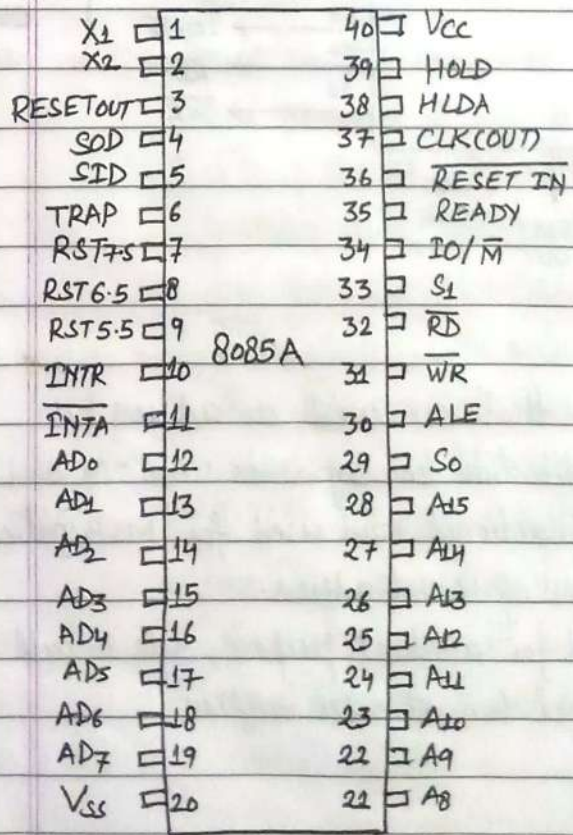
THE 8085 MPU

- The term Microprocessing unit (MPU) is similar to term Central processing unit (CPU).
- MPU is defined as a device or a group of devices that can communicate with peripherals, provide timing signals, direct data flow, and perform computing tasks as specified by the instructions in memory. The unit will have the necessary lines for the address bus, the data bus and the control signals and would require only a power supply and a crystal to be completely functional.
- 8085 microprocessor is with the following two limitations:
  (i.) The low-order address bus of the 8085 microprocessor is multiplexed with data bus. The buses need to be demultiplexed.
  (ii.) Appropriate control signals need to be generated to interface memory and I/O with the 8085.
- The 8085 Microprocessor



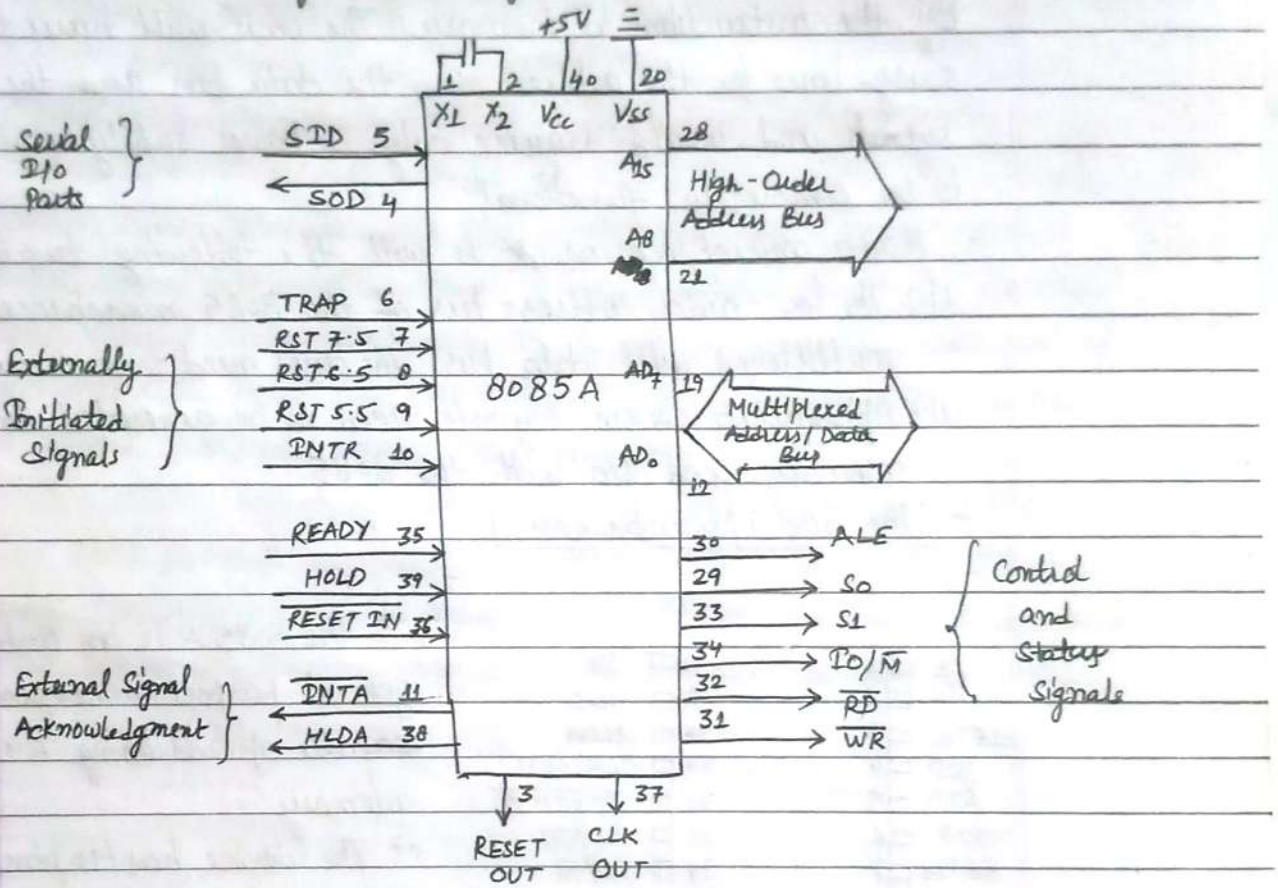
8085 Pinout

→ The 8085A is an 8-bit general-purpose microprocessor capable of addressing 64K of memory.

→ The device has 40 pins, requires a +5V signal power supply, and can operate with a 3-MHz single phase clock and can operate at maximum frequency of 5 MHz.

→ Its instruction set is upward-compatible with that of 8080A, meaning that the 8085 instruction set includes all the 8080A instructions plus some additional ones.

- Figure shows the logic pinout of 8085 microprocessor.
- All the signals can be classified into six groups:
  (1.) Address bus, (2.) Data bus,
  (3.) Control and status signals, (4.) Power supply and frequency signals,
  (5.) External initiated signals, (6.) Serial I/O ports.



(1.) Address bus

- 8085 has 16 pins that are used as address bus.
- Address lines are split into two segments: A_{15} - A_8 and AD_7 - AD_0.
  → A_{15} - A_8 are unidirectional and used for MSB, called high order address of 16-bit address.
  → AD_7 - AD_0 are used for a dual purpose, data as well as address and used for low order address.

(2.) Multiplexed Address / Data bus

- $AD_7 - AD_0$ lines are bidirectional.
- They are used as low order address bus as well as the data bus.
- During the execution of an instruction, in earlier part of the cycle, these lines are used as the low-order address bus. During the later part of the cycle, these lines are used as the data bus. This is also known as multiplexing of the bus.

(3.) Control and Status Signals

- This group includes
  - → Control signals; $\overline{RD}$ and $\overline{WR}$
  - → Status signals; IO/$\overline{M}$, S_1 and S_0, to identify the nature of the operation.
  - → One special signal; ALE to indicate the beginning of the operation.

→ ALE (Address Latch Enable): A positive going pulse generated everytime at beginning of 8085 operation. It indicates that the bits on $AD_7 - AD_0$ are address bits. This signal is used primarily to latch the low order address from the multiplexed bus and generate a separate set of eight address lines, $A_7 - A_0$.

→ $\overline{RD}$ (Read): An active low read control signal. This indicates that selected I/O or memory device is to be read and data are available on the data bus.

→ $\overline{WR}$ (Write): An active low write control signal. This indicates that data on data bus are to be written into a selected memory or I/O location.

→ IO/$\overline{M}$: A status signal used to differentiate between I/O and memory operations. When it is high it indicates I/O operation; when it is low it indicates a memory operation. This signal is combined with $\overline{RD}$ and $\overline{WR}$ to generate I/O and memory control signals.

→ S_1 and S_0: These status signals, can identify various operations, but they are rarely used in small systems.

Machine Cycle	Status			Control Signals
	IO/M	S_1	S_0	
'Opcode' Fetch	0	1	1	$\overline{RD} = 0$
Memory Read	0	1	0	$\overline{RD} = 0$
Memory Write	0	0	1	$\overline{WR} = 0$
I/O Read	1	1	0	$\overline{RD} = 0$
I/O Write	1	0	1	$\overline{WR} = 0$
Interrupt Acknowledge	1	1	1	$\overline{INTA} = 0$
Halt	Z	0	0	$\overline{RD}, \overline{WR} = Z, \overline{INTA} = 1$
Hold	Z	X	X	
Reset	Z	X	X	

(4.) Power Supply and clock Frequency

The power supply and frequency signals are as follows:

→ V_{CC}: +5 V power supply

→ V_{SS}: Ground Reference

→ X_1, X_2: A crystal is connected at these two pins. The frequency is internally divided by two.

→ CLK (OUT): (Clock output): This signal can be used as the system clock for other device.

(5.) Externally initiated signals, including interrupts

- The 8085 has five interrupt signals that can be used to interrupt a program execution.

→ INTR (Interrupt Request): is used as a general purpose interrupt.

→ $\overline{INTA}$: The microprocessor acknowledges an interrupt request by this signal.

→ RST 7.5, RST 6.5, RST 5.5: (Restart Interrupts) These are vectored interrupts that transfer the program control to specific memories locations. They have higher priorities than the INTR interrupt. Among these three the priority order is 7.5, 6.5, 5.5

→ TRAP: This is a non maskable interrupt and has the highest priority.

- In addition to the interrupts, three pins RESET, HOLD, and READY.
  → HOLD: This signal indicates that a peripheral such as a DMA controller is requesting the use of the address and data buses
  → HLDA: This signal acknowledges the HOLD request.
  → READY: This signal is used to delay the microprocessor Read or Write cycles until a slow-responding peripheral is ready to send or accept data. When this signal goes low, the microprocessor waits for an integral number of clock cycles until it goes high.
  - RESET IN: When the signal on this pin goes low the program Counter is set to zero, the buses are tri-stated and MPU is reset.
  - RESET OUT: This signal indicates that the MPU is being reset. The signal can be used to reset other devices.

(6.) Serial I/O Ports:
- 8085 has two signals to implement the serial transmission:
  → SID (Serial Input Data)
  → SOD (Serial Output Data)
- In serial transmission, data bits are sent over a single line, one bit at a time.

- Microprocessor Communication and Bus timings.
- To understand the function of various signals of 8085, it is necessary to examine the communication process between microprocessor and memory and the timings of these signals in relation to system clock.
- Communication process starts is reading from memory or fetching an instruction. Then perform the required operation to complete the communication process.

- To fetch the byte, the MPU performs the following steps:

Step 1: The microprocessor places the 16-bit address from the memory program counter (PC) on the address lines.

Step 2: The control unit sends the control signal $\overline{RD}$ to enable the memory chip.

Step 3: The byte from memory location is placed on the data bus.

Step 4: The byte is placed in the instruction decoder of microprocessor and the task is carried out according to the instruction.

eg: Fetch the instruction code stored in location 2005H.

