

The 8085 Non-Vectored Interrupt Process

1. The interrupt process should be **enabled** using the **EI** instruction.
2. The 8085 checks for an interrupt during the execution of **every** instruction.
3. If there is an interrupt, the microprocessor will **complete the executing instruction**, and start a **RESTART** sequence.
4. The RESTART sequence **resets the interrupt flip flop** and **activates the interrupt acknowledge signal** (INTA).
5. Upon receiving the INTA signal, the **interrupting device** is expected to return the **op-code** of one of the 8 RST instructions.



The 8085 Non-Vectored Interrupt Process

6. When the microprocessor executes the RST instruction received from the device, it **saves the address of the next instruction** on the stack and **jumps to the appropriate entry in the IVT**.
7. The **IVT entry** must **redirect** the microprocessor to the actual **service routine**.
8. The service routine must include the instruction **EI** to re-enable the interrupt process.
9. At the end of the service routine, the **RET** instruction **returns the execution to where the program was interrupted**.



The 8085 Non-Vectored Interrupt Process

- The 8085 recognizes 8 RESTART instructions: RST0 - RST7.
 - each of these would send the execution to a predetermined hard-wired memory location:

Restart Instruction	Equivalent to
RST0	CALL 0000H
RST1	CALL 0008H
RST2	CALL 0010H
RST3	CALL 0018H
RST4	CALL 0020H
RST5	CALL 0028H
RST6	CALL 0030H
RST7	CALL 0038H



Restart Sequence

- The restart sequence is made up of three machine cycles
 - In the 1st machine cycle:
 - The microprocessor sends the INTA signal.
 - While INTA is active the microprocessor reads the data lines expecting to receive, from the interrupting device, the opcode for the specific RST instruction.
 - In the 2nd and 3rd machine cycles:
 - the 16-bit address of the next instruction is saved on the stack.
 - Then the microprocessor jumps to the address associated with the specified RST instruction.



Restart Sequence

- The location in the IVT associated with the RST instruction can not hold the complete service routine.
 - The routine is written somewhere else in memory.
 - Only a JUMP instruction to the ISR's location is kept in the IVT block.



Hardware Generation of RST Opcode

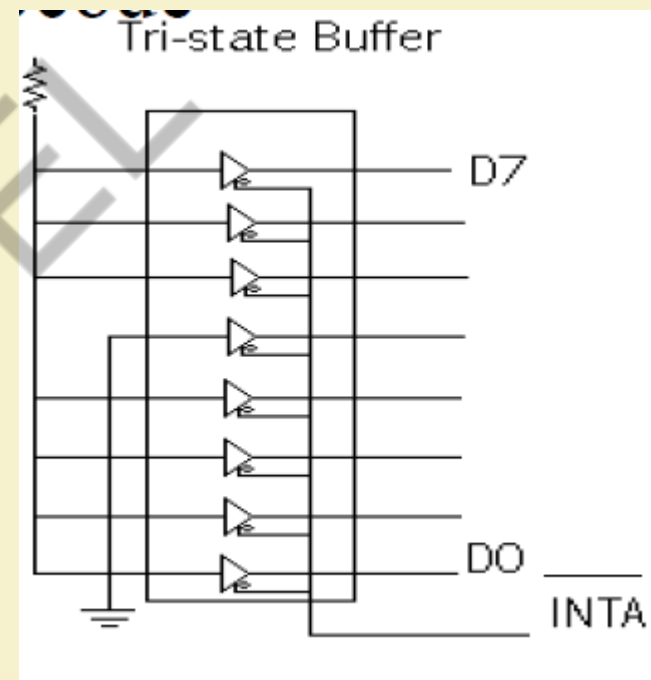
- How does the external device produce the opcode for the appropriate RST instruction?
 - The opcode is simply a collection of bits.
 - So, the device needs to set the bits of the data bus to the appropriate value in response to an INTA signal.



Hardware Generation of RST Opcode

RST 5's opcode is EF =

D							D
7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	1



Hardware Generation of RST Opcode

- During the interrupt acknowledge machine cycle, (the 1st machine cycle of the RST operation):
 - The Microprocessor activates the INTA signal.
 - This signal will enable the Tri-state buffers, which will place the value EFH on the data bus.
 - Therefore, sending the Microprocessor the RST 5 instruction.
- The RST 5 instruction is exactly equivalent to CALL 0028H



Issues in Implementing INTR Interrupts

- How long must INTR remain high?
 - The microprocessor checks the INTR line one clock cycle before the last T-state of each instruction.
 - The interrupt process is Asynchronous.
 - The INTR must remain active long enough to allow for the longest instruction.
 - The longest instruction for the 8085 is the conditional CALL instruction which requires 18 T-states.

Therefore, the INTR must remain active for 17.5 T-states.



Issues in Implementing INTR Interrupts

- How long can the INTR remain high?
 - The INTR line must be deactivated before the EI is executed. Otherwise, the microprocessor will be interrupted again.
 - The worst case situation is when EI is the first instruction in the ISR.
 - Once the microprocessor starts to respond to an INTR interrupt, INTA becomes active (=0).

Therefore, INTR should be turned off as soon as the INTA signal is received.



Issues in Implementing INTR Interrupts

- Can the microprocessor be interrupted again before the completion of the ISR?
 - As soon as the 1st interrupt arrives, all maskable interrupts are disabled.
 - They will only be enabled after the execution of the EI instruction.

Therefore, the answer is: “only if you allow it to”.
If the EI instruction is placed early in the ISR, other interrupt may occur before the ISR is done.



Multiple Interrupts & Priorities

- How do we allow multiple devices to interrupt using the INTR line?
 - The microprocessor can only respond to one signal on INTR at a time.
 - Therefore, we must allow the signal from only one of the devices to reach the microprocessor.
 - We must assign some priority to the different devices and allow their signals to reach the microprocessor according to the priority.



The Priority Encoder

The solution is to use a circuit called the priority encoder (74366).

- This circuit has 8 inputs and 3 outputs.
- The inputs are assigned increasing priorities according to the increasing index of the input.
 - Input 7 has highest priority and input 0 has the lowest.
- The 3 outputs carry the index of the highest priority active input.

