LOVELY PROFESSIONAL UNIVERSITY

Input-Output Organization -UNIT 4

- Input-Output Interface
- Asynchronous Data Transfer
- **Modes of Transfer**
- Priority Interrupt
- Direct Memory Access
- Input-Output Processor

Modes of Transfer

⍰Data transfer between central computer and I/O devices may be handled in a variety of modes.

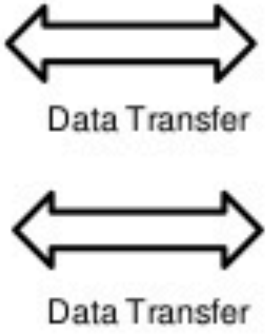
⍰Some modes use CPU as intermediate path and others transfer data directly to and from memory unit.

⍰Data Transfer to or from peripheral can be handled in one of three possible modes :

- ⍰ **Programmed I/O**
- ⍰ **Interrupt-Initiated I/O**
- ⍰ **Direct Memory Access (DMA)**

PROGRAMMED I/O

- ❑ Results of I/O instructions written in the computer program
- ❑ Data transfer initiated by an instruction in the program



❑ CPU register ⟺ Peripheral Device

Data Transfer

❑ CPU ⟺ Memory

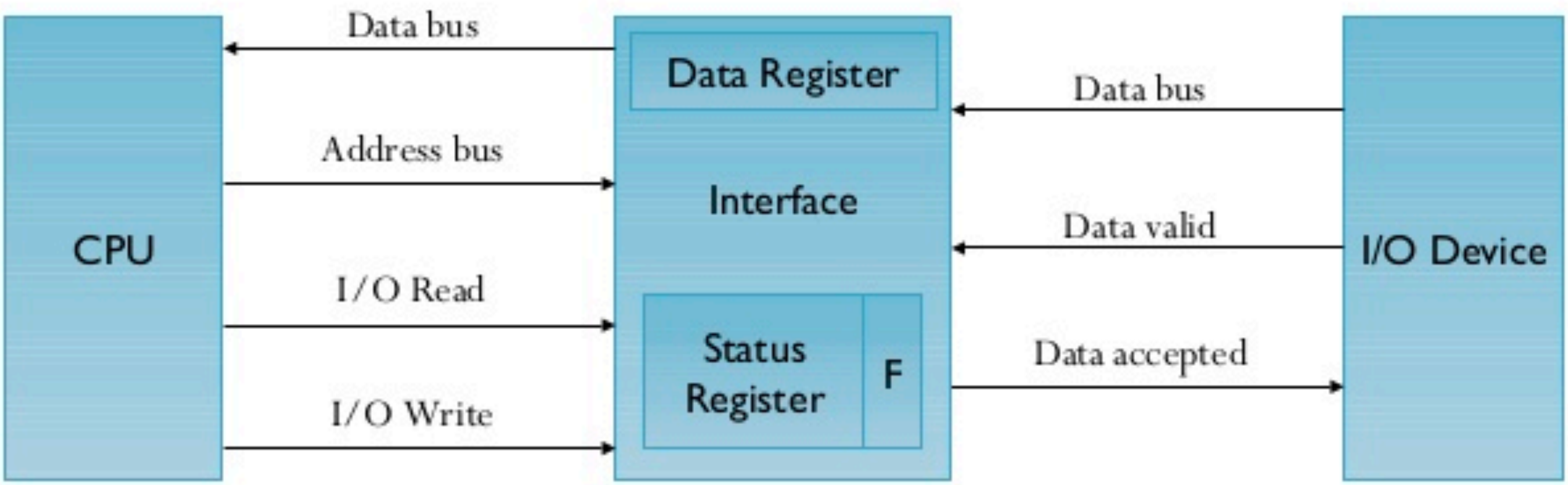
Data Transfer

- ❑ The peripheral has to be constantly monitored.
- ❑ Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made.

APPLICATIONS OF PROGRAMMED I/O METHOD

- ❑ In small low speed computers
- ❑ In systems that are dedicated to monitor a device continuously
- ❑ In the data register
- ❑ To check the status of the flag bit and branch

An example of data transfer from an I/O device through an interface into a CPU in given below:



