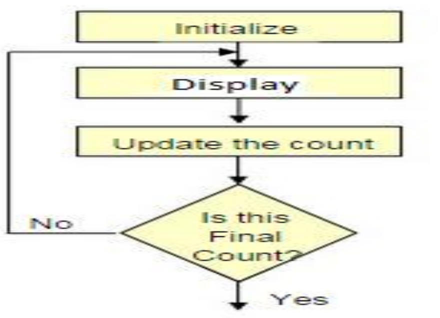
UNIT-4

- Advance Assembly Language Programming
- Counter and time delay
- types of Interrupt and their uses,RST instructions and their uses
- 8259 programmable interrupt controller
- Macros, subroutine
- Stack- implementation and uses with examples
- Memory interfacing.

2. Counter and Time Delay

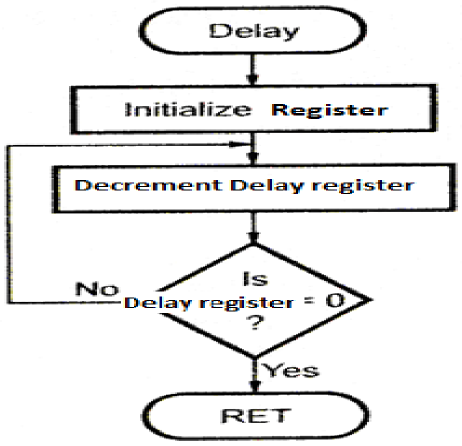
- A counter is designed simply by loading appropriate number into one of the registers and using INR or DNR instructions.
- Loop is established to update the count.
- Each count is checked to determine whether it has reached final number;if not, the loop is repeated.



Time Delay:

- Procedure used to design a specific delay.
- A register is loaded with a number , depending on the time delay required and then the register is decremented until it reaches zero by setting up a loop with conditional jump instruction.

1. Time delay using One register:



Label	Opcode	Operand	Comments	T states
	MVI	C,FFH	;Load register C	7
LOOP:	DCR	C	;Decrement C	4
	JNZ	LOOP	;Jump back to decrement C	10/7

Clock frequency of the system = 2 MHz
Clock period= 1/T= 0.5 µs
Time to execute MVI = 7 T states * 0.5= 3.5 µs
Time Delay in Loop TL= T*Loop T states * N10
= 0.5 * 14* 255
= 1785 µs = 1.8 ms
N10 = Equivalent decimal number of hexadecimal count loaded in the delay register
TLA= Time to execute loop instructions
=TL –(3T states* clock period)=1785-1.5=1783.5 µs

<u>2.Time Delay using a register pair:</u>

Label	Opcode	Operand	Comments	T states
	LXI	**B,2384H**	**Load BC with 16-bit count**	**10**
LOOP:	**DCX**	**B**	**Decrement BC by 1**	**6**
	MOV	**A,C**	**Place contents of C in A**	**4**
	ORA	**B**	**OR B with C to set Zero flag**	
	JNZ	**LOOP**	**if result not equal to 0, jump back to loop**	**10/7**