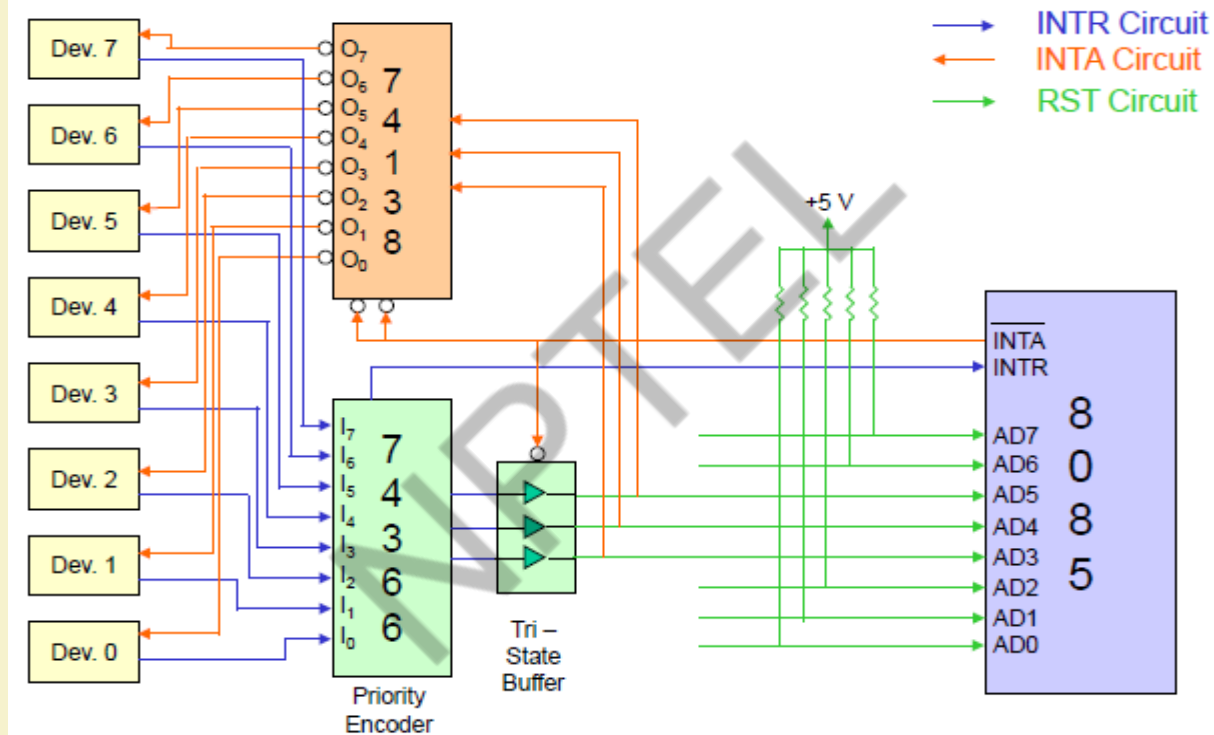


Multiple Interrupts & Priorities

- Note that the opcodes for the different RST instructions follow a set pattern.
 - Bit D5, D4 and D3 of the opcodes change in a binary sequence from RST 7 down to RST 0.
 - The other bits are always 1.
 - This allows the code generated by the 74366 to be used directly to choose the appropriate RST instruction.
- The one draw back to this scheme is that the only way to change the priority of the devices connected to the 74366 is to reconnect the hardware.



Multiple Interrupts and Priority



The 8085 Maskable/Vectored Interrupts

The 8085 has 4 Masked/Vectored interrupt inputs.

– RST 5.5, RST 6.5, RST 7.5

- They are all **maskable**.
- They are **automatically vectored** according to the following table:

Interrupt	Vector
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

- The vectors for these interrupt fall in between the vectors for the RST instructions. That's why they have names like RST 5.5 (RST 5 and a half).



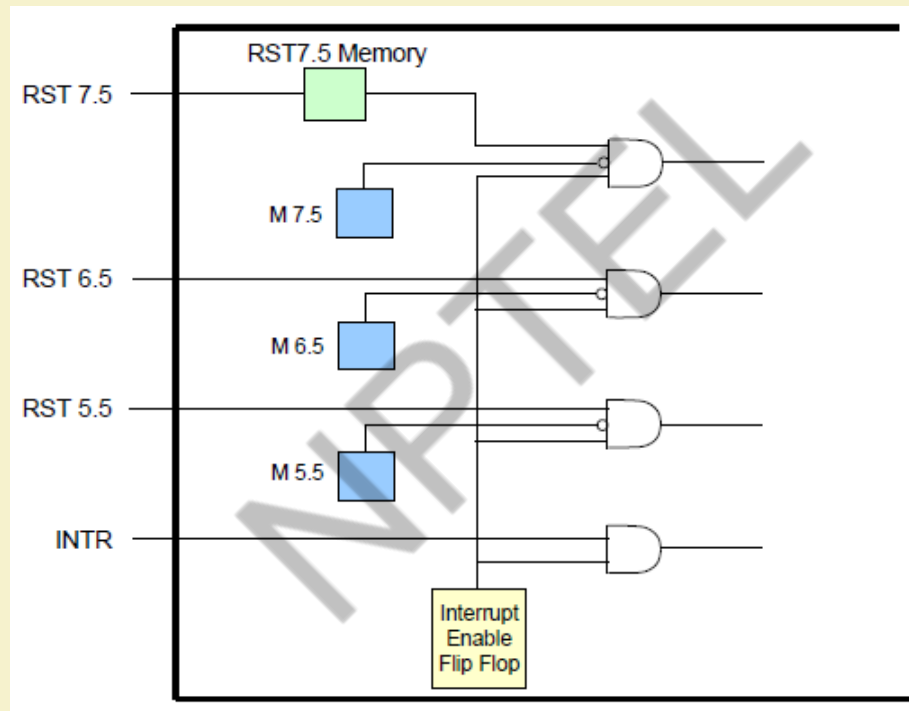
Masking RST 5.5, RST 6.5 and RST 7.5

These three interrupts are masked at two levels:

- Through the Interrupt Enable flip flop and the EI/DI instructions.
 - The Interrupt Enable flip flop controls the whole maskable interrupt process.
- Through individual mask flip flops that control the availability of the individual interrupts.
 - These flip flops control the interrupts individually.



