Computer Architecture and Operating System

Input-Output Systems

Prof. Indranil Sengupta

Department of Computer Science and Engineering

IIT Kharagpur

Introduction

- Interfacing input/output devices is more complex as compared to interfacing memory systems.
- Why?
  - Wide 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.

Magnetic Disk (Hard Disk)

- Magnetic disks constitute a traditional method for non-volatile storage of information using magnetic technology.
- Broadly three types of devices appeared:
  a) Floppy disk : made of bendable plastic
  b) Magnetic drum : made of solid metal
  c) Hard disk : made of metal or glass
- All of these rely on a rotating platter (metal or glass or plastic) coated with a thin magnetic material, and use a moveable read/write head to read and write data from / to the disk.
  - Data stored as tiny magnets.

Magnetic drum
(62.5 Kbytes)

8” floppy disk
(360 Kbytes)

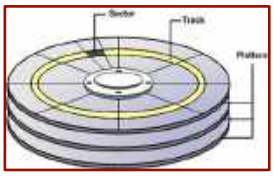
3.5” floppy disk
(1.2 Mbytes)

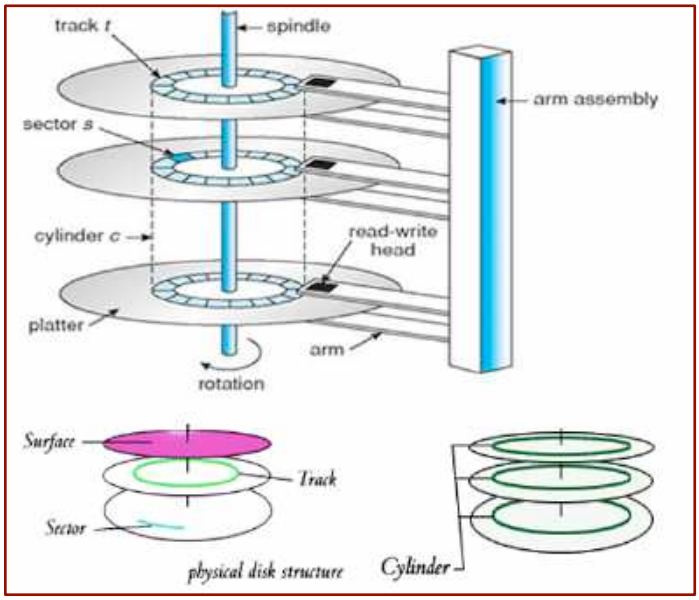
3.5” magnetic disk
(1 Tbytes)

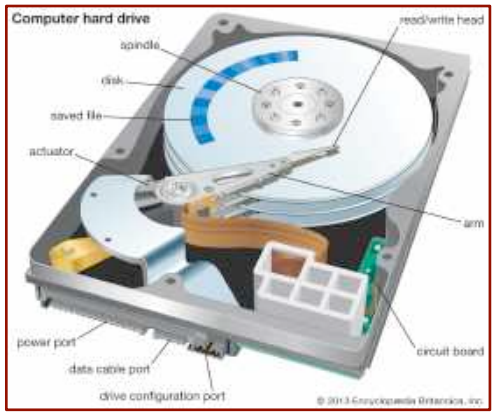
1.8” solid-state disk
(512 Gbytes)

Organization of Data on a Hard Disk

- The hard disks consists of a collection of *platters* (typically, 1 to 5), which are connected together and can spin in unison.
  - Each platter has two recording surfaces, and comes in various sizes (1 – 8 inches).
  - The stack of platter typically rotates at a speed of 5,400 to 10,000 rpm.
  - Each disk surface is divided into concentric circles called *tracks*.
    - The number of tracks per surface can vary from 1000 to 5000.
  - Each track is divided into a number of *sectors* (64 – 200 sectors/track).
    - Typical sector size: 512 – 2048 bytes.
    - Sector is the smallest unit that can be read or written.
  - The disk heads for all the surfaces are connected and move together.
  - All the tracks under the heads at a given time on all surfaces is called a *cylinder*.