Time Delay in Loop TL= T*Loop T states * N10

= 0.5 * 24* 9092

= 109 ms

Time Delay using a LOOP within a LOOP

MVI B,38H 7T **Delay in Loop TL1=1783.5 µs**

LOOP2: MVI C,FFH 7T

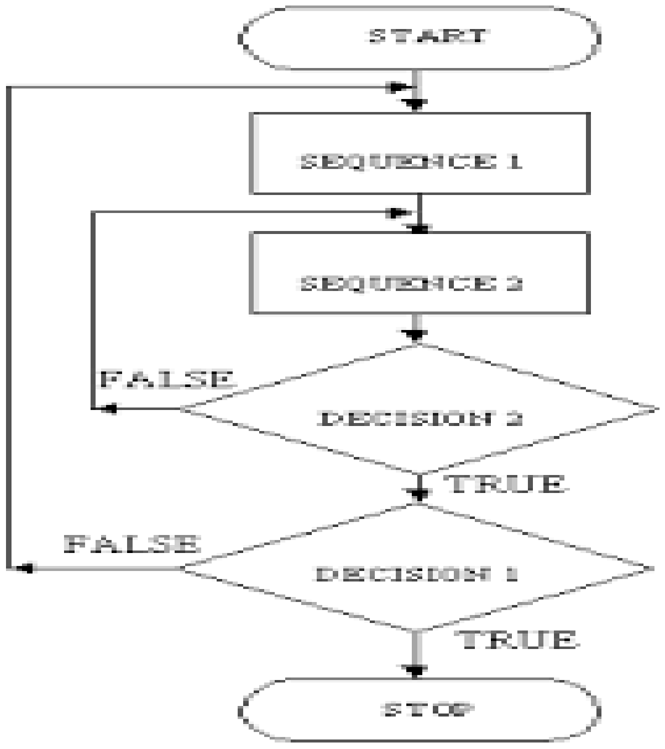
Delay in Loop TL2= (0.5*21+TL1)*56

LOOP1: DCR C 4T **=100.46ms**

JNZ LOOP1 10/7 T

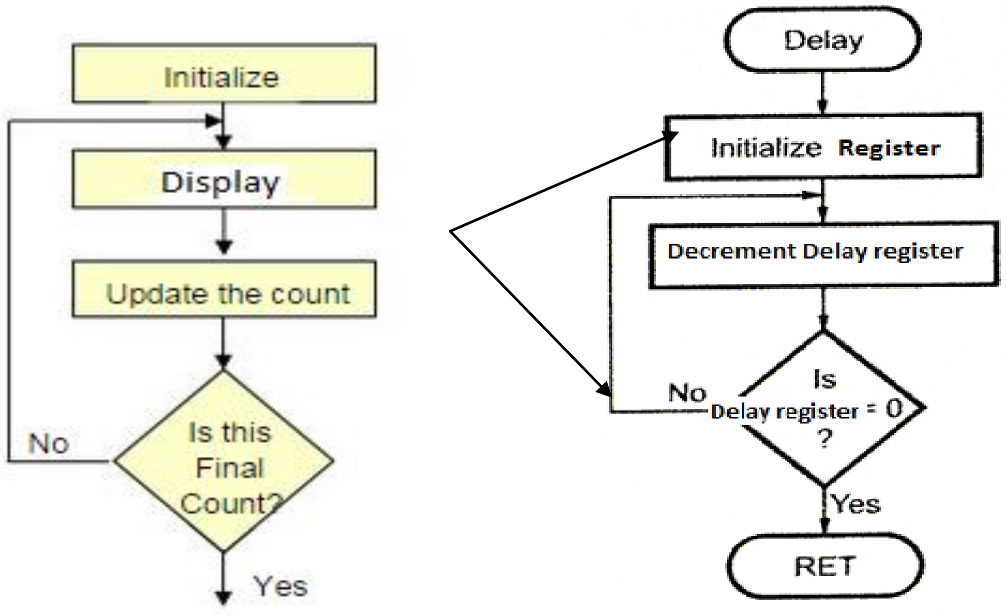
DCR B 4T

JNZ LOOP 2 10/7T



Flowchart for time delay with two loops

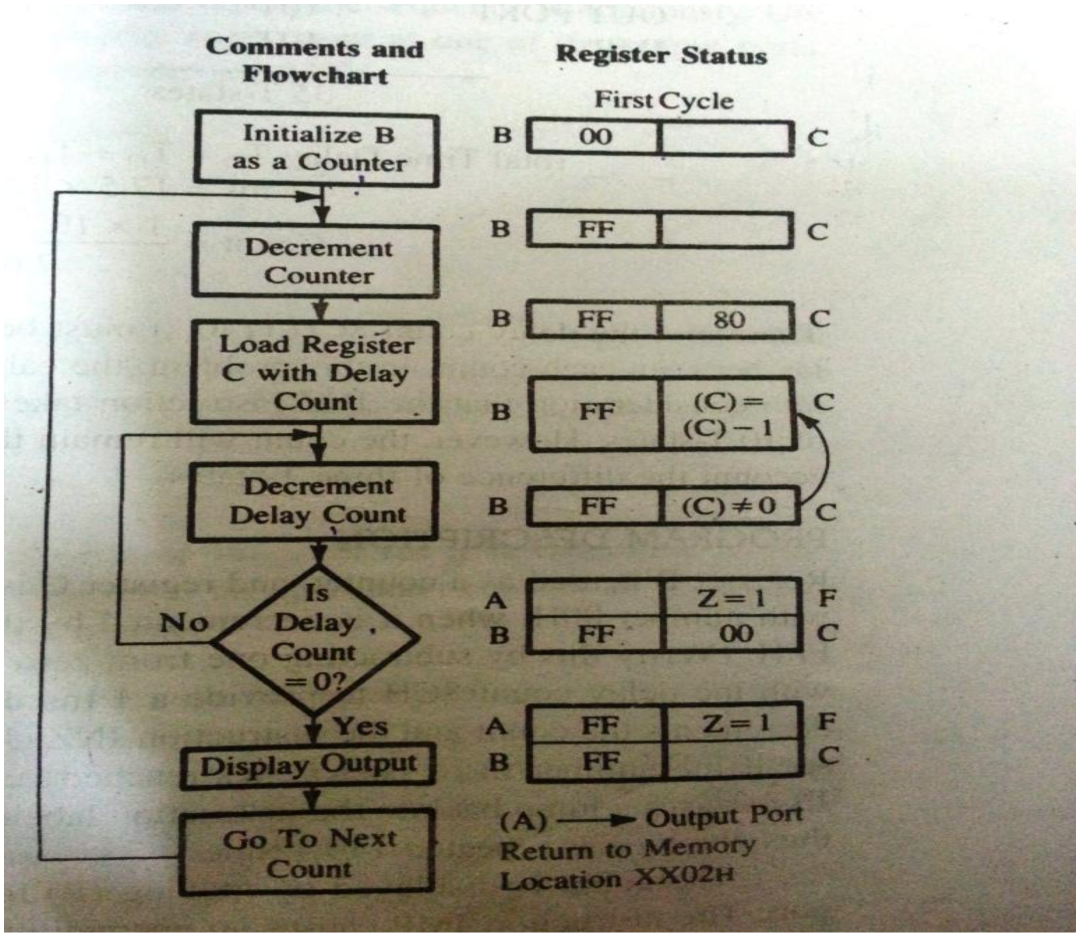
Flowchart of a counter with time delay



3.Illustrative Program: Hexadecimal Counter Write a Program to count continuously from FFH to 00H using register C with delay count 8CH between each count and display the number at one of the output ports.

```
         MVI B,00H
NEXT:    DCR B
         MVI C,8CH
DELAY:   DCR C
         JNZ DELAY
         MOV A,B
         OUT PORT#
```

```
JMP NEXT
```