Maskable Interrupts



The 8085 Maskable/Vectored Interrupt Process

1. The interrupt process should be **enabled** using the **EI** instruction.
2. The 8085 checks for an interrupt during the execution of **every** instruction.
3. If there is an interrupt, and if the interrupt is enabled using the interrupt mask, the microprocessor will **complete the executing instruction**, and **reset the interrupt flip flop**.
4. The microprocessor then executes a call instruction that sends the execution to the **appropriate** location in the interrupt vector table.



The 8085 Maskable/Vectored Interrupt Process

5. When the microprocessor executes the call instruction, it **saves the address of the next instruction** on the stack.
6. The microprocessor **jumps to the specific service routine**.
7. The service routine must include the instruction **EI** to re-enable the interrupt process.
8. At the end of the service routine, the **RET** instruction **returns the execution to where the program was interrupted**.

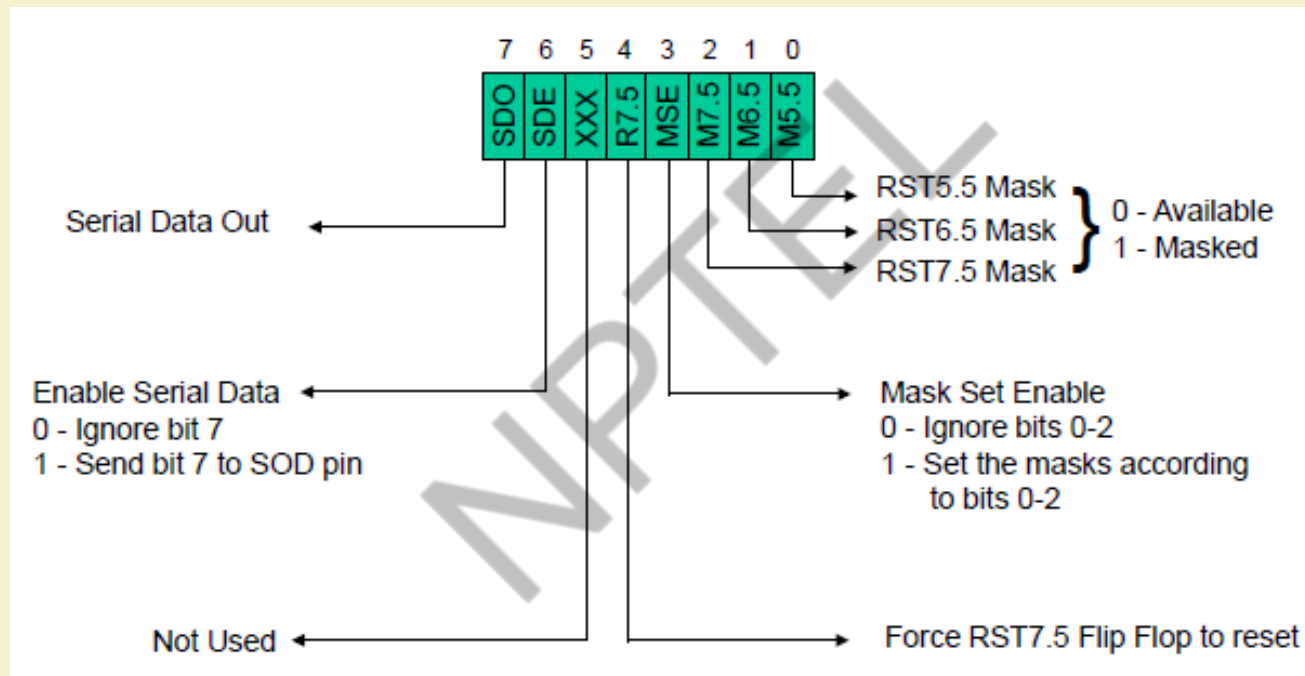


Manipulating the Masks

- The Interrupt Enable flip flop is manipulated using the EI/DI instructions.
- The individual **masks** for RST 5.5, RST 6.5 and RST 7.5 are manipulated using the **SIM** instruction.
 - This instruction takes the bit pattern in the Accumulator and applies it to the interrupt mask enabling and disabling the specific interrupts.



How SIM Interprets the Accumulator



SIM and the Interrupt Mask

- Bit 0 is the **mask** for RST 5.5, bit 1 is the **mask** for RST 6.5 and bit 2 is the **mask** for RST 7.5.
 - If the mask bit is 0, the interrupt is **available**.
 - If the mask bit is 1, the interrupt is **masked**.
- Bit 3 (Mask Set Enable - MSE) is an **enable for setting the mask**.
 - If it is set to 0 the mask is **ignored** and the old settings remain.
 - If it is set to 1, the new setting are **applied**.
 - The SIM instruction is used for multiple purposes and not only for setting interrupt masks.
 - It is also used to control functionality such as Serial Data Transmission.
 - Therefore, bit 3 is necessary to tell the microprocessor whether or not the interrupt masks should be modified



SIM and the Interrupt Mask

- The RST 7.5 interrupt is the **only** 8085 interrupt that has **memory**.

- If a signal on RST7.5 arrives while it is masked, a flip flop will remember the signal.
- When RST7.5 is unmasked, the microprocessor will be interrupted **even if the device has removed the interrupt signal**.
- This flip flop will be **automatically reset** when the microprocessor responds to an RST 7.5 interrupt.

- Bit 4 of the accumulator in the SIM instruction allows **explicitly resetting** the RST 7.5 memory even if the microprocessor did not respond to it.