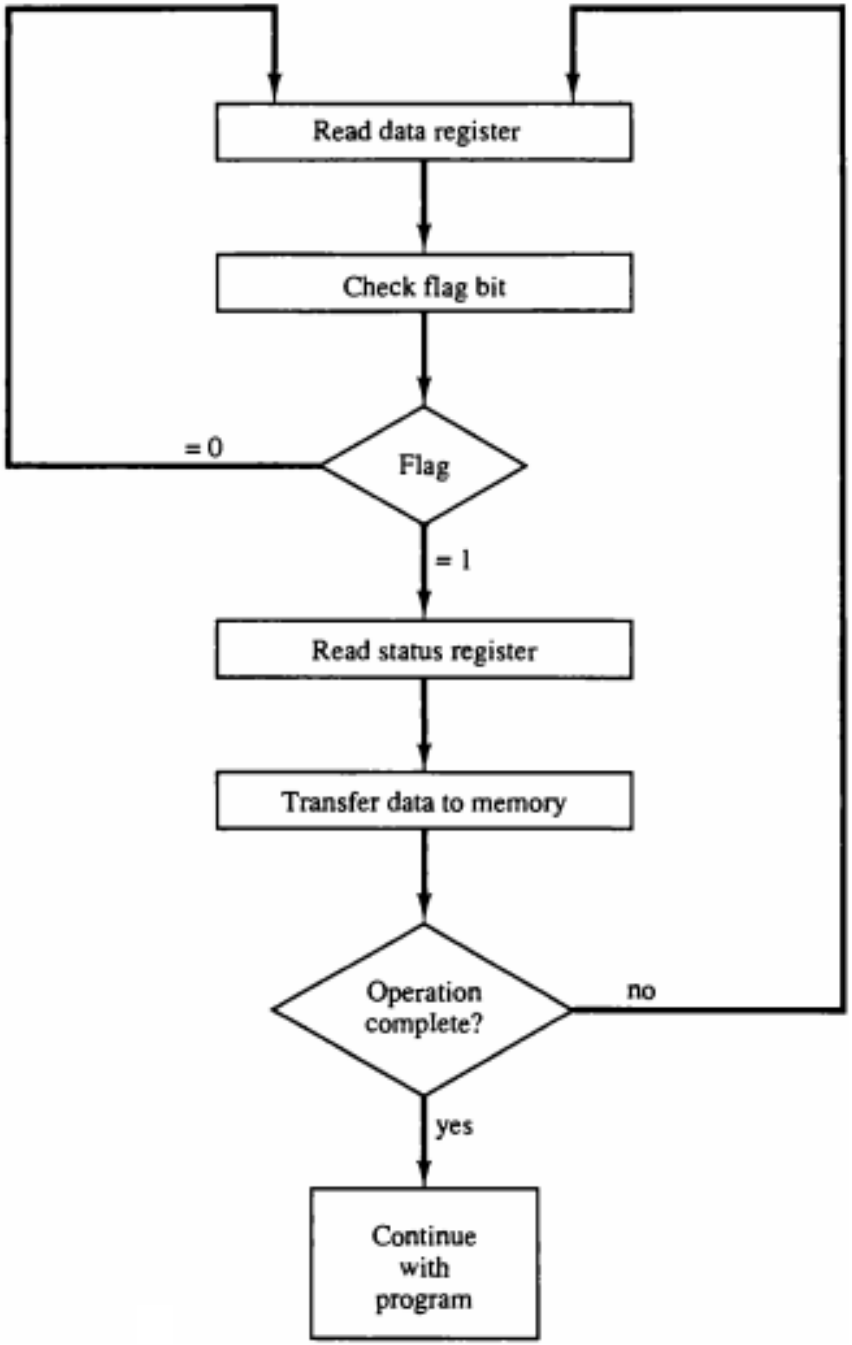


Figure ·11 Flowchart for CPU program to input data.

Programmed I/O method

- In programmed I/O method, CPU stays in a program loop until the I/O unit indicated that it is ready for data transfer.
- This is a time consuming process since it keeps the processor busy needlessly.
- It can be avoided by using **Interrupt** facility and special commands to inform the interface to issue an interrupt request signal when data are available for the device.

Q-In programmed I/O , CPU reads the data from data register , if flag is equal to

