SUB reg</td>
<td>10010SSS</td>
<td>1</td>
<td>F</td>
<td>4T</td>
<td>7</td>
</tr>
<tr>
<td>31.</td>
<td>SUB M</td>
<td>10010110</td>
<td>1</td>
<td>F,R</td>
<td>7T</td>
<td>1</td>
</tr>
<tr>
<td>32.</td>
<td>SUI d8</td>
<td>11010110</td>
<td>2</td>
<td>F,R</td>
<td>7T</td>
<td>1</td>
</tr>
<tr>
<td>33.</td>
<td>SBB reg</td>
<td>10011SSS</td>
<td>1</td>
<td>F</td>
<td>4T</td>
<td>7</td>
</tr>
<tr>
<td>34.</td>
<td>SBB M</td>
<td>10011110</td>
<td>1</td>
<td>F,R</td>
<td>7T</td>
<td>1</td>
</tr>
<tr>
<td>35.</td>
<td>SBI d8</td>
<td>11011110</td>
<td>2</td>
<td>F,R</td>
<td>7T</td>
<td>1</td>
</tr>
<tr>
<td>36.</td>
<td>INR reg</td>
<td>00SSS100</td>
<td>1</td>
<td>F</td>
<td>4T</td>
<td>7</td>
</tr>
<tr>
<td>37.</td>
<td>INR M</td>
<td>00110100</td>
<td>1</td>
<td>F,R,W</td>
<td>10T</td>
<td>1</td>
</tr>
<tr>
<td>38.</td>
<td>INX rp</td>
<td>00RP0011</td>
<td>1</td>
<td>S</td>
<td>6T</td>
<td>4</td>
</tr>
<tr>
<td>39.</td>
<td>DCR reg</td>
<td>00SSS101</td>
<td>1</td>
<td>F</td>
<td>4T</td>
<td>7</td>
</tr>
<tr>
<td>40.</td>
<td>DCR M</td>
<td>00110101</td>
<td>1</td>
<td>F,R,W</td>
<td>10T</td>
<td>1</td>
</tr>
<tr>
<td>41.</td>
<td>DCX rp</td>
<td>00RP1011</td>
<td>1</td>
<td>S</td>
<td>6T</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Instruction</td>
<td>Format</td>
<td>Function</td>
<td>Type</td>
<td>Result</td>
<td>Count</td>
</tr>
<tr>
<td>---</td>
<td>-------------</td>
<td>--------</td>
<td>----------</td>
<td>------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>42.</td>
<td>ANA reg</td>
<td><code>101000SSS</code></td>
<td>F</td>
<td>4T</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>43.</td>
<td>ANA M</td>
<td><code>10100110</code></td>
<td>F,R</td>
<td>7T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>44.</td>
<td>ANI d8</td>
<td><code>11100110</code></td>
<td>F,R</td>
<td>7T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>45.</td>
<td>ORA reg</td>
<td><code>101100SSS</code></td>
<td>F</td>
<td>4T</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>46.</td>
<td>ORA M</td>
<td><code>10110110</code></td>
<td>F,R</td>
<td>7T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>47.</td>
<td>ORI d8</td>
<td><code>11110110</code></td>
<td>F,R</td>
<td>7T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>48.</td>
<td>XRA reg</td>
<td><code>101010SSS</code></td>
<td>F</td>
<td>4T</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>49.</td>
<td>XRA M</td>
<td><code>10101110</code></td>
<td>F,R</td>
<td>7T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>50.</td>
<td>XRI d8</td>
<td><code>11110110</code></td>
<td>F,R</td>
<td>7T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>51.</td>
<td>CMP reg</td>
<td><code>101110SSS</code></td>
<td>F</td>
<td>4T</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>52.</td>
<td>CMP M</td>
<td><code>10111110</code></td>
<td>F,R</td>
<td>7T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>53.</td>
<td>CPI d8</td>
<td><code>11111110</code></td>
<td>F,R</td>
<td>7T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>54.</td>
<td>CMA</td>
<td><code>00101111</code></td>
<td>F</td>
<td>4T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>55.</td>
<td>CMC</td>
<td><code>00111111</code></td>
<td>F</td>
<td>4T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>56.</td>
<td>STC</td>
<td><code>00110111</code></td>
<td>F</td>
<td>4T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>57.</td>
<td>RLC</td>
<td><code>00001111</code></td>
<td>F</td>
<td>4T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>58.</td>
<td>RAL</td>
<td><code>00010111</code></td>
<td>F</td>
<td>4T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>59.</td>
<td>RRC</td>
<td><code>00001111</code></td>
<td>F</td>
<td>4T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>60.</td>
<td>RAR</td>
<td><code>00111111</code></td>
<td>F</td>
<td>4T</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
### Group IV: Branching instructions