SIM and the Interrupt Mask

- The SIM instruction can also be used to perform serial data transmission out of the 8085's SOD pin.
 - One bit at a time can be sent out serially over the SOD pin.
- Bit 6 is used to tell the microprocessor whether or not to perform serial data transmission
 - If 0, then do not perform serial data transmission
 - If 1, then do.
- The value to be sent out on SOD has to be placed in bit 7 of the accumulator.
- Bit 5 is not used by the SIM instruction



Using SIM Instruction to Modify Interrupt Masks

Example: Set the interrupt masks so that RST5.5 is enabled, RST6.5 is masked, and RST7.5 is enabled.

– First, determine the contents of the accumulator

- Enable 5.5
- Disable 6.5
- Enable 7.5
- Allow setting the masks
- Don't reset the flip flop
- Bit 5 is not used
- Don't use serial data
- Serial data is ignored

bit 0 = 0
bit 1 = 1
bit 2 = 0
bit 3 = 1
bit 4 = 0
bit 5 = 0
bit 6 = 0
bit 7 = 0

SDO	SDE	XX	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	0	1	0	1	0

Contents of accumulator are: 0AH

EI
MVI A, 0A
SIM

; Enable interrupts including INTR
; Prepare the mask to enable RST 7.5, and 5.5, disable 6.5
; Apply the settings RST masks



Triggering Levels

RST 7.5 is **positive edge sensitive**.

- When a positive edge appears on the RST7.5 line, a logic 1 is **stored** in the flip-flop as a “**pending**” interrupt.
- Since the value has been stored in the flip flop, the line **does not have to be high** when the microprocessor checks for the interrupt to be recognized.
- The line must **go to zero and back to one** before a new interrupt is recognized.

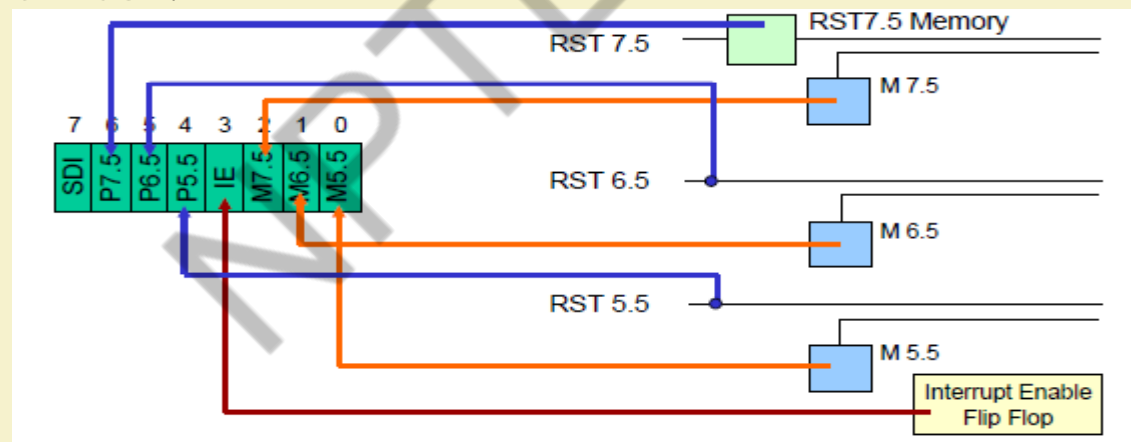
RST 6.5 and RST 5.5 are **level sensitive**.

- The interrupting signal **must remain present until the microprocessor checks for interrupts**.

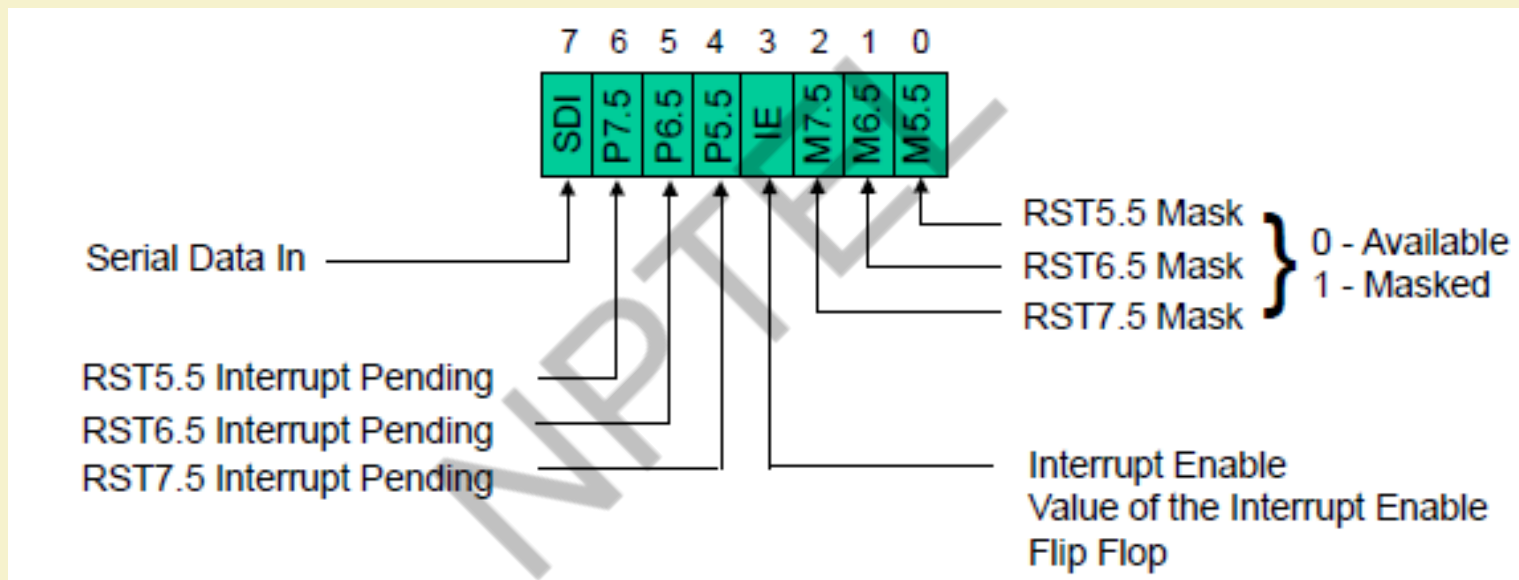


Determining the Current Mask Settings

- RIM instruction: Read Interrupt Mask
 - Load the **accumulator** with an 8-bit pattern showing the status of each interrupt pin and mask.



