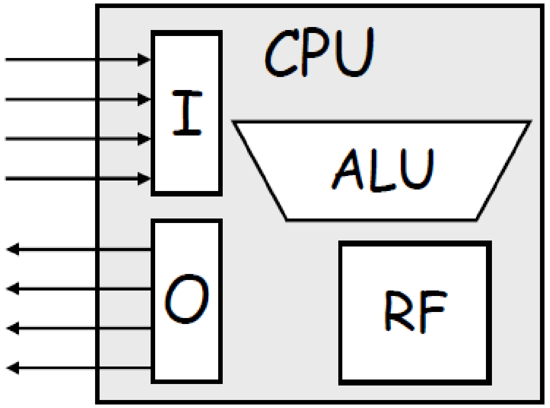
اصول سیستم‌های کامپیوتری

فصل نهم

ورودی/خروجی - وقفه



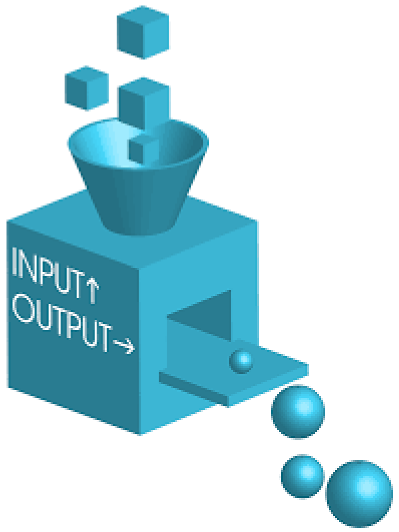
Copyright Notice

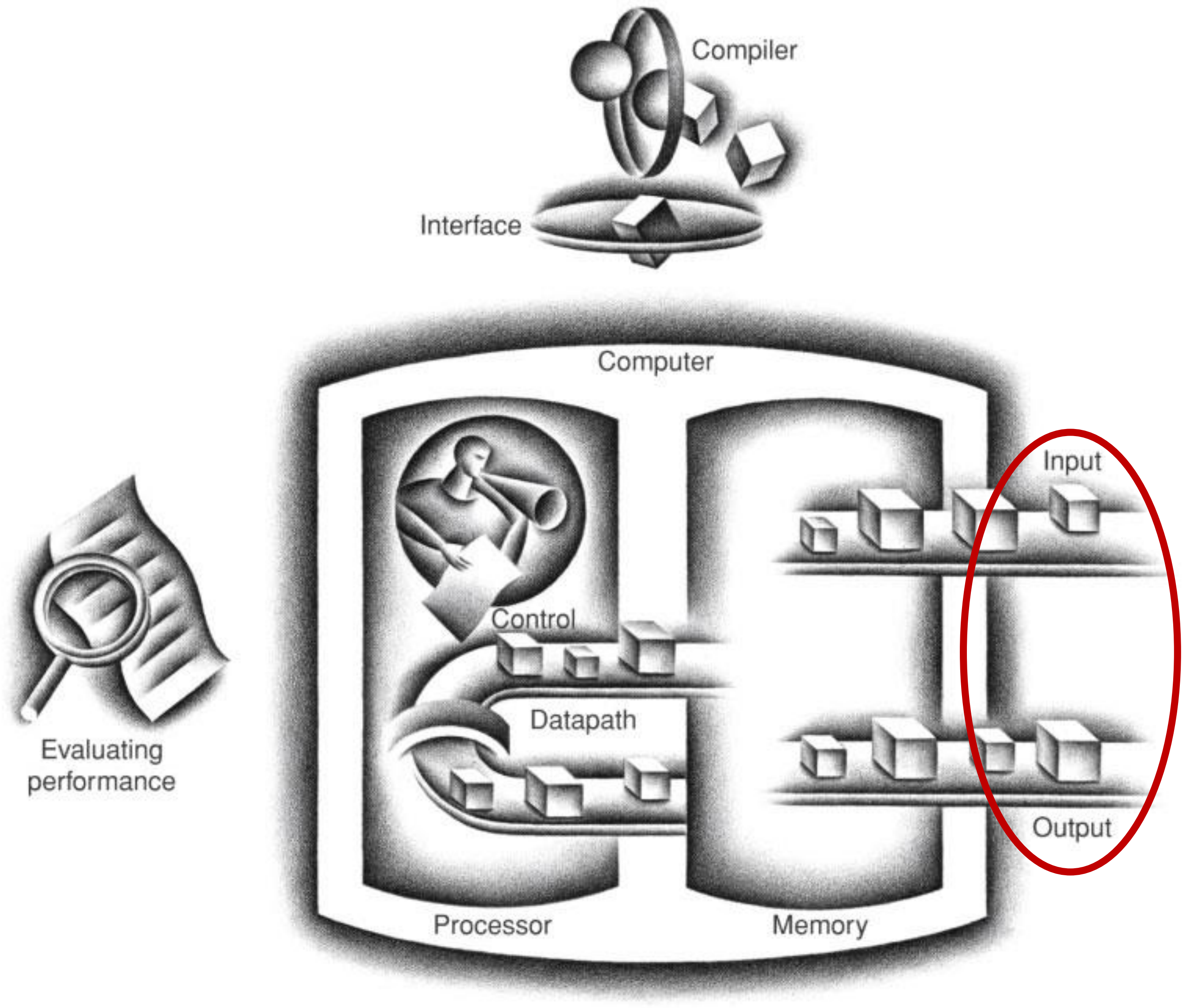
Parts (text & figures) of this lecture are adopted from :

- D. Patterson & J. Hennessey, “Computer Organization & Design, The Hardware/Software Interface”, 5th Ed., MK publishing, 2014
- M. Mano, C. Kime, T. Martin, “Logic and Computer Design Fundamentals”, 5th Ed., Pearson, 2015
- W. Stallings, “Computer Organization and Architecture, Designing for Performance”, 10th Ed., Pearson, 2016

Outlines

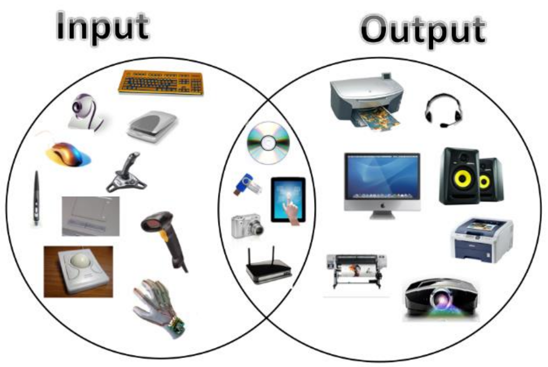
- I/O devices (Instances & Characteristics)
- I/O transactions
  - Sync & Async data transfer
  - Handshaking
- ISA perspective
  - Memory-mapped vs. isolated I/O
  - Interrupt vs. Programmed I/O
  - Direct Memory Access (DMA)




Compiler
Interface
Computer
Input
Control
Datapath
Output
Processor
Memory
Evaluating performance

I/O Devices

- Same components for all kinds of computer
  - Desktop, server, embedded
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers



I/O Devices Characteristics - 1

- Behavior:
  - Input (read only)
  - output (write only, cannot be read)
  - storage (can be reread and usually rewritten)

