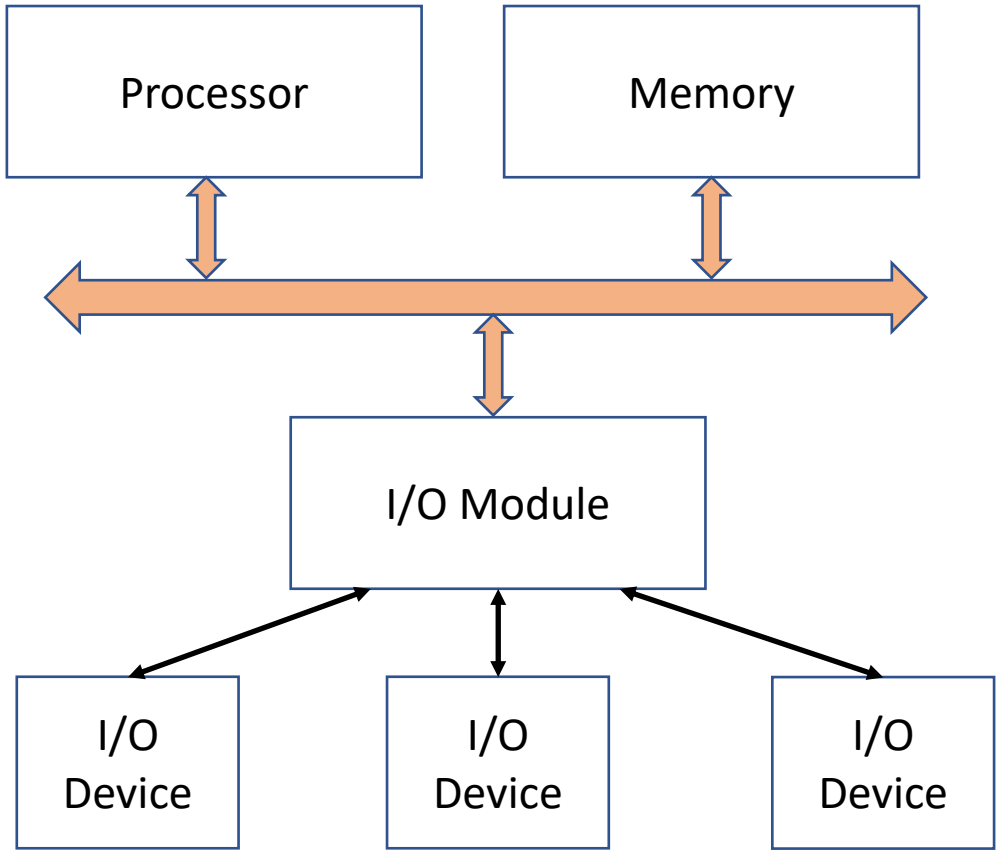
Input – Output Organization

Introduction

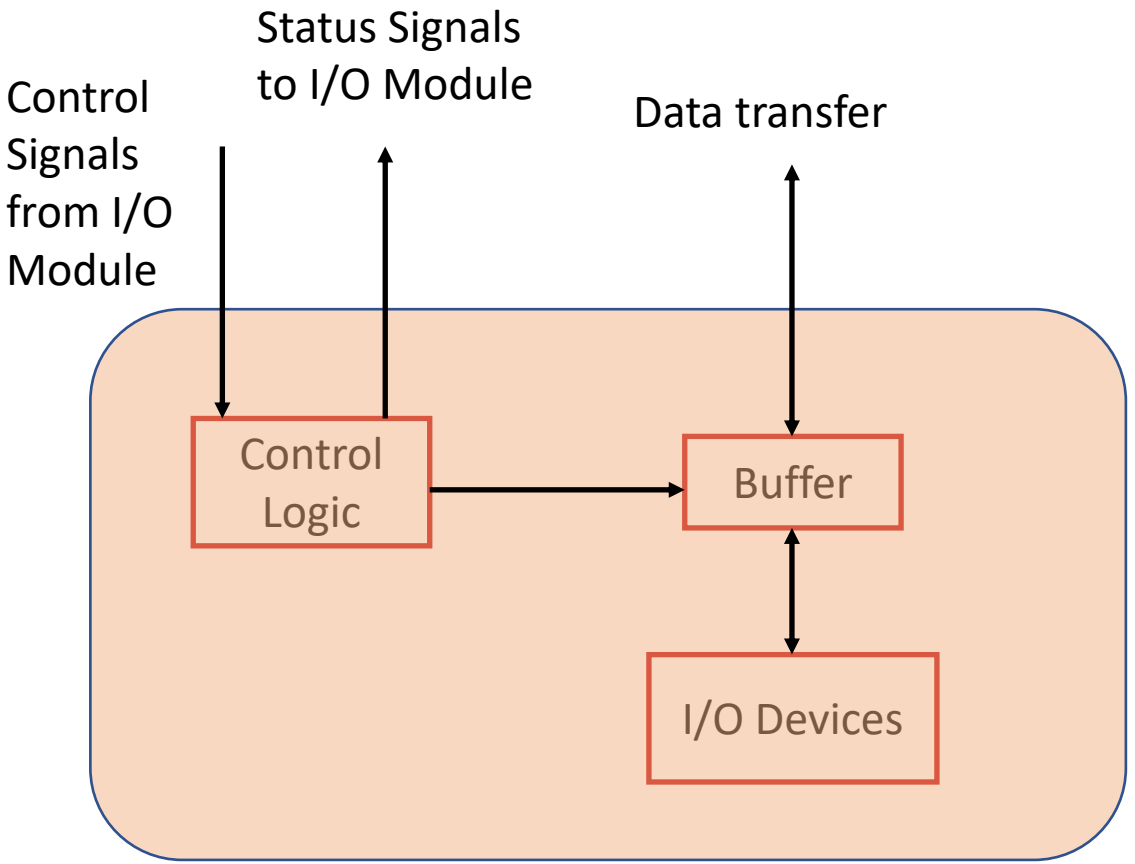
- Interfacing input/output devices is more complex as compared to interfacing memory systems.
- Why?
  - Variety of peripherals (keyboard, mouse, disk, camera, printer, scanner etc.
  - Widely varying speeds.
  - Data transfer rate can be regular or irregular.
  - Sizes of data blocks transferred at a time varies widely (few bytes to Kbytes)
  - Slower than processor and memory.