Disk Access Time

- There are three components to the access time in hard disk:
  a) **Seek time**:
     - The time required to move the head to the desired track.
     - Average seek times are in the range 8 – 20 msec.
     - Actual average can be 25 – 30% less than this number, since accesses to disks are often localized.

b) **Rotational delay**:
   - Once the head is on the correct track, we must wait for the desired sector to rotate under the head.
   - The average delay or latency is the time for half the rotation.
   - Examples:
     - For 3600 rpm, average rotational delay = 0.5 rotation / 3600 rpm = 8.30 msec
     - For 5400 rpm, average rotational delay = 0.5 rotation / 5400 rpm = 5.53 msec
     - For 7200 rpm, average rotational delay = 0.5 rotation / 7200 rpm = 4.15 msec

c) Transfer time:

- The total time to transfer a block of data (typically, a sector).
- Transfer rates are typically 15 MB/sec or more.
- Transfer time depends on:
  - Sector size
  - Rotation speed of the disk
  - Recording density on the tracks

Example

Consider a disk with sector size 512 bytes, 2000 tracks per surface, 64 sectors per track, three double-sided platters, and average seek time of 10 msec.

a) What is the capacity of the disk?

b) If the disk platters rotate at 7200 rpm, and one track of data can be transferred per revolution, what is the maximum data transfer rate?

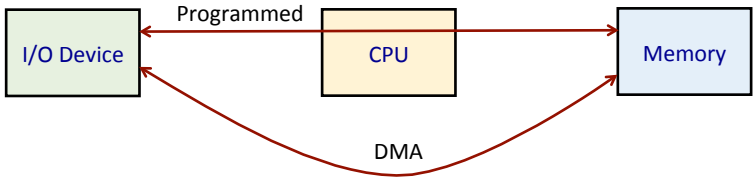
Data Transfer Techniques

Data Transfer Techniques

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

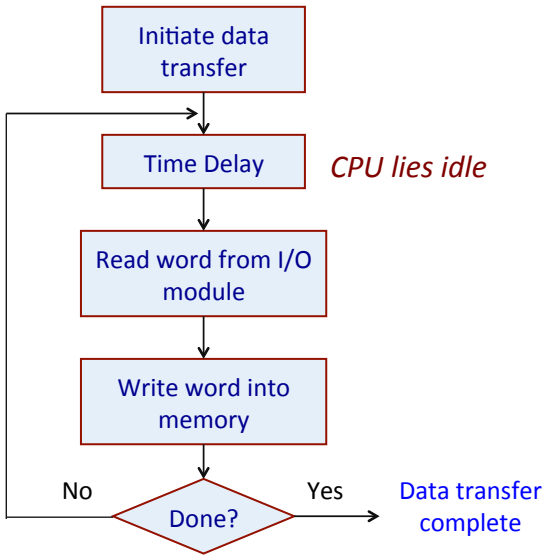
Generally slower as the execution of a program is involved.

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



(a) Synchronous Data Transfer

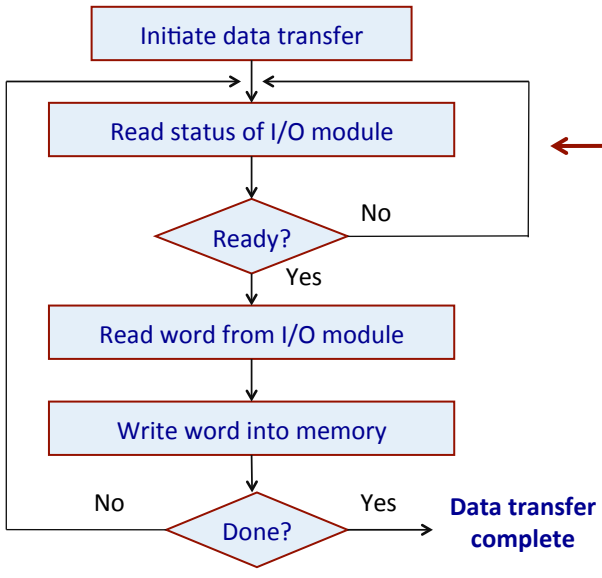
- The I/O device transfers data at a *fixed rate* that is known to the CPU.
- The CPU initiates the I/O operation and transfers successive bytes/words after giving fixed time delays.
- **Characteristics**:
  - During the time delay, CPU lies idle.
  - Not many I/O devices have strictly synchronous data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



