SRKR Engineering College
Department of Information Technology, Bhimavaram,
A.P. - 534204

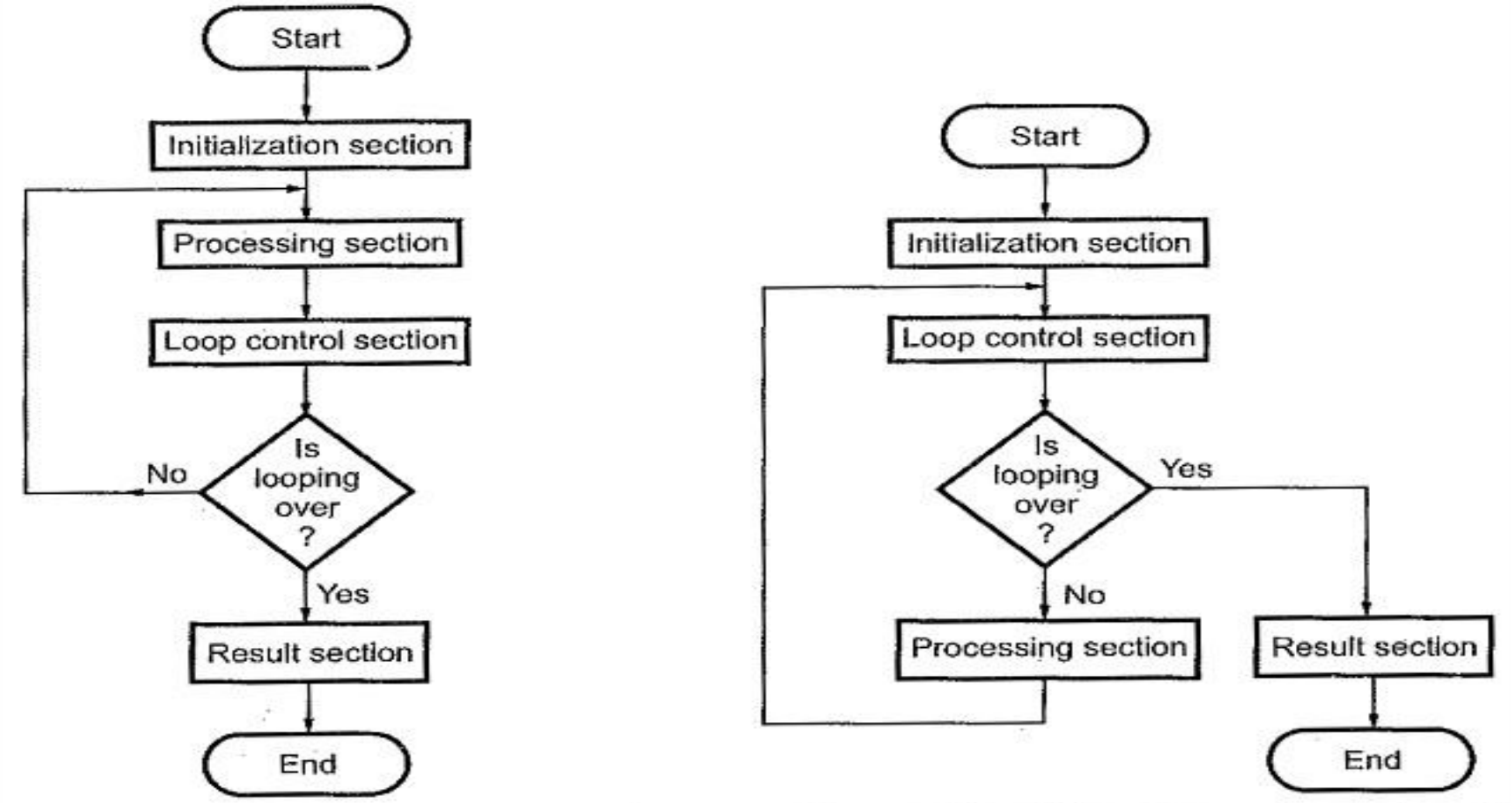
MICROPROCESSORS
UNIT-II

P.D.S.S.Lakshmi kumari
Asst.Professor

Looping, Counting and Indexing

- **Looping:** In this Programming Techniques in Microprocessor 8085, the Program is instructed to execute certain set of instructions repeatedly to execute a particular task number of times. For example, to add ten numbers stored in the consecutive memory locations we have to perform addition ten times.
- The Program loop is the basic structure which forces the processor to repeat a sequence of instructions. Loops have four sections.
- **Initialization section.**
- **Processing section.**
- **Loop control section**
- **Result section.**