4.Illustrative Program: Zero to nine (Modulo ten) Counter

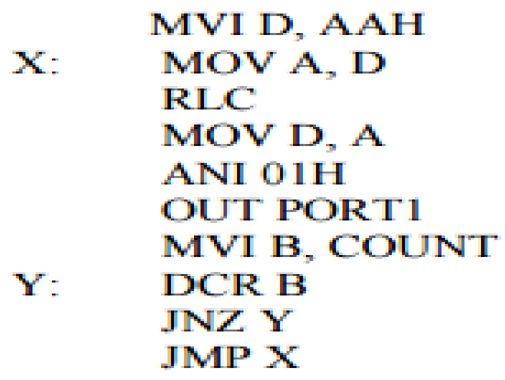
```
START:    MVI B,00H
          MOV A,B
DSPLAY:  OUT  PORT #
          LXI H,16-bit
LOOP:     DCX H
          MOV A,L
          ORA H
          JNZ LOOP
          INR B
          MOV A,B
          CPI 0AH
          JNZ DSPLAY
          JZ START
```

Start

Initialize counter

Display Output
Load Delay register
Decrement Delay register
Is Delay register=0?
Next Count
Is count =0AH?
If yes, Initialize counter
If no, Display Output

5.Illustrative Program: Generating pulse waveforms



```
        MVI D, AAH
X:      MOV A, D
        RLC
        MOV D, A
        ANI 01H
        OUT PORT1
        MVI B, COUNT
Y:      DCR B
        JNZ Y
        JMP X
```

- Generates a continuous square wave with the period of 500 Micro Sec. Assume the system clock period is 325ns, and use bit D0 output the square wave.
  - Delay outside loop: T0=46 T states * 325=14.95 micro sec.
    - Loop delay: TL=4.5 micro sec
      - Total Td=To+TL
        Count=34 H

3. 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.

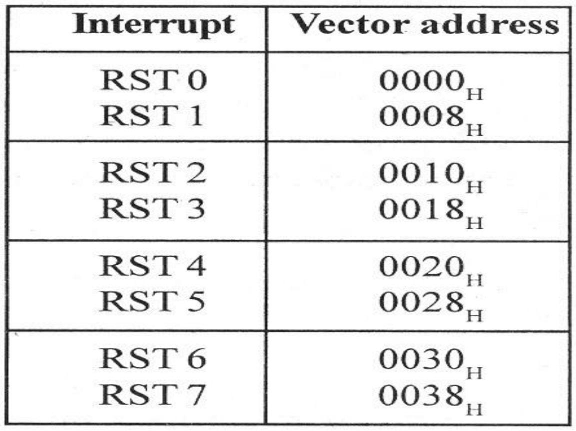
When a peripheral is ready for data transfer, it interrupts the processor by sending an appropriate signal to the interrupt pin of the processor. If the processor accepts the interrupt then the processor suspends its current activity and executes an interrupt service subroutine to complete the data transfer 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.

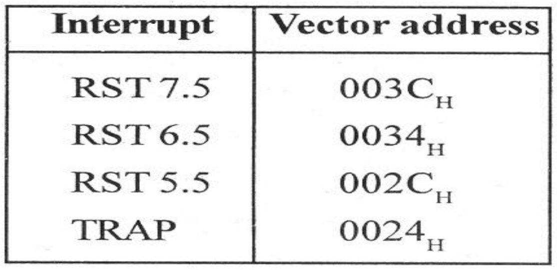
- The software interrupts are program instructions. These instructions are inserted at desired locations in a program. While running a program, lf a software interrupt instruction is encountered, then the processor executes an interrupt service routine (ISR).
- The hardware interrupts are initiated by 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, RST6 and RST 7.



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



Interrupt	Vector address
RST 7.5	$003C_H$
RST 6.5	0034_H
RST 5.5	$002C_H$
TRAP	0024_H

The vector addresses of software interrupts are given in table below.

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 8O85 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 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 an 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 INETRRUPTS:
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 InterruptMask). 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 Dl instruction. (Dl-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).

8085 interrupt response process:

interrupts should be enabled by using EI instruction, then only processor responds to all mask able interrupts.

- When microprocessor is executing a program, it checks for INTR line during execution of each instruction.
- If INTR is high then processor completes executing the current instruction, disables the interrupts and sends a INTA signal
- INTA is used by the external hardware to specify the restart instruction to processor(since INTR is a non-vectored interrupt).
- When microprocessor receives the RST instruction, it saves PC content on stack and PC is loaded with the vector address.
- Microprocessor executes the instructions at vector address.
- The interrupts should be enabled if required in the ISR(interrupt service routine)
- At the end of interrupt service routine, RET instruction loads the PC from the stack. So processor comes back to the instruction where it was interrupted previously.