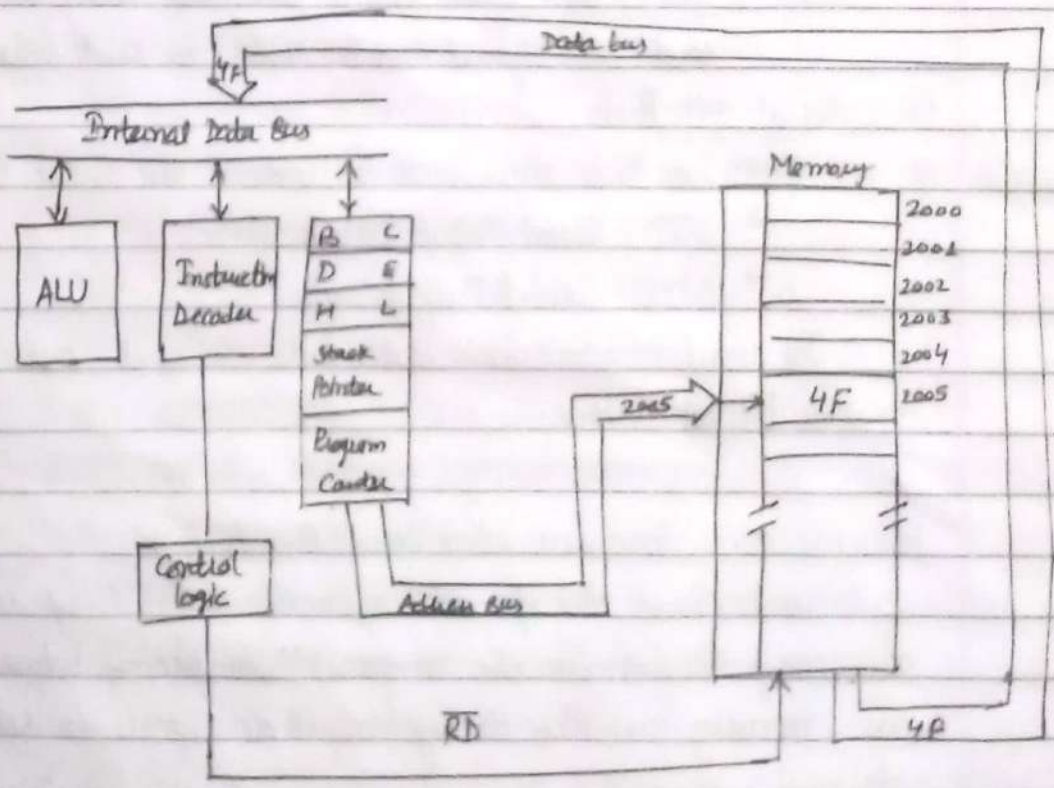
Step 1: Place the address 2005H on address bus from PC.

Step 2: Control unit sends $\overline{RD}$ signal to memory.

Step 3: Instruction code is placed on data bus.

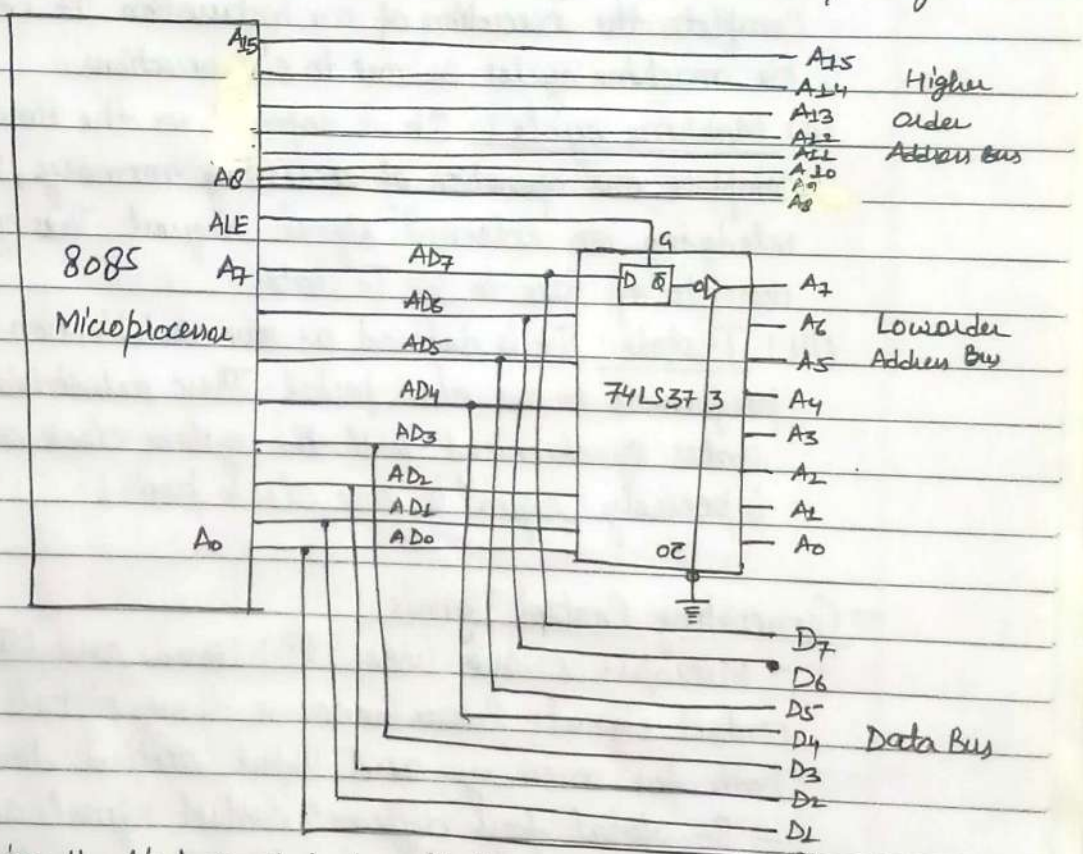
Step 4: Instruction code is placed in instruction decoder.

Suppose instruction code 4FH is stored in memory location 2005H.



- Demultiplexing the bus $AD_7 - AD_0$:

- 8085 uses a multiplexed data bus and lower order address bus ($AD_7 - AD_0$).
- The multiplexing is done to reduce the pin count of the device.
- External memory or I/O devices need the complete 16-bit address for decoding and selecting a device. So the multiplexed address and data bus must be de-multiplexed.
- The de-multiplexing can be done with the help of ALE signal given by the processor for this purpose.
- Figure shows the hardware needed for demultiplexing.



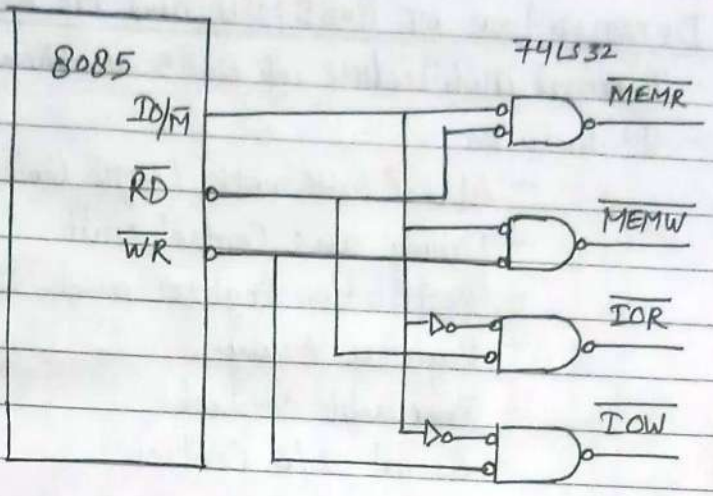
- During the first T-state in all the machine cycles, $AD_7 - AD_0$ lines act as address bus and the lower order address is available on these lines. The $AD_7 - AD_0$ lines are converted into the data bus during the second T-state.
- The latch 74LS373 which has 8 input lines connected to $AD_7 - AD_0$ and 8 output lines is [illegible] ALE signal is connected

to the enable (G) pin of latch; the output control ($\overline{OC}$) signal of the latch is grounded.

- During T_1 (first T-state) ALE goes high and the latch is transparent. When ALE goes low the lower order address bits are latched until next ALE and output of latch represents the low order address bus.
- When ALE goes low ($AD_7 - AD_0$) lines are available for data bus.

- There are three terms related to timing of operations:
  (i.) Instruction cycle: It is defined as the time required to complete the execution of an instruction. It consists of one to six machine cycles or one to six operations.
  (ii.) Machine cycle: It is defined as the time required to complete one operation of accessing memory, I/O or acknowledging an external request. This cycle may consists of two to six T-states.
  (iii.) T-state: It is defined as one subdivision of the operation performed in one clock period. These subdivisions are internal states synchronized with the system clock and each T-state is precisely equal to one clock period.

- Generating Control Signals:
  - Microprocessor uses ($\overline{RD}$) (Read) and ($\overline{WR}$) (write) as control signals. Because these signals are used both for memory and Input/output devices.
  - So total four different control signals are generated by combining the signals $\overline{RD}$, $\overline{WR}$, and IO/$\overline{M}$. The signal IO/$\overline{M}$ goes low for memory operations. This signal is ANDed with $\overline{RD}$ and $\overline{WR}$ signals by using 74LS32; and generate $\overline{MEMR}$ and $\overline{MEMW}$ control signals. When IO/$\overline{M}$ goes high, it indicates the peripheral I/O operations and generate $\overline{IOR}$ and $\overline{IOW}$ control signals.


8085
IO/M
RD
WR
74LS32
MEMR
MEMW
IOR
IOW

DETAILED LOOK AT 8085 MPU and ITS ARCHITECTURE

- Internal architecture of 8085 is shown in figure
- It includes
  - ALU (Arithmetic / Logic Unit)
  - Timing and Control unit
  - Instruction Register and Decoder
  - Register Array
  - Interrupt control
  - Serial I/O Control

→ **ALU:**

→ It performs the computing functions

→ It includes accumulator, the temporary registers, the arithmetic and logic circuits and five flags

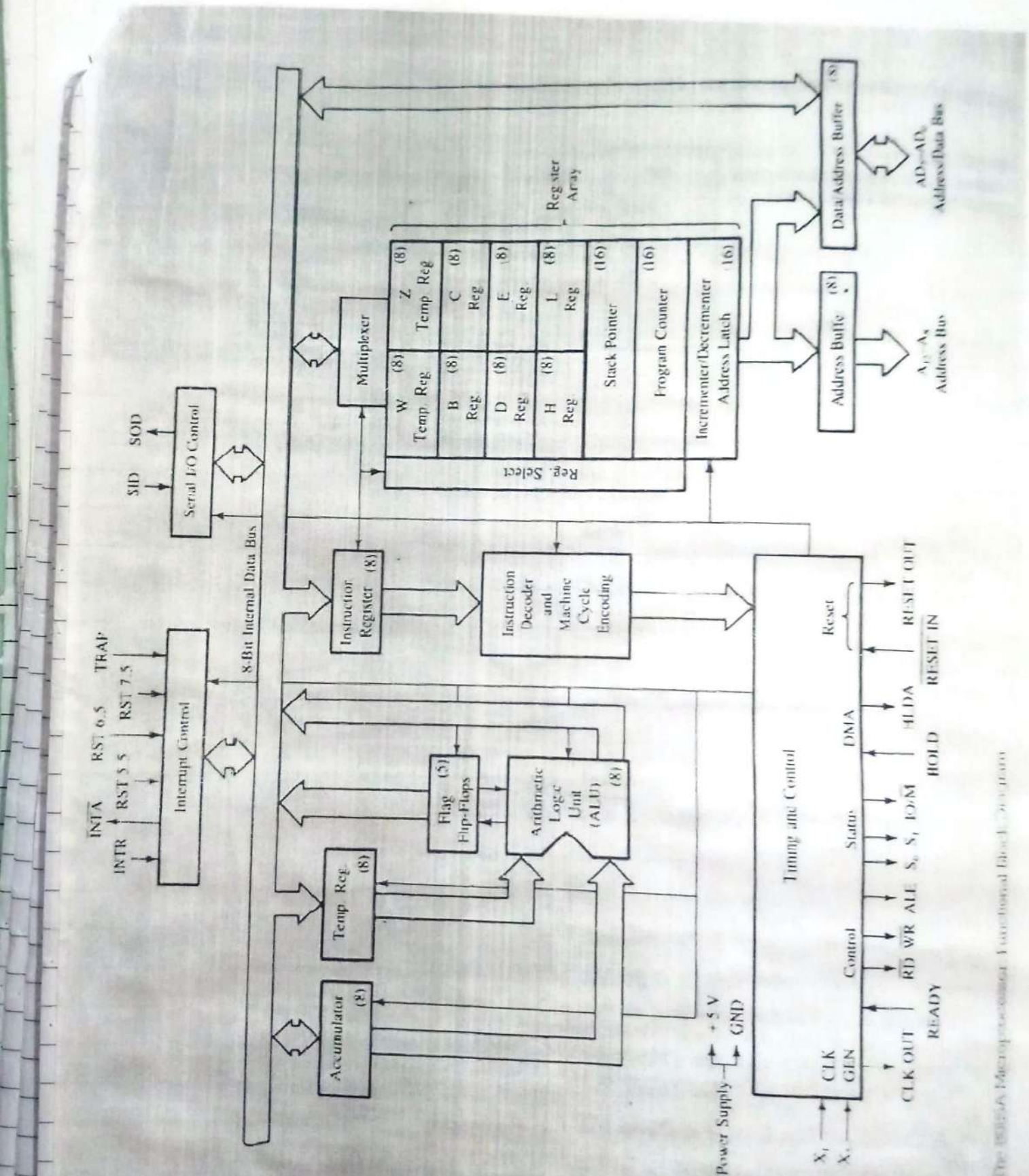
→ The temporary registers are used to hold data during an arithmetic / logic operation. The result is stored in accumulator, and flags are set or reset according to the result of operation.