- **Flowchart:**


Start
Initialization section
Processing section
Loop control section
Is looping over ?
No
Yes
Result section
End
Flowchart 1
Start
Initialization section
Loop control section
Is looping over ?
Yes
No
Processing section
Result section
End
Flowchart 2

Example For Conditional Looping And Unconditional Looping

- EX: MVI A,FF
- L1: DCR A
- JNZ L1

L1

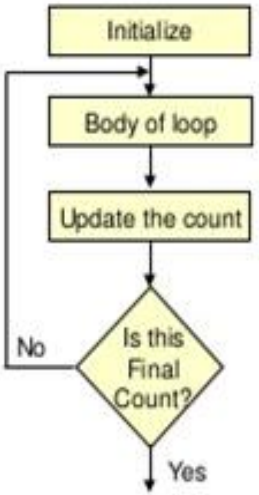
MVI A,FF

L1: DCR A

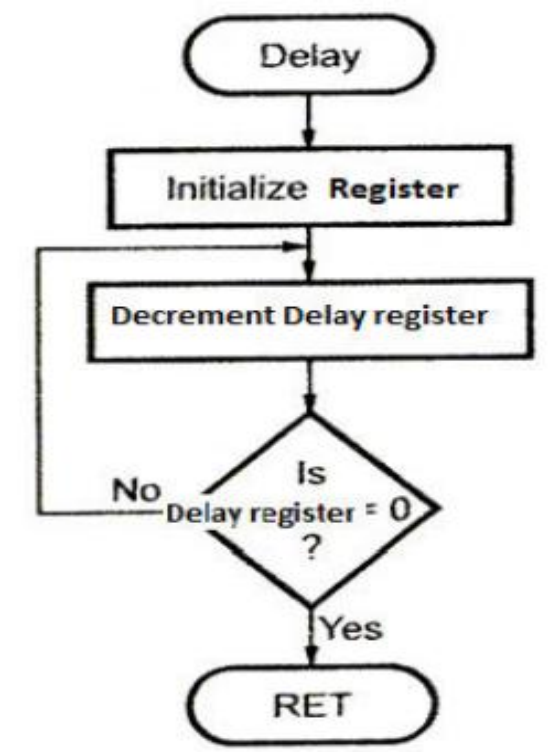
JMP

- **Counting:** This technique allows programmer to count how many times the instruction/set of instructions are executed.
- A loop counter is set up by loading a register with a certain value. Then using the DCR (to decrement) and INR (to increment) the contents of the register are updated.
- A loop is set up with a conditional jump instruction that loops back or not depending on whether the count has reached the termination count.

- Continuous loop-repeat task continuously



- Conditional loop-repeats a task until certain data conditions are met.



- **Indexing:** This Programming Techniques in Microprocessor 8085 allows programmer to point or refer the data stored in sequential memory locations one by one. Let us see the program loop to understand looping, counting and indexing.
- **EX:** LXI H,8070
- MOV A,M
- INX H
- MOV B,M
- ADD B
- INX H
- MOV M,A
- HLT

Time delays

- In the real time applications, such as traffic light control, digital clock, process control, serial communication, it is important to keep a track with time.
- For example in traffic light control application, it is necessary to give time delays between two transitions. These time delays are in few seconds and can be generated with the help of executing group of instructions number of times. These software timers are also called time delays or software delays.
- Delay Time calculation for Execution of every instruction needs some clock cycles.
- Thus microprocessor will have to execute the required number of instructions to generate the required time delay.

Register as a counter

- MVI C, FF 7T
- REP: DCR C 4T
- JNZ REP 10T/7T (incase of jump)
- RET
- Delay time in loop (TL): = No. of T states* time for 1T state* N10 (equivalent value of counts in hexa decimal) =14*T * 255 times =14 * 0.33micro sec* 255
- Delay time outside (TO): 7T = 7X0.33 micro sec
- Total delay time: TL + TO

Register pair as a counter

- LXI B, FFFF 10T
- Loop: DCX B 6T
- MOV A,C 4T
- ORA B 4T
- JNZ Loop 10/7T
- RET
- Tdl (delay time in loop) = No of T states in loop * time for 1T state* N10 in decimal = 24*0.33micrso sec*65535 = 0.5 seconds
- How to generate Required delay time ?
- Required delay time = 0.25 sec (say)
- 0.25 = 24*0.33 micro seconds * counter value counter value = 0.25/(24*0.33 micro seconds)
- This counter value is loaded in to the register pair (BC) to generate the required time delay

Nested Loops for Delay

- Instead (or in conjunction with) Register Pairs, a nested loop structure can be used to increase the total delay produced.