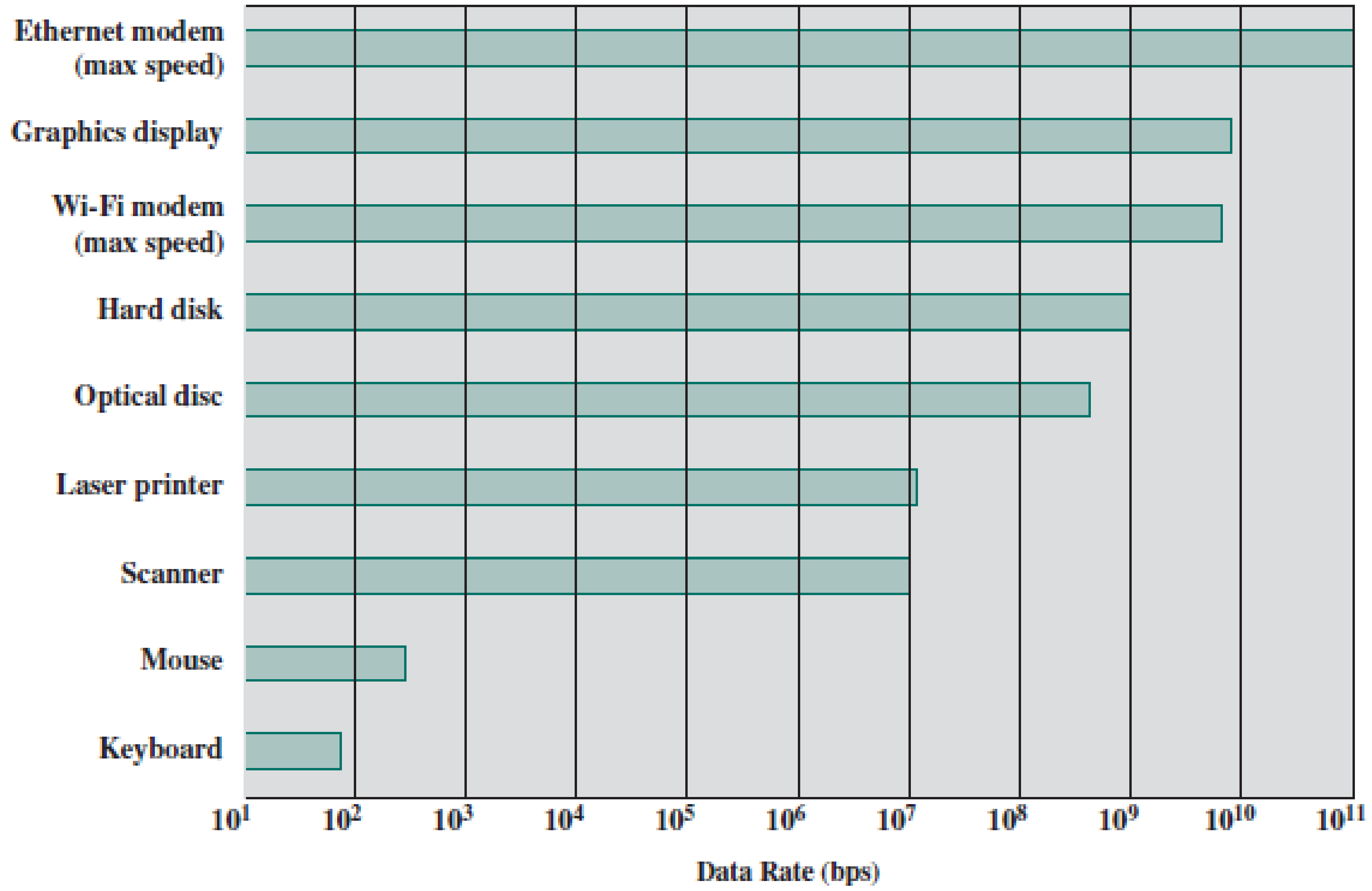
I/O Devices Characteristics - 2

- Partner:
  - A human or a machine is at the other end of the I/O device
    - feeding data on input or
    - reading data on output

I/O Devices Characteristics - 3

- Data rate:
  - The peak rate at which data can be transferred between the I/O device & memory or processor

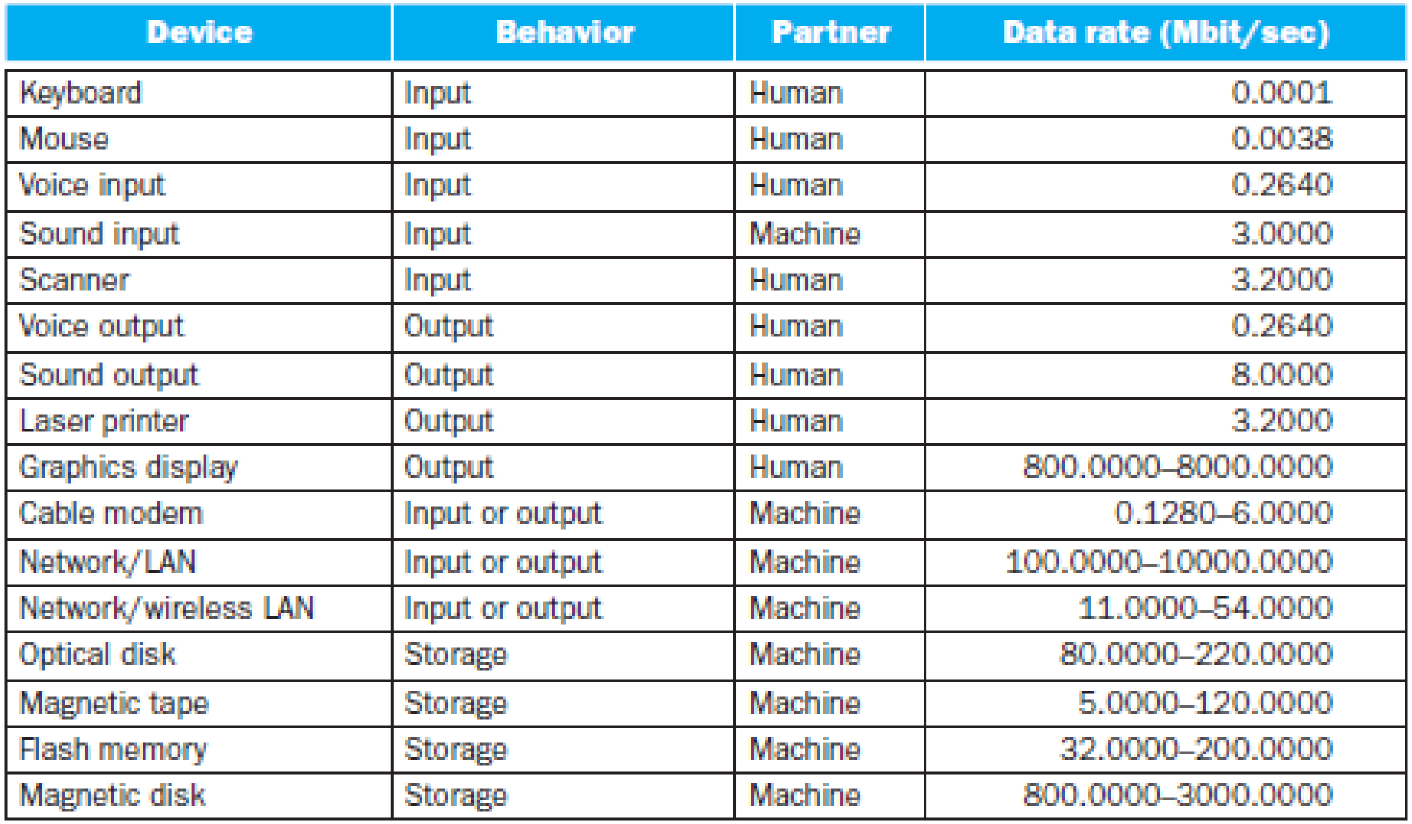
Typical I/O Device Data Rates



I/O Devices Characteristics

- Behavior (input, output, storage)
- Partner (a human or a machine)
- Data rate
- e.g. keyboard:
  - an input device
  - used by a human
  - with a peak data rate of about 10 bytes per second

