Interrupt-driven I/O

- Polling takes valuable CPU time
- Open communication only when some data has to be passed -> *Interrupt*.
- I/O interface, instead of the CPU, monitors the I/O device
- When the interface determines that the I/O device is ready for data transfer, it generates an *Interrupt Request* to the CPU
- Upon detecting an interrupt, CPU stops momentarily the task it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing

The problem with programmed I/O is that the processor has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data. The processor, while waiting, must repeatedly interrogate the status of the I/O module. As a result, the level of the performance of the entire system is severely degraded. An alternative is for the processor to issue an I/O command to a module and then go on to do some other useful work. The I/O module will then interrupt the processor to request service when it is ready to exchange data with processor. The processor then executes the data transfer, and then resumes its former processing. The interrupt can be initiated either by software or by hardware.

Interrupt Driven I/O basic operation

- CPU issues read command
- I/O module gets data from peripheral whilst CPU does other work
- I/O module interrupts CPU
- CPU requests data
- I/O module transfers data

Interrupt Processing from CPU viewpoint

- Issue read command
- Do other work
- Check for interrupt at end of each instruction cycle
- If interrupted:-
    - Save context (registers)
    - Process interrupt
    - Fetch data & store

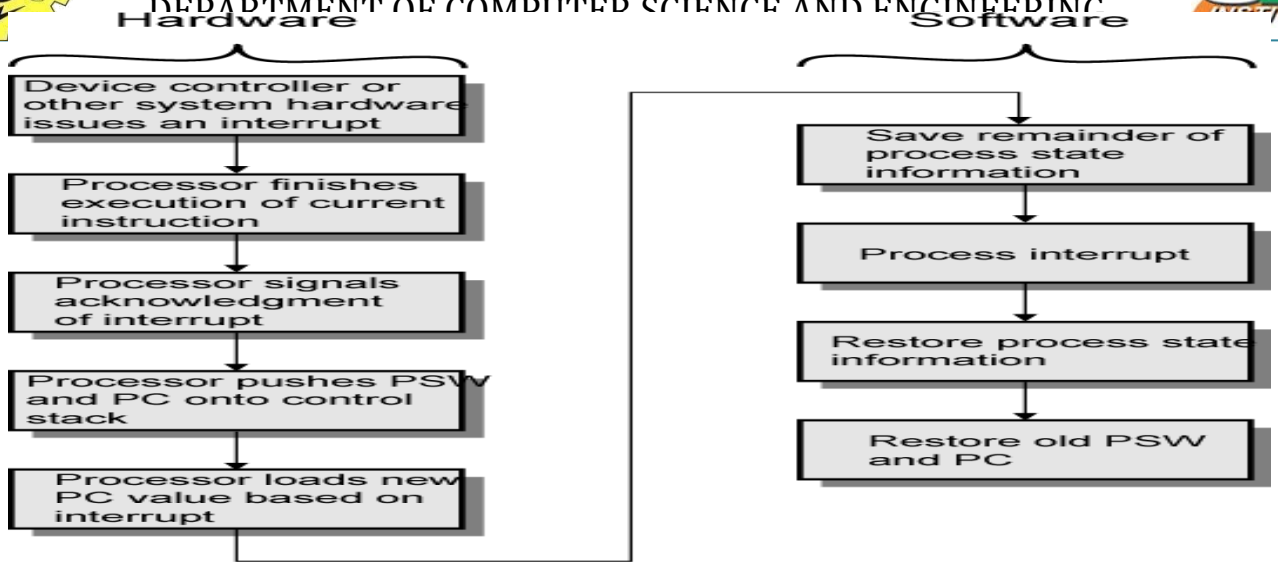


Fig: Simple Interrupt Processing

Priority Interrupt

- Determines which interrupt is to be served first when two or more requests are made simultaneously
- Also determines which interrupts are permitted to interrupt the computer while another is being serviced
- Higher priority interrupts can make requests while servicing a lower priority interrupt

Priority Interrupt by Software (Polling)

- Priority is established by the order of polling the devices (interrupt sources), that is identify the highest-priority source by software means
- One common branch address is used for all interrupts
- Program polls the interrupt sources in sequence
- The highest-priority source is tested first
- Flexible since it is established by software
- Low cost since it needs a very little hardware
- Very slow

Priority Interrupt by Hardware

- Require a priority interrupt manager which accepts all the interrupt requests to determine the highest priority request
- Fast since identification of the highest priority interrupt request is identified by the hardware
- Fast since each interrupt source has its own interrupt vector to access directly to its own service routine