Restart instructions:

RST n

These instructions are like software interrupts to 8085. When these instructions are executed processor vectors(jumps) to a specific location called restart location. The following list gives restart location for different RST instructions.

'n' value -- Vector location -- hex code
RST 0 -- 0000H -- C7
RST 1 -- 0008H -- CF
RST 2 -- 0010H -- D7
RST 3 -- 0018H -- DF
RST 4 -- 0020H -- E7
RST 5 -- 0028H -- EF
RST 6 -- 0030H --F7
RST 7 -- 0038H --FF

To get the vector location 'n' value is multiplied by 8 and the result is converted to hexadecimal notation. For example RST 3 instruction, multiply 3*8=24. 24 in hexadecimal notation is 18H. So vector address is 0018H.

8085 has 5 external interrupts. As already mentioned in this 4 are vectored interrupts and 1 is non-vectored interrupt.

RST 5.5, RST 6.5, RST 7.5, TRAP are vectored interrupts. INTR is non-vectored interrupt. TRAP is a non mask able interrupt.

Interrupt Priority:

when more than one interrupts occur at the same time, then processor responds to them according to the following priority
TRAP(highest)
RST 7.5
RST 6.5
RST 5.5
INTR (lowest)

Interrupt vector locations:

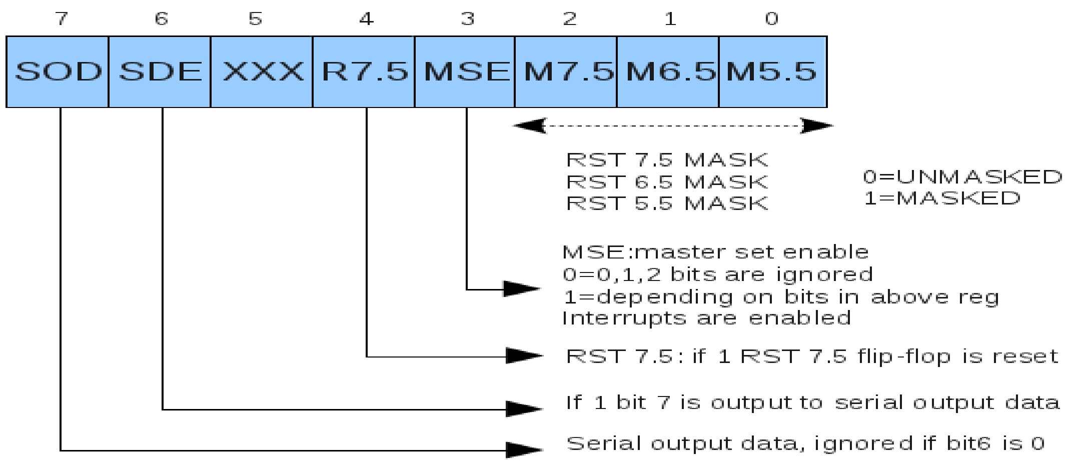
TRAP – 0024H(it is same as RST 4.5)
RST 5.5 – 002CH
RST 6.5 – 0034H
RST 7.5 – 003CH
To get the vector location for RST interrupts, interrupt value is multiplied by 8 and the result is converted to hexadecimal notation. For example RST 5.5 instruction, multiply 5.5*8=44. 44 in hexadecimal notation is 2CH. So vector address is 002CH.

Trigger levels:

TRAP is level and edge triggered
RST 7.5 is positive edge triggered
RST 6.5, RST 5.5 are level triggered.

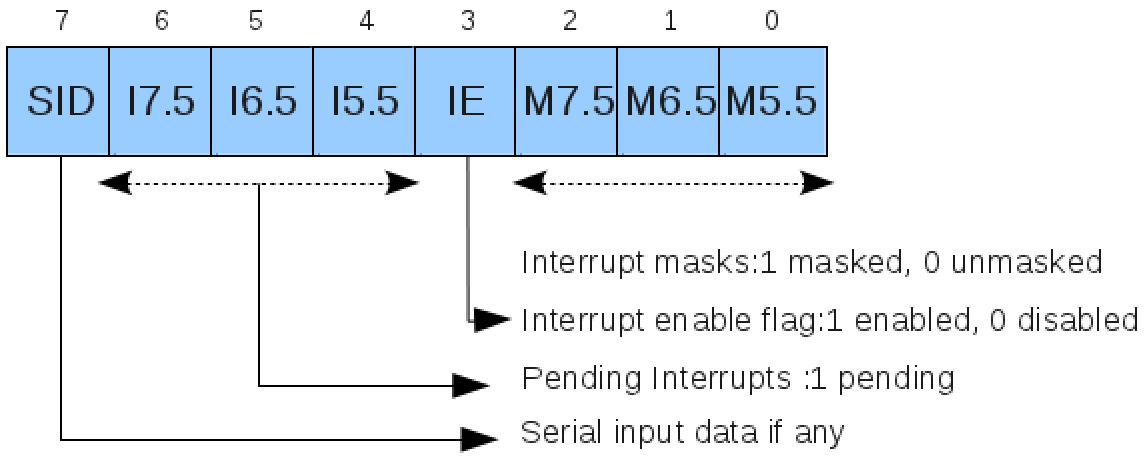
Masking of interrupts:

SIM instruction sets mask pattern for RST 5.5, RST 6.5, RST 7.5.
SIM instructions reads accumulator bit pattern and accordingly masks the interrupts. The bit pattern is shown in below figure. It also resets D flip-flop of RST 7.5 interrupt. And it also implements serial I/O.