Diversity of I/O Devices

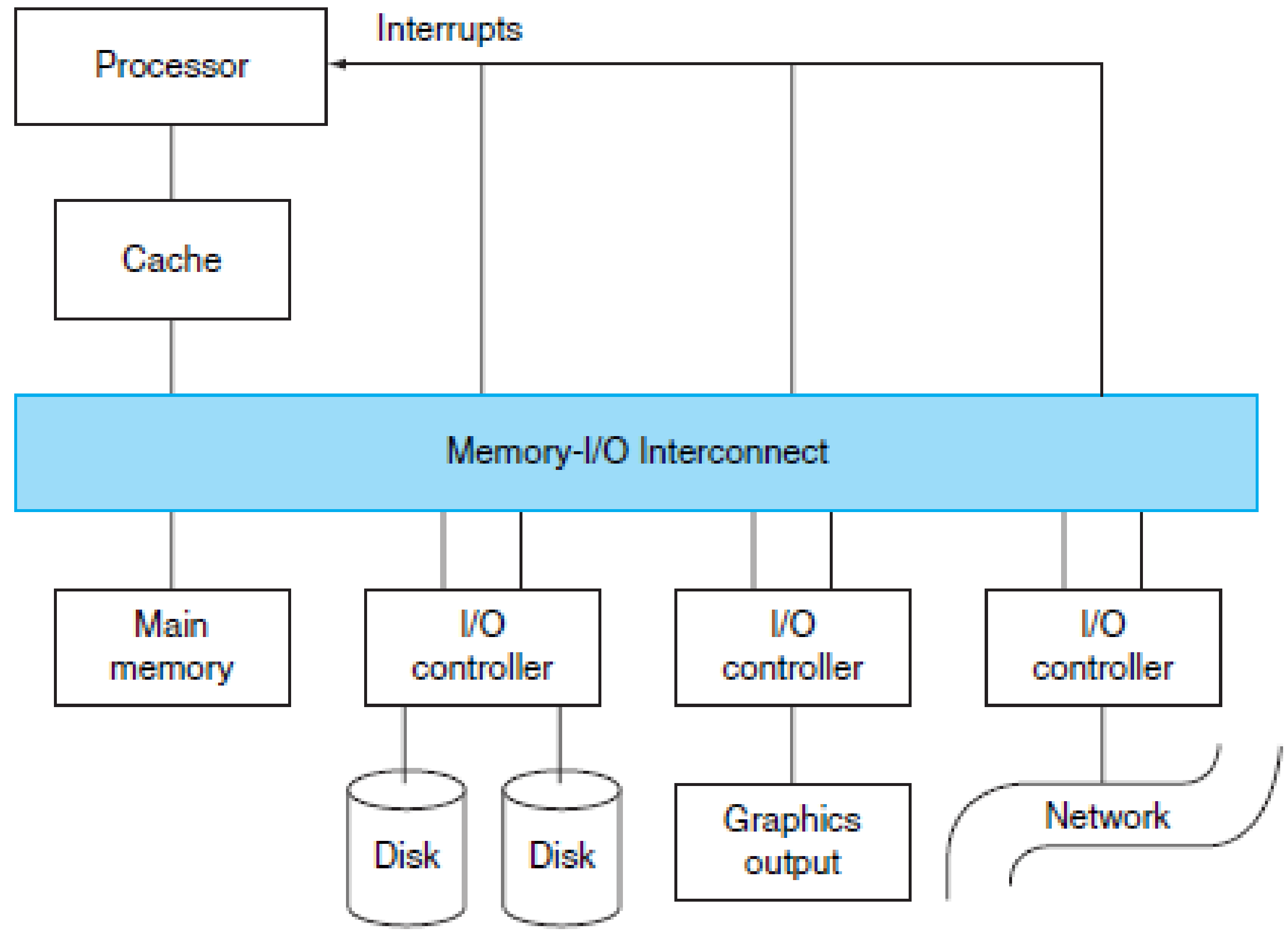


Device	Behavior	Partner	Data rate (Mbit/sec)
Keyboard	Input	Human	0.0001
Mouse	Input	Human	0.0038
Voice input	Input	Human	0.2640
Sound input	Input	Machine	3.0000
Scanner	Input	Human	3.2000
Voice output	Output	Human	0.2640
Sound output	Output	Human	8.0000
Laser printer	Output	Human	3.2000
Graphics display	Output	Human	800.0000–8000.0000
Cable modem	Input or output	Machine	0.1280–6.0000
Network/LAN	Input or output	Machine	100.0000–10000.0000
Network/wireless LAN	Input or output	Machine	11.0000–54.0000
Optical disk	Storage	Machine	80.0000–220.0000
Magnetic tape	Storage	Machine	5.0000–120.0000
Flash memory	Storage	Machine	32.0000–200.0000
Magnetic disk	Storage	Machine	800.0000–3000.0000

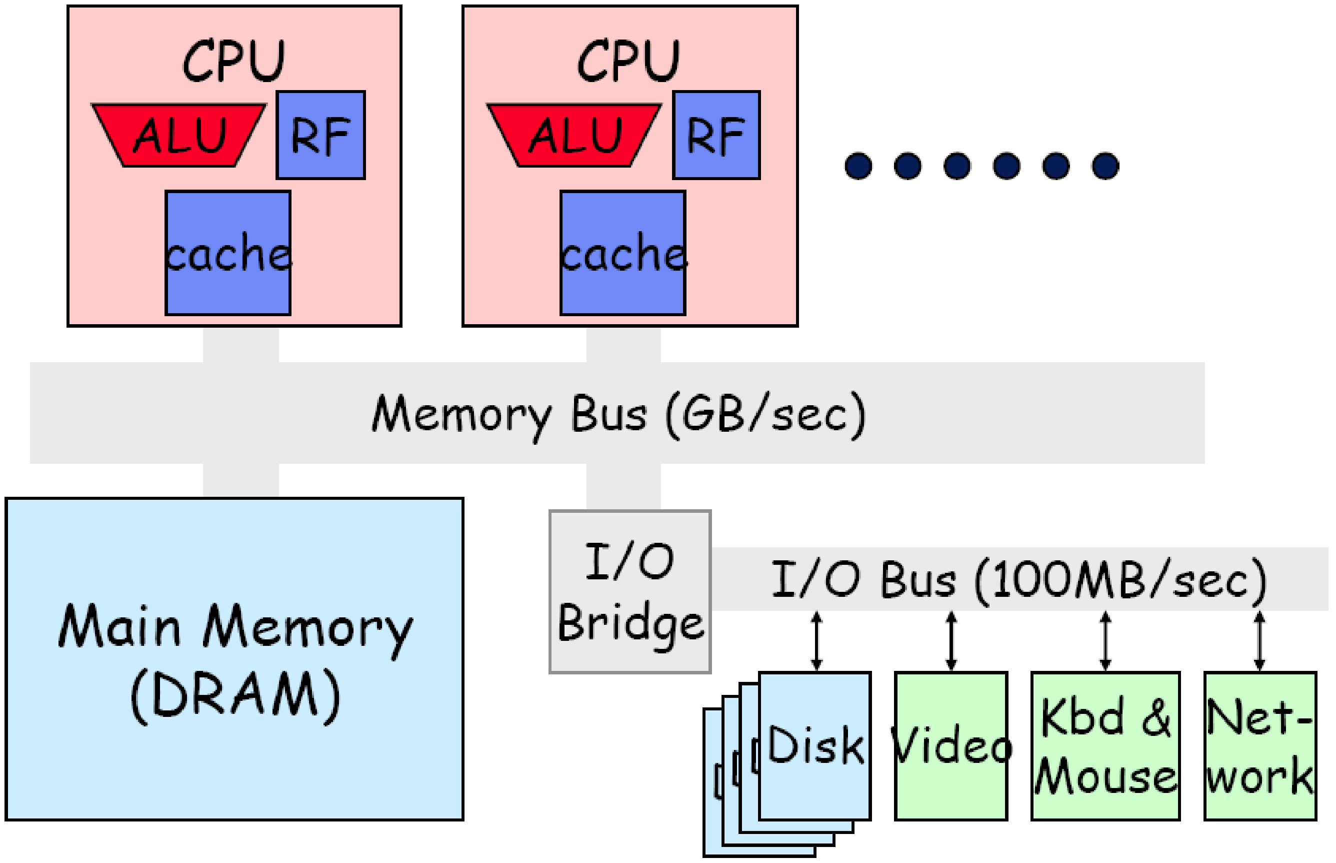
I/O Controller (I/O Bridge)