Input / Output Interface

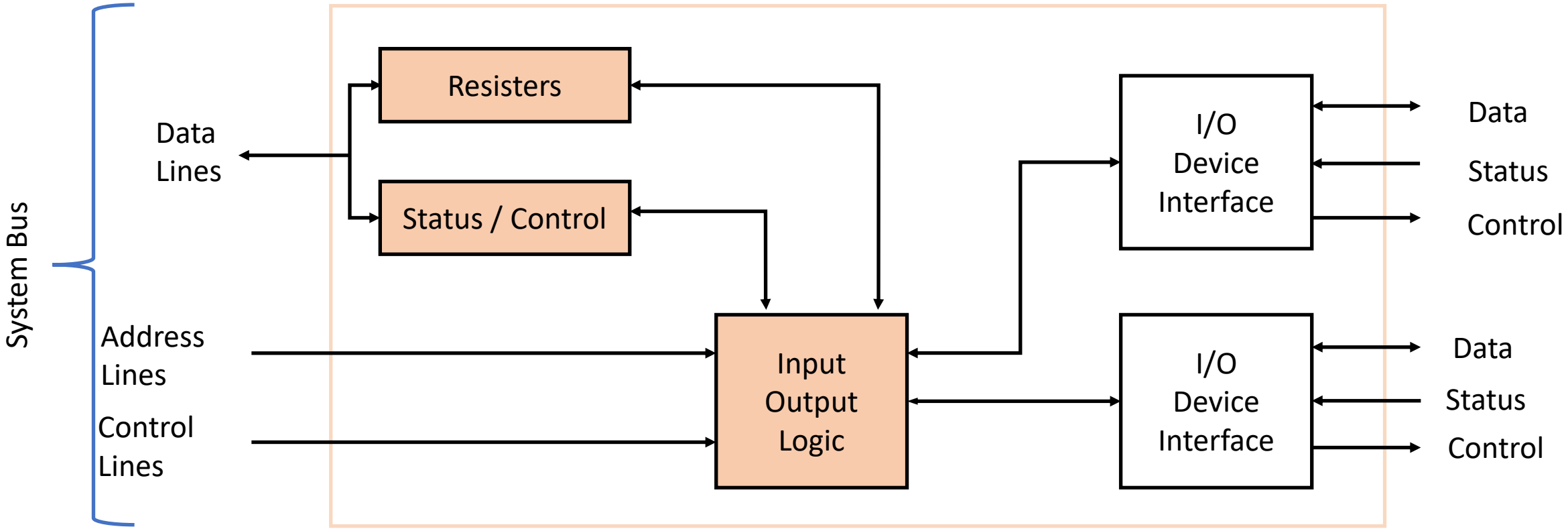
- To handle widely different types of I/O devices, we need a programmable I/O interface or *I/O Module*.
  - Interfaces to processor and memory on one side
  - Interfaces to one or more peripheral devices on the other side.