Reading pending interrupts:

RIM instruction is used to read pending interrupts. After executing this instruction accumulator is loaded with the interrupt status signals. The pattern is shown in below figure. This instruction is also 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, 0 unmasked
Interrupt enable flag:1 enabled, 0 disabled
Pending Interrupts :1 pending
Serial input data if any

5. Stack Memory:
The stack memory can be described as a set of memory locations in the R/W memory that are used to store binary information temporarily during the execution of a program. The stack memory is used by both µP and programmer.
Whenever µP comes across any interrupt or subroutine the sequence of program execution is altered, in such a situation µP automatically places the address of the memory location present in the program counter on to the stack and after executing the interrupt or the subroutine the µP returns back to the original program by checking the address present in the stack.

Similarly the contents of the register can be stored on to the stack and retrieved from the stack by the programmer. During the execution of a program sometimes it becomes necessary to save the contents of certain registers and data in memory so that the registers may be used for other operations. After completing these operations the contents saved in memory can be transferred back to the registers. Memory locations for this purpose are set aside by the programmer in the beginning. The set of memory locations kept for this purpose is called stack.

Stack pointer is initialized in the beginning of the program by using the instruction LXI SP. This instruction loads the stack pointer register with the 16-bit memory address. Any area of the RAM can be used as stack but to prevent the program from being destroyed by the stack information, it is a general practice that the beginning of the stack is at the highest available memory location. The stack memory is based on last in first out principle (LIFO). The entering of data into the stack is called ‘PUSH’ operation and retrieving data from the stack is called ‘POP’ operation. The stack pointer always holds the address of the stack top.

LXI Rp: Load register pair immediate: It is a 3-byte command and is used to load the specified register pair with 16-bit data. For example if we want to load 3075H in register pair DE then the instruction is written as:

Instruction: LXI D,3075H
This instruction loads 75H in register E and 30H in register D.
Similarly if we write LXI SP, 3075H.then it loads stack pointer register (16-bit register) with 3075H.
General form of LXI instruction: **Opcode Operand Comments**

LXI	Reg. pair,	Register pair can be
	16-bit data	B, D, H, SP (Stack pointer)

POP Rp: Pop off Stack to register pair: It is 1-byte instruction used to copy the contents from the stack memory to the register pair. When POP instruction is executed then following steps occur: Suppose stack pointer contains memory address 3050H and data byte stored in 3050H is A7. The memory location 3051H contains C2, then on using POP B following steps are executed:

- The contents of memory location 3050H i.e. A7 are transferred to low-order register C
- Stack pointer is incremented by one to get the new value (SP+1) i.e. 3051H
- The contents of 3051H i.e. C2 is transferred to high-order register B and now the contents of register pair BC = C2A7
- The stack pointer is again incremented to get a new value (SP+1) i.e. 3052H

Therefore it is seen that when POP instruction is executed the contents of the memory location pointed out by the stack pointer register are copied to the low-order register (such as C, E, L and flag) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand.

General form of POP instruction:

Opcode	Operand	Comments
POP	Reg. pair	Register pair can be B, D, H, PSW(Program Status Word)

PSW represents accumulator and flag register, the accumulator is the high-order register and flag is low order register.

PUSH Rp: Push register pair onto the stack: It is 1-byte instruction used to copy the contents of the register pair onto the stack. When PUSH instruction is executed then following steps occur: Suppose stack pointer contains memory address 2058H, register D contains 5AH and register E contains 63H, then on using PUSH D following steps are executed:

- Stack pointer (SP) is decremented by one i.e. (SP-1) which here happens to be 2057H.
- Contents of register D (High order register) i.e. 5AH are copied into memory location 2057H (MSP-1)
- Stack pointer is again decremented by one i.e. (SP-1) which now becomes 2056H.
- The contents of register E (Low order register) are copied into the new value of memory location (MSP – 1) i.e. 2056H.

Therefore in general we can say that when PUSH command is used the memory location address present in the stack pointer register is decremented and the contents of the high-order register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E, L and flag) are copied to that location.

General form of PUSH instruction:
Opcode Operand Comments
PUSH Reg. pair Register pair can be
B, D, H, PSW(Program Status Word)

Stack Instructions
The commands used for writing into the stack memory and reading from it are PUSH and POP. The details of how these commands work are discussed in the next section. These instructions that write and read from the stack are called stack instructions. In these instructions indirect addressing mode is used, because a 16-bit register called stack pointer (SP) holds the address of the desired memory location.

How data is transferred into stack and retrieved from the stack using PUSH and POP command? Suppose register B contains 6AH and register C contains B7H and we want to transfer this data to register pair DE using stack then first of all we write a short program for this purpose. Here we are explaining function of each instruction in the program with the help of diagrams.