→ The flags are affected by the arithmetic and logic operations in the ALU. The flags affects the condition of data in the accumulator - with some exceptions.

→ The conditions of flags are as follows:

(i) S - sign flag: after the execution of arithmetic and logic operations, If bit D_7 of result is 1, the Sign flag is set, the number will be viewed as a negative number; If it is 0, the number will be considered as positive.

(ii) Z - Zero flag: It is set if the ALU operation result is 0 and the flag is reset if result is not 0. This flag is modified by the results in the accumulator as well as in other registers.


Register Array
Multiplexer
W (8) Temp Reg
Z (8) Temp Reg
B (8) Reg
C (8) Reg
D (8) Reg
E (8) Reg
H (8) Reg
L (8) Reg
Stack Pointer (16)
Program Counter (16)
Incrementer/Decrementer Address Latch (16)
Reg. Select
Data Address Buffer (8)
Address Buffer (8)
AD0–AD7 Address Data Bus
A8–A15 Address Bus
Serial I/O Control
SID
SOD
8-Bit Internal Data Bus
Interrupt Control
INTA
INTR
RST 5.5
RST 6.5
RST 7.5
TRAP
Instruction Register (8)
Instruction Decoder and Machine Cycle Encoding
Flag (5) Flip-Flops
Arithmetic Logic Unit (ALU) (8)
Temp Reg (8)
Accumulator (8)
Timing and Control
Reset
RESET OUT
RESET IN
HLDA
HOLD
DMA
Status
Control
RD WR ALE S0 S1 IO/M
READY
CLK GEN
CLK OUT
X1
X2
+5 V
GND
Power Supply

→ AC - Auxiliary Carry flag:
In arithmetic operation, when a carry is generated by digit D_3 and passed to digit D_4 the AC flag is set. The flag is used internally for BCD operations.

→ P - Parity flag:
After arithmetic and logic operation, if the result has an even number of 1s the flag is set. If it has odd number of 1s, the flag is reset.

→ CY - Carry flag:
If arithmetic operation results in a carry, the Carry flag is set; otherwise it is reset.

The bit positions reserved for these flags in flag register as:

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

→ Timing and Control unit:
- This unit synchronizes all the microprocessor operations with the clock and generates the control signals necessary for communication between microprocessor and peripherals.

→ Instruction Register and Decoder:
- It is part of the ALU.
- When an instruction is fetched from memory, it is loaded in the instruction register.
- The decoder decodes the instruction and establishes the sequence of events to follow.
- Instruction register is not programmable and cannot be accessed through any instruction.

→ Register Array:
- It includes six registers, one accumulator, one flag register and two special purpose register. Two temporary registers W and Z.
- General purpose registers: 8085 has six general purpose registers to store 8-bit data; these are identified as B, C, D, E, H and L. They can be combined as register pairs

BC, DE, HL to perform 16-bit operations.

Accumulator (A)	Flag Register
B	C
D	E
H	L
Stack Pointer (SP)	
Program Counter (PC)	

- Accumulator: It is an 8-bit register that is part of ALU and used to store 8-bit data and to perform 8-bit arithmetic and logical operations. The result of an operation is stored in the accumulator and is identified as A.
- Flags: ALU includes five flip-flops which are set or reset after an operation according to data conditions of result. They are called Zero (Z), Carry (CY), Sign (S), Parity (P) and Auxiliary carry (AC) flags.
These flags are have critical importance in decision making process of the microprocessor.
- Program Counter (PC) and Stack Pointer (SP):
Also known as special purpose registers. These are 16-bit registers used to hold memory addresses.
The function of PC is to point to the memory address from which the next byte is to be fetched. When a byte is fetched the PC is incremented by one to point to the next memory location.
The stack pointer (SP) is also 16-bit register used as memory pointer. It points to a memory location in a R/W memory, called the stack. The beginning of the stack is defined by loading a 16-bit address in SP.
- Temporary Registers: Two additional registers called temporary registers W and Z. These are used to hold 8-bit data during execution of instructions and are not visible to programmer.

→ Interrupt Control:

Sometimes it is necessary to interrupt the execution of the main program to answer a request from an I/O device. An I/O device may send an interrupt signal to interrupt control unit to indicate the data is ready. The microprocessor temporarily stops what it is doing and then returns to what it was doing.

→ Serial I/O Control:

Sometimes, ~~it is~~ I/O device is working with serial data rather than parallel, In this case the serial data stream from an input device must be converted to 8-bit parallel data before computer can use it. Likewise 8-bit data out of a computer must be converted to serial ~~form~~ form before a serial output device can use it.

The SID input is where serial data enters the 8085. The SOD output is where the serial data leaves the 8085. Two instructions (SIM and RIM) allow the user to perform the serial parallel conversion needed for serial I/O devices.

INTERRUPTS

"Interrupt is a mechanism by which the processor is made to transfer control from its current program execution to another program of more importance or higher priority."

- Interrupts are generated by a variety of sources and helps I/O devices to obtain services of CPU.
- The program that is executed upon interrupts is known as Interrupt Service Routine (ISR).

"Why we need interrupts?"

The interrupts are needed due to following reasons:

→ I/O devices are slower than the memory and CPU.
→ Different device requires different amount of time from CPU.
→ Uncertainty of when device will be ready.

Types of Interrupts

- Vectored and Non-vectored Interrupts
- Maskable and Non-maskable Interrupts
- Software and Hardware Interrupts
- Edge triggered and level triggered Interrupt.

- Vectored and Non vectored Interrupts

- "In case of vectored Interrupt the address of service routine is hardwired. When a vectored interrupt arises the microprocessor transfers its control to a predefined memory location which is called the vectored location of that vectored interrupt. At this vectored location the microprocessor gets the address of ISR. In microprocessor 8085 all the interrupts except INTR are vectored interrupts.

- In Non vectored interrupts the address of ISR needs to be supplied externally by the device.
Microprocessor 8085 has INTR as the only non vectored interrupt.

– Maskable and Non Maskable Interrupts

- The maskable interrupts are the interrupts which can be ignored or masked. These interrupts can be kept pending.
- The response time of these interrupts are high.
- These interrupts are generally used to interface the peripheral devices.
- These interrupts may or may not be vectored
Microprocessor 8085 has three maskable interrupts RST 7.5, RST6.5 and RST 5.5.
- A non-maskable ~~inter~~ interrupt (NMI) is an interrupt which can not be ignored or masked. It is mostly used for the attention of system for non-recoverable hardware errors.
- An NMI is generally used when response time is critical and when an interrupt ~~can~~ should not be disabled in the normal operation of the system. NMIs are typically used to handle non-recoverable errors, which need immediate attention.
These interrupts are always vectored.
Microprocessor 8085 has on NMI called TRAP.

- Software and Hardware Interrupts

- Software Interrupts:

→ These interrupts arise in response of an instruction.
→ It is of synchronous nature.
→ This interrupt is requested by executing instruction or due to arithmetic errors.
→ The microprocessor does not execute acknowledge cycle to acknowledge this interrupt. It executes normal instruction cycle.
→ It can not be masked or ignored; it has highest priority among all.
→ It does not affect on interrupt. It is not used to interface peripherals i.e. it does not improve throughput of system.
→ It is used in program debugging.

Microprocessor 8085 has eight software interrupts.
RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6, RST 7.

- Hardware Interrupts:

→ It is an asynchronous event.
→ This interrupt is requested by an external device.
→ The microprocessor executes either interrupt acknowledge cycle or idle machine cycle to acknowledge this interrupt.
→ It can be masked.
→ It affects on interrupt control logic. It is used to interface peripherals in interrupt driven I/O and improves throughput of system.

Microprocessor 8085 has five hardware interrupts.
TRAP, RST 7.5, RST 6.5, RST 5.5, INTR.

- Edge triggered and Level triggered Interrupts.

- Edge triggered interrupts are recognized on the falling or rising edge of the input signal. They are generally used for high priority interrupts and are latched internally inside the processor.
- Level triggered interrupts overcome the problem of latching. In this the requesting device holds the interrupt line at specified logic state till processor acknowledges the interrupts. This type of interrupts can be shared by other devices.

Microprocessor 8085 has RST 6.5, RST 5.5, INTR are level trigged interrupts.

THE 8085 INTERRUPT

- The 8085 interrupt process is controlled by the Interrupt Enable flip-flop, which is internal to the processor and can be set or reset by using software instructions.
- If the flip-flop is enabled and the input to the interrupt signal INTR goes high, the microprocessor is interrupted. This is a maskable interrupt and can be disabled.
- The 8085 has a nonmaskable and three additional vectored interrupt signals as well.
- The 8085 interrupt process can be described in terms of ~~those~~ eight steps:

Step1: The interrupt process should be enabled by writing the instruction EI in the main program. The EI instruction sets the Interrupt Enable flip-flop. The instruction DI resets the flip-flop and disables the interrupt process.

Step2: When the microprocessor is executing a program, it checks the INTR line during the execution of each instruction.

Step3: If the line INTR is high and the interrupt is enabled, the microprocessor completes the current instruction, disables the Interrupt Enable flip-flop and sends a signal called $\overline{\text{INTA}}$. The processor cannot accept any interrupt requests until the interrupt flip-flop is enabled again.

Step4: The signal $\overline{\text{INTA}}$ is used to insert a restart (RST) instruction through external hardware. The RST instruction is a 1-byte call instruction that transfers the program control to a specific memory location on page 00H and restarts the execution at that memory location after executing Step5.

Step5: When the microprocessor receives RST instruction, it saves the memory address of next instruction on the stack. The program is transferred to the CALL location.

Step6: Assuming that the task to be performed is written as a subroutine at the specified location, the processor performs the task. This subroutine is known as service routine.

Step7: The service routine should include the instruction EI to enable the interrupt again.

Step8: At the end of the subroutine, the RET instruction retrieves the memory address where the program was interrupted and continues the execution.