<table>
<thead>
<tr>
<th></th>
<th>Instruction</th>
<th>Opcode</th>
<th>Function</th>
<th>Code</th>
<th>OpType</th>
<th>Code</th>
<th>Instruction Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>JMP addr16</td>
<td>11000011</td>
<td>F,R,R</td>
<td>3</td>
<td>10T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>J&lt;condition&gt; addr16</td>
<td>111100010</td>
<td>F,R/F,R,R</td>
<td>3</td>
<td>7T/10T</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>CALL addr16</td>
<td>11001101</td>
<td>S,R,R,W,W</td>
<td>3</td>
<td>18T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>C&lt;condition&gt; addr16</td>
<td>11111100</td>
<td>S, R or S,R,R,W,W</td>
<td>3</td>
<td>9T/18T</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>RET</td>
<td>11001001</td>
<td>F,R,R</td>
<td>1</td>
<td>10T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>R&lt;condition&gt;</td>
<td>11111100</td>
<td>S/S,R,R</td>
<td>1</td>
<td>6T/12T</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>RST n</td>
<td>11111111</td>
<td>S,W,W</td>
<td>1</td>
<td>12T</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>PCHL</td>
<td>11101001</td>
<td>S</td>
<td>1</td>
<td>6T*</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

### Group V: Machine control instructions

<table>
<thead>
<tr>
<th></th>
<th>Instruction</th>
<th>Opcode</th>
<th>Function</th>
<th>Code</th>
<th>OpType</th>
<th>Code</th>
<th>Instruction Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>69</td>
<td>SIM</td>
<td>00110000</td>
<td>F</td>
<td>1</td>
<td>4T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>RIM</td>
<td>00100000</td>
<td>F</td>
<td>1</td>
<td>4T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>DI</td>
<td>11110111</td>
<td>F</td>
<td>1</td>
<td>4T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>EI</td>
<td>11110111</td>
<td>F</td>
<td>1</td>
<td>4T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>HLT</td>
<td>01110110</td>
<td>F,B</td>
<td>1</td>
<td>5T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>NOP</td>
<td>00000000</td>
<td>F</td>
<td>1</td>
<td>4T</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Total instructions:** 246
INTERRUPTS

NEED FOR INTERRUPTS

Interrupt is a signal send by an external device to the processor, to the processor to perform a particular task or work. Mainly in the microprocessor based system the interrupts are used for data transfer between the peripheral and the microprocessor.

When a peripheral is ready for data transfer, it interrupts the processor by sending an appropriate signal to the interrupt pin of the processor. If the processor accepts the interrupt then the processor suspends its current activity and executes an interrupt service subroutine to complete the data transfer between the peripheral and processor. After executing the interrupt service routine the processor resumes its current activity. This type of data transfer scheme is called interrupt driven data transfer scheme.

TYPES OF INTERRUPTS

The interrupts are classified into software interrupts and hardware interrupts.

- The software interrupts are program instructions. These instructions are inserted at desired locations in a program. While running a program, if a software interrupt instruction is encountered, then the processor executes an interrupt service routine (ISR).

- The hardware interrupts are initiated by an external device by placing an appropriate signal at the interrupt pin of the processor. If the interrupt is accepted, then the processor executes an interrupt service routine (ISR).

SOFTWARE INTERRUPTS OF 8085

The software interrupts are program instructions. When the instruction is executed, the processor executes an interrupt service routine stored in the vector address of the software interrupt instruction. The software interrupts of 8085 are RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6 and RST 7.

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Vector address</th>
</tr>
</thead>
<tbody>
<tr>
<td>RST 0</td>
<td>0000&lt;sub&gt;H&lt;/sub&gt;</td>
</tr>
<tr>
<td>RST 1</td>
<td>0008&lt;sub&gt;H&lt;/sub&gt;</td>
</tr>
<tr>
<td>RST 2</td>
<td>0010&lt;sub&gt;H&lt;/sub&gt;</td>
</tr>
<tr>
<td>RST 3</td>
<td>0018&lt;sub&gt;H&lt;/sub&gt;</td>
</tr>
<tr>
<td>RST 4</td>
<td>0020&lt;sub&gt;H&lt;/sub&gt;</td>
</tr>
<tr>
<td>RST 5</td>
<td>0028&lt;sub&gt;H&lt;/sub&gt;</td>
</tr>
<tr>
<td>RST 6</td>
<td>0030&lt;sub&gt;H&lt;/sub&gt;</td>
</tr>
<tr>
<td>RST 7</td>
<td>0038&lt;sub&gt;H&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

The vector addresses of software interrupts are given in table below.
The software interrupt instructions are included at the appropriate (or required) place in the main program. When the processor encounters the software instruction, it pushes the content of PC (Program Counter) to stack. Then loads the Vector address in PC and starts executing the Interrupt Service Routine (ISR) stored in this vector address. At the end of ISR, a return instruction - RET will be placed. When the RET instruction is executed, the processor POP the content of stack to PC. Hence the processor control returns to the main program after servicing the interrupt. *Execution of ISR is referred to as servicing of interrupt.*

All software interrupts of 8085 are vectored interrupts. The software interrupts cannot be masked and they cannot be disabled.