	MVI B, 10H	7 T-States
LOOP2	MVI C, FFH	7 T-States
LOOP1	DCR C	4 T-States
	JNZ LOOP1	10 T-States
	DCR B	4 T-States
	JNZ LOOP2	10 T-States

Delay Calculation of Nested Loops

- The calculation remains the same except that it the formula must be applied recursively to each loop.
- – Start with the inner loop, then plug that delay in the calculation of the outer loop.
- Delay of inner loop– T_{O1} = 7 T-States
- MVI C, FFH instruction– T_{L1} = (255 X 14) - 3 = 3567 T-States
- 14 T-States for the DCR C and JNZ instructions repeated 255 times (FF = 255) minus 3 for the final JNZ
- Delay of outer loop– T_{O2} = 7 T-States
  - MVI B, 10H instruction– T_{L1} = (16 X (14 + 3574)) - 3 = 57405 T-States
  - 14 T-States for the DCR B and JNZ instructions and 3574 T-States for loop1 repeated 16 times ($10_{16} = 16_{10}$) minus 3 for the final JNZ.
- – T_{Delay} = 7 + 57405 = 57412 T-States
- Total Delay(T_{Delay}) = 57412 X 0.5 µSec = 28.706 mSec

Stack and Subroutines

- **Stack**
- The stack is a group of memory location in the R/W memory that is used for temporary storage of binary information during the execution of a program
- The stack is a LIFO structure.
- –Last In First Out.
- The starting location of the stack is defined by loading a 16 bit address into the stack pointer (**LXI SP,16 bit**.) that spaced is reserved, usually at the top of the memory map.
- The 8085 provide two instruction PUSH & POP for storing information on the stack and retrieving it back.
- The stack normally grows backwards into memory.

Operation of the stack

- During pushing, the stack operates in a “decrement then store” style.
- The stack pointer is decremented first, then the information is placed on the stack.
- During poping, the stack operates in a “use then increment” style.
- The information is retrieved from the top of the stack and then the pointer is incremented.
- The SP pointer always points to “the top of the stack“

PUSH PSW Register Pair

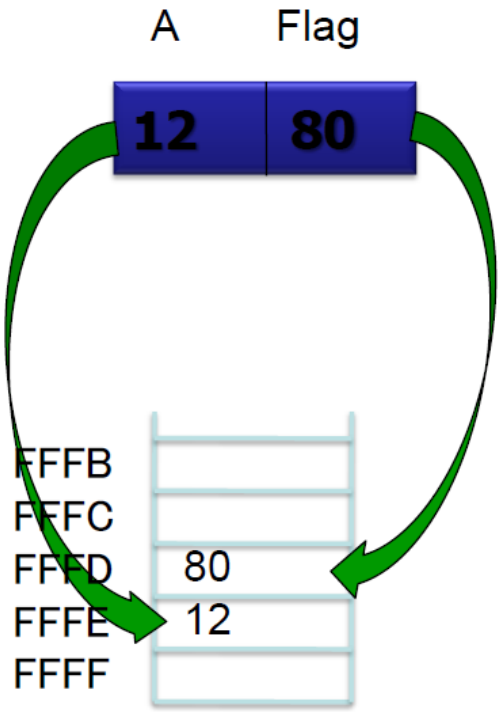
PUSH PSW (1 Byte Instruction) A Flag

Decrement SP

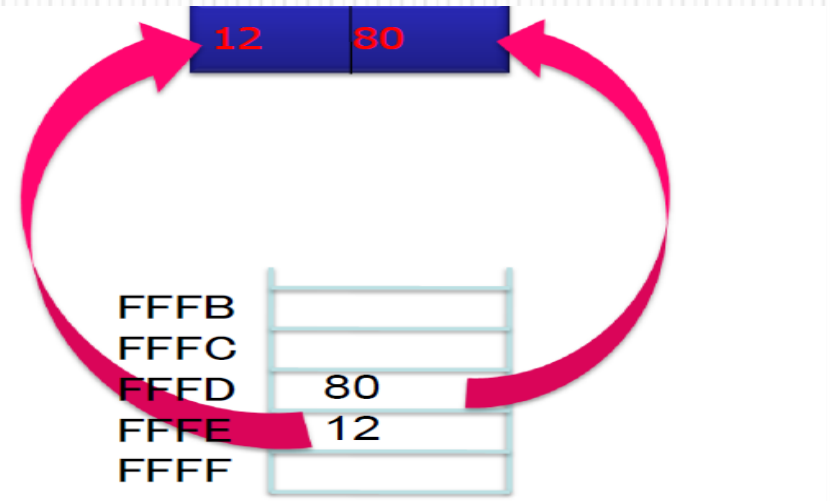
Copy the contents of register A to the memory location pointed to by SP