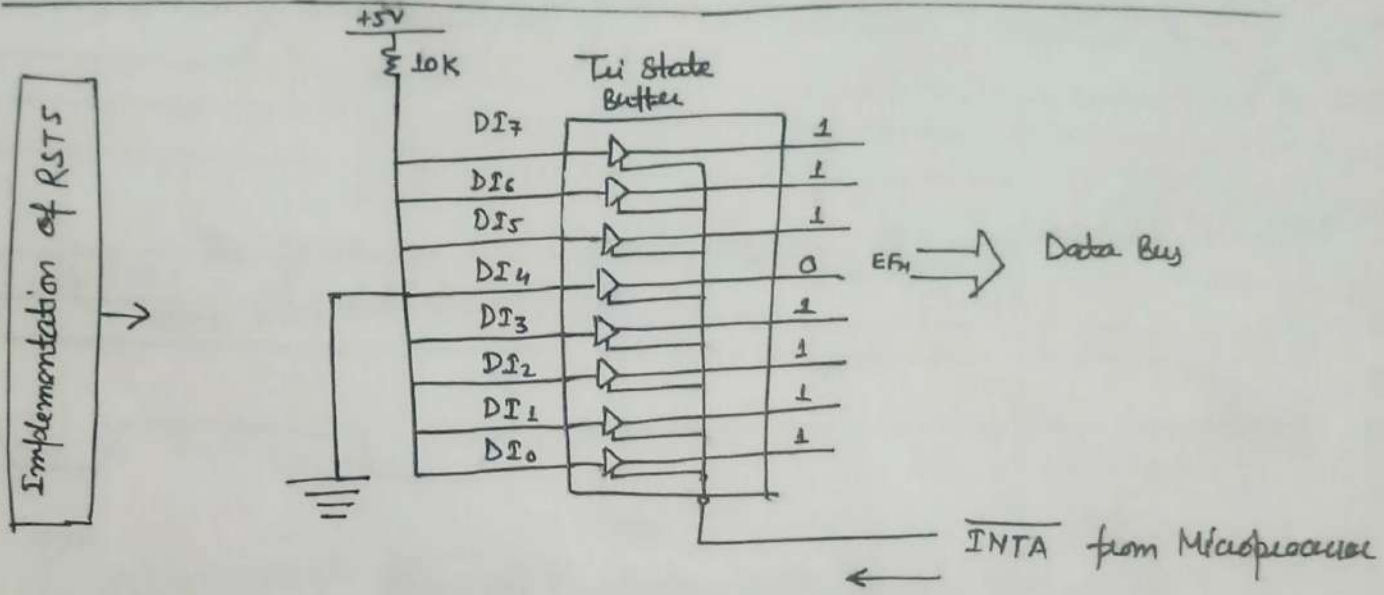
eg: You are reading an interesting novel and you get a call on telephone. There are following steps to receive and respond to a telephone call:

1. The telephone should be enabled.
2. You should glance at the light at certain intervals to check whether someone is calling.
3. If you see a blinking light, pick up the receiver, say hallo and watt for a response. Once you pick up the phone, the line is busy and no more calls can be received until receiver is replaced.
4. Assuming that caller is your roommate and requesting for shutting all the windows in his room.
5. You insert a bookmark on the page you are reading.
6. You replace the receiver on the hook.
7. You shut your roommate's windows.
8. You go back to your book, find your mark and start reading again.

RST (Restart) Instructions

- The 8085 instruction set includes eight RST (Restart) instructions.
- RST is 1-byte call instructions that transfers the program execution to a specific location on page 00H.
- The RST instructions are executed in a similar way to that of Call instructions. The address in the program counter is stored on the stack before the program execution is transferred to the RST call location. When the processor encounters a Return instruction in the subroutine associated with the RST instruction, the program returns the the address that was stored on the stack.
- In case of a hardware interrupt, RST instruction will be used to restart the program execution.
- The RST instructions are built using resistors and a tristate buffer.
- The list of RST instructions is as follows:

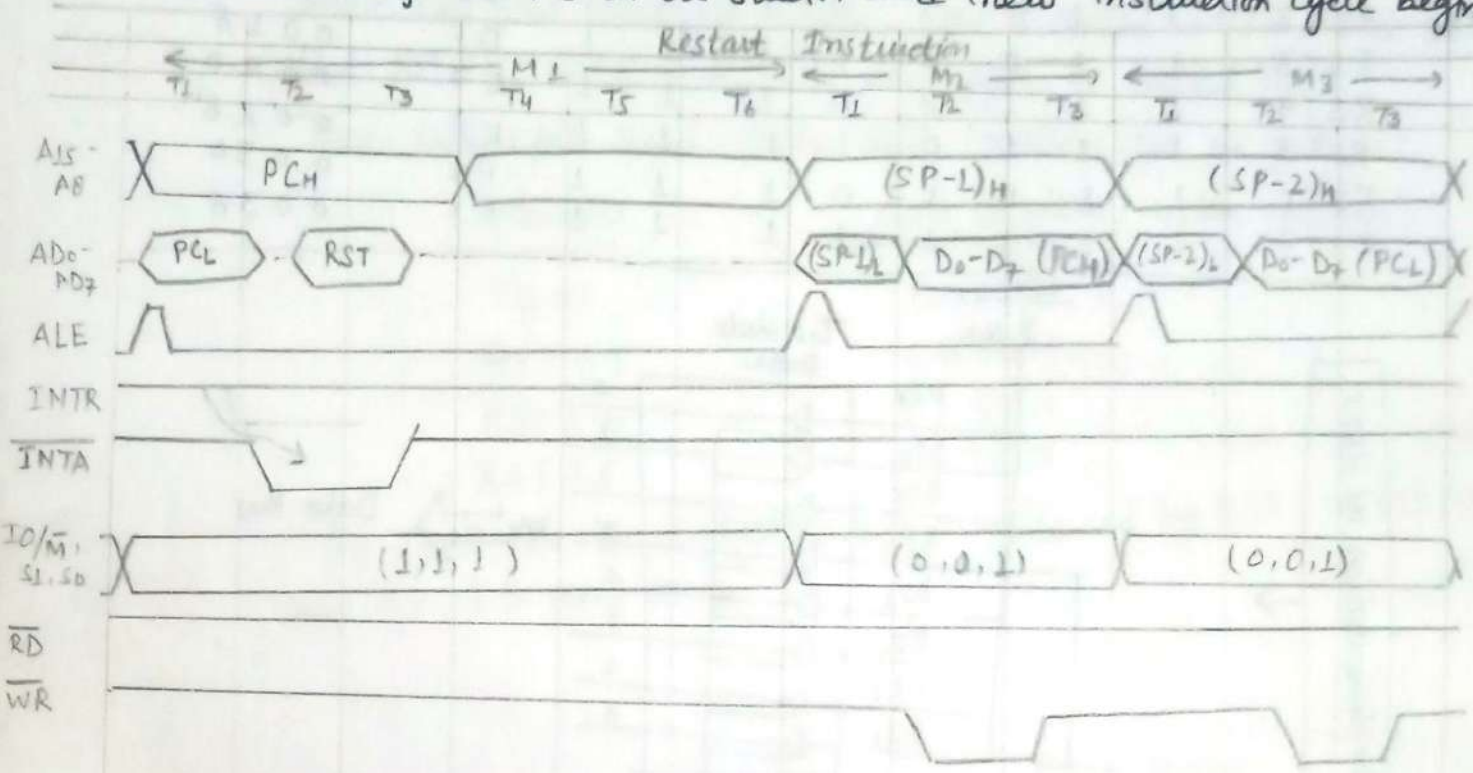
Mnemonics	Binary Code D7	D6	D5	D4	D3	D2	D1	D0	Hex Code	Call location in Hex
RST0	1	1	0	0	0	1	1	1	C7	0000
RST1	1	1	0	0	1	1	1	1	CF	0008
RST2	1	1	0	1	0	1	1	1	D7	0010
RST3	1	1	0	1	1	1	1	1	DF	0018
RST4	1	1	1	0	0	1	1	1	E7	0020
RST5	1	1	1	0	1	1	1	1	EF	0028
RST6	1	1	1	1	0	1	1	1	F7	0030
RST7	1	1	1	1	1	1	1	1	FF	0038



- Execution of RST and Interrupt Acknowledgement

* In response to the INTR high signal, 8085 sends the $\overline{INTA}$ low signal, which is used to enable a buffer and the RST instruction is placed on the data bus during M_1.
* During M_1, the program counter holds the memory address of next instruction, which should be stored on the stack so that the program can continue after the service routine.
* During M2, the address of (SP-1) location is placed on the address bus, and the high order address of the program counter is stored on the stack.
* During M_3, the low order address of the program counter is stored in the next location (SP-2) of the stack.
* The machine cycle M_1 of the interrupt acknowledge is identical with opcode fetch with two exceptions.
  - → The $\overline{INTA}$ signal is sent out instead of the $\overline{RD}$ signal.
  - → The status lines ($IO/\overline{M}$, S_0 and S_1) are 1 1 1 instead of 0 1 1.

- During M1 RST is decoded, 1-byte call instruction to location of interrupt service Routine (ISR).
- The machine cycles M_2 and M_3 are memory write cycles that store the contents of the PC on the stack, and new instruction cycle begins.

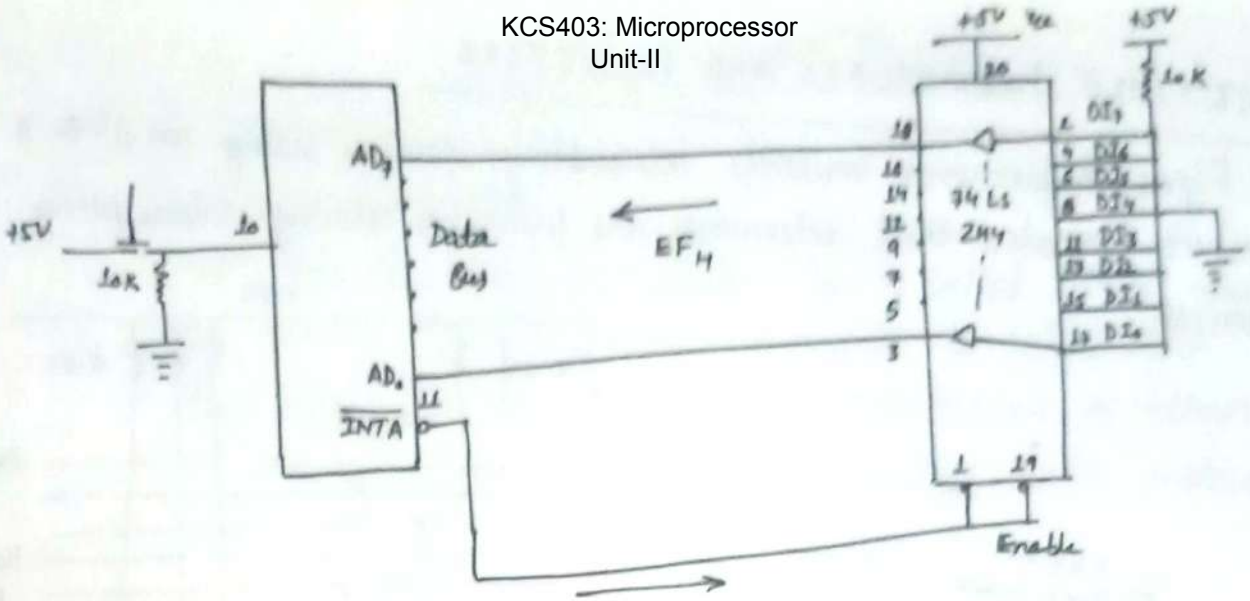


Execution of RST and Interrupt Acknowledge
(Timing diagram)

- IMPLEMENTING 8085 INTERRUPT

- In reality, you have no direct access to restart locations in EPROM or ROM, if the system has already been designed. And in single board micro-computers some restart locations are usually reserved for uses, and the system designer provides a jump ~~imp~~ instruction at a restart location to jump somewhere in R/W memory.

eg:- If instruction RST 5 is inserted, it transfers the program to location 0028H.