How RIM sets the Accumulator's different bits



The RIM Instruction and the Masks

- Bits 0-2 show the current **setting of the mask** for each of RST 7.5, RST 6.5 and RST 5.5
 - They return the contents of the three mask flip flops.
 - They can be used by a program to read the mask settings in order to modify only the right mask.
- Bit 3 shows whether the maskable interrupt process is **enabled or not**.
 - It returns the contents of the Interrupt Enable Flip Flop.
 - It can be used by a program to determine whether or not interrupts are enabled.



The RIM Instruction and the Masks

- Bits 4-6 show whether or not there are **pending interrupts** on RST 7.5, RST 6.5, and RST 5.5
 - Bits 4 and 5 return the current value of the RST5.5 and RST6.5 **pins**.
 - Bit 6 returns the current value of the RST7.5 memory **flip flop**.
- Bit 7 is used for **Serial Data Input**.
 - The RIM instruction reads the value of the **SID pin** on the microprocessor and returns it in this bit.



Pending Interrupts

- Since the 8085 has five interrupt lines, interrupts may occur during an ISR and remain pending.
 - Using the **RIM** instruction, the programmer can read the status of the interrupt lines and find if there are any pending interrupts.
- The advantage is being able to find about interrupts on RST 7.5, RST 6.5, and RST 5.5 without having to enable low level interrupts like INTR.



Using RIM and SIM to set Individual Masks

Example: Set the mask to enable RST6.5 without modifying the masks for RST5.5 and RST7.5.

- In order to do this correctly, we need to use the RIM instruction to find the current settings of the RST5.5 and RST7.5 masks.
- Then we can use the SIM instruction to set the masks using this information.
- Given that both RIM and SIM use the Accumulator, we can use some logical operations to mask the un-needed values returned by RIM and turn them into the values needed by SIM.



Using RIM and SIM to set Individual Masks

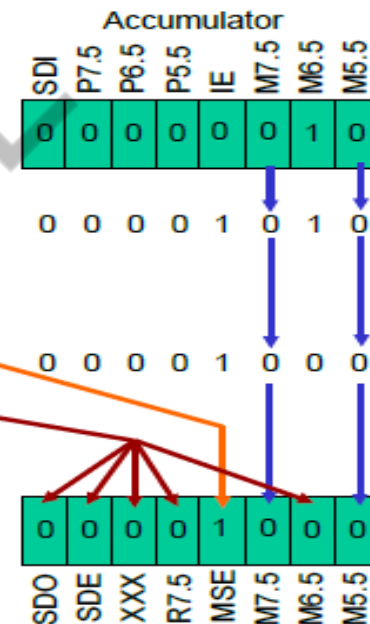
- Assume the RST5.5 and RST7.5 are enabled and the interrupt process is disabled.

RIM ; Read the current settings.

ORI 08H ; 0 0 0 0 1 0 0 0
; Set bit 4 for MSE.

ANI 0DH ; 0 0 0 0 1 1 0 1
; Turn off Serial Data, Don't reset
; RST7.5 flip flop, and set the mask
; for RST6.5 off. Don't cares are
; assumed to be 0.

SIM ; Apply the settings.



TRAP

- TRAP is the only **non-maskable** interrupt.
 - It does not need to be enabled because it **cannot be disabled**.
- It **has the highest priority** amongst interrupts.
- It is **edge and level sensitive**.
 - It needs to be high and stay high to be recognized.
 - Once it is recognized, it won't be recognized again until it goes low, then high again.
- TRAP is usually used for power failure and emergency shutoff.



Internal Interrupt Priority

- Internally, the 8085 implements an **interrupt priority scheme**.
 - The interrupts are ordered as follows:
 - TRAP
 - RST 7.5
 - RST 6.5
 - RST 5.5
 - INTR
 - However, TRAP has lower priority than the HLD signal used for DMA.



The 8085 Interrupts

Interrupt Name	Maskable	Masking Method	Vectored	Memory	Triggering Method
INTR	Yes	DI / EI	No	No	Level Sensitive
RST 5.5 / RST 6.5	Yes	DI / EI SIM	Yes	No	Level Sensitive
RST 7.5	Yes	DI / EI SIM	Yes	Yes	Edge Sensitive
TRAP	No	None	Yes	No	Level & Edge Sensitive



Direct Memory Access

This is a process where data is transferred between two peripherals directly without the involvement of the microprocessor.

– This process employs the HOLD pin on the microprocessor

The external DMA controller sends a signal on the HOLD pin to the microprocessor.

- The microprocessor completes the current operation and sends a signal on HLDA and stops using the buses.
- Once the DMA controller is done, it turns off the HOLD signal and the microprocessor takes back control of the buses.



Basic Concepts in Serial I/O

- Interfacing requirements:
 - Identify the device through a port number.
 - Memory-mapped.
 - Peripheral-mapped.
 - Enable the device using the Read and Write control signals.
 - Read for an input device.
 - Write for an output device.
 - Only one data line is used to transfer the information instead of the entire data bus.



Basic Concepts in Serial I/O

- Controlling the transfer of data:
 - Microprocessor control.
 - Unconditional, polling, status check, etc.
 - Device control.
 - Interrupt.



Synchronous Data Transmission

- The transmitter and receiver are synchronized.
 - A sequence of synchronization signals is sent before the communication begins.
- Usually used for high speed transmission.
 - More than 20 K bits/sec.
- Message based.
 - Synchronization occurs at the beginning of a long message.