Memory location	**Mnemonics**	**Comments**
3000	LXI SP, 3079H	This instruction loads stack pointer register with 3079H indicating to the µP that the memory space from memory location 3078H and moving upwards can be used as stack for temporary storage.
3003	LXI B, 6AB7H	This instruction loads register pair BC with 6AB7H.
3006	PUSH B	This instruction loads the contents of reg. B i.e. 6AH in location 3078H and contents of reg.C i.e. B7H in 3077H.

Stack related instructions:

PUSH instruction: PUSH Rp:This instruction stores the content of register pair Rp(16-bit) into stack. Stack is Last in first out data structure.
Example:
PUSH B; push BC pair on to stack
PUSH D; push DE pair on to stack
PUSH H; push HL pair on to stack
PUSH PSW; push PSW(Accumulator+Flag register) on to stack
When this instruction is executed by the processor first it decrements SP by 1 and stores higher byte of the specified register pair, Then it again decrements SP by 1 and stores lower byte of the specified register pair. SP always points to top of stack.

8085 Maintains a stack which grows downwards .i.e. Higher address to lower address. And SP points to already stored location.

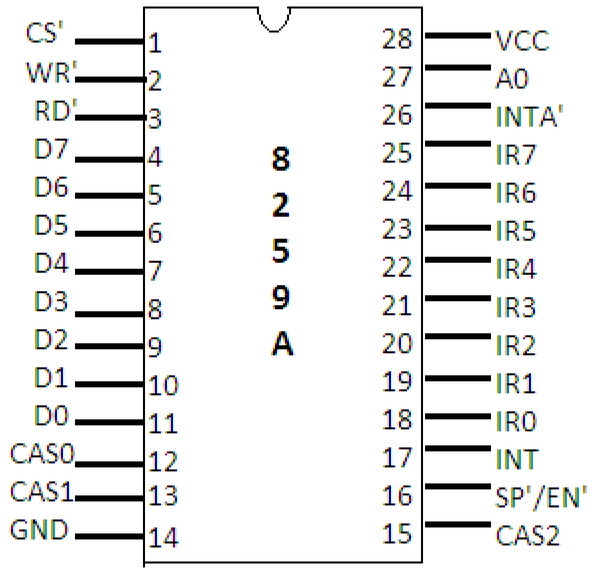
POP instruction:

POP Rp:This instruction is used to retrieve data from stack. i.e. Is 16-bit data from top of stack is stored in the specified register pair.

Example:
POP B; store data from top of stack in BC pair
POP D; store data from top of stack in DE pair
POP H; store data from top of stack in HL pair
POP PSW; store data from top of stack in Accumulator and Flag register
When this instruction is executed by the processor, it copies a byte from top of stack into lower byte of register pair and increments SP by 1, Then it again copies a byte from top of stack into higher byte of the register pair specified in the instruction and increments SP by 1.

8259 programmable interrupt controller

The Intel 8259A programmable Interrupt Controller (PIC) is one of the most common interrupt controller used in IBM PCs. It can handle eight vectored priority interrupts for the CPU. It is a 28 pin DIP package and requires a single +5V DC supply for its operation. It is designed to minimize the software and real time overhead in handling multi-level priority interrupts. It has several modes, permitting optimization for a variety of system requirements. The pin diagram of 8259A is shown in figure below.



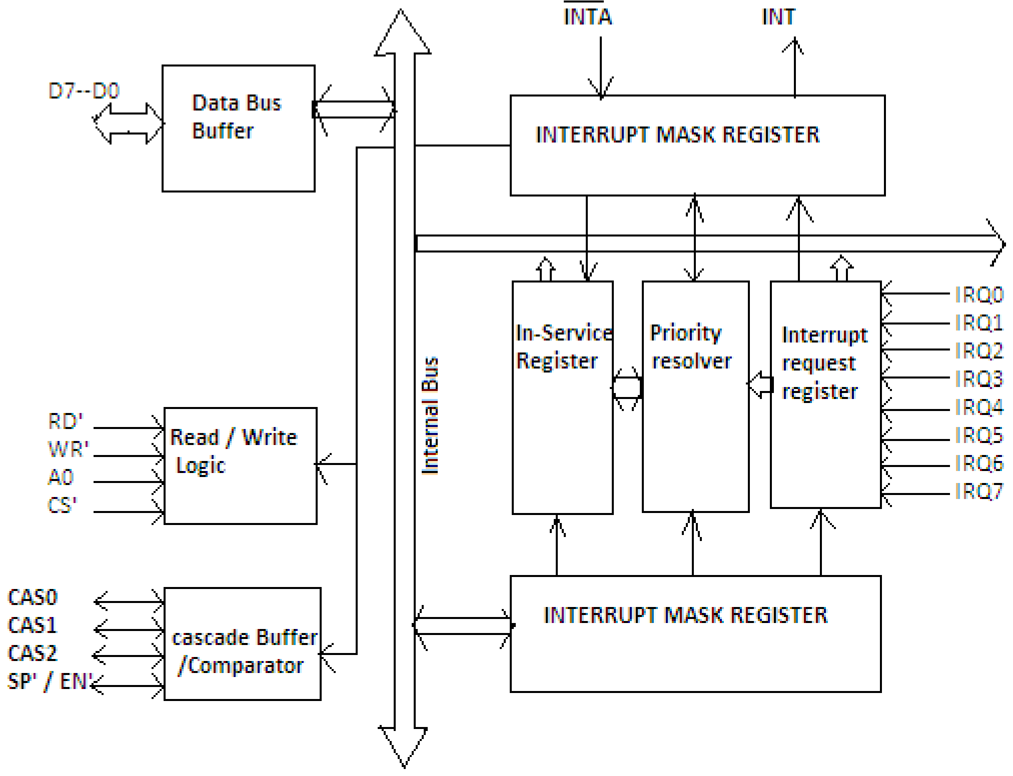
PIN Number	Pin Name	Description
1	CS’	Chip Select Input
2	WR’	Write control signal
3	RD’	Read control signal
4-11	D7-D0	Data Bus
12-13, 15	CAS0, CAS1	Cascade control
14	GND	Ground
16	SP’/EN’	
17	INT	Interrupt output for processor