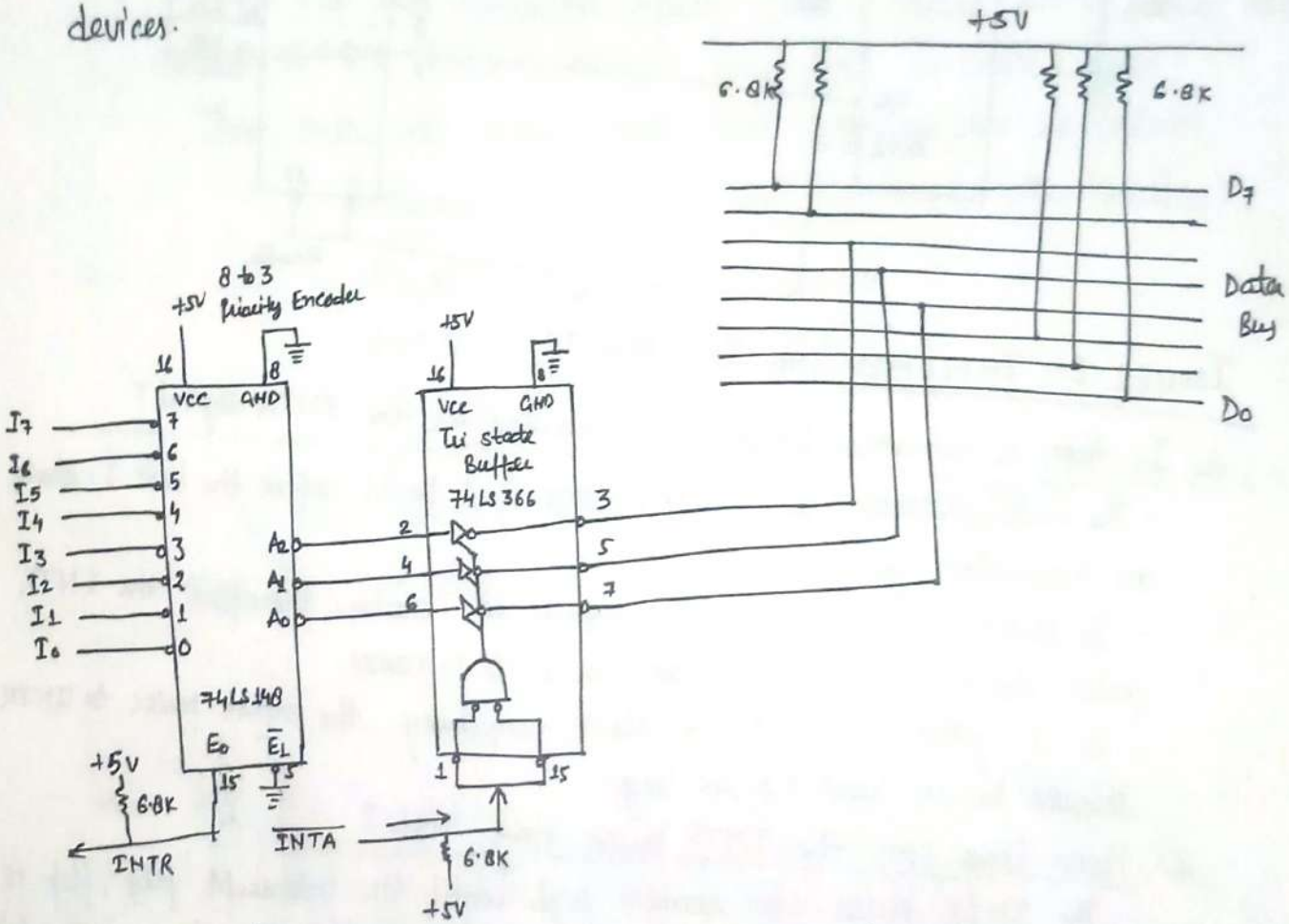


ISSUES IN IMPLEMENTING INTERRUPTS

1. **Is there a minimum pulse width required for the INTR signal?**
   - The microprocessor checks INTR, one clock period before the last T-state of an instruction cycle.
   - In 8085, the call instructions require 18-T states; therefore the INTR pulse should be high at least for 17.5 T-states.
   - In a system with 3 MHz clock frequency, the input pulse to INTR should be at least 5.8 μs long.
2. **How long can the INTR pulse stay high?**
   - The INTR pulse can remain high until the interrupt flip-flop is set by the EI instruction in the service routine. If it remains high after the execution of the EI instruction, the processor will be interrupted again, as if it were a new interrupt.
3. **Can the microprocessor be interrupted again before the completion of the first interrupt service routine?**
   - After the first interrupt, the interrupt process is automatically diabled.
     → If the service routine enables the interrupt at the end of the service routine, the microprocessor can not be interrupted before the completion of this routine.
     → If instruction EI is written at the beginning of the routine the microprocessor can be ~~written~~ interrupted again during service routine.

MULTIPLE INTERRUPTS AND PRIORITIES

- Figure implements multiple interrupting devices using an 8-to-3 priority encoder that determines the priorities among interrupting devices.



- If we examine the instruction code for eight RST instructions, we will notice that bits D_5, D_4, D_3 change in a binary sequence and that the others are always at logic 1.
- The encoder provides ~~appo~~ appropriate combinations on its output lines A_0, A_1 and A_2. which are connected to the data lines D_3, D_4 and D_5 through a tri-state buffer. The eight inputs to the encoder are connected to eight different interrupting devices.
- When an interrupting device requests service, one of the input lines goes low, which makes line E_0 high and ~~&~~ interrupts the microprocessor.
- When the interrupt is acknowledged and signal $\overline{INTA}$ enables the tri-state buffer, the code corresponding to the input is placed on D_5, D_4, D_3.
  For example, for RST 5, line I_5 goes low, the o/p of ~~deco~~ encoder will be 010. This code is inverted by buffer and combined with other data lines.

11 101 111

I_5

8085 VECTORED INTERRUPTS

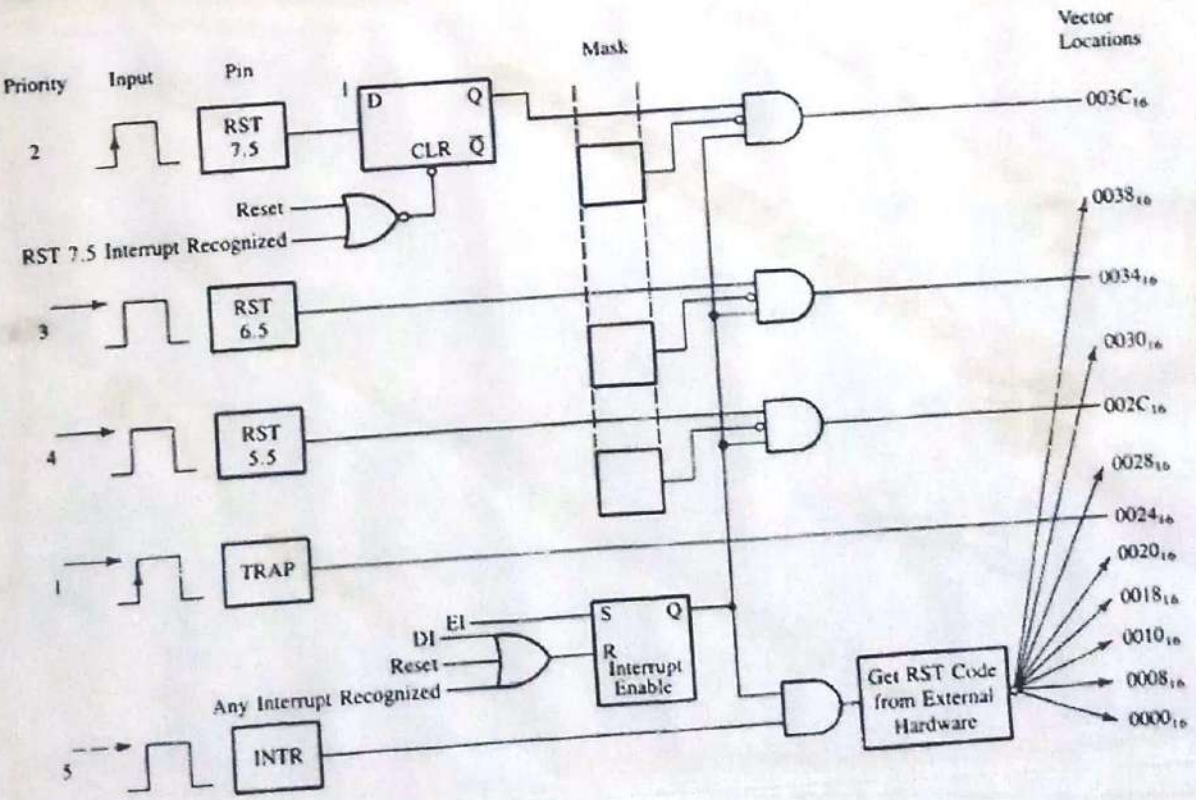
- 8085 Interrupt structure

(i.) Hardware Interrupts

- 8085 has five interrupts inputs. One is called INTR, three are called RST 7.5, RST 6.5, RST 5.5 and fifth is called TRAP.
- These interrupts and their call locations are as follows:

Interrupts	Call locations (Vector locations)
TRAP	0024H
RST 7.5	003CH
RST 6.5	0034H
RST 5.5	002CH

- The TRAP has the highest priority, followed by RST 7.5, RST 6.5, and RST 5.5 and INTR.



- TRAP

- TRAP is non-maskable interrupt. It is unaffected by any mask or interrupt enable.
- It has the highest priority among the interrupt signals, it need not be enabled, and it cannot be disabled.
- TRAP is level and edge sensitive. This means that the TRAP input should go high and stay high to be acknowledged.
- TRAP is generally used for such critical events as power failure and emergency shut-off. In sudden power failure, it executes a ISR and send the data from main memory to backup memory.
- The signal which overrides the TRAP, is HOLD signal.
- There are two ways to clear TRAP interrupt
  1. By resetting microprocessor
  2. By giving a high TRAP ACKNOWLEDGE.

- RST 7.5

- The RST 7.5 interrupt is a maskable interrupt.
- It has second highest priority.
- RST 7.5 interrupt is maskable interrupt and enabled under program control with two instructions:
  1. EI instruction
  2. SIM instruction
- RST 7.5 interrupt is disabled by
  1. DI instruction, SIM instruction
  2. System or processor reset
  3. After recognition of interrupt.
- This is positive-edge sensitive and can be triggered with a short pulse. The request is stored internally by D-flip-flop until the microprocessor responds to the request or until it is cleared by Reset or by bit D_4 in the SIM instruction.

- RST 6.5 and RST 5.5

- The RST 6.5 has the third priority whereas RST 5.5 has the fourth priority.
- RST 6.5 and RST 5.5 both are maskable interrupt; and Enabled by EI or SIM instruction.

These interrupts are disabled by
1. DI instruction, SIM instruction
2. System or processor reset.
3. After recognition of interrupt

- These are level triggered, meaning that the triggering level should be on until the microprocessor completes the execution of the current instruction. If microprocessor is unable to respond to these requests immediately, they should be stored or held by external hardware.

- INTR
  - It has lowest priority.
  - It is non-vectored interrupt. After receiving $\overline{INTA}$ (Active low) signal, it has to supply the address of ISR.
  - INTR is a maskable interrupt and Enabled by EI instruction.
  - INTR is disabled by
    1. DI, SIM instruction
    2. System or processor reset
    3. After recognition of interrupt.
  - It is a level triggered interrupt, means that input goes to high and it is necessary to maintain high state until it recognized.
  - The following sequence of events occurs when INTR signal goes high:
    1. The 8085 checks the status of INTR during execution of each instruction.
    2. If INTR signal is high, then 8085 complete its current instruction and sends active low interrupt acknowledge signal, if the interrupt is enabled.
    3. In response to the acknowledge signal, external logic places an instruction code on the data bus. In the case of multibyte instruction, additional interrupt acknowledge machine cycles are generated by the 8085 to transfer the additional bytes into the microprocessor.
    4. On receiving the instructions the 8085 save the address of next instruction on stack and execute received instruction.

(ii) Software Interrupts

- The software interrupts are program instructions. These instructions are inserted at desired locations in a program.
- The 8085 has eight software interrupts from RST0 to RST7. The vector address for these interrupts can be calculated as follows:
  → (Interrupt number × 8) = Vector address

eg: For RST 3,
Vector address = $3 \times 8 = (24)_{10} = (18)_{16}$
Vector address for RST 3 = 0018_H

The table shows the vector addresses of all interrupts.