**The software interrupts are RST0, RST1, … RST7 (8 Nos).**

**HARDWARE INTERRUPTS OF 8085**

An external device, initiates the hardware interrupts of 8085 by placing an appropriate signal at the interrupt pin of the processor. The processor keeps on checking the interrupt pins at the second T-state of last machine cycle of every instruction. If the processor finds a valid interrupt signal and if the interrupt is unmasked and enabled, then the processor accepts the interrupt. The acceptance of the interrupt is acknowledged by sending an INTA signal to the interrupted device.

The processor saves the content of PC (program Counter) in stack and then loads the vector address of the interrupt in PC. (If the interrupt is non-vectored, then the interrupting device has to supply the address of ISR when it receives INTA signal). It starts executing ISR in this address. At the end of ISR, a return instruction, RET will be placed. When the processor executes the RET instruction, it POP the content of top of stack to PC. Thus the processor control returns to main program after servicing interrupt.

**The hardware interrupts of 8085 are TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR.**

Further the interrupts may be classified into VECTORED and NON-VECTORED INTERRUPTS.

**VECTORED INTERRUPT**

In vectored interrupts, the processor automatically branches to the specific address in response to an interrupt.

**NON-VECTORED INTERRUPT**

But in non-vectored interrupts the interrupted device should give the address of the interrupt service routine (ISR).

*In vectored interrupts, the manufacturer fixes the address of the ISR to which the program control is to be transferred. The vector addresses of hardware interrupts are given in table above in previous page.*
The TRAP, RST 7.5, RST 6.5 and RST 5.5 are vectored interrupts.
The INTR is a non-vectored interrupt. Hence when a device interrupts through INTR, it has to supply the address of ISR after receiving interrupt acknowledge signal.

*The type of signal that has to be placed on the interrupt pin of hardware interrupts of 8085 are defined by INTEL.*

The TRAP interrupt is edge and level sensitive. Hence, to initiate TRAP, the interrupt signal has to make a low to high transition and then it has to remain high until the interrupt is recognized.
The RST 7.5 interrupt is edge sensitive (positive edge). To initiate the RST 7.5, the interrupt signal has to make a low to high transition and it need not remain high until it is recognized.
The RST 6.5, RST 5.5 and INTR are level sensitive interrupts. Hence for these interrupts the interrupting signal should remain high, until it is recognized.

**MASKABLE & NON-MASKABLE INTRUPTS:**

*The hardware vectored interrupts are classified into maskable and non-maskable interrupts.*

- TRAP is non-maskable interrupt
- RST 7.5, RST 6.5 and RST 5.5 are maskable interrupt.

Masking is preventing the interrupt from disturbing the main program. When an interrupt is masked the processor will not accept the interrupt signal. The interrupts can be masked by moving an appropriate data (or code) to accumulator and then executing SIM instruction. (SIM - Set Interrupt Mask). The status of maskable interrupts can be read into accumulator by executing RIM instruction (RIM - Read Interrupt Mask).

All the hardware interrupts, except TRAP are disabled, when the processor is resetted. They can also be disabled by executing DI instruction. (DI-Disable Interrupt).

- When an interrupt is disabled, it will not be accepted by the processor. (i.e., INTR, RST 5.5, RST 6.5 and RST 7.5 are disabled by DI instruction and upon hardware reset).
- To enable (to allow) the disabled interrupt, the processor has to execute El instruction (El-Enable Interrupt).
INTERRUPT DRIVEN DATA TRANSFER SCHEME

The interrupt driven data transfer scheme is the best method of data transfer for effectively utilizing the processor time. In this scheme, the processor first initiates the I/O device for data transfer. After initiating the device, the processor will continue the execution of instructions in the program. Also at the end of an instruction the processor will check for a valid interrupt signal. If there is no interrupt then the processor will continue the execution.

When the I/O device is ready, it will interrupt the processor. On receiving an interrupt signal, the processor will complete the current instruction execution and saves the processor status in stack. Then the processor calls an interrupt service routine (ISR) to service the interrupted device. At the end of ISR the processor status is retrieved from stack and the processor starts executing its main program. The sequence of operations for an interrupt driven data transfer scheme is shown in figure below.

Fig (a) : Main program execution sequence
Fig (b) : ISR execution sequence
TIMING DIAGRAM for various machine cycles

The machine cycles are the basic operations performed by the processor, while instructions are executed. The time taken for performing each machine cycle is expressed in terms of T-states.