Typical I/O Device Interface



I/O Module Schematic

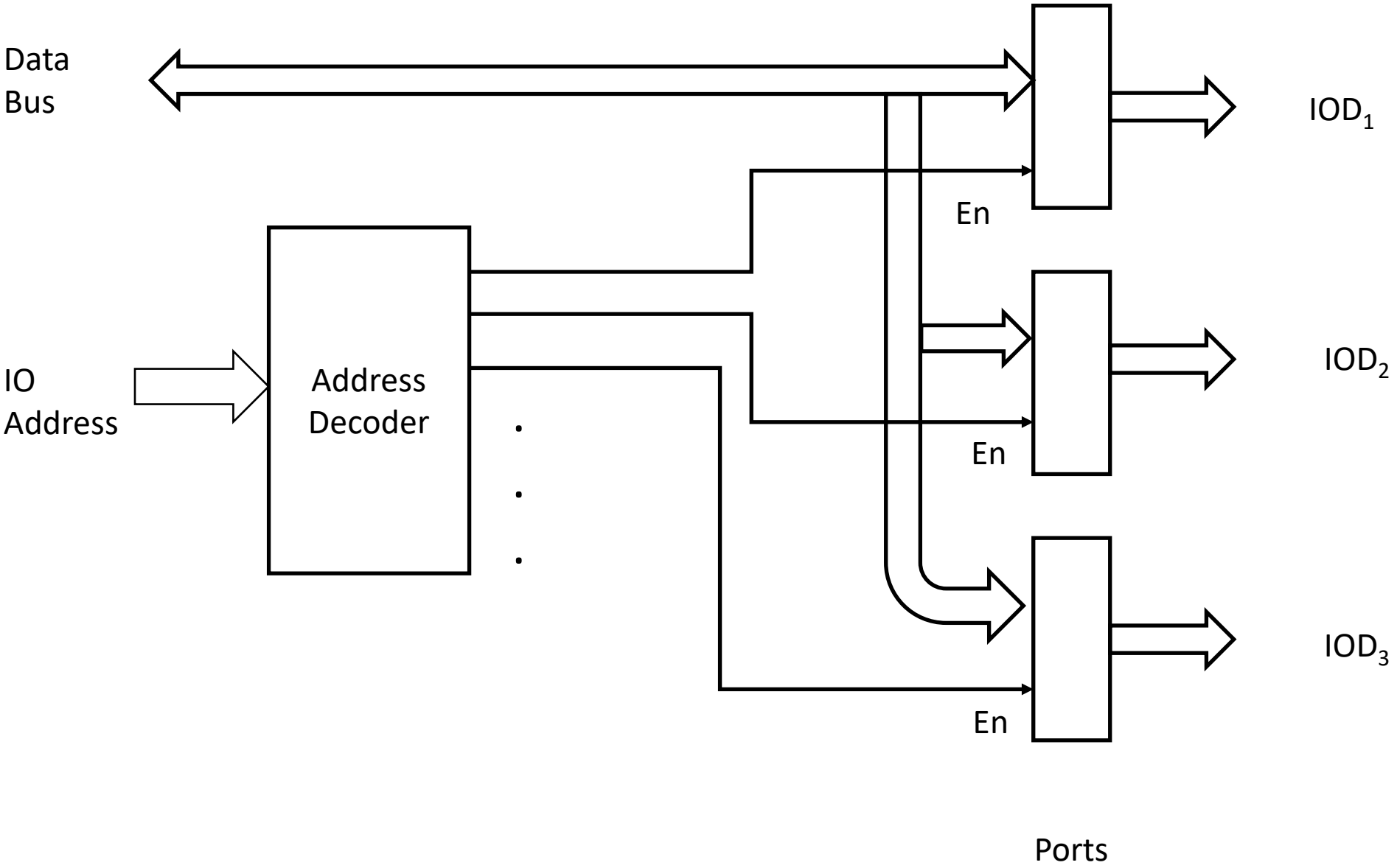


Typical Steps During I/O

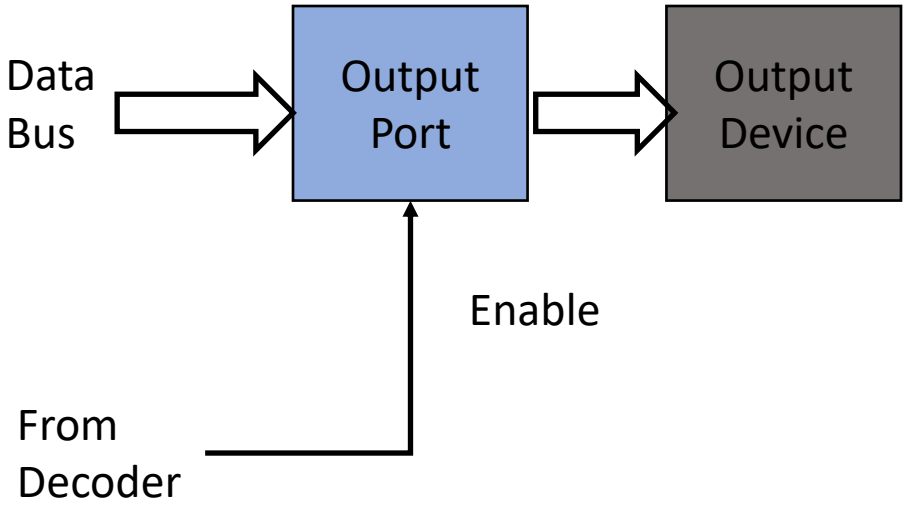
a) Processor requests the I/O Module for device status.
b) I/O Module returns the status to the processor.
c) If the device is ready, processor requests data transfer.
d) I/O Module gets data from device (say, input device).
e) I/O Module transfers data to the processor.
f) Processor stores the data in memory.

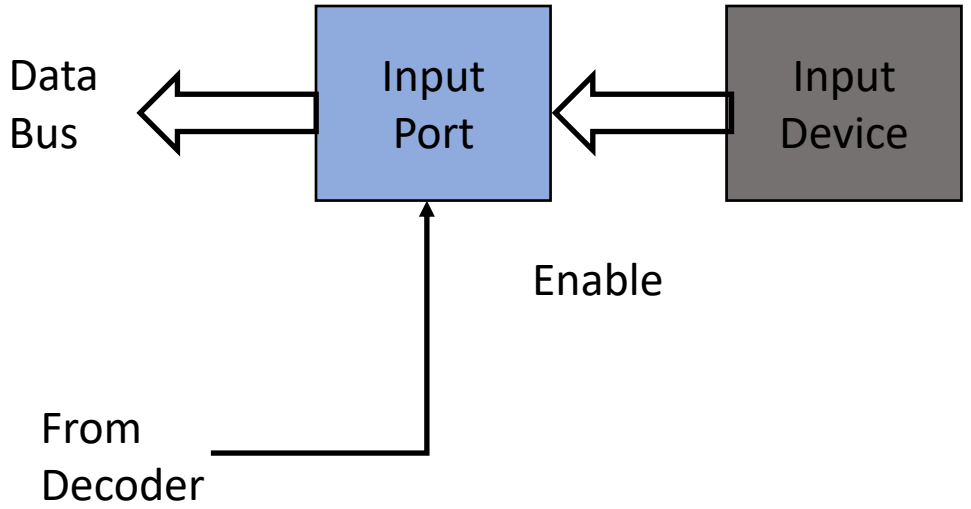
How are I/O devices typically interfaced

- Through input and output ports.
- Output Port:
  - Basically a parallel in parallel out (PIPO) register that is enabled when a particular output device address is given.
  - The register inputs are connected to the data bus, and the register outputs are connected to the output device
- Input Port:
  - Basically a parallel tristate bus driver that is enabled when a particular input device address is given.
  - The driver outputs are connected to the data bus, while the inputs are connected to the input device.


Data
Bus
IO
Address
Address
Decoder
En
En
En
IOD_1
IOD_2
IOD_3
Ports

Example of output and input ports





Memory-Mapped and I/O Mapped Device Interface

- Memory-mapped device interface:
  - The same address decoder selects memory and I/O ports.
  - Some of the memory address space is occupied by I/O ports.
  - All data transfer instructions to/from memory can be used to transfer data to/from I/O devices.
  - The processor need not have separate instructions for I/O , nor it need to specify whether and address generated by the CPU is a memory address or an I/O address.
- I/O mapped device interface:
  - Separate instructions for I/O data transfer (say, IN and OUT).
  - A processor signal identifies whether a generated address refers to an memory location or an I/O device.
  - Separate address decoders for selecting memory and I/O ports
  - The complete memory address space can be utilized.