- Error may occur if the input device and the processor get 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.

(b) 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.

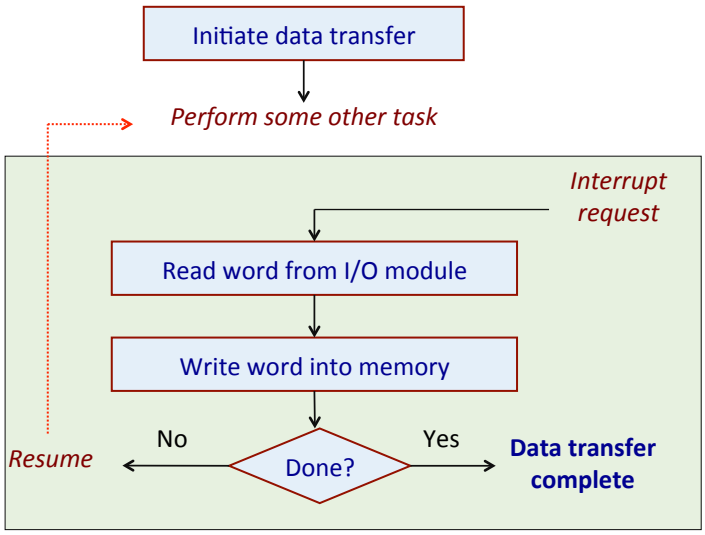


Lot of CPU time is wasted in this loop

- Cannot 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.

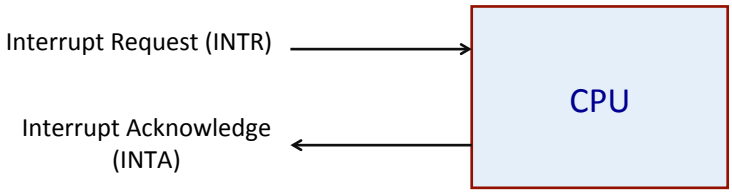
(c) Interrupt-Driven Data Transfer

- 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 not 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.



This part of the program that gets activated when interrupt request comes is called interrupt handler (IH) or interrupt service subroutine (ISS).

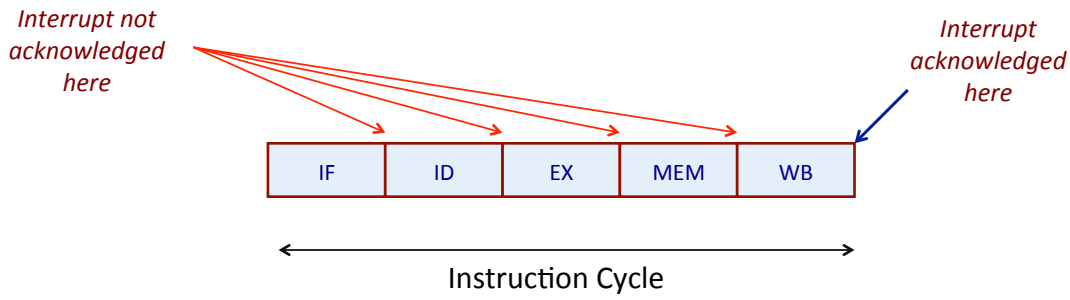
- Typical interfaces for interrupt-driven data transfer:
  a) Interrupt request (INTR) – External device raises this signal when it wants to draw attention of the CPU.
  b) Interrupt acknowledge (INTA) – CPU sends this signal back to the external device to indicate that it has received the INTR signal.