One T-state is the time period of one clock cycle of the microprocessor.

The various machine cycles are

1. Opcode fetch …………….. - 4 / 6 T
2. Memory Read …………….. - 3 T
3. Memory Write …………….. - 3 T
4. I/O Read ………………….. - 3 T
5. I/O Write …………………. - 3 T
6. Interrupt Acknowledge …- 6 / 12 T
7. Bus Idle …………………….. - 2 / 3 T
Opcode fetch machine cycle of 8085

Each instruction of the processor has one byte opcode. The opcodes are stored in memory. The opcode fetch machine cycle is executed by the processor to fetch the opcode from memory. Hence, every instruction starts with opcode fetch machine cycle.

The time taken by the processor to execute the opcode fetch cycle is either 4 T or 6 T. In this time, the first, 3 T-states are used for fetching the opcode from memory and the remaining T-states are used for internal operations by the processor. The timings of various signals during opcode fetch cycle is shown in fig 2.2.

1. At the falling edge of first T-state (T1), the microprocessor outputs the low byte address on ADo-AD3 lines and high byte address on A5 to A11 lines. ALE is asserted high to enable the address latch. The other control signals are asserted as follows.

\[ \text{IO/M = 0, } S_0 = 1, S_1 = 1. \]

2. At the middle of T1, the ALE is asserted low and this enables the latch to take low byte of the address and keep on its output lines.

3. In the second T-state (T2), the memory is requested for read by asserting read line low. When read is asserted low, the memory is enabled for placing the data on the data bus. The time allowed for memory to output the data is the time during which read remains low.

4. In the third T-state (T3), the read signal is asserted high. On the rising edge of read signal the data is latched into microprocessor. Other control signals remains in the same state until the next machine cycle.

5. The fourth T-state (T4) is used by the processor for internal operations to decode the instruction and encode into various machine cycles, and also for completing the task specified by 1 byte instructions. During this cycle the address and data bus will be in high impedance state.
Memory Read Machine Cycle of 8085

The memory read machine cycle is executed by the processor to read a data byte from memory. The processor takes, 3T states to execute this cycle. The timings of various signals during memory read cycle are shown in fig 2.3.

1. At the falling edge of T₁, the microprocessor outputs the low byte address on AD₀ - AD₃ lines and high byte address on A₈ to A₁₅ lines. ALE is asserted high to enable the address latch. The other control signals are asserted as follows:
   \[ IO / M = 0, S₀ = 0, S₁ = 1. \]

2. At the middle of T₂, the ALE is asserted low and this enables the latch to take low byte of address and keep on its output lines.

3. In the second T-state (T₃) the memory is requested for read by asserting read line low. When read is asserted low, the memory is enabled for placing the data on the data bus. The time allowed for memory to output the data is the time during which read remains low.

4. At the end of T₃, the read signal is asserted high. On the rising edge of read signal the data is latched into microprocessor. Other control signals remains in the same state until the next machine cycle.
Memory Write Machine Cycle of 8085

The memory write machine cycle is executed by the processor to write a data byte in a memory location. The processor takes, 3 T states to execute this machine cycle. The timings of various signals during memory write cycle are shown in fig 2.4.

( RD will be high; READY is tied high either permanently or temporarily in the system.)

Fig 2.4 : Memory write machine cycle of 8085

1. At the falling edge of T₁, the microprocessor outputs the low byte address on AD₀ - AD₇ lines and high byte address on A₈ to A₁₅ lines. ALE is asserted high to enable the address latch. The other control signals are asserted as follows.
   IO / M = 0, S₀ = 1, S₁ = 0.

2. At the middle of T₁, the ALE is asserted low and this enables the D latch for latching the low byte address into its output lines.

3. In the falling edge of T₂ the processor outputs data on AD₀ to AD₇ lines and then request memory for write operation by asserting the write control signal WR to low.

4. At the end of T₃, the processor asserts WR high. This enables the memory to latch the data into it. The memory should prepare itself to accept the data within the time duration in which write control signal remains low. Other control signals remains in the same state until the next machine cycle.
I/O Read Cycle of 8085

The I/O Read cycle is executed by the processor to read a data byte from I/O port or from the peripheral which is I/O mapped in the system. The processor takes 3T states to execute this machine cycle. The timings of various signals during this machine cycle are shown in fig 2.5.

![Diagram of I/O Read Cycle of 8085](image)

(\(W_R\) will be high; \(READY\) is tied high either permanently or temporarily in the system.)