A= 0

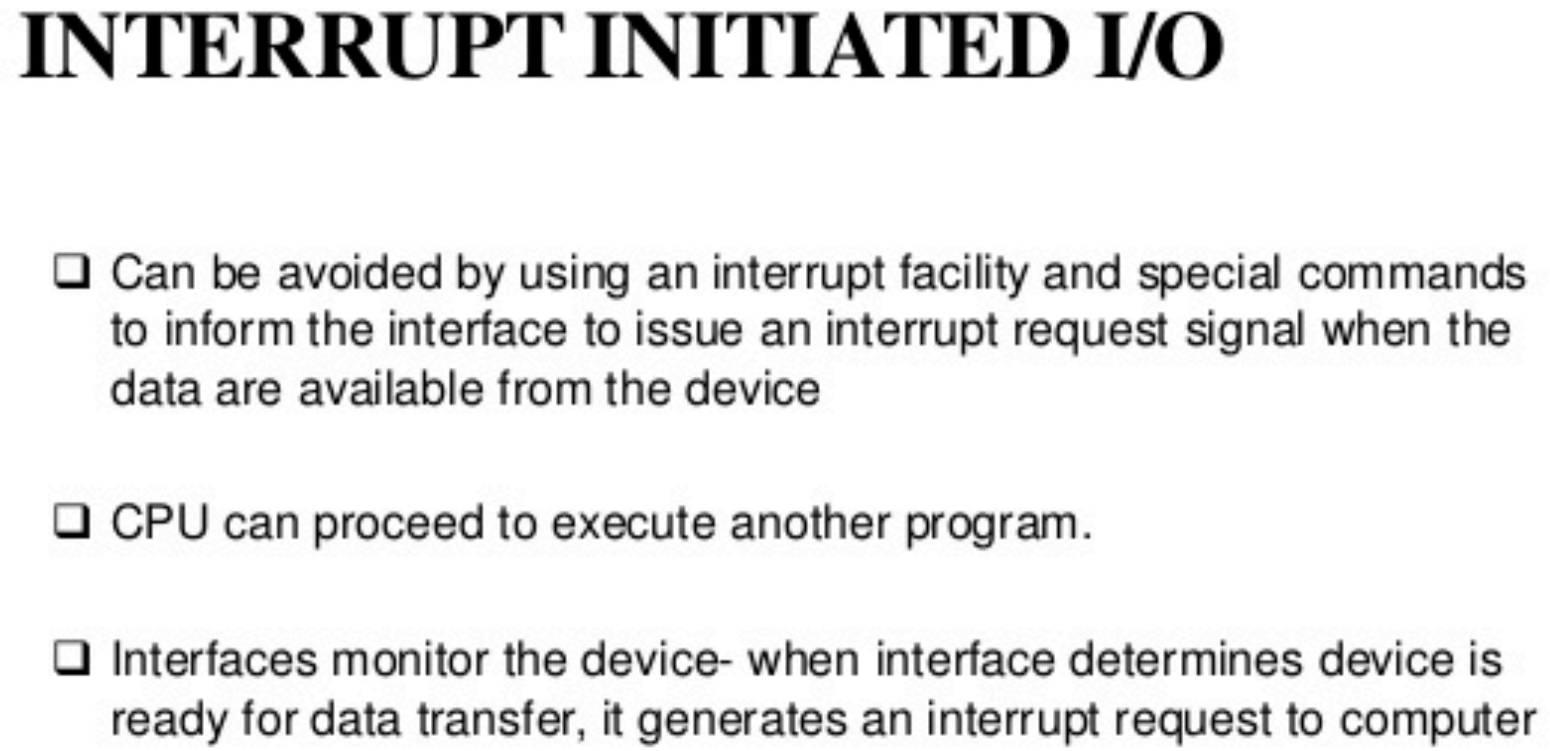
B=1

- **An alternative** to the CPU constantly monitoring the flag is to let the interface inform the computer when it is ready to transfer data.
- **This mode of transfer** uses the interrupt facility. While the CPU is running a program, it does not check the flag.
- **However**, when the flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed of the fact that the flag has been set.
- **The CPU deviates** from what it is doing to take care of the input or output transfer.
- **After the transfer** is completed, the computer returns to the previous program to continue what it was doing before the interrupt.
- **The CPU responds** to the interrupt signal by storing the return address from the program counter into a memory stack and then control branches to a service routine that processes the required I/O transfer.

- **In principle**, there are two methods for accomplishing the way that the processor chooses the branch address of the service routine varies from one unit to another..

- **One is called vectored** interrupt and the other, **nonvectored interrupt.** In a non vectored interrupt, the branch address is assigned to a fixed location in memory.

- **In a vectored interrupt**, the source that interrupts supplies the branch information to the computer. This information is called the interrupt vector.



INTERRUPT INITIATED I/O

- ❑ Can be avoided by using an interrupt facility and special commands to inform the interface to issue an interrupt request signal when the data are available from the device
- ❑ CPU can proceed to execute another program.
- ❑ Interfaces monitor the device- when interface determines device is ready for data transfer, it generates an interrupt request to computer

Priority Interrupts

Priority

- 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

A priority interrupt is a system that establishes priority over the various sources to determine

- **which condition is to serviced first when two or more requests arrive simultaneously**
- **which conditions are permitted to interrupt the computer while another request is being serviced**

Priority Interrupts

Priority Interrupt by Software **(Polling)**

Polling procedure is used to identify highest priority source by software means

- common branch address for all the interrupts

- **Priority is established by the order of polling the devices(interrupt sources)**
  - highest priority device is tested first and if interrupt is on , control branches to service routine for this source otherwise next lower priority source is tested

- Flexible since it is established by software
- Low cost since it needs a very little hardware

- Very slow
- if there are many interrupt time required to poll may exceed time available to service IO device

Priority Interrupts

Priority Interrupt by Hardware

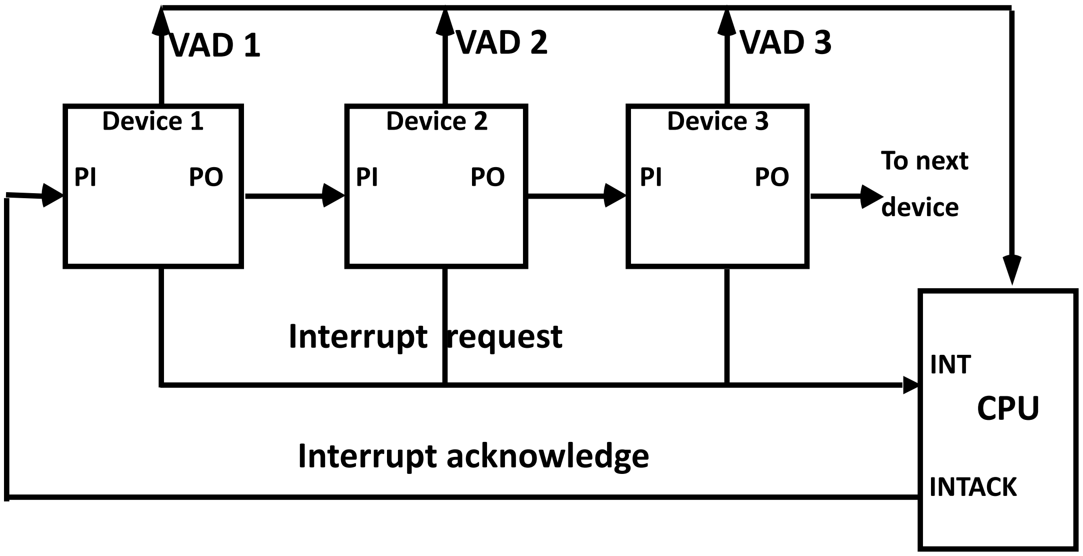
- Require a **priority interrupt manager** which accepts all the interrupt requests to determine the highest priority request.

- 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

- Can be addressed using serial or parallel connection of interrupt lines. Example of serial is Daisy chaining Priority

Hardware Priority Interrupts – Daisy Chain



*** Serial hardware priority function**

*** Interrupt Request Line**

- Single common line

*** Interrupt Acknowledge Line**

- Daisy-Chain

-Serial connection of all device that request an interrupt

-Device with highest priority placed in first position followed by devices with lower priority and so on.