Interrupt	Vector Address
RST 0	0000_H
RST 1	0008_H
RST 2	0010_H
RST 3	0018_H
RST 4	0020_H
RST 5	0028_H
RST 6	0030_H
RST 7	0038_H

MASKING / UNMASKING OF INTERRUPTS

- ~~Masking~~
- Maskable interrupts are enabled and disabled under program control.
- Maskable interrupts are enabled with two instructions:
  → EI (Enable Interrupt)
  → SIM (Set Interrupt Mask)

→ EI (Enable Interrupt) Instruction:

- This is a 1-byte instruction
- The instruction sets the Interrupt Enable flip-flop and enables the interrupt process
- System reset or an interrupt disables the interrupt process.

- DI (Disable Interrupt) Instruction:

- This is a 1-byte instruction.
- The instruction resets the Interrupt Enable flip-flop and disables the interrupts.
- It should be included in a program segment where an interrupt from an outside source cannot be tolerated.

PENDING INTERRUPTS

- Because there are several interrupt lines, when one interrupt is being served, other interrupt requests may occur and remain pending. The 8085 has an additional instruction called RIM (Read Interrupt Mask) to sense these pending interrupts.
- The instruction RIM checks for a pending interrupt.
- RIM (Read Interrupt Mask): Instruction
  → This is a 1-byte instruction that can be used for following function:
  * To read interrupt mask. This instruction loads the accumulator with 8-bits indicating the current states of interrupt masks.
  * To identify pending interrupts: Bits D_4, D_5, and D_6 identify the pending interrupts.
  * To receive serial data - Bit D_7 is used to receive serial data

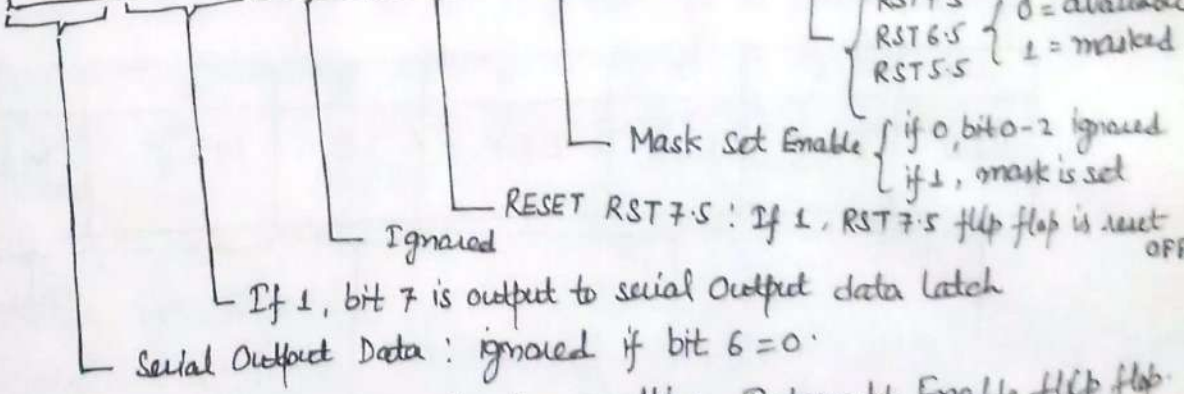
7	6	5	4	3	2	1	0
SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5

- Interrupt Masks: 1 = masked
- Interrupt Enable Flag: 1 = enabled
- Pending Interrupts: 1 = pending
- Serial Input data bit, if any

→ SIM (Set Interrupt Mask) Instruction:

- This is a 1-byte instruction and can be used for three different functions.
  - → One function is to set mask for RST 7.5, RST 6.5, RST 5.5 interrupts. This instruction reads the contents of the accumulator and enables or disables the interrupts according to the content of accumulator. Bit D_3 is a control bit and should = 1 for bits D_0, D_1, and D_2 to be effective. Logic 0 on D_0, D_1 and D_2 will enable the corresponding interrupts, and logic 1 will disable the interrupt.
  - → The second function is to reset RST 7.5 flip-flop. Bit D_4 is additional control for RST 7.5. If $D_4 = 1$ RST 7.5 is reset. This is used to override RST 7.5 without servicing it.
  - → The Third function is to implement serial I/O. Bits D_7 and D_6 of the accumulator are used for serial I/O and do not affect the interrupts. Bit $D_6 = 1$ enables the serial I/O and bit D_7 is used to transmit (output) bits.
- The status of these interrupts can be read by executing RIM instruction.
- The format of the 8-bit data is shown below:

7	6	5	4	3	2	1	0
SOD	SDE	XXX	R7.5	MSE	M7.5	M6.5	M5.5



- Maskable interrupts are disabled ~~with~~ by resetting Interrupt Enable flip flop. The flip-flop is reset in one of the three ways:
  1. Instruction DI
  2. System Reset
  3. Recognition of an Interrupt Request

INSTRUCTION SETS

- An instruction is a binary pattern designed inside a microprocessor to perform a specific function.
- The entire group of instructions called the instruction set. The group of instructions determine what functions the microprocessor can perform.
- The 8085 instructions can be classified into five functional categories:
  - → Data transfer operations
  - → Arithmetic operations
  - → Logic operations
  - → Branching operations
  - → Machine Control operations

→ Data Transfer (copy) operations:

* This group of instructions copies data from a location to another location without modifying the content of source.
* The term data transfer is used for this copying function.
* The various types of data transfer are:
  - ⇒ Between Registers
  - ⇒ Specific data byte to a register or a memory location
  - ⇒ Between a memory location and a register.
  - ⇒ Between an I/O device and the accumulator.
* Instructions are, MOV, MVI, LDA, STA.

→ Arithmetic operations:

These instructions perform arithmetic operations such as addition, subtraction, increment and decrement.

* Addition: Any 8 bit number, or contents of register, or the contents of a memory location can be added to the contents of accumulator and result is stored in accumulator. No two other 8-bit registers can be added directly.

Date: _____ Page: _____

* Subtraction: Any 8-bit data or contents of a register or the contents of a memory location can be subtracted from the contents of accumulator. The result is stored in the accumulator.

The subtraction is performed in 2's complement form. No two other 8-bit register can be subtracted directly.

* Increment/ Decrement:

- The 8-bit contents of any memory location or any register can be directly incremented or decremented by 1.
- Similarly 16 bit contents of a register pair can be incremented or decremented by 1.
- Increment/ Decrement operation can be performed in any one of the registers or memory location.

→ Logical Operations:

- These instructions perform logic operations on the contents of accumulator.

⇒ AND, OR, Exclusive-OR:

Source: Accumulator and An 8 bit number or the contents of a memory location.

Destination: Accumulator.

⇒ Rotate: Each bit in accumulator can be shifted either left or right to next position.

⇒ Compare: Any 8-bit or the content of a register, or the contents of memory location compared with the content of accumulator.

⇒ Complement: The contents of accumulator can be complemented all 0s and 1s are replaced by 1s and 0s respectively.

→ Branch Operations:

These operations alter the sequence of program execution

either conditionally or unconditionally.

* Jump: Conditional jumps are an important aspect of the decision making process in programming. The instructions test for a certain condition and alter the program sequence accordingly. In addition, the instruction set includes an instruction called unconditional jump.

* Call, Return and Restart: These instructions change the sequence of a program either by calling a subroutine or returning from a subroutine. The conditional Call and Return instruction also can reset condition flag.

→ Machine Control Operation:

These instructions control machine functions such as

* Halt: Stop executing the program.
* Interrupt
* Do nothing: No operation is done, usually used for delay or to replace instructions during debugging.

- Addressing Modes

- To perform any operation, we have to give the corresponding instructions to the microprocessor.
- In each instruction, programmer has to specify 3 things:
  - → Operation to be performed
  - → Address of source of data
  - → Address of destination of result
- The method by which the address of source of data or the address of destination of data (result) is given in the instruction is called addressing mode.

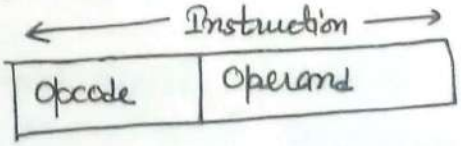
or.

The term addressing mode refers to the way in which the operand of instruction is specified.

The 8085 has the following five types of addressing:

(i.) Immediate addressing mode
(ii.) Direct addressing mode
(iii.) Register direct addressing mode
(iv.) Register indirect addressing mode
(v.) Implicit addressing mode

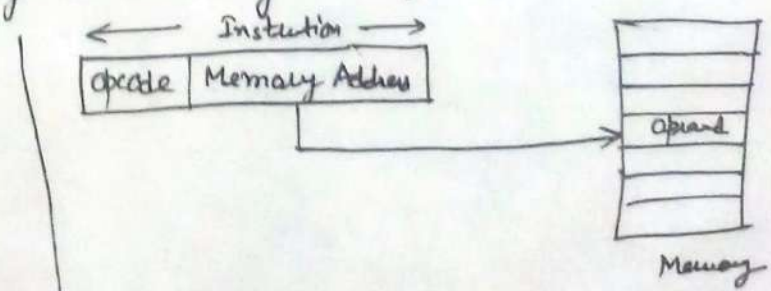
(i.) **Immediate addressing mode:** In this mode, the operand is specified within the instruction itself. The format of immediate addressing mode is as



eg: MVI A, 9AH

(ii.) **Direct addressing mode:** In this mode, the address of the operand is given in the instruction itself. The instruction set does not support memory to memory transfer.

eg:
LDA 2500H

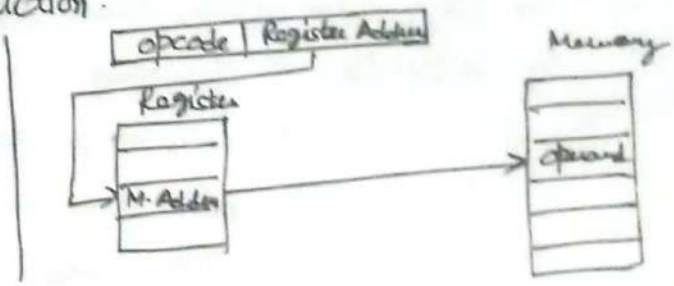


(iii.) Register direct addressing mode: In this mode, a copy of a byte or word from the source register is transferred to destination register. The operand is in the register named in the instruction. It executes very fast.

eg: MOV A,B
(B ← Source, A ← destination)

(iv.) Register indirect addressing mode: In this mode, a byte is transferred between a register and a memory location. The address of a memory location is stored in a register and that register is specified in the instruction.

eg: MOV A,M



(V.) Implicit addressing mode: In this mode, the instruction itself specifies the data to be operated upon.

eg: CMA.

Instruction Formats

"Instruction is a command to microprocessor to perform a given task on specified data."

Each instruction has two parts.

* The first part is task or operation to be performed. This part is known as opcode (operation code).
* The second part is the data to be operated on called the operand. Operand can be specified in a number of ways.

The instructions are classified into three groups:

→ 1-byte instruction: Includes the opcode and operand in the same byte.

eg:- MOV C, A
ADD B

→ 2-byte instruction, the first byte specifies the operation code and the second byte specifies the operand.

eg: MVI A, 30H

→ 3-byte instruction, the first byte specifies the opcode and following two bytes specify 16-bit address.

eg:- JMP 2085 H

Opcode Format: To understand operation codes, we need to examine how an instruction is designed into the microprocessor.

In design of 8085 microprocessor chip all operations, registers and status flags are identified with a specific code.

eg:- For all internal Registers.

Code	Register
000	B
001	C
010	D
011	E
100	H
101	L
110	Reserved for memory related operations
111	A

Code	Register Pair
00	BC
01	DE
10	HL
11	AF OR SP.