Fig 2.5 : I/O read machine cycle of 8085

1. At the falling edge of \(T_1\), the microprocessor outputs the 8 bit port address on both the low order address lines \((AD_0 - AD_7)\) and high order address lines \((A_8 \text{ to } A_{11})\). ALE is asserted high to enable the address latch. The other control signals are asserted as follows.
   \[ IO / \bar{M}, S_0, S_1 \]
   \[ IO / \bar{M} = 1, S_0 = 0 \text{ and } S_1 = 1. \text{ (} IO / \bar{M} \text{ is asserted high to indicate I/O read operation).} \]

2. At the middle of \(T_1\), the ALE is asserted low and this enables the latch to take the port address and keep on its output lines.

3. In the second T-state \((T_2)\) the I/O device is requested for read by asserting read line low. When read is asserted low, the I/O port is enabled for placing the data on the data bus. The time allowed for I/O port to output the data is the time during which read remains low.

4. At the end of \(T_3\), the read signal is asserted high. On the rising edge of read signal the data is latched into microprocessor. Other control signals remains in the same state until the next machine cycle.
I/O Write Cycle of 8085

The I/O write machine cycle is executed by the processor to write a data byte in the I/O port or to a peripheral which is I/O mapped in the system. The processor takes, 3T states to execute this machine cycle. The timings of the various signals of I/O write cycle is shown in fig 2.6.

![Diagram of I/O write cycle of 8085]

( RD will be high; READY is tied high either permanently or temporarily in the system.)

Fig 2.6 : I/O write machine cycle of 8085

1. At the falling edge of T₁, the microprocessor outputs the 8 bit port address on both the low order address lines (AD₀ - AD₇) and high order address lines (A_a to A₁₅). ALE is asserted high to enable the address latch. The other control signals are asserted as follows.

IO / M = 1, S₀ = 1 and S₁ = 0. (IO / M is asserted high to indicate I/O read operation).

2. At the middle of T₁, the ALE is asserted low and this enables the D-latch for latching the port address into its output lines.

3. In the falling edge of T₁, the processor outputs data on AD₀ - AD₇ lines and then request I/O port for write operation by asserting the write control signal \( \overline{WR} \) to low.

4. At the end of T₁, the processor asserts \( \overline{WR} \) high. This enables the I/O port to latch the data into it. The I/O port should prepare itself to accept the data within the time duration in which write control signal remains low. Other control signals at the same state until the next machine cycle.
DELAY ROUTINE

Delay routines are subroutines used for maintaining the timings of various operations in microprocessor.

In control applications, certain equipment needs to be *ON/OFF* after a specified time delay. In some applications, a certain operation has to be repeated after a specified time interval. In such cases, simple time delay routines can be used to maintain the timings of the operations.

DELAY ROUTINE PROCESS

*A delay routine is generally written as a subroutine (It need not be a subroutine always. It can be even a part of main program). In delay routine a count (number) is loaded in a register of microprocessor. Then it is decremented by one and the zero flag is checked to verify whether the content of register is zero or not. This process is continued until the content of register is zero. When it is zero, the time delay is over and the control is transferred to main program to carry out the desired operation.*

The delay time is given by the total time taken to execute the delay routine. It can be computed by multiplying the total number of T-states required to execute subroutine and the time for one T-state of the processor. The total number of T-states can be computed from the knowledge of T-states required for each instruction. The time for one T-state of the processor is given by the inverse of the internal clock frequency of the processor.

For example, if the 8085 microprocessor has 5 MHz quartz crystal then,

The internal clock frequency = $\frac{5}{2} = 2.5$ MHz

Time for one T-state = $\frac{1}{2.5 \times 10^6} = 0.4\mu\text{sec}$

- For small time delays (< 0.5 msec) an 8-bit register can be used.
- For large time delays (< 0.5 Sec) 16-bit register should be used.
- For very large time delays (> 0.5 sec), a delay routine can be repeatedly called in the main program.

The disadvantage in delay routines is that the processor time is wasted. An alternate solution is to use dedicated timer like 8253/8254 to produce time delays or to maintain timings of various operations.

Two example delay routines are presented in this section with details of timing calculations.
EXAMPLE DELAY ROUTINE -1
Write a delay routine to produce a time delay of 0.5 msec in 8085 processor-based system whose clock source is 6 MHz quartz crystal.

Solution
The delay required is 0.5 msec, hence an 8-bit register of 8085 can be used to store a Count value and then decrement to zero. The delay routine is written as a subroutine as shown below.

Delay routine
MVI D, N ; Load the count value, N in D-register.
Loop: DCR D ; Decrement the count.
JNZ Loop ; If count is zero go to
RET ; Return to main program.