18-25	IRQ0 – IRQ7	Interrupt request inputs
26	INTA	Interrupt Acknowledge input
27	A0	Address input
28	VCC	+5V supply

Functional Description of 8259A

8259A is used to manage the interrupt requirement of the system. It has 8 interrupt input lines through which it accepts interrupt requests from external devices and determines the priority of the incoming request and issues a interrupt to the CPU. The interrupt inputs are extendable up to 64 levels. And subsequently inputs information related to ISR so that the processor can initialize the program counter with the ISR address of the interrupting device. It is programmed as an i/o device and provides number of interrupt modes to the programmer so that the manner in which the requests are processed by 8259 can be configured to match the system requirement. Figure below shows the functional diagram of 8259A.

The block diagram shows the following sub components of 8259A



i. Data Bus Buffer
ii. Interrupt Request Register
iii. In-Service register
iv. Priority Resolver
v. Interrupt mask Register
vi. Read / Write Logic
vii. Cascade Buffer/Comparator

Data Bus Buffer: It is a bidirectional 8-bit register. CPU transfers the Control and status information through this buffer to 8259A; similarly 8259 transfers the vector address information to data bus through this buffer. On the CPU side it is connected to the 8-bit data bus and on the other side it is connected internally to the internal data bus.

Interrupt Request register and In-Service register:

The interrupt requests from the external devices IRQ lines are handled by these two registers in cascade. IRR is used to latch the incoming request and, in conjunction with priority resolver, allows unmasked requests with sufficient priority to put a ‘1’ on the INT pin. ISR is used to store all the interrupts that are being serviced by the CPU.

Priority resolver:

The logic block determines the priority of the services being in-request in the Interrupt Request Register. The highest priority is selected and sent to the ISR during INTA pulse.

Interrupt Mask register:

The IMR stores the masking information for the interrupt lines. The IMR operates on the IRR. Masking of a higher priority interrupt will not disable or mask the interrupt request lines of lower priority.

Interrupt (INT) and Interrupt Acknowledgement (INTA) :

INT output from 8259A goes to CPU interrupt input. This line is used to inform the CPU about the interrupting device that is interrupting the processor. INTA pulse will make the 8259A to release vectored address information of the interrupt service routine (ISR) onto the data bus.

Read/Write Control Logic:

The function of the R/W logic is to accept the commands from the CPU. It consist of an initialization Command Word (ICW) registers and Operation Command register (OCW) registers which store various control formats. It also allows the status of 8259 to be transferred to data bus. It has the following control signals:

Chip Select (CS’): It is a low active signal is used to select the chip. No operation is possible until Chip is selected through this input.

Write (WR’) and Read (RD’):

These are two low active signal used for read and write operations. WR is used to write control words (ICWs and OCWs) to 8259A and read (RD’) is used to read the status information of IRR, ISR, IMR or the interrupt level to the data bus.

Address input (A0):

The Address input A0 from the processor can be directly connected to A0 pin of 8259A and is used with RD’ and WR’ to write commands and to select various status registers for read operation.

Cascade Buffer / Comparator:

For one of the two purposes; either as an input to determine whether 8259A is to be master (SP'/EN' = 1) or as slave (SP'/EN' = 0), or as an output to disable the data bus transceiver when data are being transferred from the 8259A to the CPU.

This block is of importance when more than eight interrupts are to be used. This allows multiple 8259A to be cascaded for this purpose. It compares the IDs of all 8259As. CAS0-to-CAS2 are outputs from 8259A when used as master and are inputs when used as slave. As a master, 8259A sends the IDs of the interrupting slave device onto the CAS0-CAS2 lines. The selected slave device will send its preprogrammed subroutine address onto the data bus during the next one or two consecutive INTA pulses.