-Interrupt generated by any device [?] signals low state interrupt line

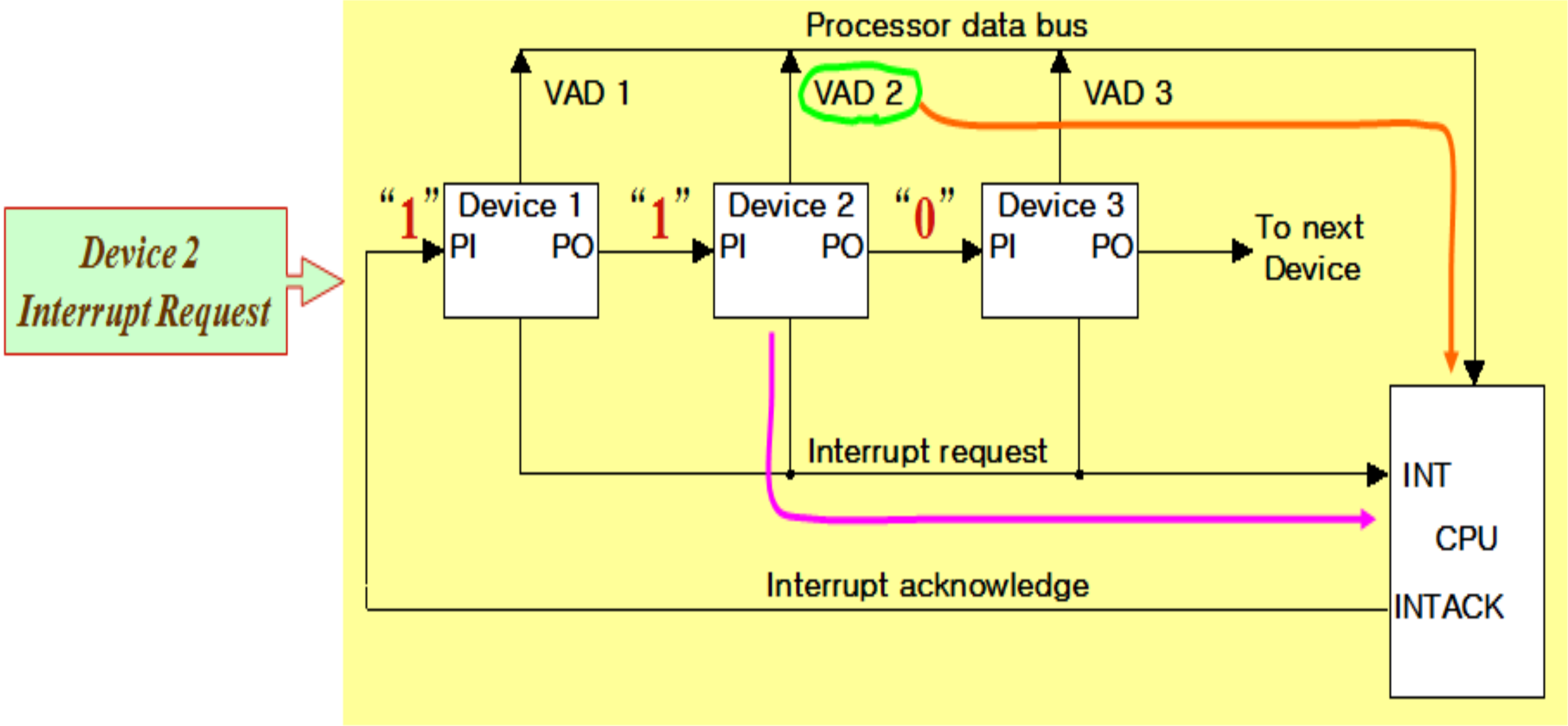
-CPU responds by enabling interrupt acknowledgement (INTACK) line.

- device receives PI=1 and passes to next only when not requesting else PI=0

-Thus device with PI=1 and PO=0 is one with highest priority requesting interrupt