The following table shows the T-state required for execution of the instructions in the subroutine.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>T-State required for execution of an instruction</th>
<th>Number of times the instruction is executed</th>
<th>Total T-States</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL addr16</td>
<td>18</td>
<td>1</td>
<td>18 x 1 = 18</td>
</tr>
<tr>
<td>MVI D, N</td>
<td>7</td>
<td>1</td>
<td>7 x 1 = 7</td>
</tr>
<tr>
<td>DCR D</td>
<td>4</td>
<td>N times</td>
<td>4 x N = 4N</td>
</tr>
<tr>
<td>JNZ LOOP</td>
<td>10 (or)</td>
<td>(N-1) times</td>
<td>10 x (N-1) = 10N - 10</td>
</tr>
<tr>
<td>RET</td>
<td>10</td>
<td>1</td>
<td>10 x 1 = 10</td>
</tr>
</tbody>
</table>

TOTAL T-STATES FOR DELAY SUBROUTINE 14N + 32

Calculation to find the count value, N:

External clock frequency = 6 Mhz

Internal clock frequency = External Frequency / 2
= 6 / 2
= 3 Mhz

Time period for 1 T-State = 1 / Internal clock frequency
= 1 / 3x10^6
= 0.333µS

No. of T-states required for delay of 0.5mS = Required time delay / Time for one T-state
= 0.5mS / 0.333µS
= 1500.10
≈ 1500 = 1500

From above table, we know that;
14N + 32 = 1500
N = (1500 – 32) / 14 = 104.857;10 ≈ 105;10 = 69;10

Therefore by replacing the count value, N by 69;10 in the above program, a delay of 0.5mSec can be produced.
PROGRAMMING EXAMPLES:

1. Write an ALP using 8085 to multiply two 8-bit numbers by repeated addition.

   MVI A, OO ; Accumulator contents are cleared
   MVI C, OO ; C Register contents are cleared
   MVI B, data#1 ; I Operand is loaded into B Register
   MVI D, data#2 ; II Operand is loaded into D Register

   Loop: ADD B |}
       JNC next |
       INR C | Multiplication by repeated addition.
   Next: DCR D |
          JNZ loop |

   STA 4200H ; Storing of results into memory location
   MOV A,C
   STA 4201H ; Storing of carry into next memory location

2. Write an ALP for 8085 to count from AAH to 00H, with a time delay of 2ms for each count. Assume the external frequency given to the processor is 2MHz.

   Internal Frequency in 8085 = External frequency / 2
ie., = 2Mhz / 2
= 1Mhz

   1 T-State = 1 / f (internal frequency)
= 1 µ S

   Main program for counting from AA to 00
   MVI C, AAH
   Loop: CALL Delay
          DCR C
          JNZ Loop
          HLT

   Delay program for delay of 2ms
   Delay: MVI D, 4AH
   Next: NOP
          NOP
          NOP
          NOP
          DCR D
          JNZ Next
          RET
3. Write an ALP using 8085 to evaluate the expression $C=A^2+B^2$

Let ‘A’ be Data#1 and ‘B’ be Data#2

```
MVI B, Data#1 ; Data #1 is stored in register B
MOV C, B ; Copy of Data #1 is made in register C
MVI D, Data#2 ; Data #2 is stored in register D
MOV E, D ; Copy of Data #2 is made in register E

XRA A ; Accumulator content is cleared
Again: ADD B ]
DCR C } $A^2$ is calculated by repeated Addition
JNZ Again ]

MOV H, A ; Calculated $A^2$ value is stored in register H

XRA A ; Accumulator content is cleared
Loop: ADD D ]
DCR E } $B^2$ is calculated by repeated Addition
JNZ Loop ]

ADD H ; $A^2 + B^2$ is determined, by adding result in A and register content H

STA 4200H ; Result is stored in memory location 4200H
```
INTERFACING EXAMPLES:

Draw the circuit diagram of an 8085 system, having a 4 KB EPROM and two 8 KB RAM ICs. The starting address of the EPROM is 0000H and that of RAM is 8000H. The address of the decoder circuits should be clearly shown.

Answer:

EPROM - 4 KB (Address lines required is 12 - A_0 to A_{11})
RAM-I - 8 KB (Address lines required is 13 - A_0 to A_{12})
RAM-II - 8 KB (Address lines required is 13 - A_0 to A_{12})

Mapping of Addresses to Memory ICs

<table>
<thead>
<tr>
<th>ICs</th>
<th>Binary Address</th>
<th>Hex Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>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}</td>
<td></td>
</tr>
<tr>
<td>EPROM 4 KB</td>
<td>0 0 x x 0 0 0 0 0 0 0 0 0 0 0</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>. . . . . . . . . . . . . . . . . . .</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>0 0 x x 1 1 1 1 1 1 1 1 1 1 1 1</td>
<td>0FFF</td>
</tr>
<tr>
<td>RAM-I 8 KB</td>
<td>0 1 x 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>4000</td>
</tr>
<tr>
<td></td>
<td>. . . . . . . . . . . . . . . . . . .</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>0 1 x 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
<td>5FFF</td>
</tr>
<tr>
<td>RAM-II 8 KB</td>
<td>1 0 x 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>8000</td>
</tr>
<tr>
<td></td>
<td>. . . . . . . . . . . . . . . . . . .</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>1 0 x 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
<td>9FFF</td>
</tr>
</tbody>
</table>
Part A:

1. **State the function of HOLD and HLDA pins in 8085.**
   
   The HOLD and HLDA pins in 8085 are used in interfacing the 8257-DMA controller IC with the processor.
   