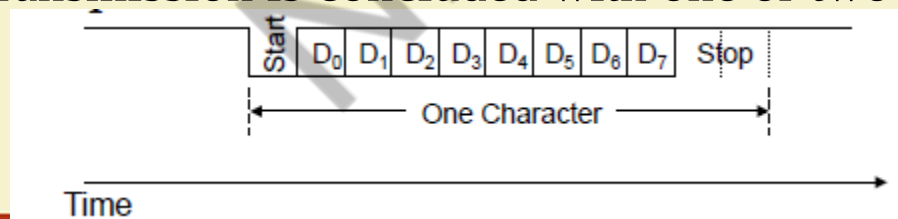
Asynchronous Data Transmission

- Transmission occurs at any time.
- Character based.
 - Each character is sent separately.
- Generally used for low speed transmission.
 - Less the 20 K bits/sec.



Asynchronous Data Transmission

- Follows agreed upon standards:
 - The line is normally at logic one (mark).
 - Logic 0 is known as space.
 - The transmission begins with a start bit (low).
 - Then the seven or eight bits representing the character are transmitted.
 - The transmission is concluded with one or two stop bits.



Simplex and Duplex Transmission

- Simplex.
 - One-way transmission.
 - Only one wire is needed to connect the two devices
 - Like communication from computer to a printer.
- Half-Duplex.
 - Two-way transmission but one way at a time.
 - One wire is sufficient.
- Full-Duplex.
 - Data flows both ways at the same time.
 - Two wires are needed.
 - Like transmission between two computers.



Rate of Transmission

- For parallel transmission, all of the bits are sent at once.
- For serial transmission, the bits are sent one at a time.
 - Therefore, there needs to be agreement on how “long” each bit stays on the line.
- The rate of transmission is usually measured in bits/second or baud.



Length of Each Bit

- Given a certain baud rate, how long should each bit last?
 - Baud = bits / second.
 - Seconds / bits = 1 / baud.
 - At 1200 baud, a bit lasts $1/1200 = 0.83$ m Sec.



Transmitting a Character

- To send the character A over a serial communication line at a baud rate of 56.6 K:
 - ASCII for A is 41H = 01000001.
 - Must add a start bit and two stop bits:
 - 11 01000001 0
 - Each bit should last $1/56.6\text{K} = 17.66 \mu \text{ Sec}$. Known as bit time.
 - Set up a delay loop for $17.66 \mu \text{ Sec}$ and set the transmission line to the different bits for the duration of the loop.



Error Checking

- Various types of errors may occur during transmission.
 - To allow checking for these errors, additional information is transmitted with the data.
- Error checking techniques:
 - Parity Checking.
 - Checksum.
- These techniques are for error checking not correction.
 - They only indicate that an error has occurred.
 - They do not indicate where or what the correct information is



Parity Checking

Make the number of 1's in the data Odd or Even.

- Given that ASCII is a 7-bit code, bit D the parity information.

- Even Parity

The transmitter counts the number of ones in the data. If there is an odd number of 1's, bit D is set to 1 to make the total number of 1's even.

- The receiver calculates the parity of the received message, it should match bit D .

- If it doesn't match, there was an error in the transmission.



Checksum

Used when larger blocks of data are being transmitted.

The transmitter adds all of the bytes in the message without carries. It then calculates the 2's complement of the result and send that as the last byte.

The receiver adds all of the bytes in the message including the last byte. The result should be 0.



RS 232

A communication standard for connecting computers to printers, modems, etc.

- The most common communication standard.
- Defined in the 1950's.
- It uses voltages between +15 and –15 V.
- Restricted to speeds less than 20 K baud.
- Restricted to distances of less than 50 feet (15 m).

The original standard uses 25 wires to connect the two devices. **However, in reality only three of these wires are needed.**