Decrement SP

Copy the contents of Flag register to the memory location pointed to by SP



- **Pop PSW Register Pair**
- POP PSW (1 Byte Instruction) A FLAG
- Copy the contents of the memory location pointed to by the SP to Flag register
- Increment SP
- Copy the contents of the memory location pointed to by the SP to register A
- Increment SP



- Let, We want to Reset the Zero Flag
- Program:
  - LXI SP FFFF
  - PUSH PSW
  - POP H
  - MOV A L
  - ANI BFH (BFH= 1011 1111) * Masking
  - MOV L A
  - PUSH H
  - POP PSW

8085 Flag :

S	Z	X	AC	X	P	X	Cy

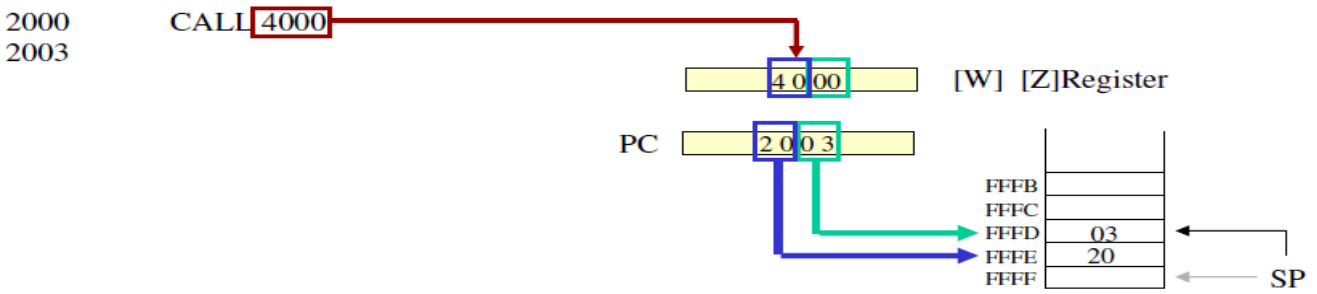
Subroutines

- A subroutine is group of instruction written separately from the main program to perform a function that occurs repeatedly in the main program.
- When a main program calls a subroutine the program execution is transferred to the subroutine. after the completion of the subroutine ,the program execution returns to the main program.
- The microprocessor uses the stack to store the return address of the subroutine.
- The 8085 has two instructions for dealing with subroutines.
- –The CALL instruction is used to redirect program execution to the subroutine.
- –The RET instruction is used to return to the main program at the end of the subroutine .

- **CALL 16 bit address**
- Call subroutine in conditionally located at the memory address specified by the 16 bit operand.
- This instruction places the address of the next instruction on the stack and transfers the program execution to the subroutine address.

CALL 4000H (3 byte instruction)

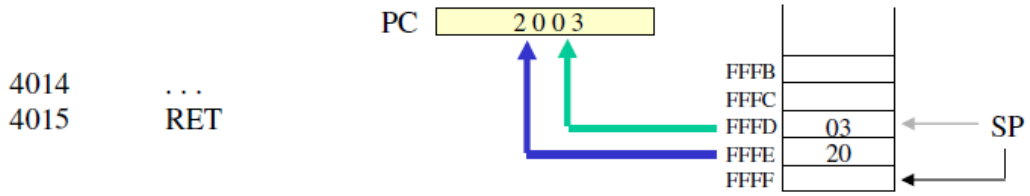
– When CALL instruction is fetched, the MP knows that the next two Memory location contains 16bit subroutine address in the memory.



- **RET**
- Return unconditionally from the subroutine.
- This instruction locates the return address on the top of the stack and transfers the program execution back to the calling program.