- Fact
  - Input/outputs very slow devices
    - Average response time: few milliseconds
  - CPUs very fast devices
    - Average running time: nanoseconds
- Question:
  - How are I/O devices connected to CPU?
    - I/O controller (also called I/O bridge)

Typical Collection of I/O Devices

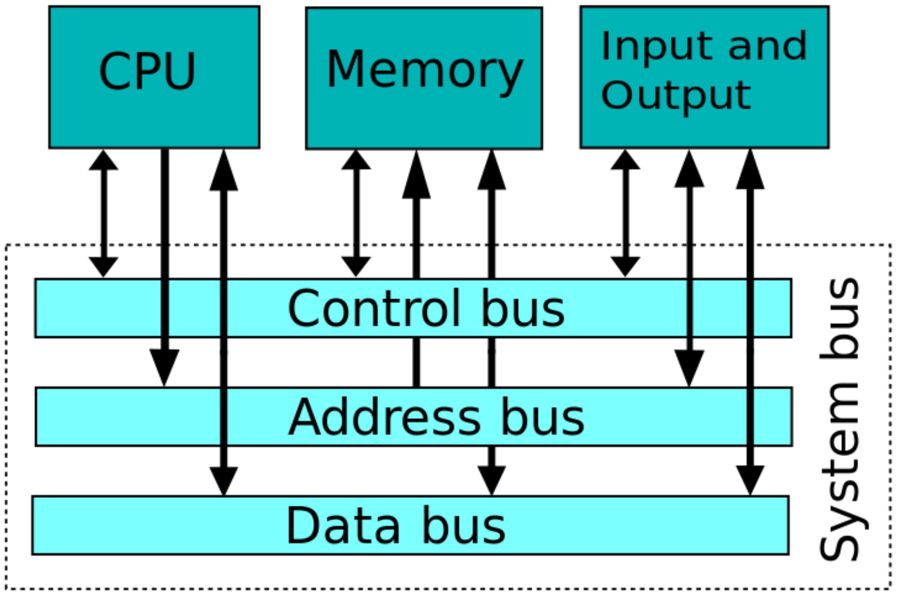


Memory-I/O Separate Bus



BUS

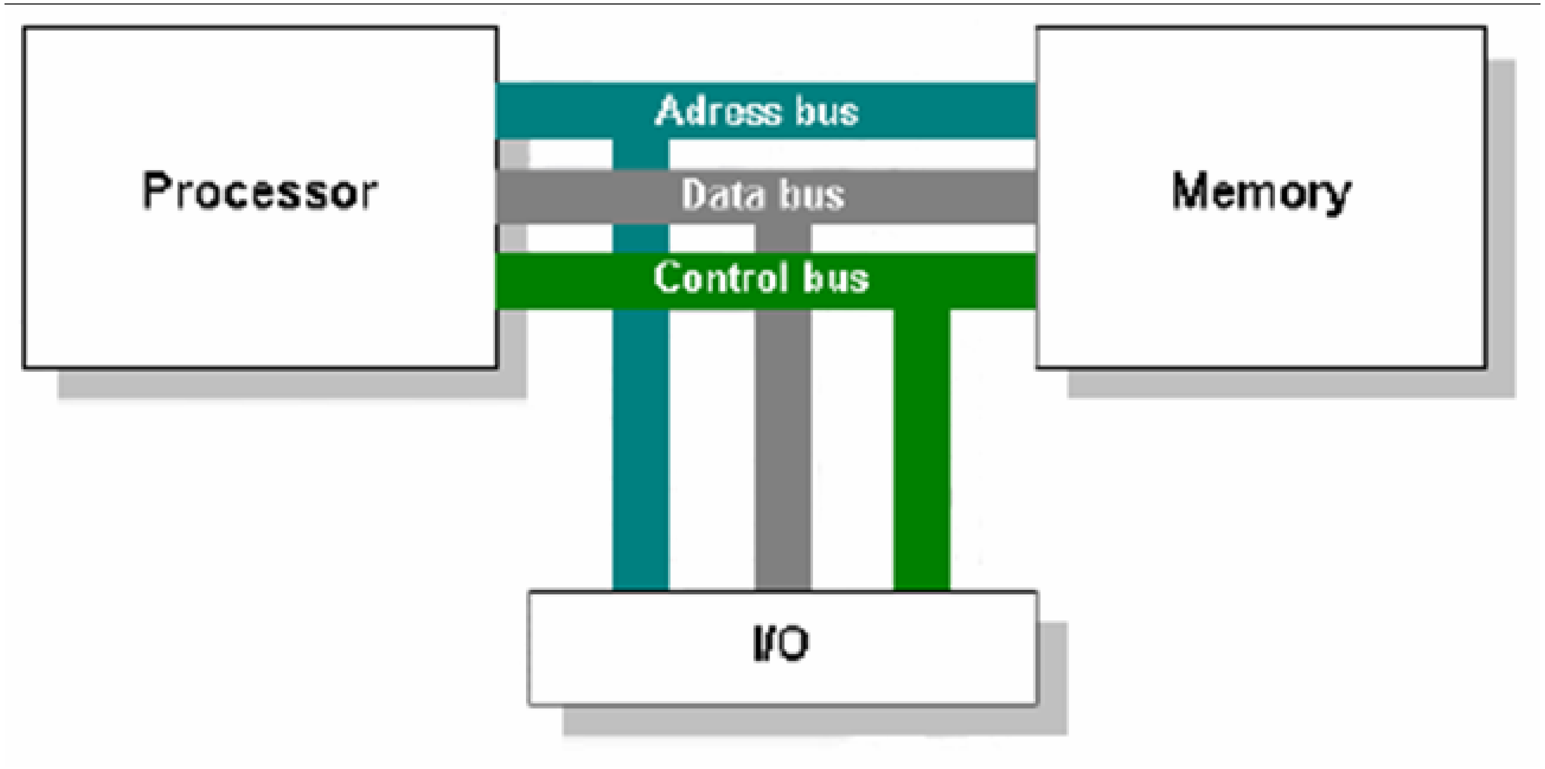
Memory, CPU & I/O devices communicate via BUS



What is a Bus?