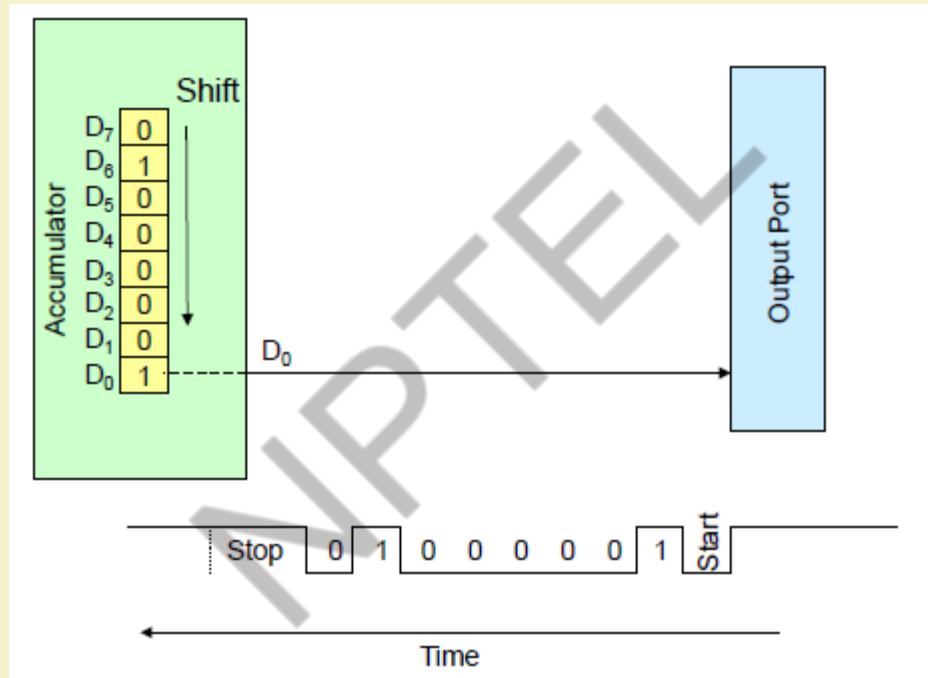


Software-Controlled Serial Transmission

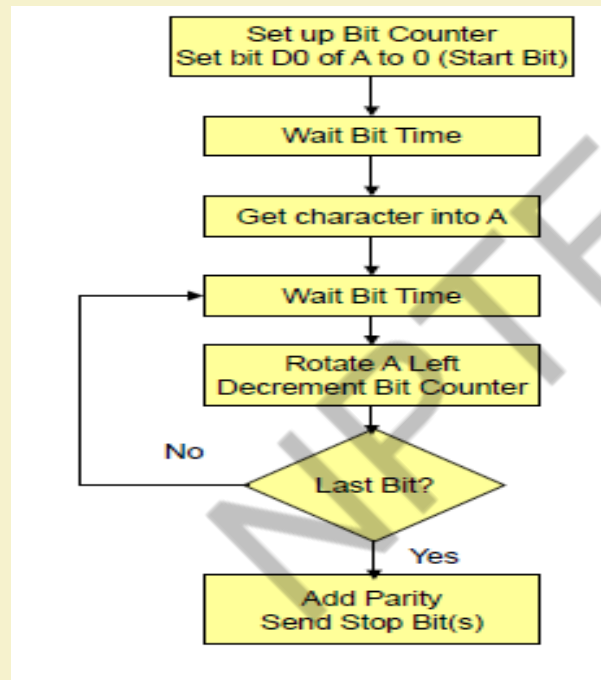
- The main steps involved in serially transmitting a character are:
 - Transmission line is at logic 1 by default.
 - Transmit a start bit for one complete bit length.
 - Transmit the character as a stream of bits with appropriate delay.
 - Calculate parity and transmit it if needed.
 - Transmit the appropriate number of stop bits.
 - Transmission line returns to logic 1.



Serial Transmission



Flowchart of Serial Transmission

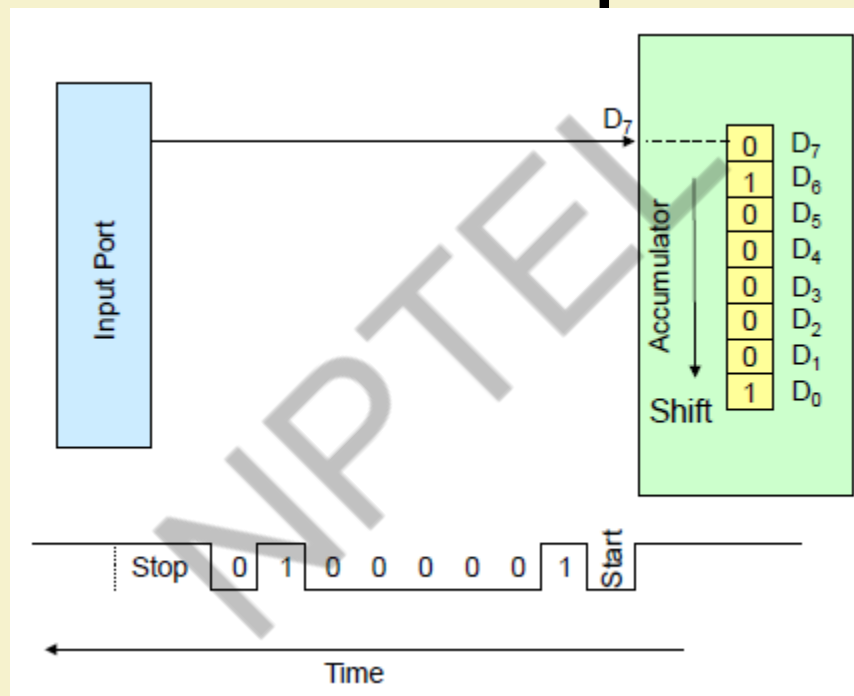


Software-Controlled Serial Reception

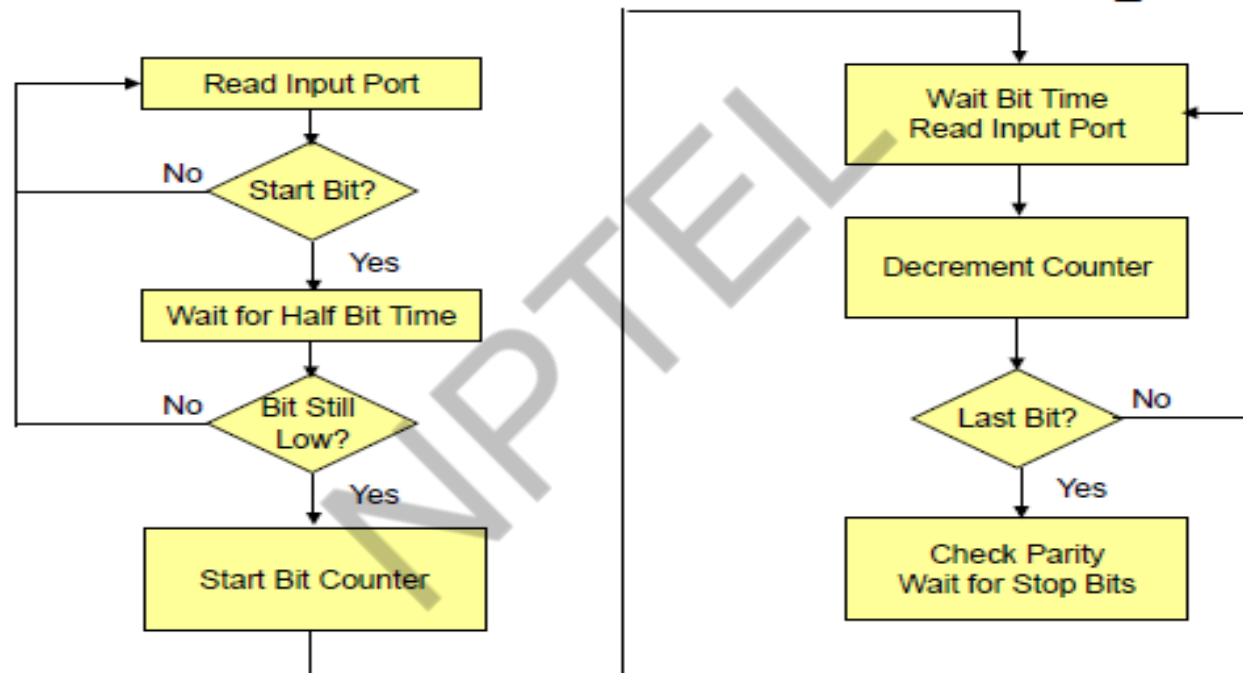
- The main steps involved in serial reception are:
 - Wait for a low to appear on the transmission line.
 - Start bit
 - Read the value of the line over the next 8 bit lengths.
 - The 8 bits of the character.
 - Calculate parity and compare it to bit 8 of the character.
 - Only if parity checking is being used.
 - Verify the reception of the appropriate number of stop bits.