Hardware Priority Interrupts – Daisy Chain

Example: Daisy chain working



Parallel Priority Interrupts

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

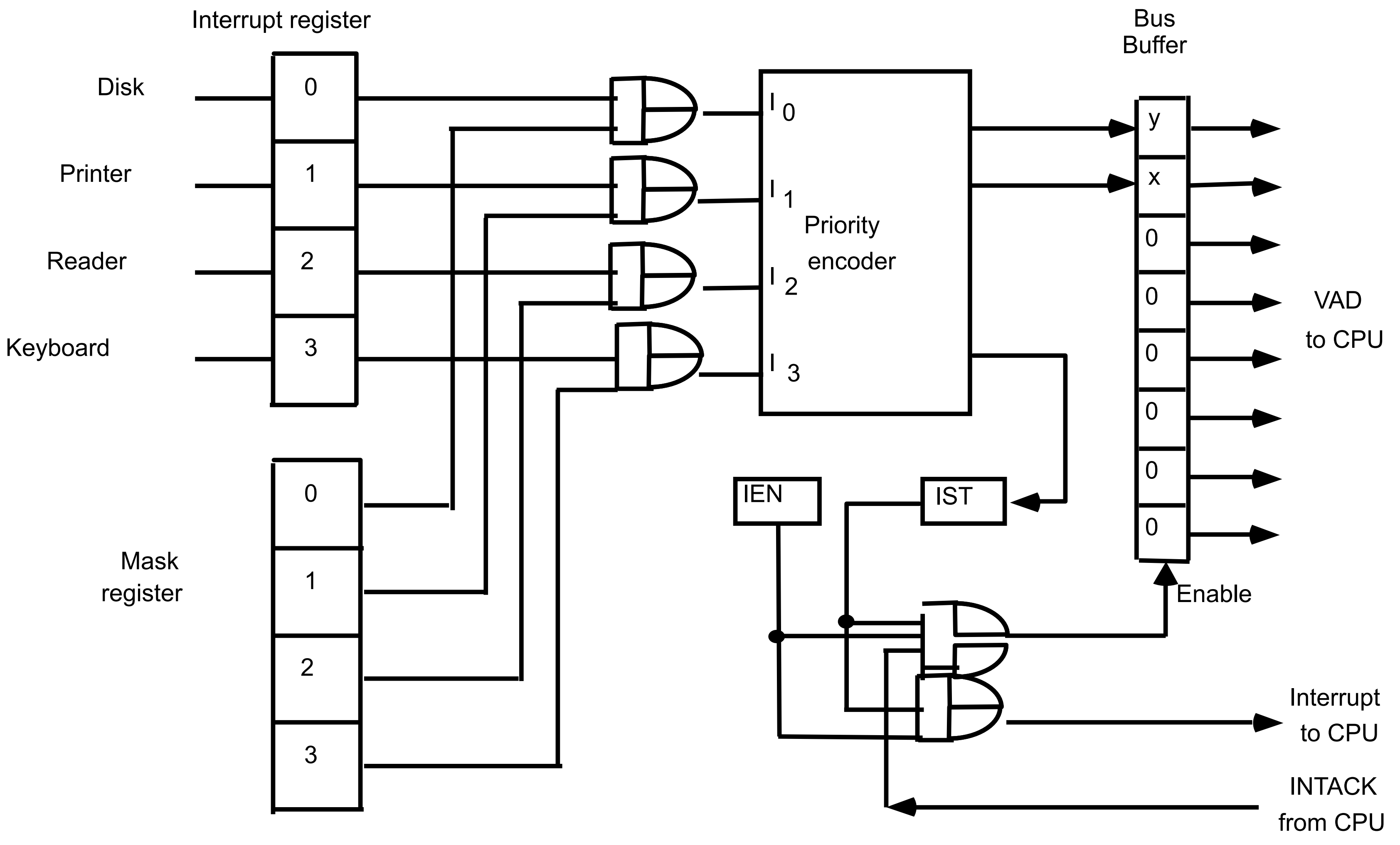
- Each bit can be cleared by a program instruction

Mask Register:

- Mask Register is associated with Interrupt Register

- Each bit can be set or cleared by an Instruction

Parallel Priority Interrupts



Priority Encoder

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

Priority Encoder Truth table

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	$(IST) = I_0 + I_1 + I_2 + I_3$

Interrupt Cycle

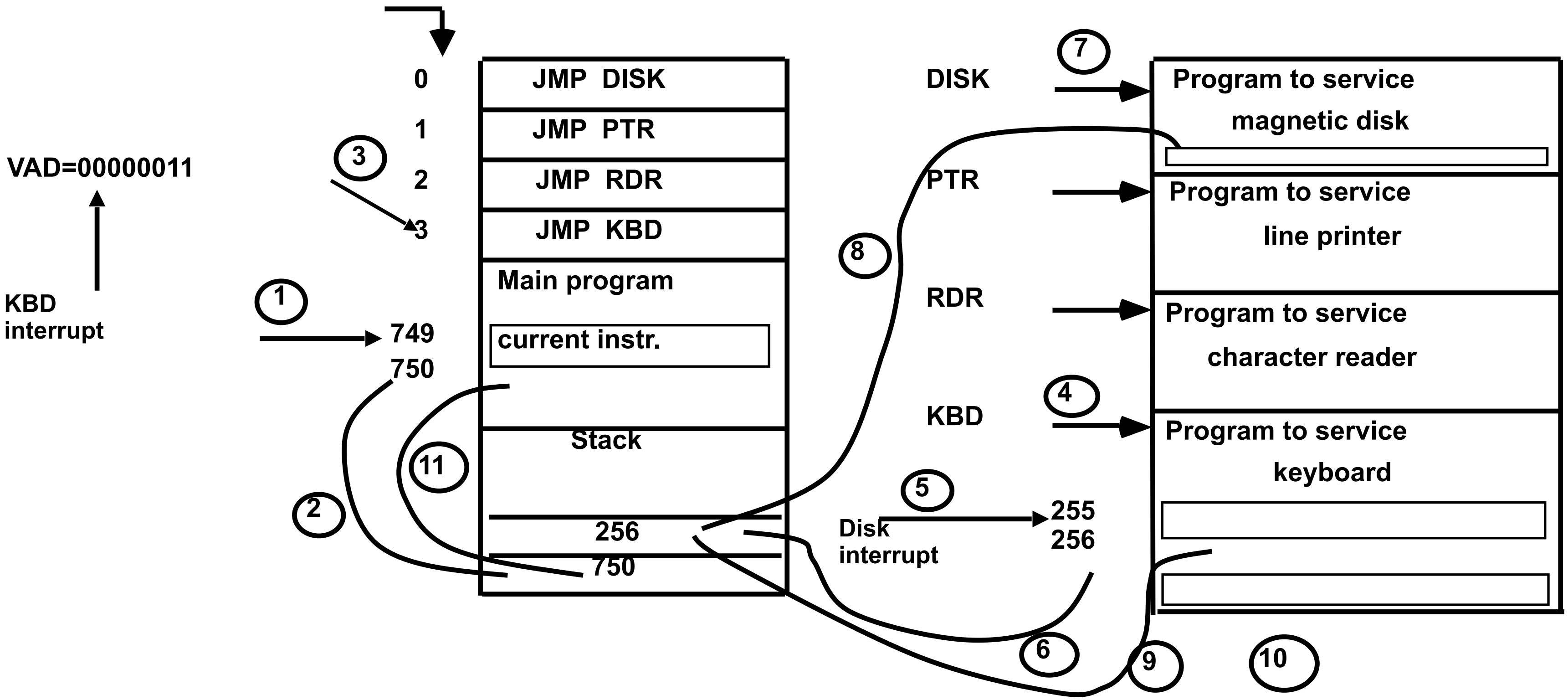
At the end of each Instruction cycle

- **CPU checks IEN and IST**
- **If IEN · 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