- A communication pathway connecting two or more devices
  - Usually broadcast
  - Often grouped
  - A number of channels in one bus
- e.g. 32 bit data bus is 32 separate single bit channels
- Power lines may not be shown

Specific Buses



Data Bus

- Carries data
  - There is no difference between “data” and “instruction” at this level
- Width is a key determinant of performance
  - 8, 16, 32, 64 bit

Address Bus

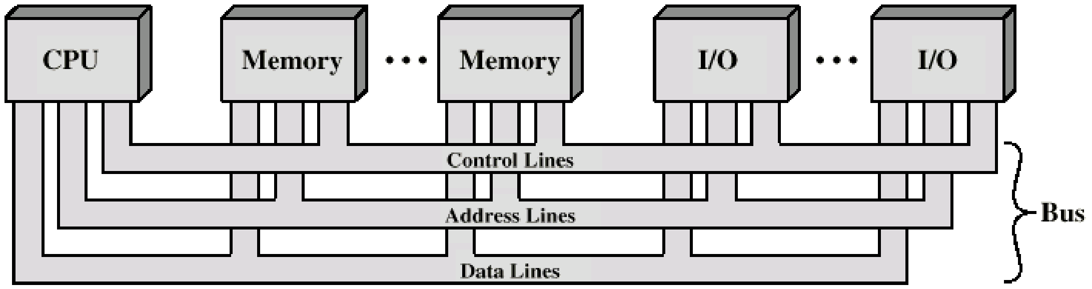
- Identify the source/ destination of data
- e.g. CPU needs to read an instruction (data) from a given location in memory
- Bus width determines maximum memory capacity of system
    - e.g. MIPS has 32 bit address bus giving 4GB address space

Control Bus

- Control and timing information
  - Memory read/write signal
  - Interrupt request
  - Clock signals

Disadvantage of Buses

- Creates a communication bottleneck
  - Bus bandwidth limits maximum I/O throughput
- Maximum bus speed is largely limited by:
  - Length of bus
  - Number of devices on bus
  - Slowest device on bus



I/O Transactions

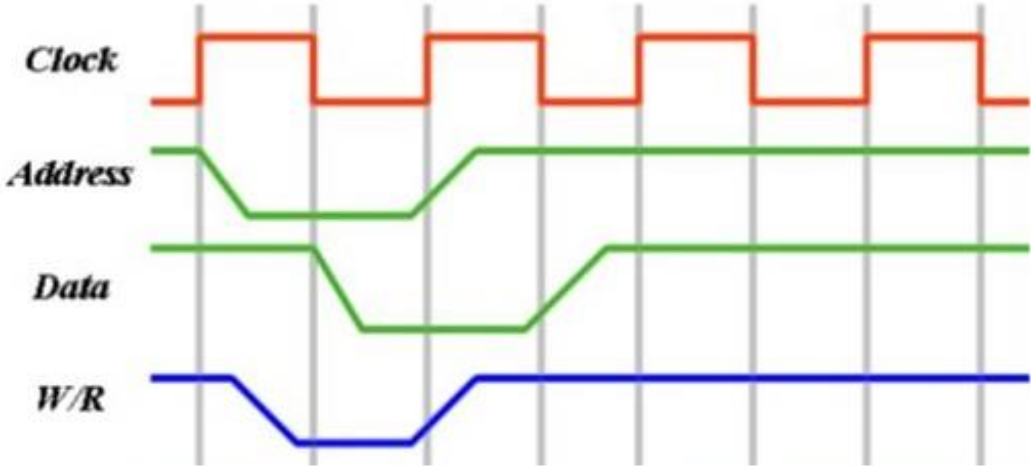
- An I/O Transaction Consists of:
  - sending address & command
  - sending or receiving data
- Types of I/O Transactions:
  - Input
    - from input devices to memory or processor
  - Output
    - from memory or processor to output devices

Sync/Async Interconnection

- Synchronous interconnection
  - includes a clock in the control lines
  - uses a fixed protocol for communicating that is relative to the clock
- Asynchronous interconnection
  - not clocked
  - uses a handshaking protocol for coordinating usage rather than a clock

Synchronous Interconnection

- A clock in the control lines
- Fixed communicating protocol relative to the clock



Sync Interconnection (pros/cons)

- ☺ can be implemented easily in a small finite-state machine
- ☺ the bus can run fast, and the interface logic will be small
- 😐 Every device on the bus must run at the same clock rate
- ☹ cannot be long if fast (due to clock skew problems)

Asynchronous Interconnection

- not clocked
- uses a handshaking protocol for coordinating usage rather than a clock
  - consists of a series of steps in which the sender and receiver proceed to the next step only when both parties agree
  - The protocol is implemented with an additional set of control lines

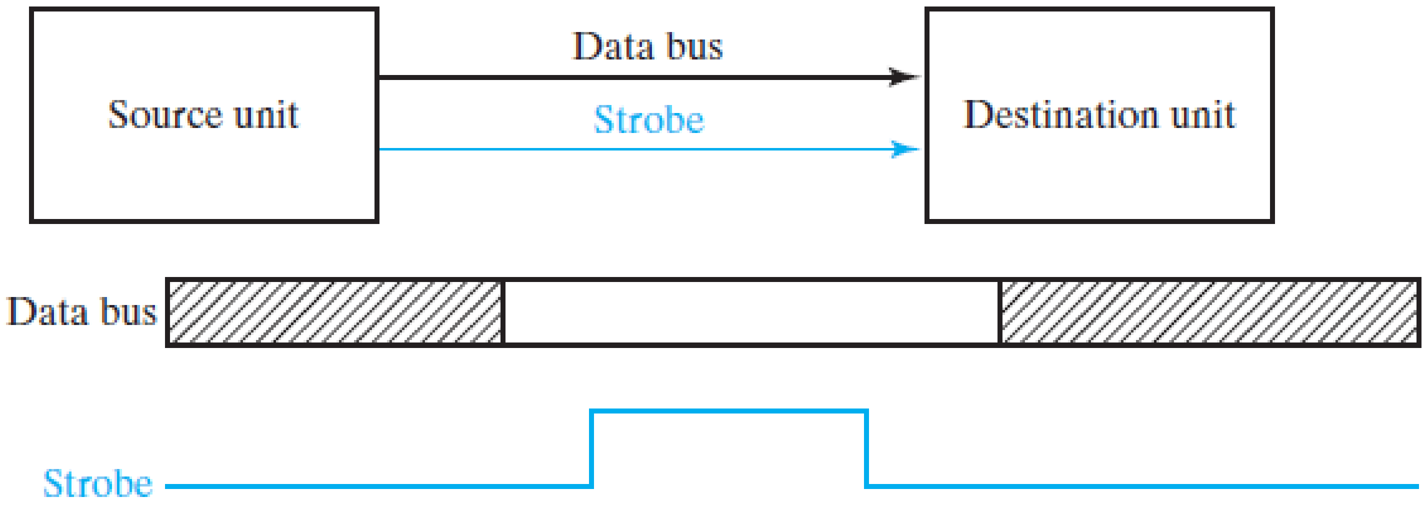
Async Interconnection (pros/cons)

- ☺ can accommodate a wide variety of devices of different speeds
- ☺ the bus can be lengthened without worrying about clock skew or synchronization problems
- ☹ ?