Some of the operation codes are identified as

Function	Operation Code
1. Rotate Accumulator to left by one position	00000 111 = 07 H
2. Add content of register to the accumulator.	10000 SSS
3. MOVE the content of register Rs to register Rd.	01 DDD SSS

where SSS ⇒ Source Register, DDD ⇒ Destination Register

eg:-

ADD B	10000 000	= 80H
MOV C, A	01 001 111	= 4FH

Data Format:- Microprocessor processes data only in binary form. However, in real world decimal and alphabets are used.

- In 8-bit microprocessor, data can be represented in following formats:
  - → ASCII
  - → BCD Code
  - → Signed Integer
  - → Unsigned Integer

→ ASCII:
- 7-bit alphanumeric code
- Represents decimal number, English alphabets, and non printable characters.

→ BCD Code:
- Used for decimal numbers having ten digits 0 to 9.
- We need 4 bits to represent ten digits from 0000 to 1001. Remaining 6 numbers are invalid.
- A register can accomodate two BCD numbers.

→ Signed Integers:
- It is either (+ve) or (-ve) number.
- In 8-bit processor D_7 is used as sign bit and all other bits are used to represent magnitude of an integer.
- All negative numbers are represented in 2's complement form.

→ Unsigned Integers:
- An integer without a sign can be represented by all the 8-bits in a microprocessor register.
- The largest number processed at a time is FFH.

INSTRUCTION CLASSIFICATION

The entire group of instructions that a microprocessor supports is called instruction set. There are 246 instructions (74 types) in the 8085 microprocessor. Based on the function of the instruction, the instructions are classified into the following five types:

1. Data transfer instruction
2. Arithmetic instruction
3. Logic and bit manipulation instructions
4. Branch instruction
5. Machine control instructions

1. Data Transfer instruction:

These instructions move data between registers or between memory and registers. These instructions copy data from source to destination. While copying, the contents of source are not modified. Data transfer instructions do not affect any of the flag.

- MOV Rd, Rs
  M, Rs
  Rd, M

  } copy from source to destination. if one of the operand is a memory location, its location is specified by the contents of HL Register.

eg:- MOV B,C MOV B, M

- MVI Rd, Data
  M, Data

  } The 8-bit data is stored in destination register or memory.
  - Memory location is specified by contents of HL Register

eg:- MVI B, 57H MVI M, 57H

- LDA 16-bit data } Load Accumulator. The content of memory location specified by 16-bits address are copied to the accumulator.

eg:- LDA 8000H

- LDAX B/D Register Pair } Load accumulator indirect

The contents of designated register pair point to a memory location. This instruction copies the contents of memory location into accumulator.

eg: LDAX B LDAX D.

- LXI Register Pair, 16-bit data } Load register pair immediate

This instruction loads 16-bit data in the register pair.

eg: LXI H, 8050H

- LHLD 16-bit address } Load HL Register direct

This instruction copies the contents of memory location pointed out by 16-bit address into register L and contents of next memory location into register H.

eg: LHLD 8040H

- STA 16-bit address } Store accumulator direct.

The contents of accumulator are copied into the memory location specified by operand.

eg: STA 2500H

- STAX Register Pair } Store accumulator indirect

The contents of accumulator are copied into the memory location specified by the contents of register pair.

eg: STAX B

- SHLD 16-bit address } Store H-L registers direct

The contents of register L are stored into memory location specified by the 16-bit address. The contents of register H are stored into the next memory location.

eg: SHLD 2050H

- XCHG } Exchange HL with DE

The contents of register H are exchanged with contents of register D. and contents of register L are exchanged with contents of register E.

eg: XCHG

- SPHL } Copy H-L pair to the stack pointer (SP)

eg: SPHL

- XTHL } Exchange HL with to of the stack.

The contents of L register are exchanged with the location pointed out by the SP; and contents of H registers are exchanged with the next location SP+1.

eg:- XTHL

- PCHL } Load program counter with HL contents

The contents of register H and L are copied into the PC. The contents of H are placed as the high order byte and contents of L as low order byte

eg:- PCHL

- PUSH Register Pair } Push Register Pair onto stack.

The contents of register pair are copied onto the stack. SP is decremented and contents of high order registers (B, D, H, A) are copied into stack. SP is again decremented and the contents of low order registers (C, E, L, Flags) are copied into stack.

eg:- PUSH B

- POP Register Pair } Pop stack to register pair

The contents of top of stack are copied into register pair. The contents of location pointed out by SP are copied to the low order register (C, E, L, Flags). SP is incremented and the contents of location are copied to the high order register (B, D, H, A).

eg:- POP H.

- IN 8-bit port address } copy data to accumulator from a port with 8-bit address

The contents of I/O port are copied into accumulator.

eg:- IN 8CH

- OUT 8-bit port address } copy data from accumulator to a port with 8-bit address

The contents of accumulator are copied into the I/O port

eg:- OUT 78H.

2. Arithmetic Instructions

Addition
→ Any 8-bit number or the contents of Register or contents of memory location can be added to the contents of Accumulator
→ Result is stored in Accumulator

Subtraction
→ Any 8-bit number or the contents of Register or contents of memory location can be subtracted from the contents of Accumulator
→ Result is stored in Accumulator
→ Subtraction is performed in 2's complement form
→ If result is (-ve), it is stored in 2's complement form.

Increment/Decrement
→ 8 bit content of a register or a memory location can be incremented/decremented by 1.
→ 16-bit contents of register pair can be incremented/decremented by 1.
→ Increment/Decrement can be performed on any register or a memory location.

All Flags are modified to reflect the result of addition.

- ADD R / M — Add register or memory contents to accumulator
- ADC R / M — Add register or memory contents and Carry Flag (CY) to the content of accumulator.
- ADI 8-bit data — The 8-bit data is added to the contents of accumulator.
- ACI 8-bit data — The 8-bit data and carry Flag (CY) are added to the contents of accumulator.

DAD Reg. Pair
- The 16 bit Contents of register pair are added to contents of HL pair.
- If result is larger than 16 bits then CY is set.
- No other flags are changed.

All Flags are modified to reflect the result of subtraction.

- SUB R / M — Subtract register or memory contents from accumulator
- SBB R / M — The contents of register or memory location and borrow Flag (CY) are subtracted from the contents of accumulator.
- SUI 8-bit Data — The 8-bit data is subtracted from the contents of accumulator.
- SBI 8-bit Data — The 8-bit data and the Borrow Flag (CY) subtracted from the contents of the accumulator.

All flags except CY flag are affected.

- INR R/M } The contents of register or memory location are incremented by 1
- INX Rp } The contents of register pair are incremented by 1.
- DCR R/M } The contents of register or memory location are decremented by 1.
- DCX Rp } The contents of register pair are decremented by 1

3. Logical Instructions:

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

- **AND, OR, XOR**
  → Any 8-bit data, or the contents of register or memory location can logically perform these operations with the contents of accumulator.
  → The result is stored in accumulator.
  → S, Z, P flags are modified to reflect the result of operation.
  → * CY is reset and AC is set in AND operation
  ** CY and AC are reset in OR and XOR operations.

* - ANA R/M } The contents of accumulator are logically ANDed with the contents of register or Memory
* - ANI 8-bit data } The contents of accumulator are logically ANDed with 8-bit data
** - ORA R/M } The contents of accumulator are logically ORed with the contents of register or memory.
** - ORI 8-bit data } The contents of accumulator are logically ORed with 8-bit data.
** - XRA R/M } The contents of accumulator are logically XORed with the contents of register or memory.
** - XRI 8-bit data } The contents of accumulator are logically XORed with 8-bit data.

• **Rotate**

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

→ S, Z, P, AC are not affected.

• RAL } Rotate accumulator left through carry. Bit D_7 is placed in the carry and carry flag is placed in the least significant position D_0. The Cy flag is modified according to bit D_7.

• RAR } Rotate accumulator right through carry. Bit D_0 is placed in the carry flag and carry flag is placed in the most significant position D_7. The Cy flag is modified according to bit D_0

• RLC } Rotate accumulator left. Bit D_7 is placed in the position of D_0 as well as Carry flag. Cy is modified according to bit D_7

• RRC } Rotate accumulator right. Bit D_0 is placed in position of D_7 as well as in carry flag. Cy is modified according to bit D_0

• **Compare**

→ Any 8-bit data, or the contents of register, or memory location can be compared with the contents of accumulator for equality, Greater than, Less than.

→ The result is reflected in status flags.

if (A) < (reg / Memory) or if (A) < data :
Carry Flag is set

if (A) = (reg / Memory) or if (A) = data :
Zero Flag is set

if (A) > (reg / Memory) or if (A) > data :
Carry and Zero flags are reset

→ The values being compared remain unchanged

• CMP R } The contents of register or memory are compared
M } with the contents of accumulator.

• CPI 8-bit data } The 8-bit data is compared with the contents of accumulator.

• Complement
→ The contents of accumulator are complemented. Each 0 is replaced by 1 and each 1 is replaced by 0.
- CMA } The contents of accumulator are complemented. No flags are affected
- CMC } The carry flag is complemented and No other flags are affected
- STC } The carry flag is set to 1 and No other flags are affected.

4. Branch instruction
The branch instruction alter the normal sequential flow. These instruction transfers the control of program from one location to another location conditionally or unconditionally.
- JMP 16-bit Address } Jump unconditionally

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand.
~~- JMP~~
- Jx 16-bit Address } Jump Conditionally

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on specified flag of the PSW.
The four flags used by the Jump instructions are:
1. Carry Flag
2. Zero Flag
3. Sign Flag
4. Parity Flag

Two ~~jumps~~ Jump instructions are associated with each flag. The sequence of a program can be changed either because the condition is present or absent.
The following instructions transfer the program sequence to the memory location specified under the given conditions:

JC	16-bit	Jump on Carry	CY = 1
JNC	16-bit	Jump on No Carry	CY = 0
JZ	16-bit	Jump on Zero	Z = 1
JNZ	16-bit	Jump on No Zero	Z = 0
JP	16-bit	Jump on Plus	S = 0
JM	16-bit	Jump on Minus	S = 1
JPE	16-bit	Jump on Even Parity	P = 1
JPO	16-bit	Jump on Odd Parity	P = 0

- CALL 16-bit addess } Call unconditionally

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand.
Before the transfer, the address of next instruction after CALL is pushed onto the stack.
Conditional Call :- Conditional call instructions are based on four data conditions : Carry, Zero, Sign, and Parity.
In case of conditional call instruction, the program is transferred to the subroutine if condition met; otherwise the main program is continued.

CC	Call subroutine if Carry flag is set	CY = 1
CNC	Call subroutine if carry flag is not set (reset)	CY = 0
CZ	Call subroutine if zero flag is set	Z = 1
CNZ	Call subroutine if zero flag is reset	Z = 0
CM	Call subroutine if Sign flag is set	S = 1
CP	Call subroutine if Sign flag is reset	S = 0
CPO	Call subroutine if Parity flag is reset	P = 0
CPE	Call subroutine if Parity flag is set	P = 1

- RET } Return unconditionally

The program sequence is transfered from the subroutine to calling program.
The top two bytes from the top of the stack are copied into the PC; and program execution begins at the new address.

Conditional Return: Instructions are based on four data conditions (flags) CY, Z, S, P.
In case of conditional return instruction, the sequence returns to the main program if condition met; otherwise the sequence in subroutine is continued.

RC	Return if Carry flag is set	CY = 1
RNC	Return if Carry flag is reset	CY = 0
RZ	Return if Zero flag is set	Z = 1
RNZ	Return if Zero flag is reset	Z = 0
RM	Return if Sign flag is set	S = 1
RP	Return if Sign flag is reset	S = 0
RPE	Return if Parity flag is set	P = 1
RPO	Return if Parity flag is reset	P = 0

5. Machine Control instruction

Instructions that control machine functions are known as machine control instruction.
There are six basic machine control instructions. They are
→ EI (Enable Interrupt) → DI (Disable Interrupt)
→ NOP (No operation) → HLT (Halt)
→ SIM (Set Interrupt Mask) → RIM (Read Interrupt Mask).