Initial and Final Operations



Initial and Final Operations

Each interrupt service routine must have an initial and final set of operations for controlling the registers in the hardware interrupt system

Initial Sequence

[1] Clear lower level Mask reg. bits

[2] IST <- 0

[3] Save contents of CPU registers

[4] IEN <- 1

[5] Go to Interrupt Service Routine

Final Sequence

[1] IEN <- 0

[2] Restore CPU registers

[3] Clear the bit in the Interrupt Reg

[4] Set lower level Mask reg. bits

[5] Restore return address, IEN <- 1

Direct Memory Access

*** Block of data transfer between high speed devices like Disk and Memory**
*** DMA controller - Interface which takes over the buses to manage the transfer directly between**
Memory and I/O Device, freeing CPU for other tasks
*** CPU initializes DMA Controller by sending memory address and the block size (number of words)**

_**Address register:**

Contains an address to specify Desired location in memory

Word count register

Holds no. of words to be transferred

Control register

Specifies the mode of transfer

Direct Memory Access

Fig 1: CPU bus signals for DMA transfer

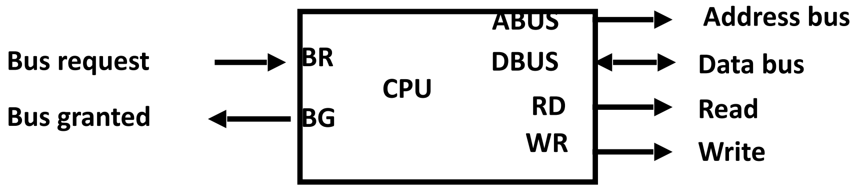
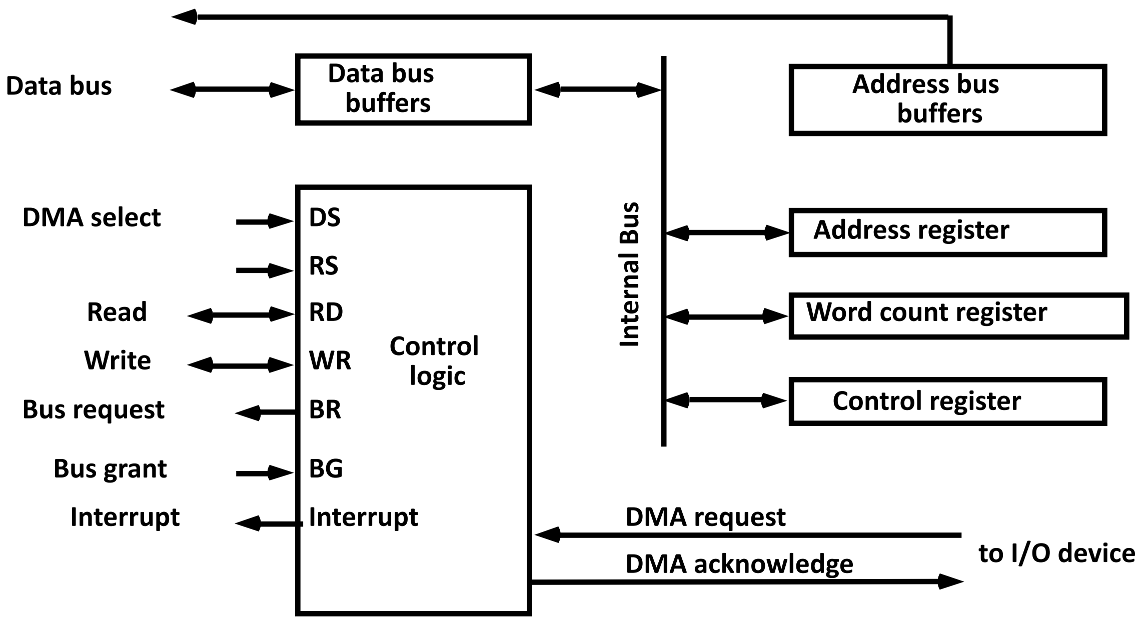


Fig 2: Block diagram of DMA controller



Direct Memory Access

RD and WR is bidirectional

When BG=0 CPU can communicate with DMA Register
When BG=1 CPU left the buses and DMA can communicate directly with memory

DMA Transfer can be made in several ways

(1)Burst Transfer : a block sequence consisting of memory words is transferred in continuous burst while the DMA controller is master of memory bus

- This mode of transfer is needed for fast devices such as magnetic disk where data transmission cannot be stopped or slowed down until an entire block is transferred

(2) Cycle stealing : Alternative technique called cycle stealing allows DMA controller to transfer one data word at time after which it must return control of the buses to the CPU.

- CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to "steal" one memory cycle

DMA I/O Operation

DMA is first initialized by CPU. After that DMA starts and continues to transfer data between memory and peripheral unit until an entire block is transferred.