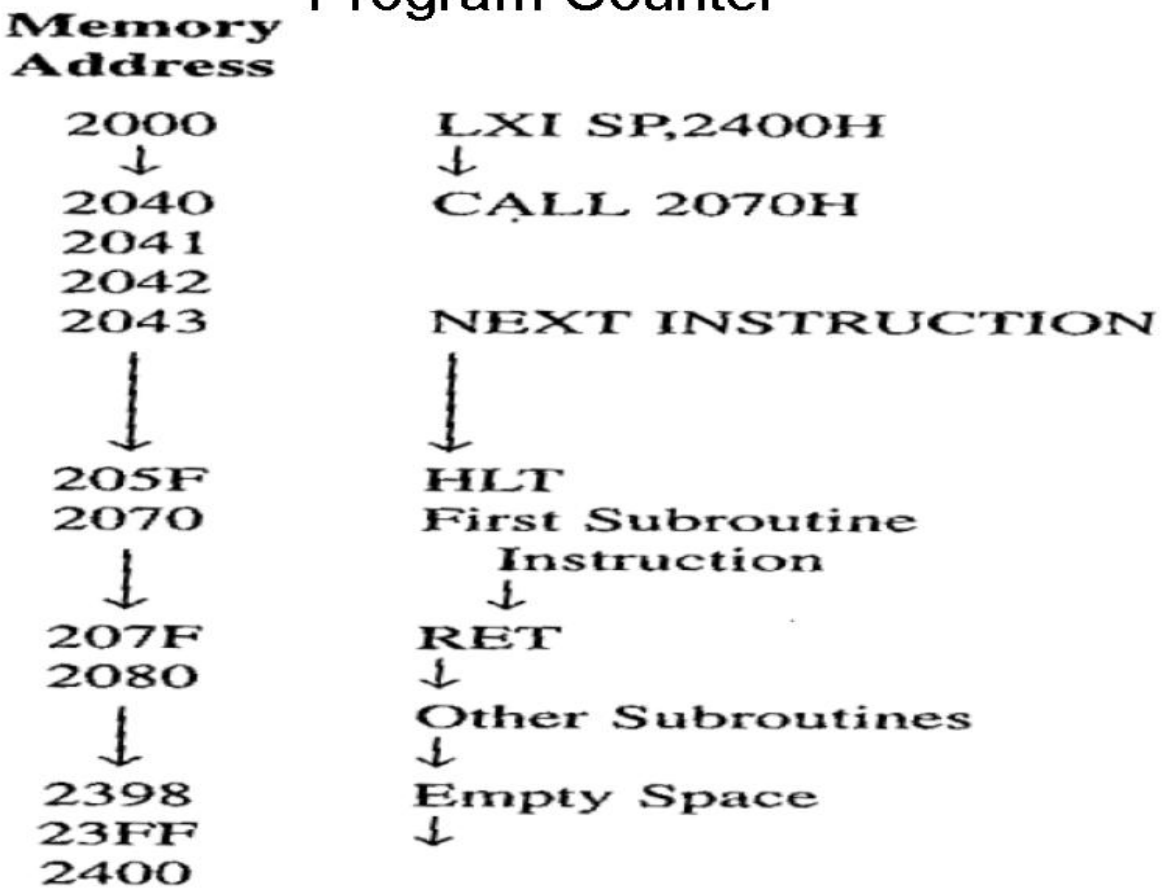
RET (1 byte instruction)

- Retrieve the return address from the top of the stack
- Load the program counter with the return address.

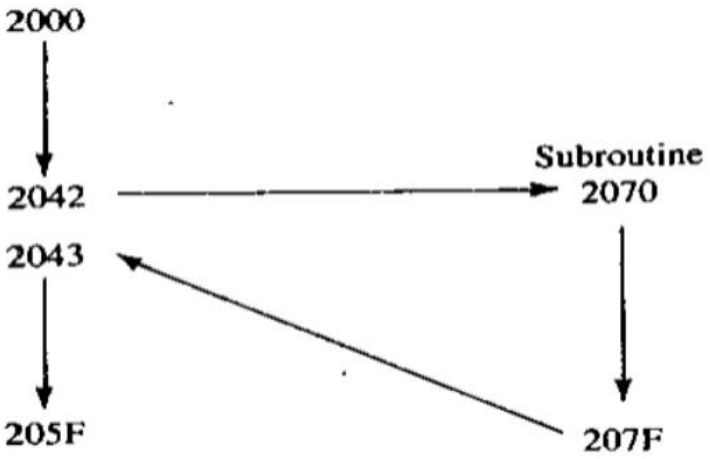


Sequence the execution of instructions

Illustrates the exchange of information between stack and Program Counter



Program Execution



Memory Address	Machine Code	Mnemonics	Comments
2040	CD	CALL 2070H	;Call subroutine located at the memory
2041	70		; location 2070H
2042	20		
2043	NEXT	INSTRUCTION	

CALL Execution

- **Instruction requires five machine cycles and eighteen T-states:** Call instruction is fetched, 16-bit address is read during M2 and M3 and stored temporarily in W/Z registers. In next two cycles content of program counter are stored on the stack (address from where microprocessor continue it execution of program after completion of the subroutine.)