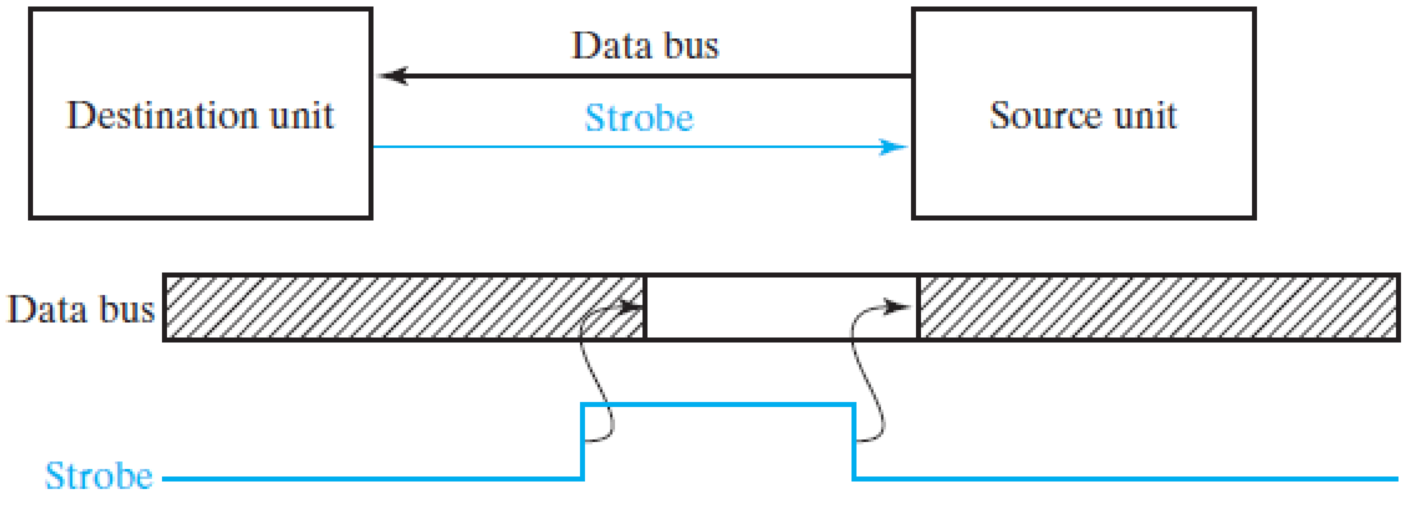
Typical Async Protocols

- Strobing
  - Source-initiated
  - Destination-initiated
- Handshaking
  - Source-initiated
  - Destination-initiated

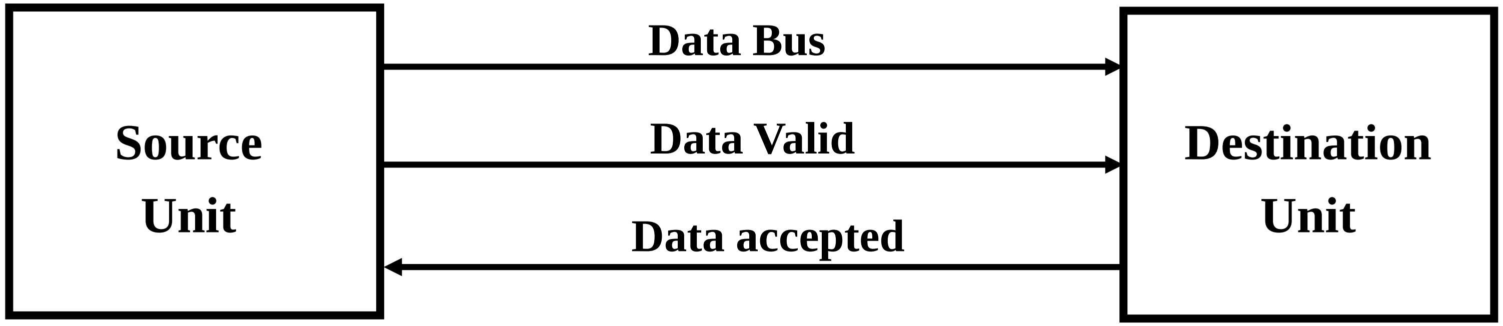
Source-Initiated Strobe

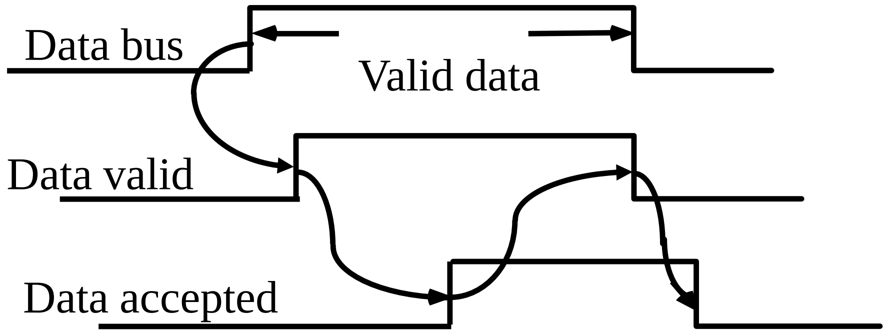


Destination-Initiated Strobe

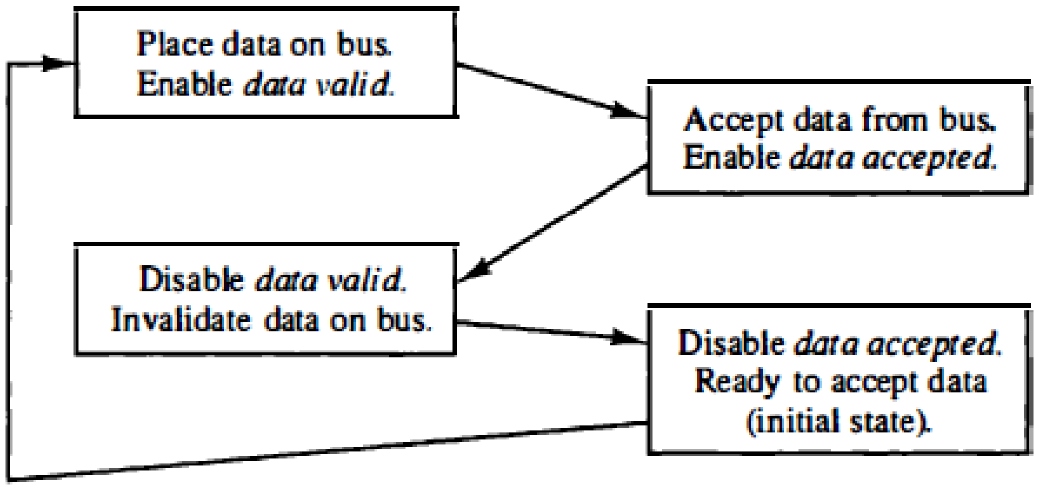


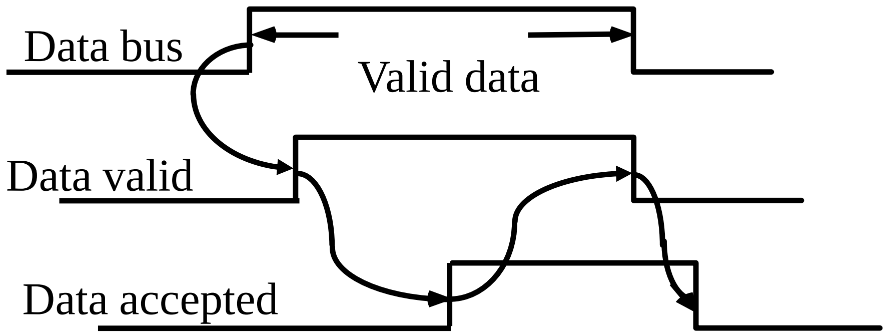
Src-Initiated Transfer via Handshake - 1