Data Transfer Techniques

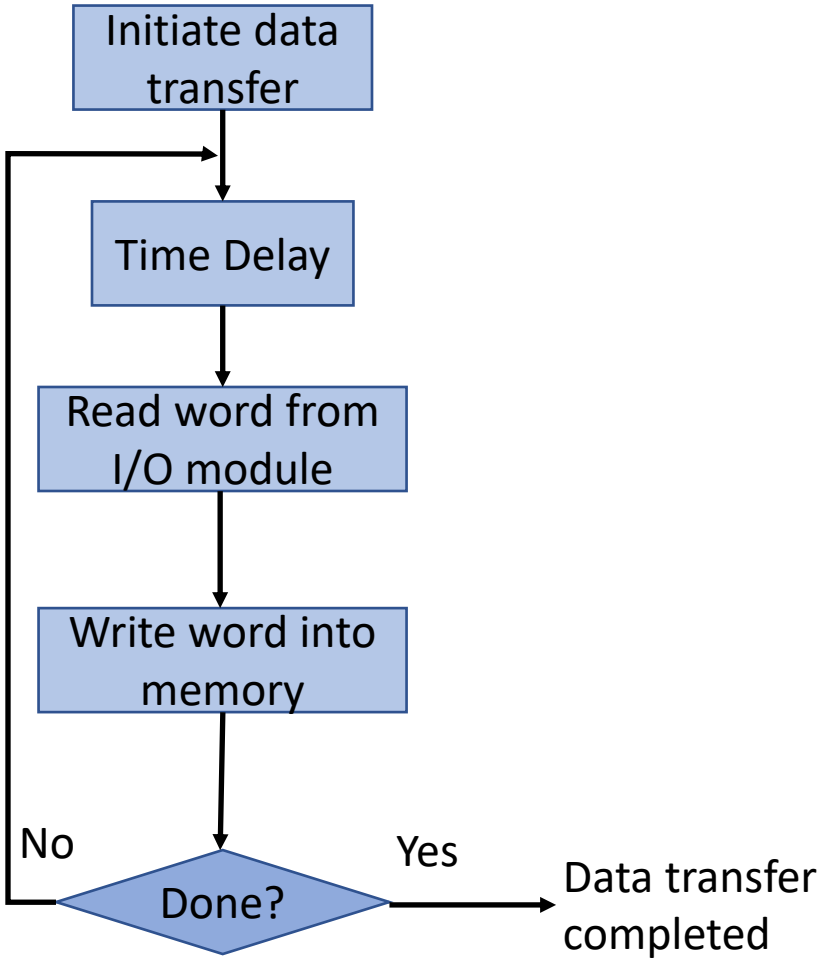
1. **Programmed:** CPU executes a program that data transfers data between I/O device and memory.
   a) Synchronous
   b) Asynchronous
   c) Interrupt-driven

2. **Direct Memory Access(DMA):** An external controller directly transfer data between I/O device and memory without CPU intervention.

(a) Synchronous Data Transfer

- The I/O device transfers data at a fixed rate that is known to the CPU.
- The CPU initiates the I/O operation and transfers successive bytes/words after giving fixed time delays.
- Characteristics:
  - During the time delay, CPU lines idle.
  - Not many I/O devices have strictly synchronous data transfer characteristics.
- A flowchart for synchronous data transfer from an input device is shown on the next slide.