What happens when an interrupt request arrives?

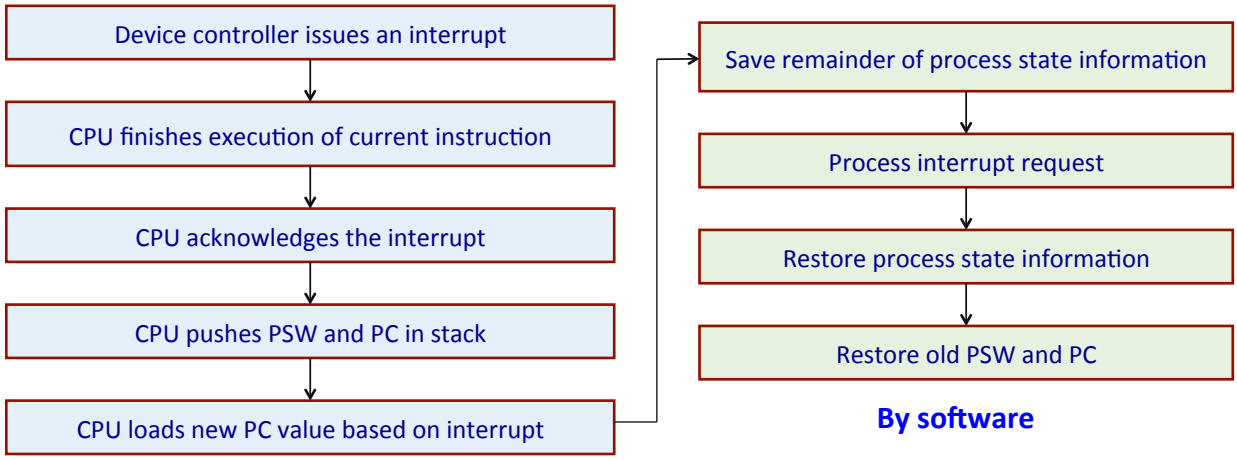
- At the end of the current instruction execution, the PC and *program status word* (PSW) are saved in stack automatically.
  - PSW contains status flags and other processor status information.
- The interrupt is acknowledged, the *interrupt vector* obtained, based on which control transfers to the appropriate ISS.
  - Different interrupting devices may have different ISS's.
- After handling the interrupt, the ISR executes a special *Return From Interrupt* (RTI) instruction.
  - Restores the PSW and returns control to the saved PC address.
  - Unlike normal RETURN where PSW is not restored.

- An instruction cycle typically consists of several machine cycles.