Interrupt Sequence of 8259A:

The main feature of 8259A is its programmability and the interrupt routine addressing capability. The addressing capability allows direct or indirect jumping to specific ISR based on the interrupt number and the interrupting device. The normal sequence of events during an interrupt depends on the type of the CPU used and are given below:

i. After a bit in the IRR is set to '1' it is compared with the corresponding mask bit in IMR. If the mask bit is 0, the request is passed to the priority resolver, but if it is 1, the request is blocked.
ii. When an interrupt is input to the priority resolver its priority is examined and. If according to the current state of the priority resolver the interrupt is to be sent to the CPU, the INT line is activated.
iii. Assuming that the IF flag in the CPU was set to 1, the CPU will enter its interrupt sequence at the completion of the current instruction and return two negative pulses over the INTA' line
iv. Upon receiving the 1st INTA' pulse, the IRR latches are disabled, so that the IRR will ignore further signals on the IR7-IR0 lines. This state is maintained until the end of the 2nd INTA' pulse.
v. Also the 1st INTA' pulse will cause the appropriate ISR bits to be set and the corresponding IRR bit to be cleared.
vi. The second INTA' pulse cause the current content of ICW2 to be placed on D7-D0, and the CPU uses this byte as the interrupt type.
vii. Now if the automatic end of interrupt (AEOI) bit ICW4 is 1, at the end of the second INTA' pulse the ISR bit that was set by 1st INTA' pulse is cleared; otherwise, the ISR bit is not cleared until the proper end of interrupt (EOI)command is sent to OCW2

Command Words

8259A has two types of command words, initialization and operational control word.

i. Initialization command word:
   a. ICW1 for chip (8259A) control
   b. ICW2 for type
   c. ICW3 for status control
   d. ICW4 for mode control

ii. Operational Command Word: OCW1, OCW2, OCW3

The initialization control words are normally set by the initialization routine when the computer system is first brought up and remains constant throughout its operation. The operational command words are used to dynamically control the processing of the interrupt.

The formats of various command words are given below:

Initialization command word: ICW1 (chip control):

7	6	5	4	3	2	1	0
Not Used by 8086/8088 system **But used in 8080/8085 system**			Always '1'	LITM	ADI	SNGL	IC4
			It directs received by to ICW1 **(it should be OCW1)**	Whether edge trigger(LITM=0) or level trigger(LITM=1) mode is used	Used **only in 8085**	Single(SNGL=1) or cascade(SNGLE=0)	IC4=1, if ICW4 is to be output during initialization sequence

ICW2 (Type)

7	6	5	4	3	2	1	0
Filled from bit 7-3 of the 2nd byte output by the CPU during the initialization of 8259A					Bit 2-0 are set according to the level of the interrupt request		
					IR6 would cause bit 2-0 to 110 and so on		

ICW3,

ICW3 is significant only in systems including more than one 8259A and is output only if SNGL=0

ICW4

ICW4 is output to only if IC4 (ICW1) is set to 1; otherwise, the content of ICW4 is cleared. The bits in ICW4 are as follows

7	6	5	4	3	2	1	0
Always set to 0			SFNM	BUF	M/S (1/0)	AEOI	uPM
			=1, if	=1, indicate	Ignored if	If 1, the	=1 to

	special fully nested mode is used in more than one 8259A	SP’/EN’ is to be used as an o/p to disable system 8286 txr while CPU i/p data from 8259A. SP’/EN’ should be =1	BUF=0; otherwise 1 for master, 0 for slave	ISR bit that caused the interrupt is cleared at the end of 2nd INTA’ pulse	indicate 8259A is in 8086 based system. 0 indicate that 8259A is in 8085 based system

Operational Command Word:

OCW1 is used for masking the interrupt request; when the mask bit corresponding to an interrupt request is 1, then the request is blocked. OCW2 and OCW3 are used for controlling the mode of the 8259A and receiving EOI commands. A byte is transferred to OCW1 by using the odd address (A0=1) associated with 8259A and bytes are output to OCW2 and OCW3 by using even address (A0=0). OCW2 is distinguished from OCW3 by the contents of bit-3 of data byte. If bit-3 is 0, the byte is put in OCW2, and if it is 1, the byte is put in OCW3. Both OCW2 and OCW3 are distinguished from ICW1, which also uses even address, by the content f bit-4 of data byte. If content of bit-4 of data byte is 0, then byte is put in OCW2 or OCW3 according to bit-3. There is no ambiguity in ICW2, ICW3, ICW4 and OCW1 all using the odd address because initialization word must always follow ICW1 as dictated by initialization sequence.

OCW1:

7	6	5	4	3	2	1	0

OCW2

7	6	5	4	3	2	1	0
R	SL	EOI	0	0	L2	L1	L0
Used for controlling the IR levels		Used for giving EOI command			SED FOR DESIGNING ir LEVEL		

R	SL	
0	0	Nonspecific, normal priority mode
0	1	Specifically clears the ISR bit indicated by L2-L0

1	0	Rotate priority so that a device after being serviced has the lowest priority
1	1	Rotate priority until position specified by L2-L0 is lowest

OCW3

7	6	5	4	3	2	1	0
0	ESMM	SMM	0	1	P	RR	RIS

8086 Example:

A typical program sequence for setting the content of ICWs, which assume the even address of the 8259A is 0080h, is given below:

```
MOV    AL, 13h            ; ICW1 indicating request to be edge trigger, use one 8259A, ICW4 to be output

OUT    80h, AL

MOV    AL, 18h ;  cause 5 MSBs of interrupt type to be set to 00011

OUT    81h

MOV    AL, 0Dh; informs the 8259A that SFNM is not to be used, SP'/EN' used to disable TXR, and

                 ; 8259A is master, EOI used to clear ISR bit, and 8259A is part of 8086 system

OUT    81h, AL
```

First two instructions causes the requests to be edge triggered, denote that only one 8259A is used