   A signal is sent by 8257 to HOLD pin in µ P, to request the µ P to stop its current process and allocate the buses for DMA data transfer.
   
   µ P acknowledges the request for DMA data transfer by 8257, by sending a signal in HLDA to 8257.

2. **Distinguish I/O mapped I/O and memory mapped I/O.**

   Mapping is the process by which the addresses are allocated to the I/O devices.
   
   The two kinds of mapping are
   
   a) Memory mapped I/O
   
   b) I/O mapped I/O

<table>
<thead>
<tr>
<th>S.No</th>
<th><strong>Memory mapped I/O</strong></th>
<th><strong>I/O mapped I/O</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16 bit address is given to each I/O device</td>
<td>8 bit address is given to each I/O device</td>
</tr>
<tr>
<td>2</td>
<td>Each I/O device is treated like a memory location and they are accessed using instructions related to memory operations.</td>
<td>All I/O devices are accessed using only two instructions viz., IN and OUT.</td>
</tr>
<tr>
<td>3</td>
<td>Data can be transferred between I/O devices and all registers in µ P.</td>
<td>Data can be transferred only between I/O devices and accumulator in µ P.</td>
</tr>
<tr>
<td>4</td>
<td>This scheme is used in system, where memory requirement is small.</td>
<td>This scheme is used in system, where complete memory capacity is required.</td>
</tr>
<tr>
<td>5</td>
<td>Only Memory Read &amp; Write machine cycles are involved during data transfer with I/O devices.</td>
<td>Only I/O Read &amp; Write machine cycles are involved during data transfer with I/O devices.</td>
</tr>
<tr>
<td>6</td>
<td>Large number of I/O devices can be connected in this scheme.</td>
<td>Only maximum of 256(=2^8) I/O devices can be connected in this scheme.</td>
</tr>
</tbody>
</table>

3. **Explain the execution of the instruction CMA instruction in 8085.**

   CMA instruction is used to perform 1’s complement of the contents of Accumulator in 8085.
4. **What is the function performed by SIM and RIM instruction.**

   **SIM Instruction:**
   The SIM instruction is used to mask the hardware interrupts RST7.5, RST6.5 and RST5.5. It is also used to send data through SOD line.

   **RIM Instruction:**
   The RIM instruction is used to check whether an interrupt (RST7.5, RST6.5 and RST5.5) is masked or not. It is also used to read data from SID line.

5. **What will be the outcome, in execution of instructions LXI H,4600H and LHLD 4600H?**

   When LXI H,4600 is executed, the number 4600 will be loaded to HL register pair.
   When LHLD 4600 is executed, the contents of memory location 4600H will be transferred to HL register pair.

6. **Explain the concept of demultiplexing AD0-7 lines in 8085?**

   Demultiplexing is the process of separating the low byte address A0-7 and 8-bit data D0-7 from AD0-7 lines of 8085, using a latch and Address latch enable (ALE) signal.

   ![Diagram of 8085 demultiplexing](attachment:image.png)

   When low byte address (A0-7) comes out of AD0-7 lines, the processor asserts HIGH in the ALE pin, enabling the latch to separate the low byte address.
7. **Compare System bus and CPU bus.**

Bus is a set of conducting wires in a microprocessor based system, which helps to carry various information like DATA, ADDRESS and other CONTROL

<table>
<thead>
<tr>
<th>System Bus</th>
<th>CPU Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>It will <em>not</em> be directly connected to CPU</td>
<td>It will be directly connected to CPU</td>
</tr>
<tr>
<td>There will be <em>separate</em> data, address &amp; control buses</td>
<td>The data and address may be <em>multiplexed</em></td>
</tr>
</tbody>
</table>

8. **State the significance of X₁ and X₂ pins of 8085.**

The clock signal is supplied to the microprocessor 8085 by connecting quartz crystal through the pins X₁ and X₂.

9. **What is Processor (machine) cycle? List the various machine cycles with its T-states.**

The machine cycles are the basic operations performed by the processor, while instructions are executed. The time taken for performing each machine cycle is expressed in terms of T-states.