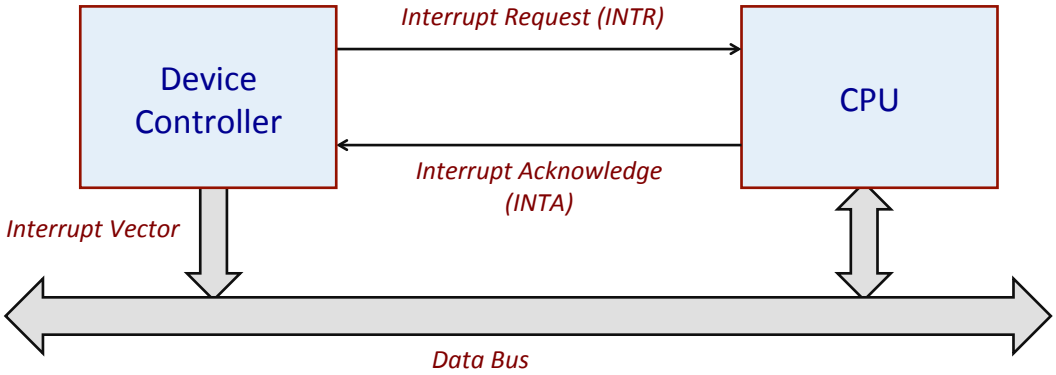


General Interrupt Processing

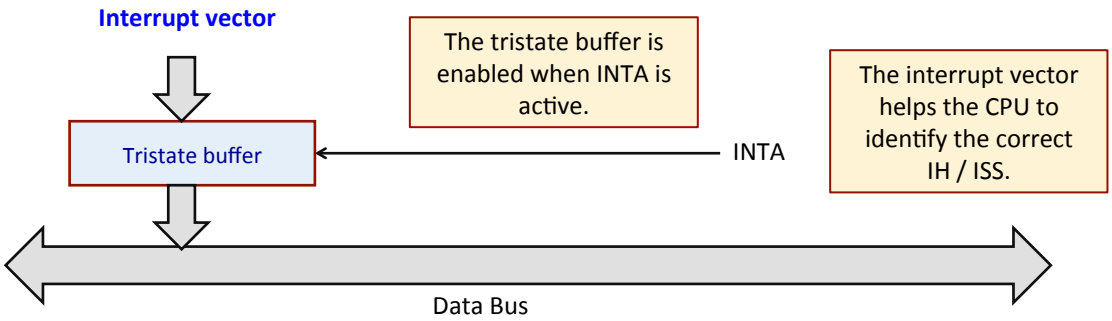
By hardware



By software

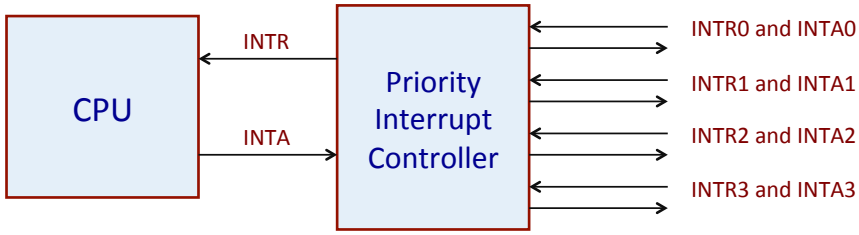


- How is the interrupt vector sent on the data bus in response to INTA?
  a) Device controller sends *INTR* to the CPU.
  b) CPU finishes the current instruction and sends back *INTA*.
  c) Device controller sends *interrupt vector* (or number) over data bus.
  d) CPU reads the interrupt vector, and identifies the device.



Multiple Devices Interrupting the CPU

- A common solution is to use a *priority interrupt controller*.
  - The interrupt controller interacts with CPU on one side and multiple devices on the other side.
  - For simultaneous interrupt requests, interrupt priority is defined.
  - The interrupt controller is responsible for sending the interrupt vector to CPU.



- How it works?
  - The *INTR* line is made active when one or more of the device(s) activate their interrupt request line.

    $$INTR = INTR0 + INTR1 + INTR2 + INTR3$$

  - When the CPU sends back *INTA*, the interrupt controller sends back the corresponding acknowledge to the interrupting device, and puts the interrupt vector on the data bus.
  - The interrupt controller is programmable, where the interrupt vectors for the various interrupts can be programmed (specified).
  - For more than one interrupt request simultaneously active, a priority mechanism is used (e.g. *INTR0* is highest priority, followed by *INTR1*, etc.).

Cases that make interrupt handling difficult

- For some interrupts, it is not possible to finish the execution of the current instruction.
  - A special *RETURN* instruction is required that would return and restart the interrupted instructions.
- Some examples:
  a) **Page fault interrupt**: A memory location is being accessed that is not presently available in main memory.
  b) **Arithmetic exception**: Some error has occurred during some arithmetic operation (e.g. division by zero).