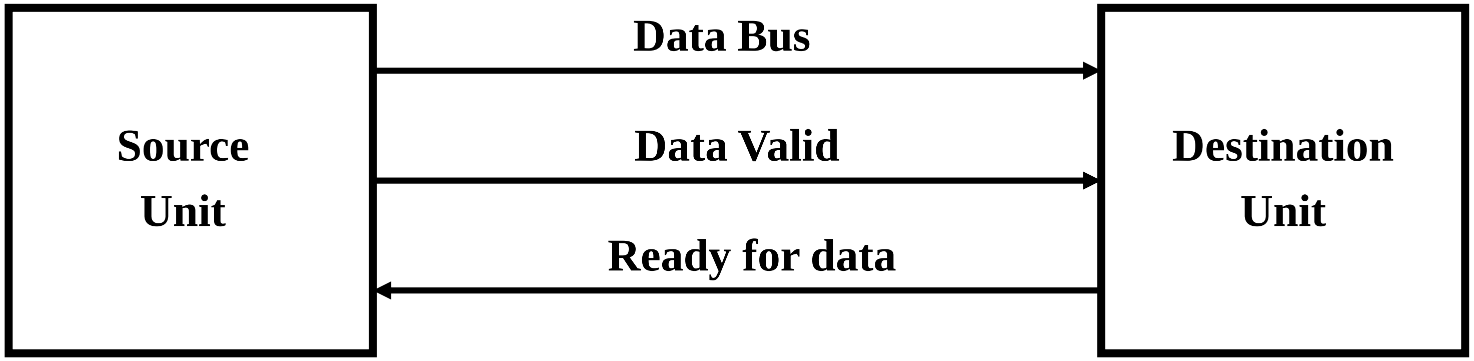


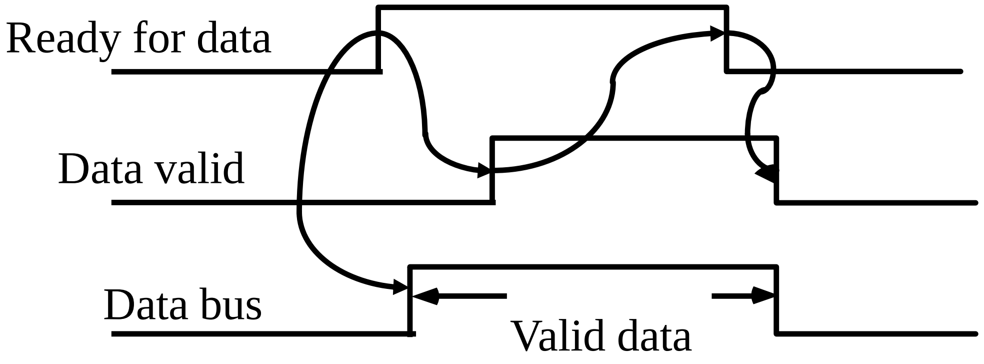
Src-Initiated Transfer via Handshake - 2



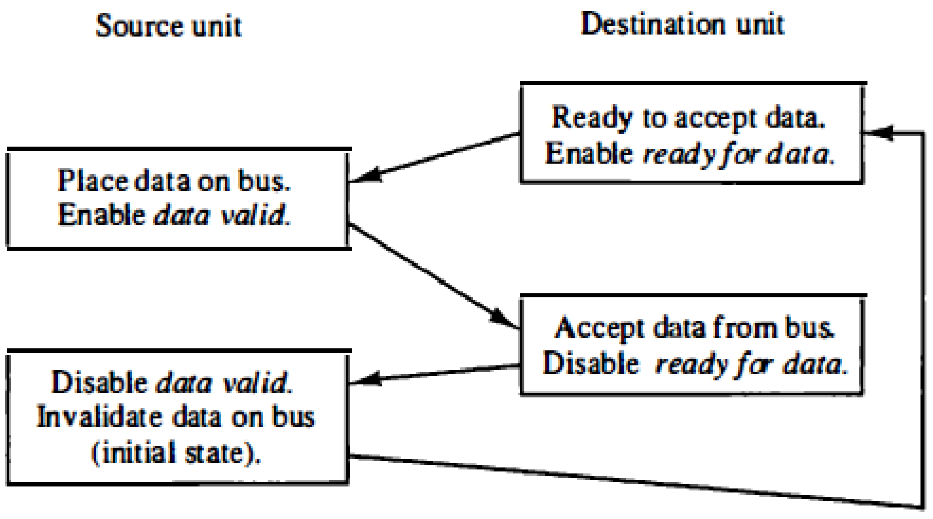


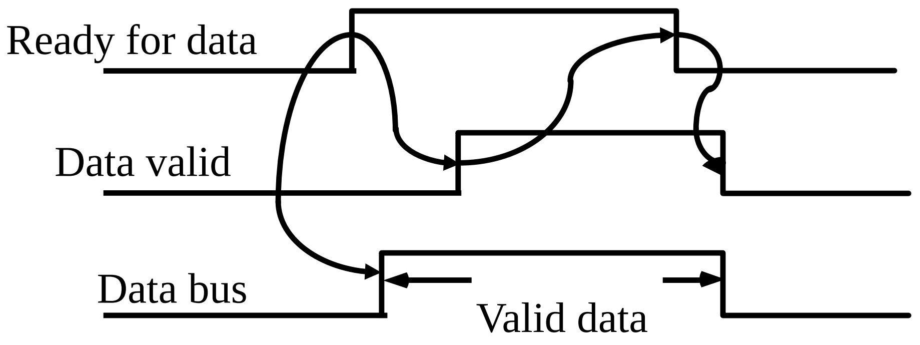
Dst-Initiated Transfer via Handshake - 1





Dst-Initiated Transfer via Handshake - 2


Source unit
Destination unit
Ready to accept data. Enable *ready for data.*
Place data on bus. Enable *data valid.*
Accept data from bus. Disable *ready for data.*
Disable *data valid.* Invalidate data on bus (initial state).



Still More …

- How is a user I/O request transformed into a device command and communicated to the device?
- What is the role of the operating system?

Operating System?

- OS responsibilities arise from I/O characteristics:
  - Multiple programs using the processor share the I/O system
  - I/O systems often use interrupts to communicate information about I/O operations. Interrupts cause a transfer to kernel or supervisor mode, so they must be handled by the OS
  - The low-level control of an I/O device is complex:
    - requires managing a set of concurrent events
    - requirements for correct device control are detailed

Operating System's Role

- Guarantees that a user's program accesses only the portions of an I/O device to which the user has rights
  - e.g., must not allow a program to read or write a file on disk if the owner of the file has not granted access to this program
  - In a system with shared I/O devices, protection could not be provided if user programs could perform I/O directly
- …

Operating System's Role (cont.)

- Guarantees that a user's program accesses only the portions of an I/O device to which the user has rights
- Provides abstractions for accessing devices by supplying routines that handle low-level device operations
- Handles the interrupts generated by I/O devices, just as it handles the exceptions generated by a program
- Tries to provide equitable access to the shared I/O resources and schedule accesses to enhance system throughput