(a) Synchronous Data Transfer

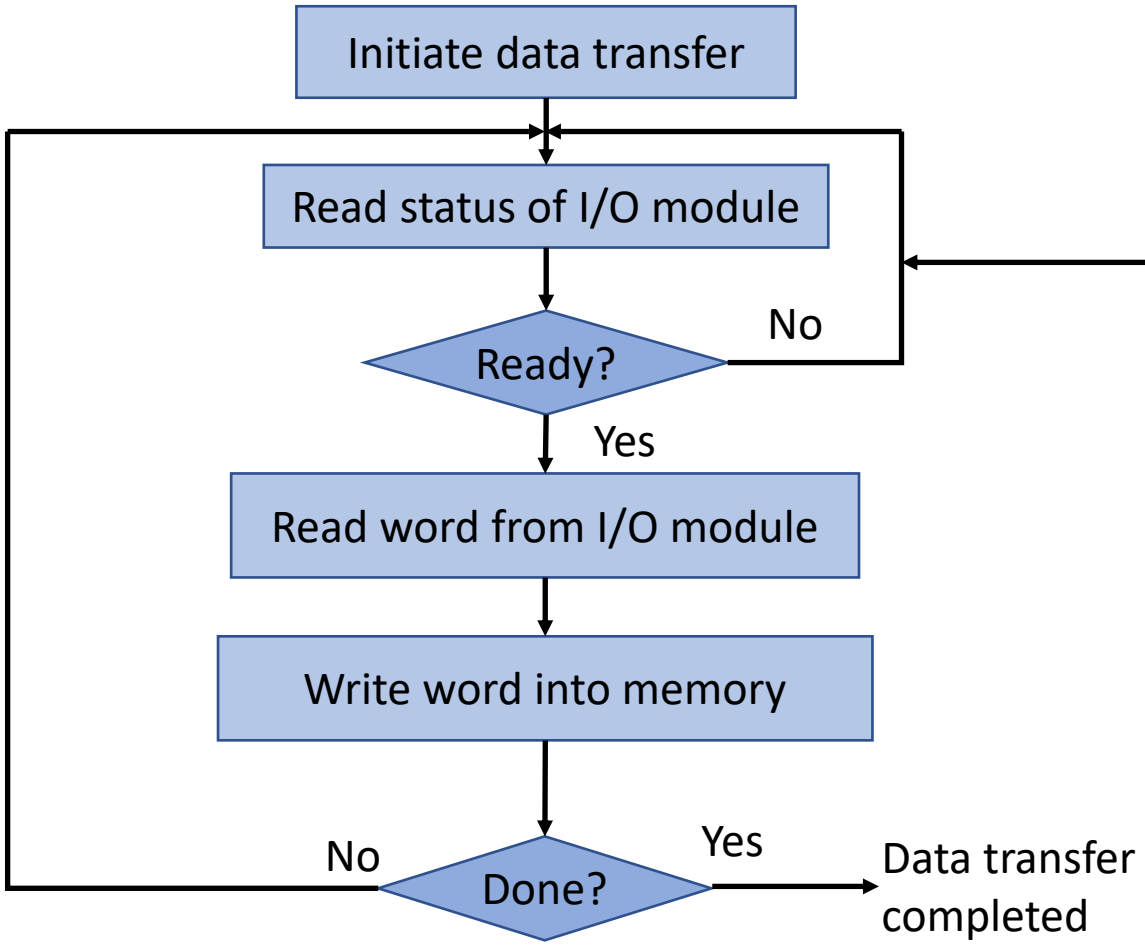


- Error may occur if the input device and the processor gey out of synchronization.
- Large number of words cannot be transferred in one go.
- Speed of data transfer depends not only on the speed of I/O device and memory, but also on the execution time of the code.

(a) Asynchronous Data Transfer

- The CPU does not know when the I/O module will be ready to transfer the next word.
- CPU has to check the status of the I/O module to know when the device is ready to transfer the next word.
  - Called handshaking.
- Characteristics:
  - While the CPU is checking whether the I/O module is ready, it cannot do anything else.
  - Wasteful of CPU time for slow devices like keyboard or mouse.

(b) Asynchronous Data Transfer

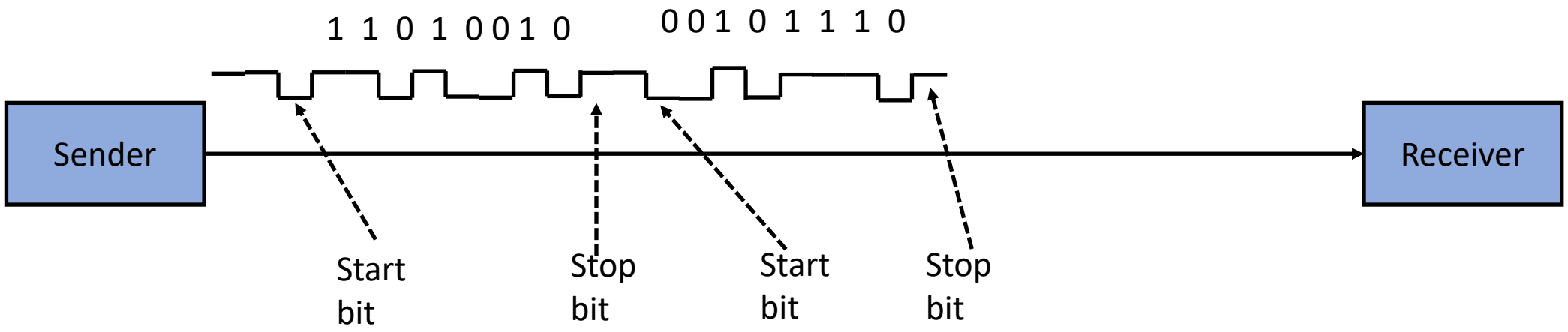


Lot of CPU time is wasted in this process

- Can not be used for high-speed devices.
- Speed of data transfer depends not only on the speed of I/O device and memory but also on the execution time of the code

(b) Asynchronous Data Transfer

- An example of asynchronous data transfer:
  - Serial data transfer between two device using start and stop bits.
  - The devices are asynchronous at the level of bytes, but are synchronous at the level of bits within the bytes.