CPU initializes the DMA by sending following information through data bus:

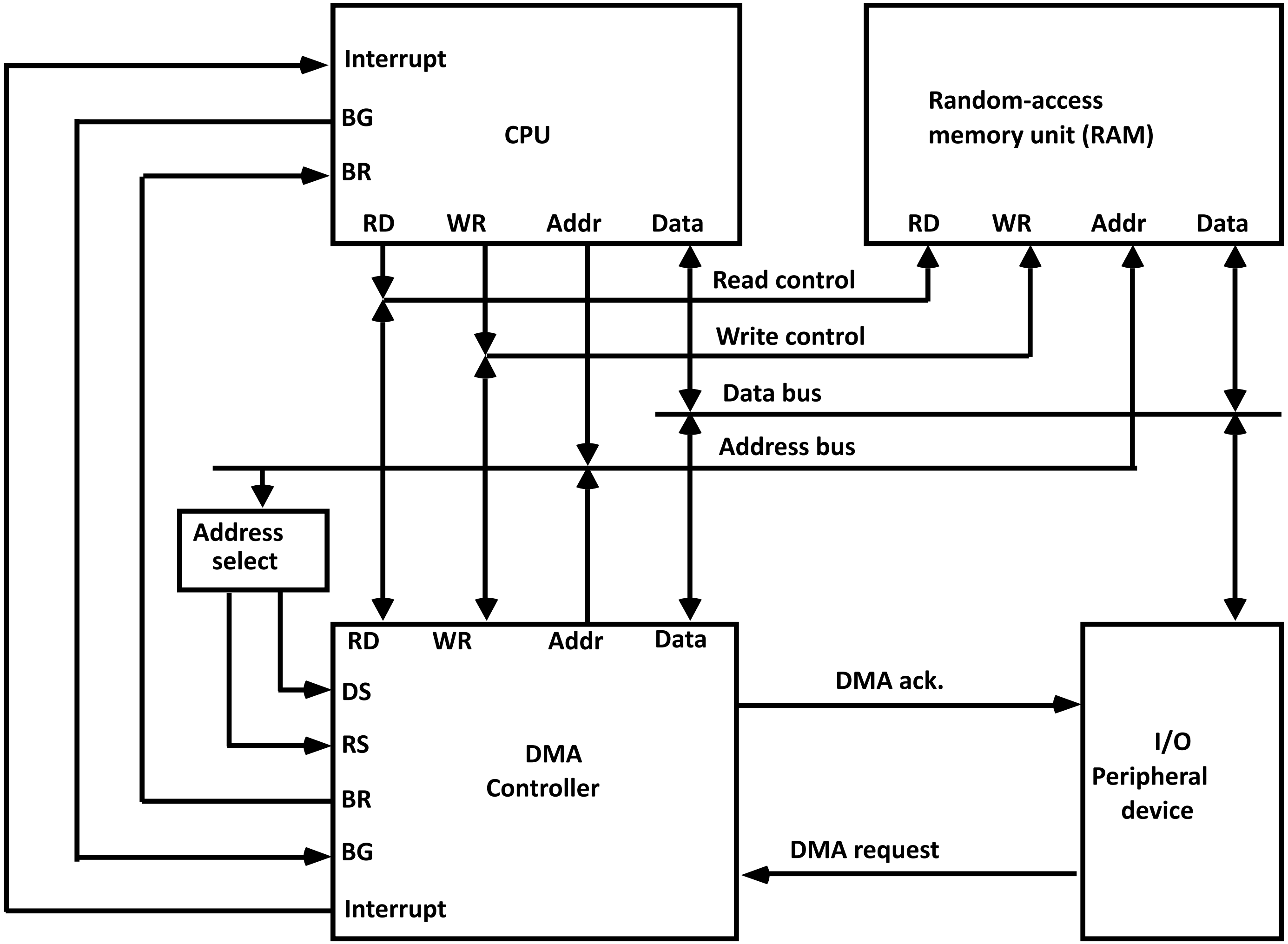
(1) Starting address of the memory block (for read/write)

(2) Word Count (no. of words in memory block)

(3) Control to specify mode of transfer (E.g. read/write)