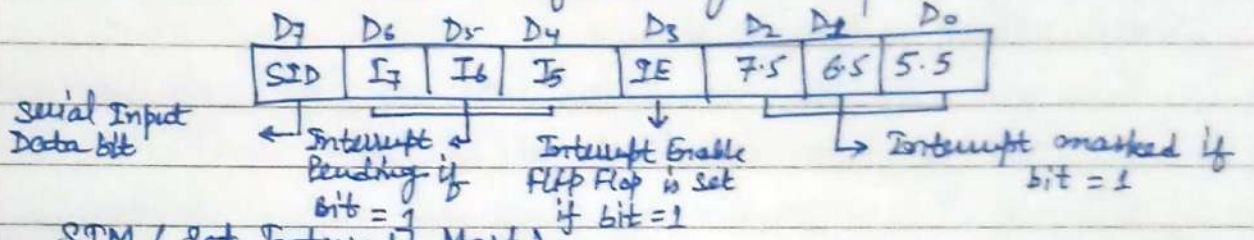
→ EI (Enable Interrupt): This is a one byte instruction used to enable the interrupt. This is used to enable the interrupt when the microprocessor is reset or the interrupt enable flag is reset after interrupt acknowledge. This instruction takes one machine cycle with four states.

→ DI (Disable Interrupt): This is a one byte instruction which resets the interrupt enable flag to disable all the interrupt except TRAP.

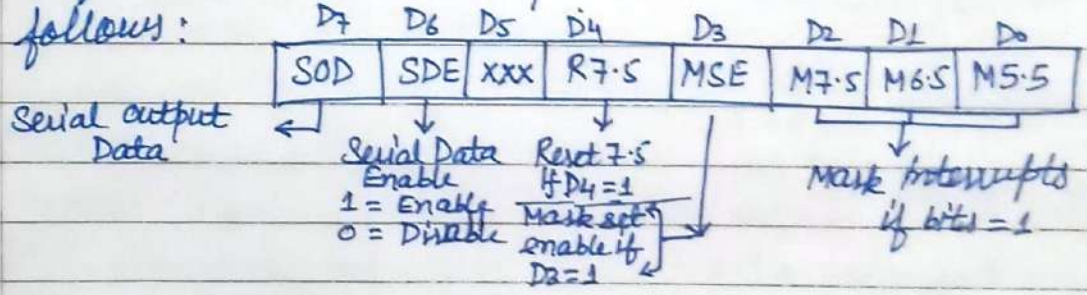
→ NOP (No operation): It is one byte instruction. No operation is performed. The instruction is fetched and decoded. No operation is executed. No Flags are affected during execution of NOP. This instruction is used to fill in time delay or to delete and insert instructions while troubleshooting.

→ HLT (Halt): The microprocessor finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state.

→ RIM (Read Interrupt Mask): This is a multipurpose instruction used to read the status of interrupts ~~7.5~~, RST 7.5, RST 6.5, RST 5.5 and read serial data input bit. The instruction loads eight bits in the accumulator with the following interpretations.



→ SIM (Set Interrupt Mask): This is a multipurpose instruction and used to implement the 8085 interrupts 7.5, 6.5, 5.5 and serial data output. The instruction interprets the accumulator contents as follows:



Assembler Directives

Assembler: It is a tool for developing programs with the assistance of computer. Assemblers are essential for writing industry standard software.

"Assembler is a program that turns symbols into machine instructions. Additionally it also allocates storage and initializes the data."

Assembler Directives: Assembler directives are the instructions to assembler concerning the program being assembled; and also called pseudo instructions or pseudo opcodes.

These are neither translated into machine code nor assigned any memory location in the object file.

Some of the Assembler directives are grouped in two different groups:

General Directives

→ Symbol Definition

- Assembler automatically assigns values to symbols that appear as instruction labels.
- It is possible to define symbols and assign them values by using EQU and SET directives.

• **EQU Directive**: → It assigns the value of 'expression' to the name specified in the label field.

→ Symbol defined by EQU can not be redefined during assembly.

•

label	opcode	operand	
name	EQU	expression	eg: ONES EQU OFFH

• SET Directive: → It assigns the value of 'expression' to the name specified.
→ The assembler enters the value of 'expression' into the symbol table. Whenever 'name' is encountered subsequently in the assembly, the assembler substitutes its value from the symbol table.
→ The value remains unchanged until altered by a subsequent SET directive.

label	opcode	operand	eg:	IMMED SET 5
name	SET	expression		ADI IMMED

→ Data Definition

These directives enables you to define data to be stored. Data can be specified in the form of 8-bit or 16-bit values or as a string of text.

• DB Directive: → It stores the specified data in consecutive memory locations.

label	opcode	operand
optional:	DB	expression(s) or string(s)

→ The operand field of DB can contain a list of expressions or strings. The list can contain upto eight total items, list items must be separated by commas.
→ Expression must evaluate to 1-byte number. Text strings can consist of maximum of 128 ASCII characters.
→ If optional label is present, it is assigned the starting value of the location counter and thus references the first byte stored by the DB directives.
eg:

```
STR:    DB      'TIME'
```

Label STR refers to the letter T of TIME.

• DW Directive: → DW directive stores 16-bit value from the expression list as an address. The values are stored starting at the current setting of the location counter.

label	opcode	operands
optional:	DW	expression list

→ In expression list least significant eight bits of the first value in the expression list are stored at current setting of location counter; the most significant 8 bits are stored at next higher location. This process is repeated for each item in expression list.
Expressions evaluate to 1 word (16-bit).

→ List items must be separated by commas. List can contain 8 items.
→ If the optional label is present, it is assigned the starting address of the location counter and thus references the first byte stored by the DW directive.

eg: ADDR1: DW COMP
COMP ⇒ label defined elsewhere.

→ Memory Reservation

• **DS Directive:** → It is used to define a block of storage.

Label	opcode	operand
optional:	DS	expression

→ The value of 'expression' specifies the numbers of bytes to be reserved for data storage.
→ Any symbol appearing in the operand expression must be defined before the assembler reaches the DS directive.
→ DS assembels no data into the program. The contents of the reserved storage are unpredictable when program execution is initiated.
→ If the optional label is present, it is assigned the current value of the location counter and references the first byte of the reserved memory block.
→ If the value of operand is zero, no memory is reserved.
→ The DS directive reserves memory by incrementing the location counter by the value of the operand expression.

```
TTYBUF: DS 72 ; Reserves 72 bytes for a
              ; terminal output buffer.
```

→ Conditional Assembly:

Conditional Assembly is especially useful when your application requires custom programs for a number of common options.

• **IF, ELSE, ENDIF Directives:**
→ These directives enable you to assemble portions of your program conditionally.

Because these directives are used in conjunction, they are described as,

label	opcode	operand
optional:	IF	expression
optional:	ELSE	--
optional:	ENDIF	--

→ The assembler evaluates the expression in the operand field of IF directive. If bit 0 of the resulting value is one, all instructions between IF directive and the next ELSE or ENDIF directive are assembled. When bit 0 is zero these instructions are ignored.

⇒ Operands are not allowed with the ELSE and ENDIF directives.

⇒ IF-ENDIF block ~~may~~ can be nested to eight levels.

⇒ Macro definitions may appear within IF-ENDIF block and vice versa

eg: 1.
```
COND1: IF TYPE EQ 0
       :
       ENDIF
```

2.
```
COND2: IF TYPE EQ 0
       :          ; TRUE
       ELSE
       :          ; FALSE
       ENDIF
```

→ Assembler Termination

• END Directive: → It identifies the end of the source program and terminates each pass of the assembler.

label	opcode	operand
optional:	END	expression

⇒ Only one END statement may appear in a source program and it must be the last source statement.

⇒ If expression is present, its value is used as the starting address for program execution. If no expression is given the assembler assumes zero as the starting address.

Location Counter Control and Relocation

- **Location Counter Control:** The location counter performs the same function as the program counter. It tells the assembler the next memory location available for instruction or data assembly.

Initially, it is set to zero.

- **ORG Directive:** → It sets the location counter to the value specified by the operand expression.

label	opcode	operand
optional:	ORG	expression

→ location counter is set to the value of operand expression. Assembly time evaluation of ORG expression always gives a modulo 64K address.

→ If no ORG directives is included before the first instruction, assembly begins at location zero.

→ Any number of ORG directives can be used in a program.

eg: PAG1: ORG OFFH

- **ASEG Directive:**

→ It directs the assembler to use location counter for the absolute program segment.

label	opcode	operand
optional:	ASEG	---

Operands are not permitted with the ASEG directive.

→ All instruction and data following the ASEG directives are assembled into absolute mode.

→ ASEG directive remains in effect until a CSEG or DSEG directive is encountered.

→ ASEG location counter has an initial value of zero. The ORG directive can be used to assign a new value to ASEG location counter.

• CSEG Directive:

→ It directs the assembler to assemble subsequent instructions and data in the relocatable mode using the code segment location counter.

Lable	opcode	operand
optional:	CSEG	{ blank / PAGE / INPAGE }

If there are multiple CSEG directives in a program, all must specify the same operand.

→ The meaning of the operand is as follows:

⇒ blank This code segment may be relocated to the next available byte boundary.

⇒ PAGE This code segment must begin on a page boundary when relocated.

⇒ INPAGE This code segment must fit within single page when relocated.

→ The Code segment location counter has initial value of zero. ORG directive can be used to assign a new value to CSEG.

→ CSEG remains in effect until an ASEG or DSEG directive is encountered.

• DSEG Directive:

→ It directs the assembler to assemble subsequent instructions and data in the relocatable mode using the data segment location counter.

Lable	opcode	operand
optional:	DSEG	{ blank / PAGE / INPAGE }

When multiple DSEG directive appear in a program, they must all specify the same operand throughout the program.

→ There is no interaction between the operands specified for DSEG and CSEG directives. Thus a Code segment can be byte relocatable while the data segment is page relocatable.

→ DSEG remains in effect until an ASEG or CSEG is encountered.

→ The data segment location counter has an initial value of zero. The ORG directive can be used to assign a new value to the DSEG location counter.

- Program Linkage:

Modular programming and relocation feature enable you to assemble and test a number of separate program that are to be joined together and executed as a single program. Eventually it becomes necessary for these separate program to communicate information among themselves. Establishing such communication is the function of the program linkage directives.

• PUBLIC Directive:

→ PUBLIC directive makes each of the symbols listed in the operand field available for access by other program.

Label	Opcode	Operand
optional:	PUBLIC	name-list

→ Each item in the operand must be the name or label assigned to data or an instruction elsewhere in this program.
→ Multiple names appear in the list must be separated by commas.
→ Each name may be declared PUBLIC only once in a program module.
→ PUBLIC directive may appear anywhere within a program module.

eg:- PUBLIC SIN, COS, TAN, SQRT

~~• EXTERN~~

• EXTRN Directive:

→ EXTRN directive provides the assembler with a list of symbols referenced in this program but defined in a different program.

Label	Opcode	Operand
optional:	EXTRN	name-list

→ When multiple items appear in the list they must be separated by commas.
→ EXTRN may appear anywhere in the program module.
→ A symbol may be declared to be external only once in a program module.

eg:- EXTRN ENTRY, ADDRTN, BEGIN

Date
Page

• NAME Directive:

→ It is used to assign a name to the object module generated by this assembly.

label	opcode	operand
optional:	NAME	module-name

→ It requires the presence of a module-name in operand field.
→ Module names are necessary so that you can refer to a module and specify the proper sequence of modules when a number of modules are to be bound together.
→ If NAME directive is missing from the program the assembler supplies a default NAME directive with the module-name MODULE.
→ The module-name assigned by NAME directive appears as a part of the page heading in the assembly listing.

eg:- NAME MAIN

• STKLN Directive:

→ One stack is generated for multiple bounded program modules. The STKLN directive allows to specify the number of bytes to be reserved for the stack for each module.

label	opcode	operand
optional:	STKLN	expression

→ The operand expression must evaluate to a number which will be used as the maximum size of the stack.
→ When STKLN is omitted the assembler provides a default STKLN of zero.
→ If program includes more than one STKLN directive, only the last value assigned is retained.

eg:- STKLN 100