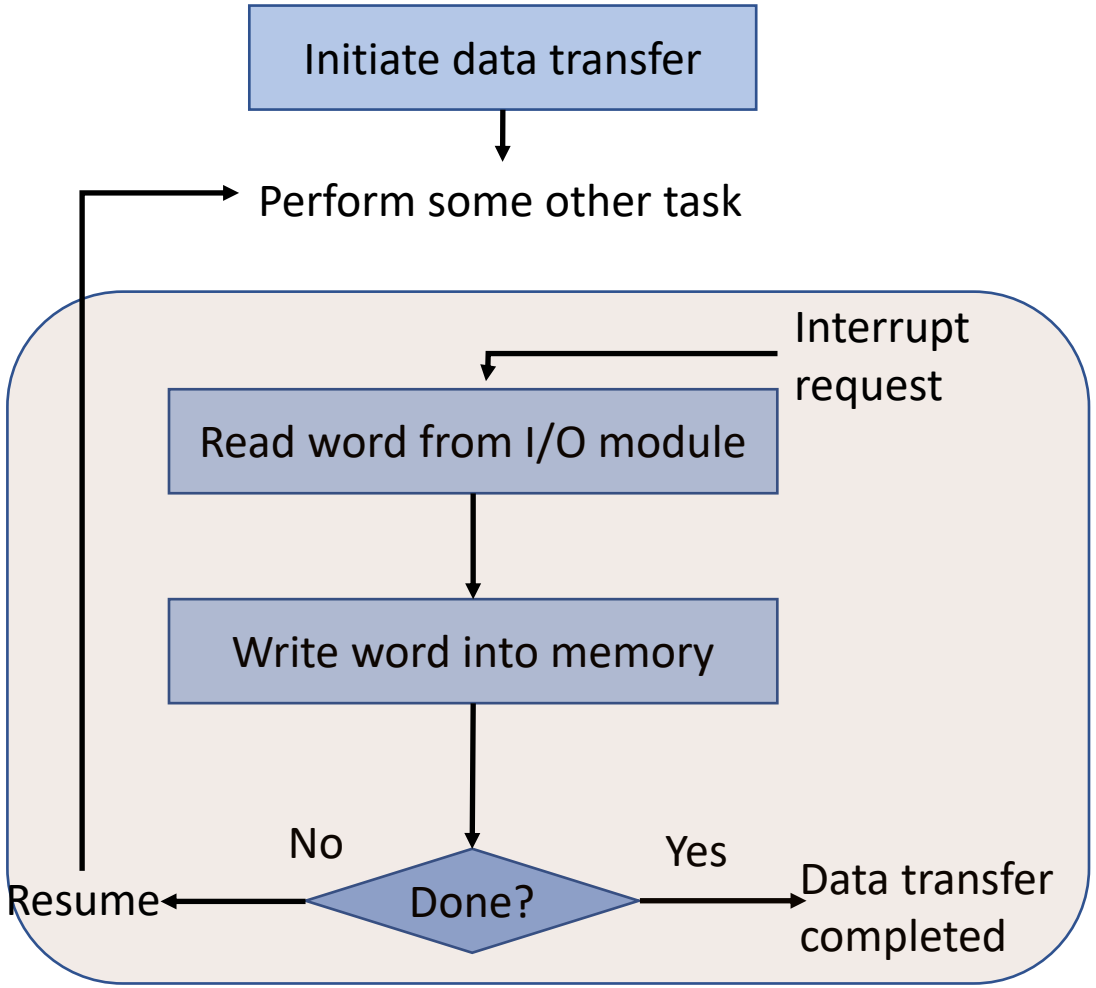
(b) Asynchronous Data Transfer

- Just like asynchronous data transfer, receiver waits for the next START bit, which indicates the beginning of a new byte transfer.
- After the START bit is received , receiver gives (known) bit delays and reads out the 8 bits of the bytes
- The STOP bits between the bytes serves to synchronize the data transmission.

(c) Interrupt -Driven Data Transfer

- The CPU initiates the data transfer and proceeds to perform *some other task*.
- When the I/O module is ready for data transfer, it informs the CPU by activating a signal (called *interrupt request).*
- The CPU suspends the task it was doing, services the request (that is, carries out the data transfer), and returns back to the task it was doing.
- Characteristics:
  - CPU time is nor wasted while checking the status of the I/O module.
  - CPU time is required only during data transfer, plus some overheads for transferring and returning control.

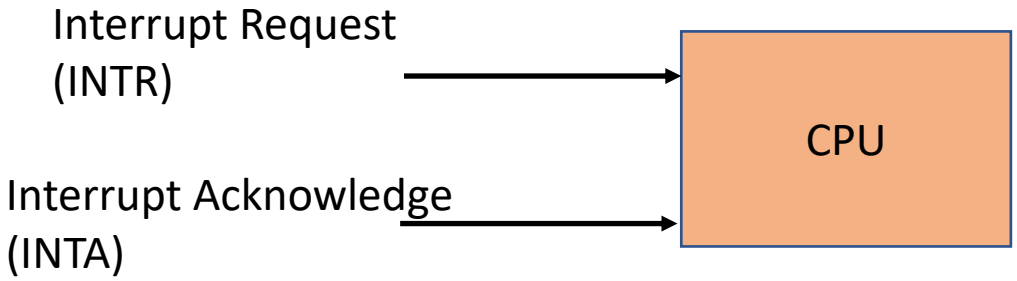
(c) Interrupt -Driven Data Transfer



- This part of the program that gets activated when interrupt request come is called ***interrupt handler*** or ***interrupt service routine*** (ISR)

Some Features of Interrupt -Driven Data Transfer

- How is ISS different from a normal subroutine or functions?
  - A function is called from well-defined places in the calling program.
    - Only the relevant registers need to be saved on entry to the function, and restored before return.
  - The ISS can get invoked from anywhere in the program that was executing.
    - Depending on when the interrupt request signal arrived.
    - So potentially all the registers that are used in the ISS needs to be saved and restored.



Some Challenges in Interrupt

- For multiple sources of interrupts, how the address of the ISR?
- How to handle multiples interrupts?
  - While an interrupt request is being processed , another interrupt request might come.
  - Enabling, disabling and masking the interrupts.
- How to handle simultaneously arriving interrupts.
- Sources of interrupts other than I/O.