Direct Memory Access (DMA)

Introduction

- In the data transfer methods discussed 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 of a disk block).
- An alternate approach is *Direct Memory Access* (DMA).
  - Allows transfer of a block of data directly between an I/O device and memory, without continuous CPU intervention.

- Why programmed I/O is not suitable for high-speed data transfer?

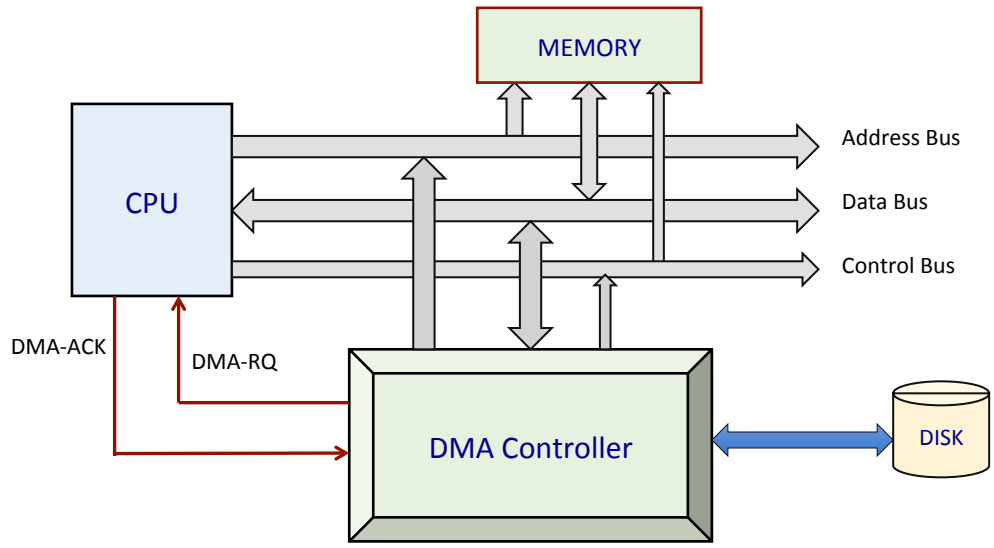
  a) Several program instructions have to be executed for each data word transferred between the I/O device and memory.
     - Suppose 20 instructions are required for each word transfer.
     - The CPI of the machine running at 1 GHz clock is 1.
     - So, 20 nsec is required for each word transfer → 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 a fixed rate.

- Consider a disk rotating at 7200 rpm, with rotational delay of 8.30 msec.
- Suppose there are 64 Kbytes of data recorded in every track.
- Once the disk head reaches the desired track, there will be a sustained data transfer at maximum rate 64 Kbytes / 8.30 msec = 7.7 MB/s.
- This sustained data transfer rate is comparable to the memory bandwidth, and cannot be handled by programmed I/O.

DMA Controller

- A hardwired controller called the *DMA controller* can enable direct data transfer between I/O device (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.



Steps Involved

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

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

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

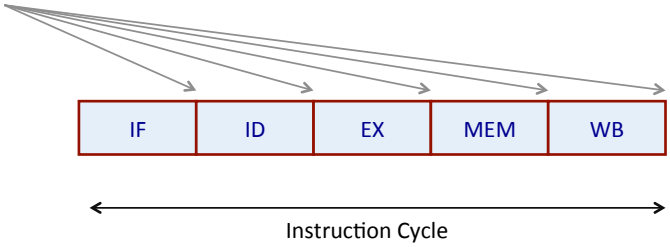
d) Now DMA controller enables its bus interface, and transfers data directly to/from memory.

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

f) The CPU again begins to use the bus to access memory.