Serial Reception



Flowchart of Serial Reception



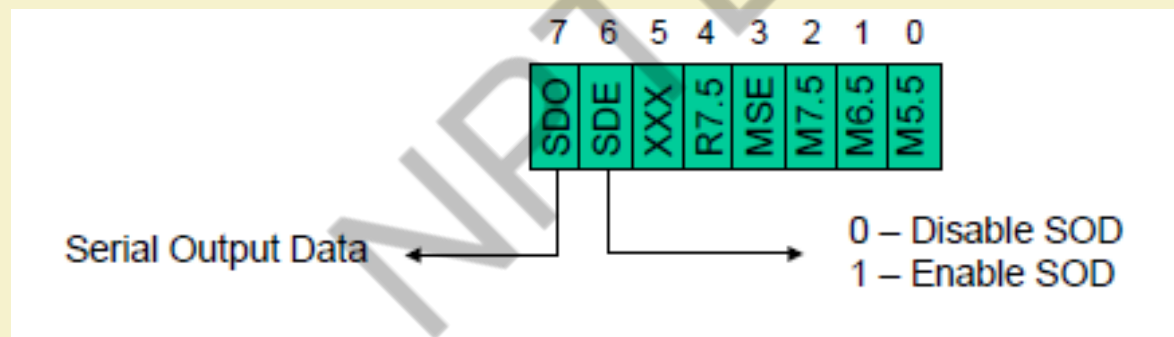
The 8085 Serial I/O Lines

- The 8085 Microprocessor has two serial I/O pins:
 - SOD – Serial Output Data
 - SID – Serial Input Data
- Serial input and output is controlled using the RIM and SIM instructions respectively.



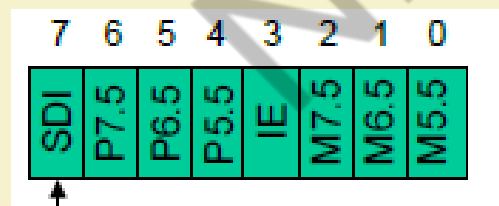
SIM and Serial Output

- The figure below shows how SIM uses the accumulator for Serial Output.



RIM and Serial Input

- Again, the RIM instruction has dual use
 - Reading the current settings of the Interrupt Masks
 - Serial Data Input
- The figure below shows how the RIM instruction uses the Accumulator for Serial Input



Ports?

- Using the SOD and SID pins, the user would not need to bother with setting up input and output ports.
 - The two pins themselves can be considered as the ports.
 - The instructions SIM and RIM are similar to the OUT and IN instructions except that they only deal with the 1-bit SOD and SID ports.



Example

- Transmit an ASCII character stored in register B using the SOD line.

```
SODDATA    MVI    C, 0BH    ; Set up counter for 11 bits
            XRA     A        ; Clear the Carry flag
NXTBIT      MVI     A, 80H    ; Set D7 =1
            RAR        ; Bring Carry into D7 and set D6 to 1
            SIM        ; Output D7 (Start bit)
            CALL     BITTIME
            STC         ; Set Carry to 1
            MOV      A, B    ; Place character in A
            RAR        ; Shift D0 of the character to the carry
                        ; Shift 1 into bit D7
            MOV      B, A    ; Save the interim result
            DCR      C      ; decrement bit counter
            JNZ      NXTBIT
```



Count number of 1's in the contents of D register and store the count in the B register

```
        MVI B, 00H
        MVI C, 08H
        MOV A, D
BACK:   RAR
        JNC SKIP
        INR B
SKIP:   DCR C
        JNZ BACK
        HLT
```



Sort given 10 numbers from memory location 2200H in the ascending order

```
                MVI B, 09    ;Initialize counter 1
START :         LXI H, 2200H ;Initialize memory pointer
                MVI C, 09H   ;Initialize counter 2
BACK:           MOV A, M     ;Get the number
                INX H        ;Increment memory pointer
                CMP M        ;Compare number with next number
                JC SKIP      ;If less, don't interchange
                JZ SKIP      ;If equal, don't interchange
                MOV D, M
                MOV M, A
```



Sort given 10 numbers from memory location 2200H in the ascending order (Contd.)

```
SKIP:      DCX H
           MOV M, D
           INX H      ;Interchange two numbers
           DCR C      ;Decrement counter 2
           JNZ BACK   ;If not zero, repeat
           DCR B      ;Decrement counter 1
           JNZ START
           HLT        ;Terminate program execution
```



Calculate the sum of series of even numbers from the list of numbers. The length of the list is in memory location 2200H and the series itself begins from memory location 2201H. Assume the sum to be 8 bit number so you can ignore carries and store the sum at memory location 2210H.

	LDA 2200H	
	MOV C, A	;Initialize counter
	MVI B, 00H	;sum = 0
	LXI H, 2201H	;Initialize pointer
BACK:	MOV A, M	;Get the number
	ANI 0FH	;Mask Bit 8 to Bit 7
	JNZ SKIP	;Don't add if number is ODD
	MOV A, B	;Get the sum
	ADD M	;SUM = SUM + data
	MOV B, A	;Store result in B register
SKIP:	INX H	;increment pointer
	DCR C	;Decrement counter
	JNZ BACK	;if counter 0 repeat
	STA 2210H	;store sum
	HLT	;Terminate program execution



Unpack the packed BCD number

```
LDA 3000H //Get the packed BCD number from the memory
MOV B,A
MVI C,04
ANI F0 // A = 90H
L1:  RRC // Need to be rotated right for 4 times to get A = 09H
     DCR C
     JNZ L1
     STA 3001
     MOV A,B
     ANI 0F // A = 08H
     STA 3002
     HLT
```