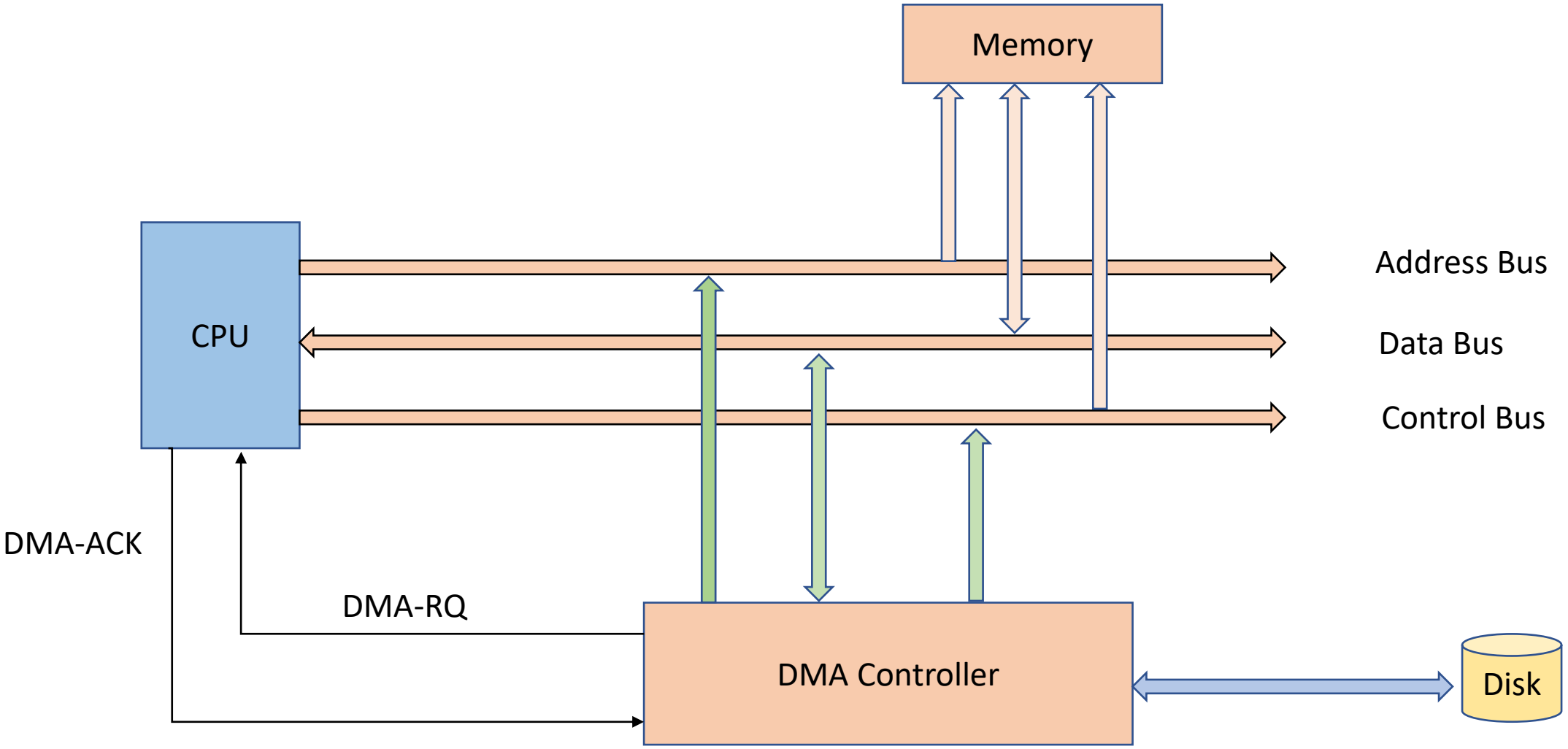
Direct Memory Access

- In the data transfer methods under programmed I/O, it is assumed that machine instructions are used to transfer the data between I/O device and memory.
  - Not very suitable when large blocks of data are required to be transferred at high speed (e.g. transfer from disk block).
- An alternate approach ***Direct Memory Access (DMA).***
  - Allows transfer of block of data directly between an I/O devices and memory, without continuous CPU intervention.

- Why programmed I/O is not suitable for high-speed data transfer?
  a) Several program instructions have to be executed for each data word transferred between I/O device and memory.
     - Suppose 20 instructions are required for each word transfer.
     - The CPI of each machine running at 1GHZ clock 1.
     - So, 20 nsec is required for each word transfer → maximum 50 M words/sec.
     - Data transfer rates of fast disks are higher than this figure.
  b) Many high speed peripheral devices like disk have a synchronous mode of operation, where data are transferred at fixed rate.
     - Consider a disk rotating ar 7200 rpm, with average rotatory delay of 4.15msec.
     - Suppose there are 64Kbytes of data recorded in every track.
     - Once the disk head is reaches the desired track, there will be a sustained data transfer at rate 64Kbytes/ 4.15 msec= 15.4 MBps.
     - This sustained data transfer rate is compatible to the memory bandwidth, and cannot be handled by programmed I/O

DMA Controller

- A hardware controller called the DMA controller can enable direct data transfer between I/O devices (e.g. disk) and memory without CPU intervention.
  - No need to execute instructions to carry out data transfer.
  - Maximum data transfer speed will be determined by the rate with which memory read and write operations can be carried out.
  - Much faster than programmed I/O.


Memory
CPU
Address Bus
Data Bus
Control Bus
DMA-ACK
DMA-RQ
DMA Controller
Disk

Step involved

a) When the CPU wants to transfer data, it initialize the DMA controller.
   - How many bytes to transfer, address in memory for transfer.

b) When the I/O device is ready for data transfer, the DMA controller sends DMA-RQ signal to the CPU.

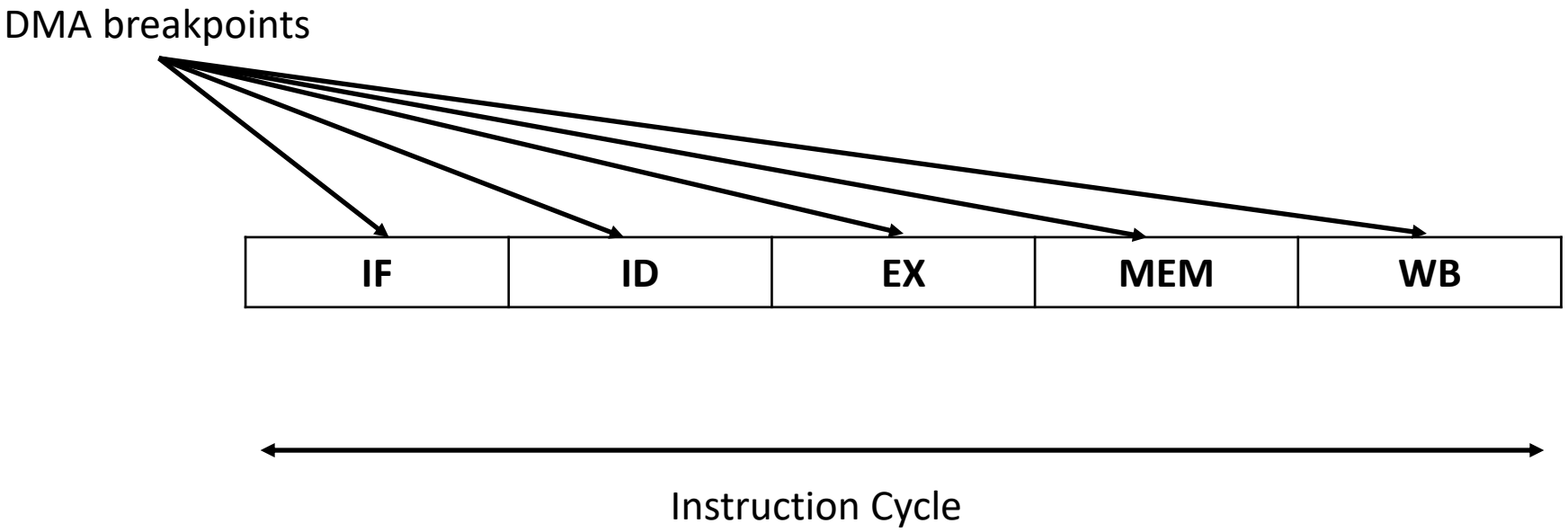
c) CPU waits till the next DMA breakpoint, relinquishes the control of the bus (i.e. puts them in high impedance state), and send DMA-ACK to DMA controller.

d) Now DMA controller enables its bus interface, and transfer data directly to or from memory.

e) When done, it deactivate the DMA-RQ signal.

f) The CPU again begins the buss to access memory.