1. Daisy Chain Priority (Serial)

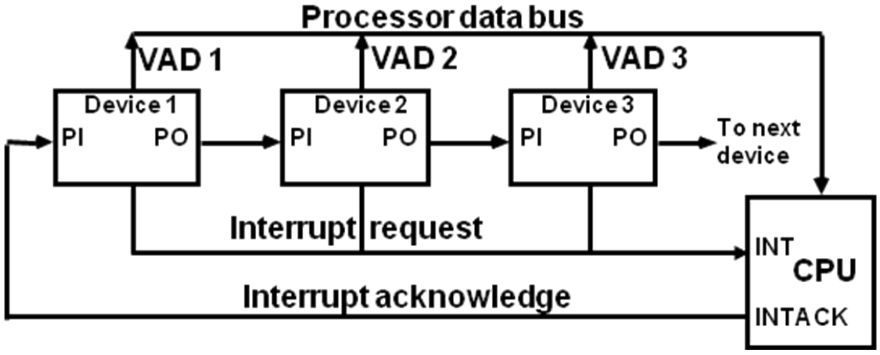
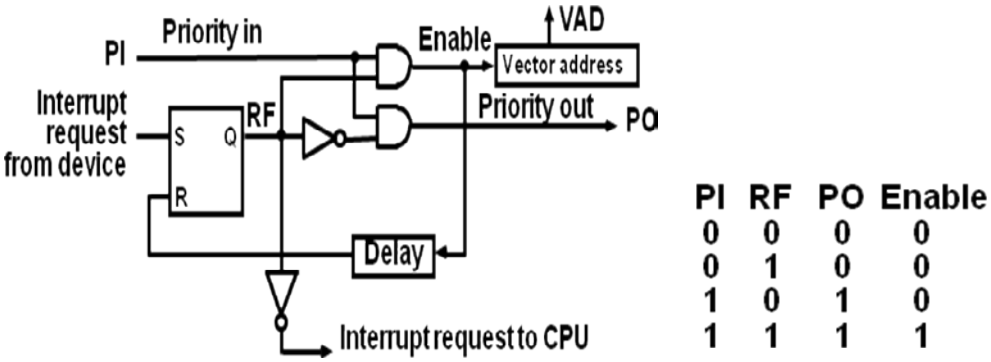


Fig: Daisy Chain priority Interrupt

- Interrupt Request from any device
- CPU responds by INTACK
- Any device receives signal(INTACK) at PI puts the VAD on the bus
- Among interrupt requesting devices the only device which is physically closest to CPU gets INTACK and it blocks INTACK to propagate to the next device



PI	RF	PO	Enable
0	0	0	0
0	1	0	0
1	0	1	0
1	1	1	1

Fig: One stage of Daisy chain priority arrangement

2. Parallel Priority

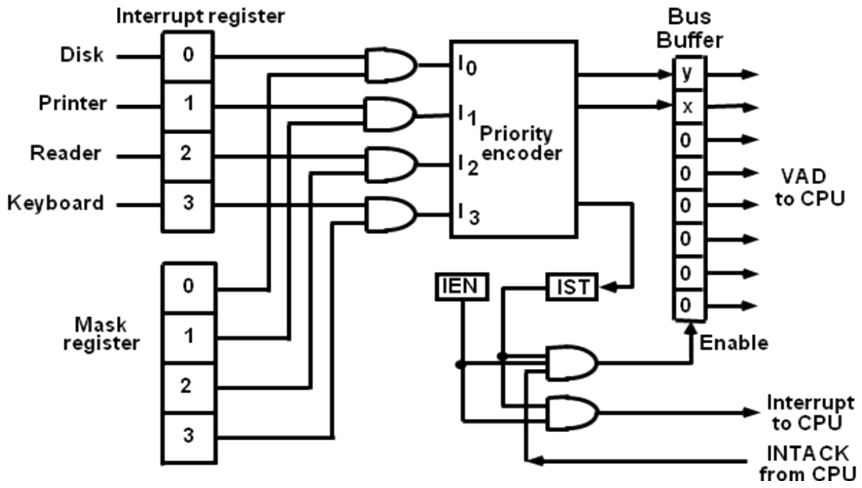


Fig: Parallel priority interrupts hardware

- IEN: Set or Clear by instructions ION or IOF
- IST: Represents an unmasked interrupt has occurred. INTACK enables tristate Bus Buffer to load VAD generated by the Priority Logic
- Interrupt Register:
  - Each bit is associated with an Interrupt Request from different Interrupt Source - different priority level
  - Each bit can be cleared by a program instruction
- Mask Register:
  - Mask Register is associated with Interrupt Register
  - Each bit can be set or cleared by an Instruction

Priority Encoder

- Determines the highest priority interrupt when more than one interrupts take place

Inputs				Outputs			
I_0	I_1	I_2	I_3	x	y	IST	Boolean functions
1	d	d	d	0	0	1	
0	1	d	d	0	1	1	
0	0	1	d	1	0	1	$x = I_0' I_1'$
0	0	0	1	1	1	1	$y = I_0' I_1 + I_0' I_2'$
0	0	0	0	d	d	0	$(IST) = I_0 + I_1 + I_2 + I_3$

Fig: Priority Encoder Truth Table

Interrupt Cycle

At the end of each Instruction cycle

- CPU checks IEN and IST

- If IEN and IST = 1, CPU -> Interrupt Cycle
    - SP □ SP – 1; Decrement stack pointer
    - M[SP] □ PC; Push PC into stack
    - INTACK □ 1; Enable interrupt acknowledge
    - PC □ VAD; Transfer vector address to PC
    - IEN □ 0; Disable further interrupts
    - Go To Fetch to execute the first instruction in the interrupt service routine