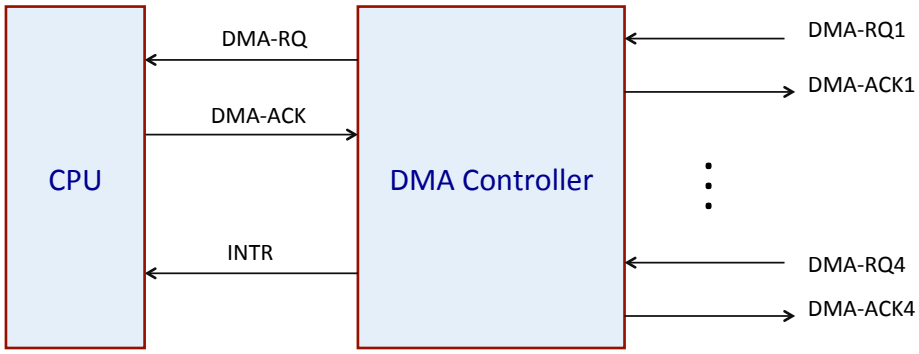
- The DMA breakpoints:
  - DMA request can be acknowledged at the end of any machine cycle.

DMA breakpoints



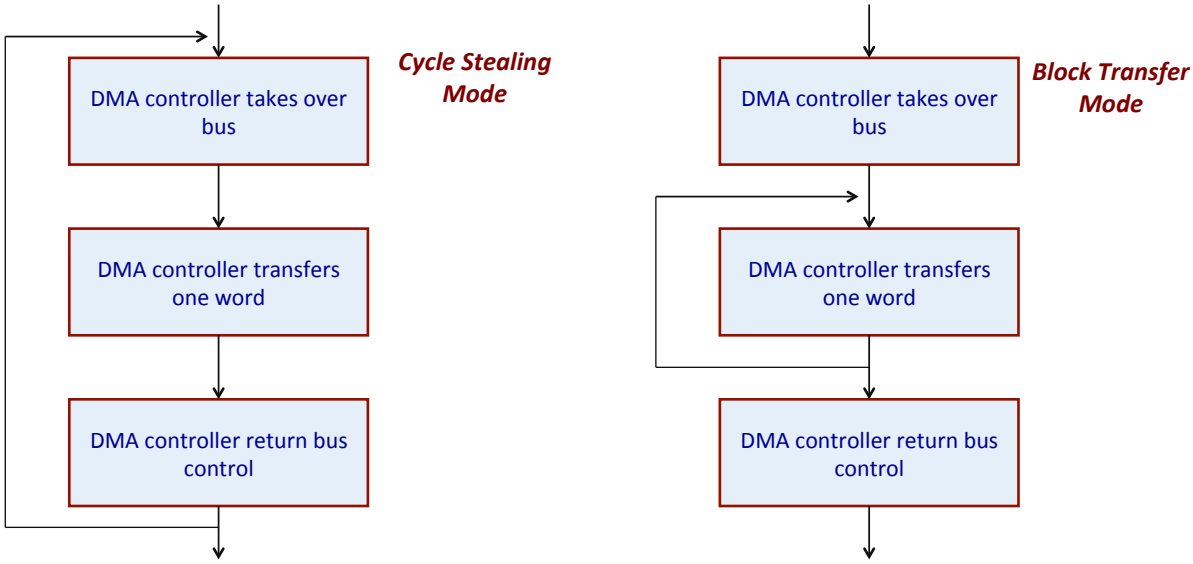
Why cannot we have interrupt breakpoints at the end of any machine cycle?

- For every DMA channel, the DMA controller will have three registers:
  a) Memory address
  b) Word count
  c) Address of data on disk
  - CPU initializes these registers 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 sends an interrupt signal to the CPU.



DMA Transfer Modes

- DMA transfer can take place in two modes:
  - **a) DMA cycle stealing**
    - The DMA controller requests for the 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 in maximum possible data transfer rate.
    - CPU will lie idle during this period as it cannot fetch any instructions from memory.



Others Applications of DMA

- Other than data transfer to/from high-speed peripheral devices, DMA can be used in some other areas as well:
  - High-speed memory-to-memory block move.
  - Refreshing dynamic memory systems, by periodically generating dummy read requests to the columns.