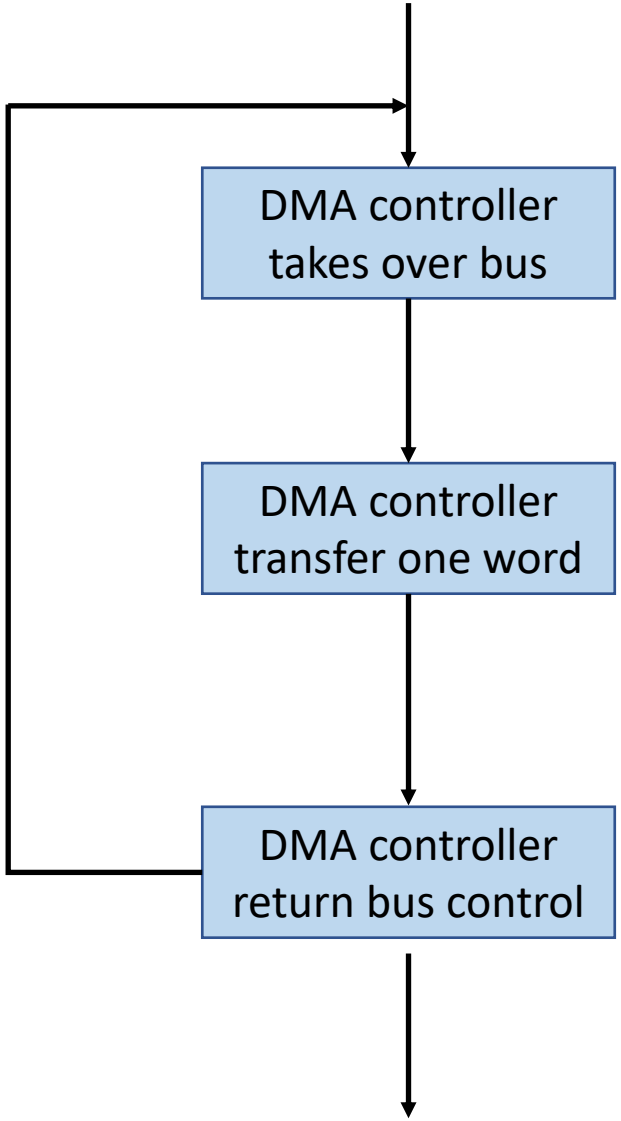
- The DMA breakpoints
  - DMA request can be acknowledged at the end of any machine cycle.



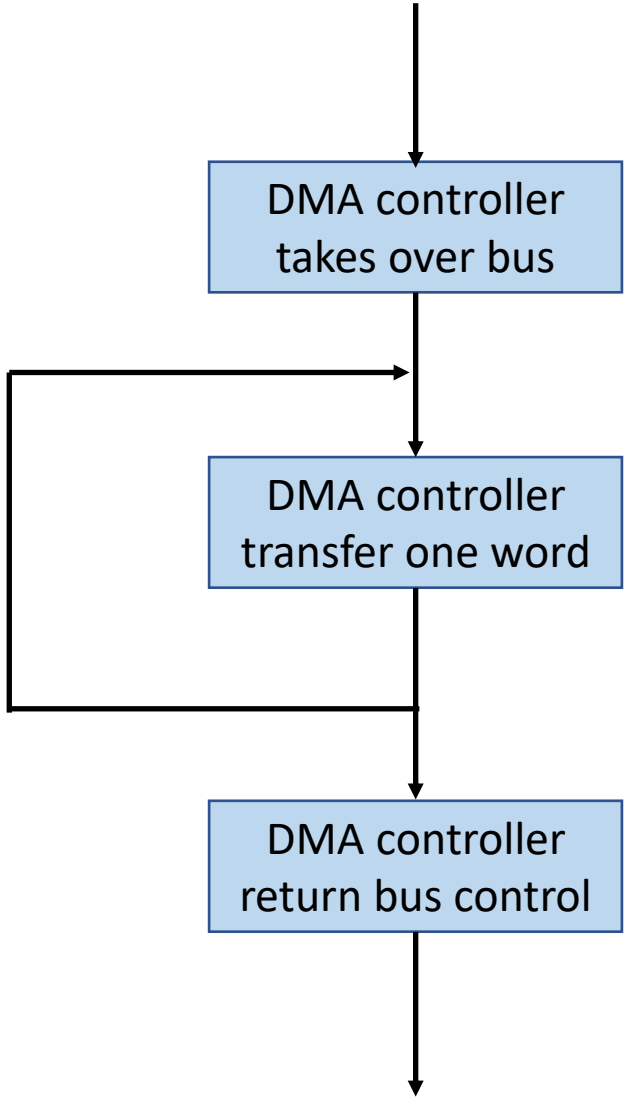
- For every DMA channel, the DMA controller will have three registers:
  a) Memory address
  b) Word count
  c) Address of data disk
- CPU initialize these register before each DMA transfer operation.
- Before the data transfer, DMA controller requests the memory bus from the CPU.
- When the data transfer is complete, the DMA controller send an interrupt signal to the CPU.

DMA Transfer Modes

- DMA transfer can take place in two modes:
  a) DMA cycling stealing
     - The DMA controller requests for a few cycles 1 or 2.
     - Preferably when the CPU is not using memory
     - DMA controller is said to steal cycles from the CPU without the CPU knowing it.
  b) DMA block transfer
     - The DMA controller transfers the whole block of data without interruption.
     - Results is maximum possible data transfer rate.
     - CPU will lie during this perios as it cannot fetch any instructions from memory.



Block Transfer mode



Cycle Stealing mode