(4) A control to start DMA Transfer

DMA Transfer



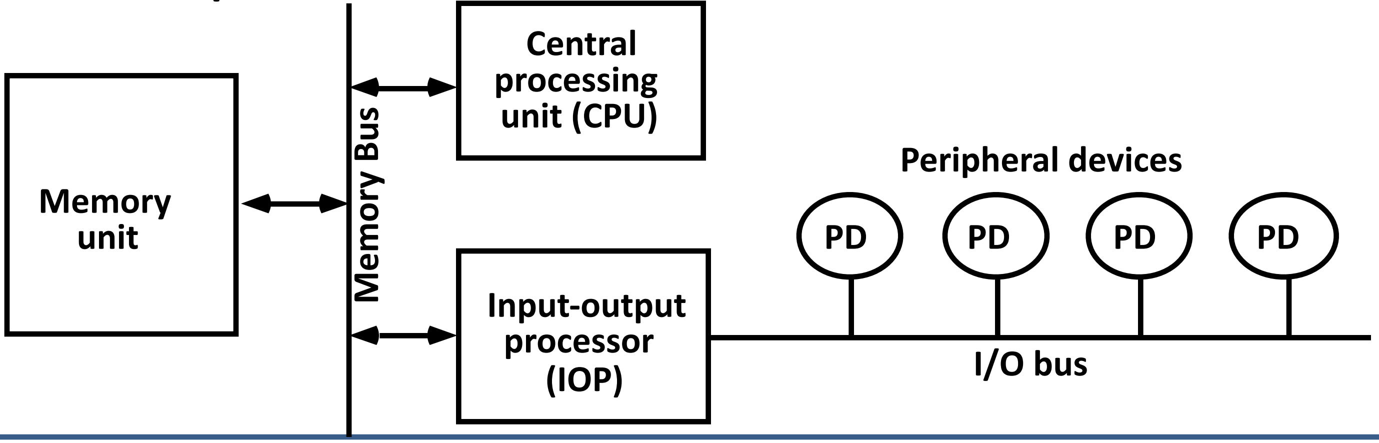
I/O Processor - Channel

Channel

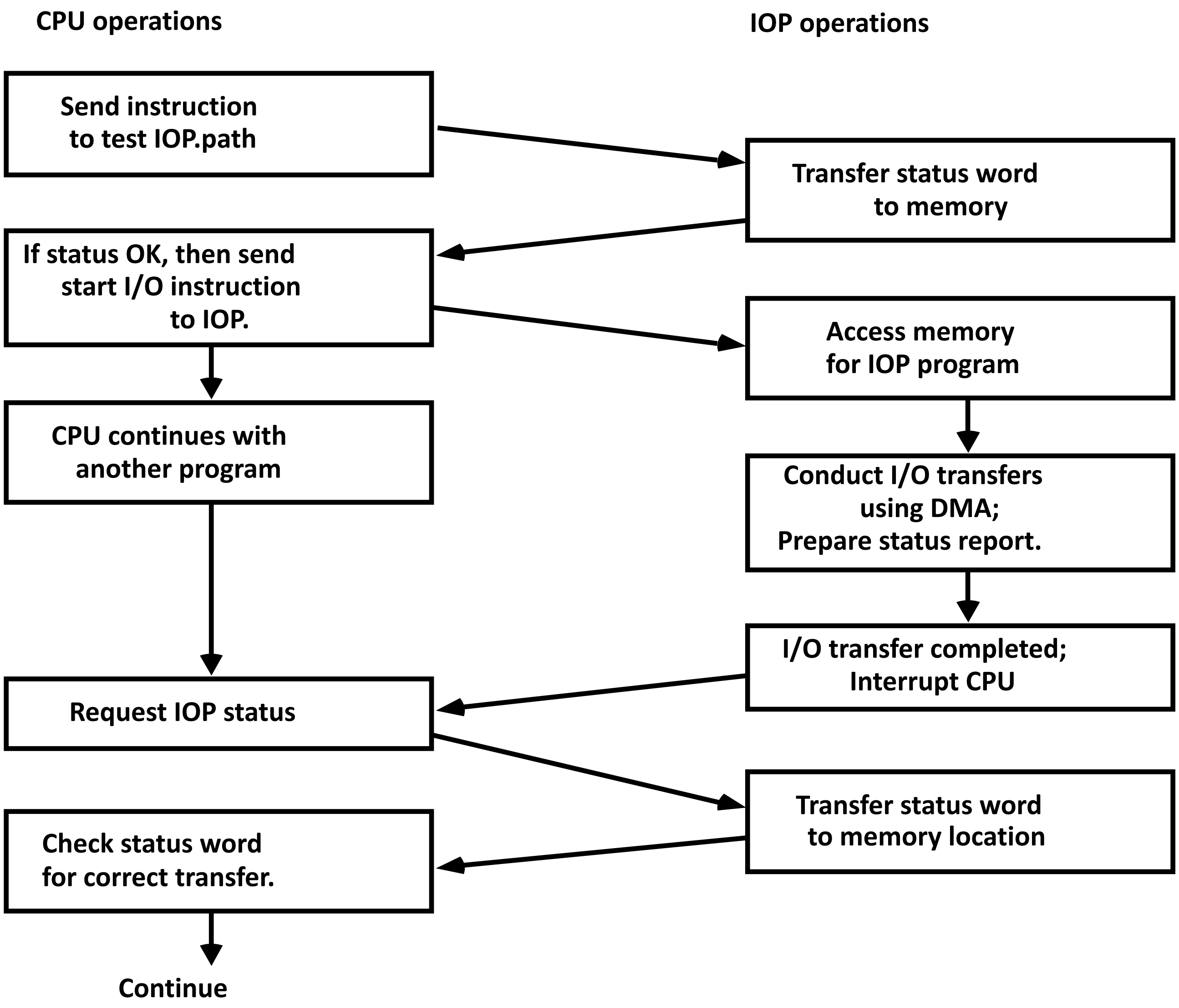
- Processor with direct memory access capability that communicates with I/O devices
- Channel accesses memory by cycle stealing
- Unlike DMA Controller, IOP can fetch and execute its own instruction
- IOP Instructions (Commands) specially designed to facilitate I/O transfer.

- Data gathered in IOP at device rate and bit capacity while CPU executing own program
- Transfer between IOP and Device similar to Programmed I/O and transfer between IOP and Memory similar to DMA
- CPU is master while IOP is slave processor

- CPU initiates the channel by executing a channel I/O class instruction and once initiated, channel operates independent of the CPU



Channel CPU Communication



CPU operations
IOP operations
Send instruction to test IOP.path
Transfer status word to memory
If status OK, then send start I/O instruction to IOP.
Access memory for IOP program
CPU continues with another program
Conduct I/O transfers using DMA; Prepare status report.
I/O transfer completed; Interrupt CPU
Request IOP status
Transfer status word to memory location
Check status word for correct transfer.
Continue