Machine Cycles	Stack Pointer (SP) 2400	Address Bus (AB)	Program Counter (PCH) (PCL)	Data Bus (DB)	Internal Registers (W) (Z)
M_1 Opcode Fetch	23FF (SP–1)	2040	20 41	CD Opcode	—
M_2 Memory Read		2041	20 42	70 Operand	→ 70
M_3 Memory Read	23FF	2042	20 43	20 Operand	→ 20
M_4 Memory Write	23FE (SP–2)	23FF	20 43	20 (PCH)	
M_5 Memory Write	23FE	23FE	20 43	43 (PCL)	(20) (70)
M_1 Opcode Fetch of Next Instruction		20 70 → (W)(Z)	2071		(2070) → (W)(Z)

Memory Address	Code (H)
2040	CD
2041	70
2042	20

Data Transfer During the Execution of the CALL Instruction

RET Execution

- Program execution sequence is transferred to the memory location 2043H location.M1 is normal fetch cycle during M2 contents of stack pointer are placed on address bus so 43H data is fetched and stored on Z register and SP is upgraded. Similarly for M3. Program sequence is transfered to2043H by placing contents of W/Z on address bus.

Memory Address	Code (H)
207F	C9

Contents of Stack Memory	
23FE	43
23FF	20

Machine Cycles	Stack Pointer (23FE)	Address Bus (AB)	Program Counter	Data Bus (DB)	Internal Registers (W) (Z)
M_1 Opcode Fetch	23FE	207F	2080	C9 Opcode	
M_2 Memory Read	23FF	23FE		43 (Stack) →	→ 43
M_3 Memory Read	2400	23FF		20 (Stack−1) →	→ 20
M_1 Opcode Fetch of Next Instruction		2043 (W) (Z) ←	2044	←	2043 (W) (Z)

Data Transfer During the Execution of the RET Instruction

- The 8085 supports conditional CALL and conditional RTE instructions.
- The same conditions used with conditional JUMP instructions can be used.
- CC, call subroutine if Carry flag is set.
- CNC, call subroutine if Carry flag is not set
- RC, return from subroutine if Carry flag is set
- RNC, return from subroutine if Carry flag is not set
- CC call on carry
- CNC call on no carry
- CP call on positive
- CM call on Minus
- CPE call on Parity even
- CPO call on Parity odd
- CZ call on Zero
- CNZ call on No zero

INTERRUPTS

- Mainly in the microprocessor based system the interrupts are used for data transfer between the peripheral and the microprocessor.
- The processor will check the interrupts always at the 2nd T-state of last machine cycle.
- If there is any interrupt it accept the interrupt and send the INTA (active low) signal to the peripheral.
- The vectored address of particular interrupt is stored in program counter.
- The processor executes an interrupt service routine (ISR) addressed in program counter.
- It returned to main program by RET instruction.