The various machine cycles are

1. Opcode fetch ……………. - 4 / 6 T
2. Memory Read ……………. - 3 T
3. Memory Write ……………. - 3 T
4. I/O Read ………………….. - 3 T
5. I/O Write ………………….. - 3 T
6. Interrupt Acknowledge …… - 6 / 12 T
7. Bus Idle ……………………. - 2 / 3 T
10. List the various addressing modes in 8085 with two examples in each.

Addressing is the method of specifying the location of data in an instruction. The different types of addressing modes in 8085 are

a) **Direct:**
   The data is stored in memory and 16 bit address of data in memory location is specified in the instruction.
   Eg.: LDA 4500, LHLD 4200

b) **Immediate:**
   The required data for processing is given next to the Opcode, in the instruction itself.
   Eg.: MVI A, 55  CPI 64,  ADI 0A

c) **Register:**
   The data is placed in a register and the register name is given in the instruction to access the data.
   Eg.: MOV A,B     ADD B,      SUB C

d) **Register Indirect:**
   The data is stored in memory and the 16-bit address of the data location in memory is placed in a register pair. This register pair holding the 16-bit address is given in the instruction to access the data.
   Eg.: LXI, H 4250
        MOV A, M

e) **Implied:**
   The data location & the operation to be performed is given in the instruction itself.
   Eg.: CMA, RAR,   XCHG

11. Define stack and stack pointer.

**Stack:**
A small portion of the RAM memory is declared as stack and it is used for temporary storage of the register contents, using instructions like PUSH and POP. The contents are stored and retrieved in LIFO (Last In First Out) form.

**Stack Pointer:**
It is a 16-bit memory pointing register, having the last address of the stack in RAM.
12. Compare CALL and JMP instructions.

**CALL Instruction:**

Execution of a CALL instruction will transfer the program control from existing program to another program. i.e., Sub program specified by the 16-bit address in CALL instruction will be executed. The called program should have RET – return instruction as its last instruction. Time taken for its execution is $9 / 18$ T

Main __________  addr16: __________
__________  __________
__________  __________
__________  __________
CALL addr16  __________
__________  __________
__________  __________
__________  RET

**JMP Instruction**

Execution of a JMP instruction will transfer the program control from one location to another location within the same program. Time taken for its execution is $7 / 10$ T

Main __________
__________
__________
__________
JMP addr16 __________
__________
__________
addr16: __________
__________

13. What is an interrupt and list the various interrupts in 8085.

**Interrupt:**

Interrupt is a signal send by an external device to the processor (or special instruction executed in a program), to stop the execution of the current process in the microprocessor and perform a particular task (i.e., data transfer) to the called device.

Various HARDWARE interrupts are TRAP, RST7.5, RST6.5, RST5.5, INTR (5 Nos)

Various SOFTWARE interrupts are RST0, RST1, RST2 …… RST7 (8 Nos)
14. Explain the function of IN and OUT instructions.

Execution of an IN instruction will transfer one byte of data from an Input device to Accumulator of microprocessor.

Execution of an OUT instruction will transfer one byte of data from Accumulator of microprocessor to an Output device.

15. Write an ALP for time delay using a register pair available in 8085.

```
Main __________ Delay: LXI D, data16
________________ NOP
________________ NOP
________________ NOP
CALL Delay DCX D
________________ JNZ loop
________________ RET

The register pair used is DE.
The total time delay made is as follows.
One T-state = 1 / F_{internal}
T – states (in execution of one loop) = 4T + 4T + 4T + 4T + 7T = 23T
Total T-states = 23T x data16 (stored in DE register pair)
```

16. Write an assembly language program to store the contents of the flag register in memory location 2000_{H}.

```
PUSH PSW - Stores the contents of Accumulator & Flag register in Stack
POP D - Restores the stored contents of stack to DE register pair
MOV A, E - Move the contents of E register to Accumulator
STA 2000_{H} - Contents of Accumulator is now stored to memory location 2000_{H}
```

17. Explain the Instruction format of 8085.

The 8085 have 74 basic instructions. The size of 8085 instructions can be 1 byte, 2 bytes or 3 bytes.

1 Byte instruction has Opcode alone.
2 Bytes instructions have 1 byte Opcode followed by 8 bit data.
3 Bytes instruction have 1 byte Opcode followed by 16 bit data.
There are five flags in 8085, they are sign flag (S), zero flag (Z), auxiliary carry flag (AC), parity flag (P) and carry flag (CY). The bit positions reserved for these flags in the flag register are shown in fig 1.7.

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

Fig 1.7: Bit positions of various flags in the flag register of 8085

After an ALU operation if the most significant bit of the result is 1, the sign flag is set. The zero flag is set if the ALU operation results in zero and it is reset if the result is non-zero. In an arithmetic operation, when a carry is generated by the lower nibble the auxiliary carry flag is set. After an arithmetic or logical operation if the result has an even number of 1’s the parity flag is set otherwise it is reset.