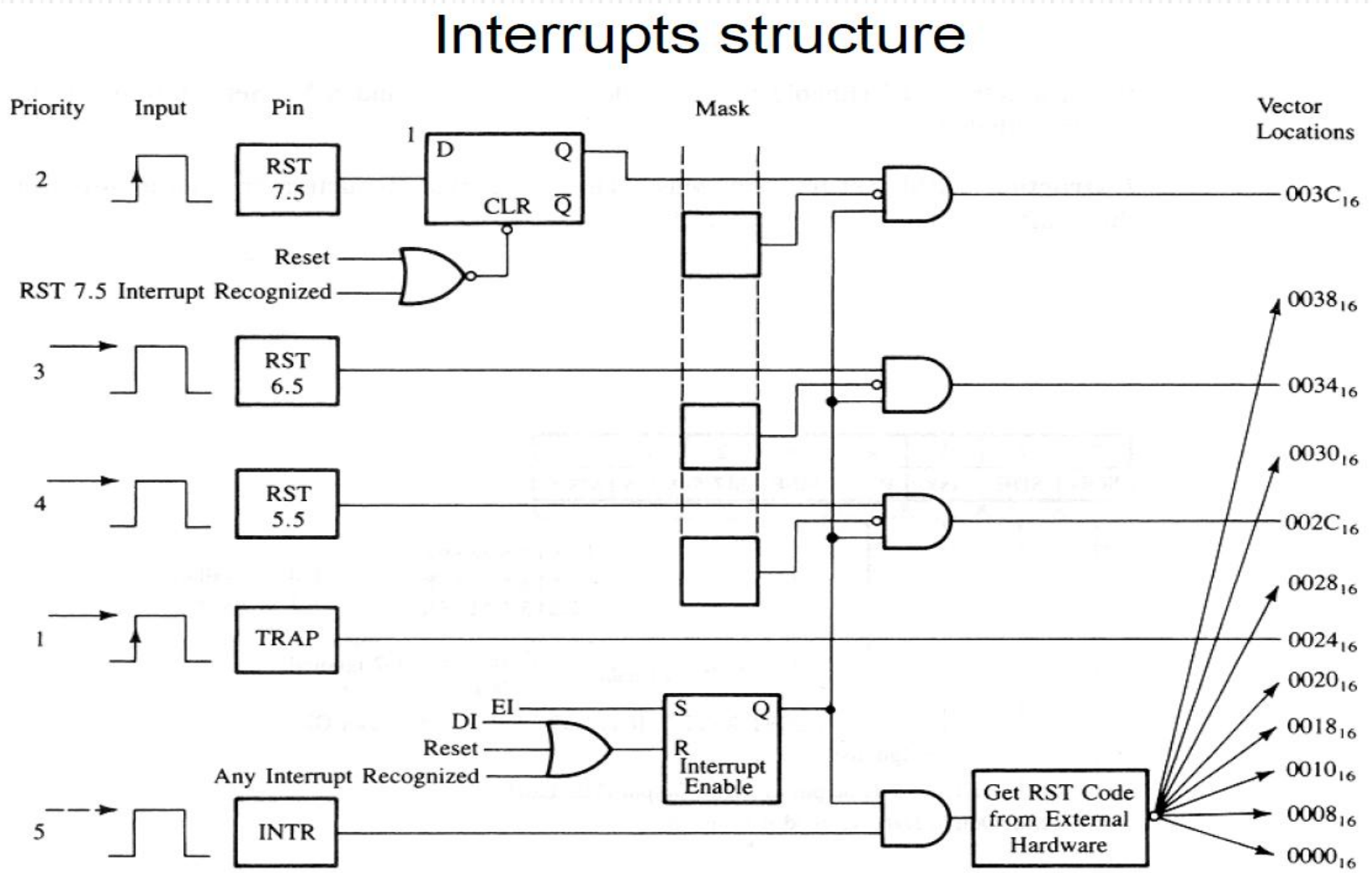
- **Types of Interrupts:**
- It supports two types of interrupts.
- Hardware
- Software
- **Software interrupts:**
- The software interrupts are program instructions. These instructions are inserted at desired locations in a program.
- The 8085 has eight software interrupts from RST 0 to RST 7. The vector address for these interrupts can be calculated as follows.
- Interrupt number * 8 = vector address
- For RST 5,5 * 8 = 40 = 28H

- **Vector addresses of all interrupts**

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

- **Hardware interrupts:**
- An external device initiates the hardware interrupts and placing an appropriate signal at the interrupt pin of the processor.
- If the interrupt is accepted then the processor executes an interrupt service routine.
- The 8085 has five hardware interrupts
- (1) TRAP (2) RST 7.5 (3) RST 6.5 (4) RST 5.5 (5) INTR

Interrupt Structure of 8085



- **(1)TRAP:**This interrupt is a non-maskable interrupt. It is unaffected by any mask or interrupt enable.
- TRAP bas the highest priority and vectored interrupt.
- TRAP interrupt is edge and level triggered. This means hat the TRAP must go high and remain high until it is acknowledged.
- 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. (i.e., If the processor receives HOLD and TRAP at the same time then HOLD is recognized first and then TRAP is recognized).
- There are two ways to clear TRAP interrupt.
- 1.By resetting microprocessor (External signal)
- 2.By giving a high TRAP ACKNOWLEDGE (Internal signal)

- **(2)RST 7.5:**The RST 7.5 interrupt is a maskable interrupt. It has the second highest priority. It is edge sensitive. ie. Input goes to high and no need to maintain high state until it recognized.
- Maskable interrupt. It is disabled by,
- 1.DI instruction
- 2.System or processor reset.
- 3.After reorganization of interrupt.
- Enabled by EI instruction**.**
- **(3)RST 6.5 and 5.5:**The RST 6.5 and RST 5.5 both are level triggered. . ie. Input goes to high and stay high until it recognized.
- Maskable interrupt. It is disabled by,
- 1.DI, SIM instruction
- 2.System or processor reset.
- 3.After reorganization of interrupt.

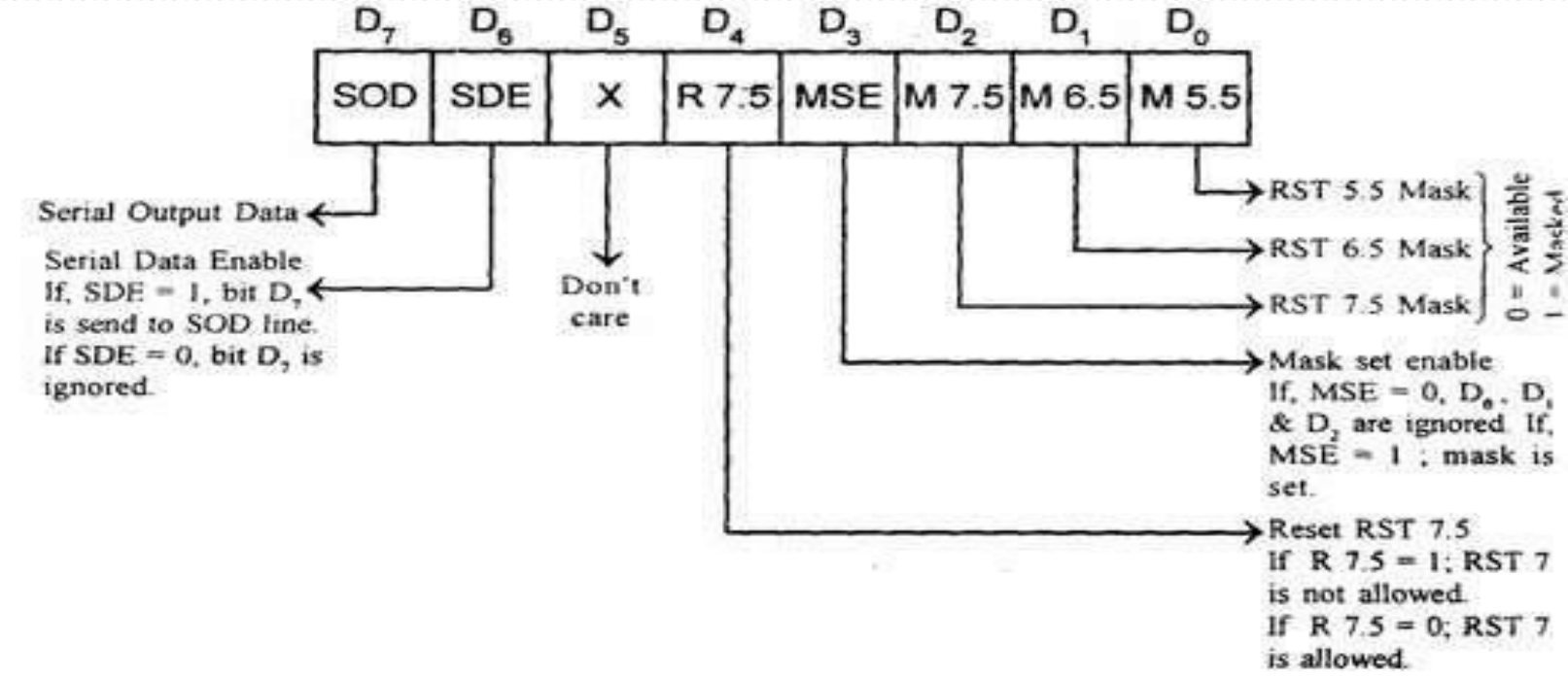
- Enabled by EI instruction.
- The RST 6.5 has the third priority whereas RST 5.5 has the fourth priority.
- INTR is a maskable interrupt. It is disabled by,
- 1.DI, SIM instruction
- 2.System or processor reset.
- 3.After reorganization of interrupt
- Enabled by EI instruction.
- Non- vectored interrupt. After receiving INTA (active low) signal, it has to supply the address of ISR.It has lowest priority.
- It is a level sensitive interrupts. ie. Input goes to high and it is necessary to maintain high state until it recognized.

Interrupt Process

- The following sequence of events occurs when INTR signal goes high.
- The 8085 checks the status of INTR signal during execution of each instruction.
- If INTR signal is high, then 8085 complete its current instruction and sends active low interrupt acknowledge signal, if the interrupt is enabled.
- In response to the acknowledge signal, external logic places an instruction OPCODE 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.
- On receiving the instruction, the 8085 save the address of next instruction on stack and execute received instruction.

SIM and RIM for interrupts

- **SIM instruction.**
- The status of these interrupts can be read by executing RIM instruction.
- The masking or unmasking of RST 7.5, RST 6.5 and RST 5.5 interrupts can be performed by moving an 8-bit data to accumulator and then executing SIM



- **RIM Instruction**
- The status of pending interrupts can be read from accumulator after executing RIM instruction.
- When RIM instruction is executed an 8-bit data is loaded in accumulator, which can be interpreted as shown in fig.