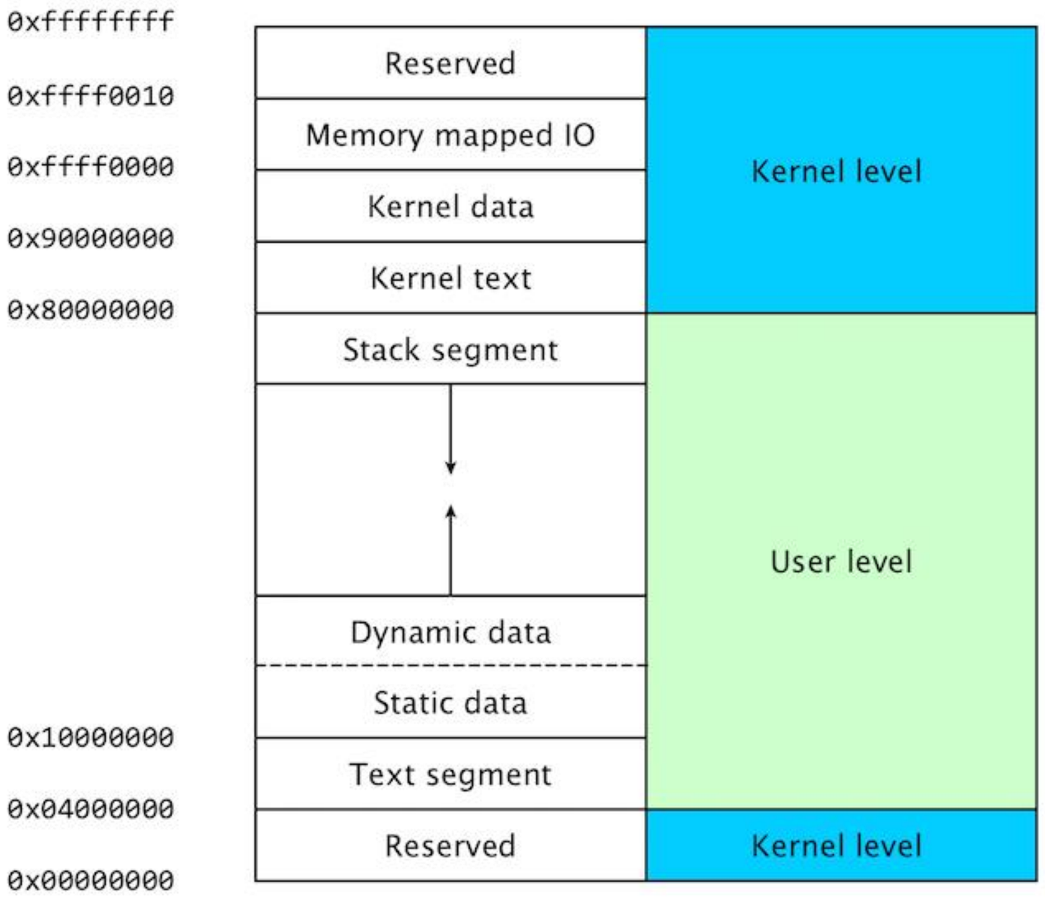
OS Communication with I/O

- Give commands to the I/O devices:
  - read, write, disk seek, …
- Be notified when the I/O device has completed an operation or has encountered an error
  - e.g., when a disk completes a seek, it will notify OS
- Transfer data between memory and I/O device
  - e.g., the block being read from a disk must be moved from disk to memory

Giving Commands to I/O Devices

- How to Access I/O?
  - From ISA perspective
- I/O Configuration
  - Isolated I/O
  - Memory-Mapped I/O

Memory-Mapped I/O in MIPS



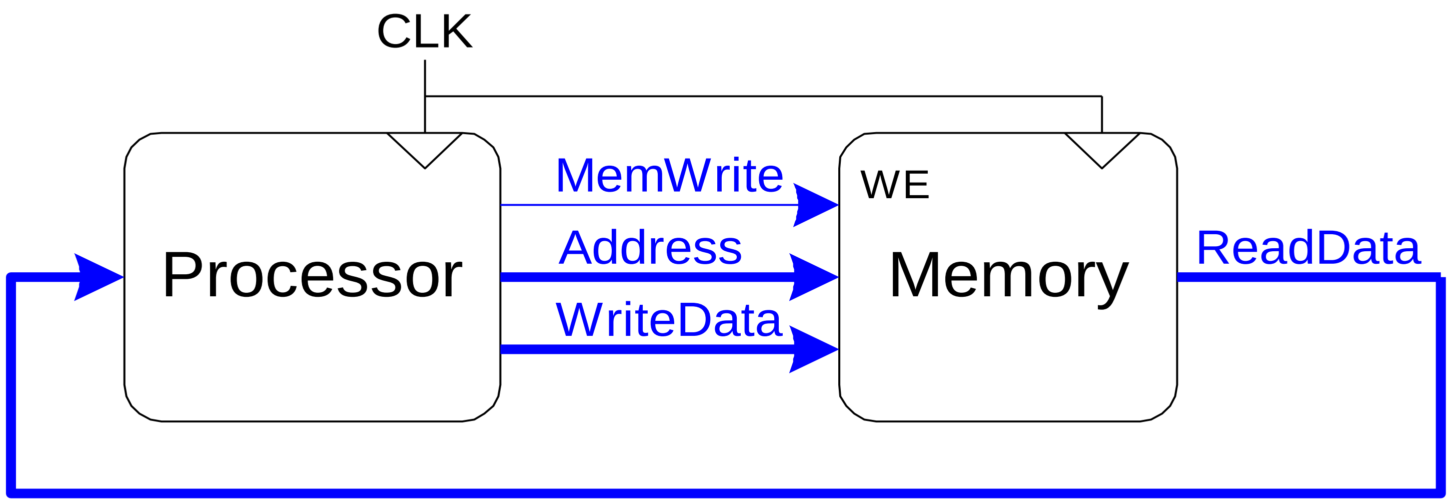
Memory-Mapped I/O

- Performed by simple load/ store instructions
- A subset of unused memory addresses mapped to registers of external devices
- Memory & devices on bus programmed to respond only to their own address ranges

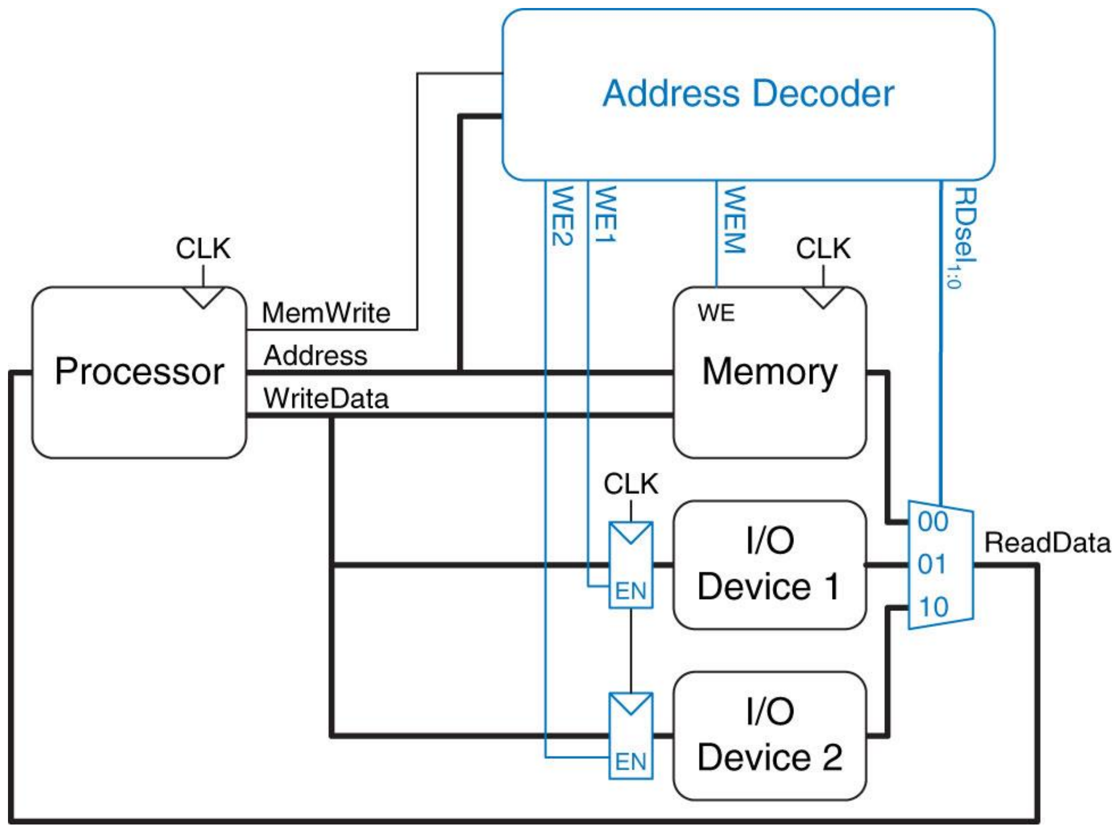
Memory-Mapped I/O

- ☺ Pros
  - Easy to handle memory locations (software perspective)
  - Wide range of addressing modes
- ☹ Cons
  - Caching may cause to memory incoherency issue
  - Slow (consumes CPU cycles)

Memory Interface

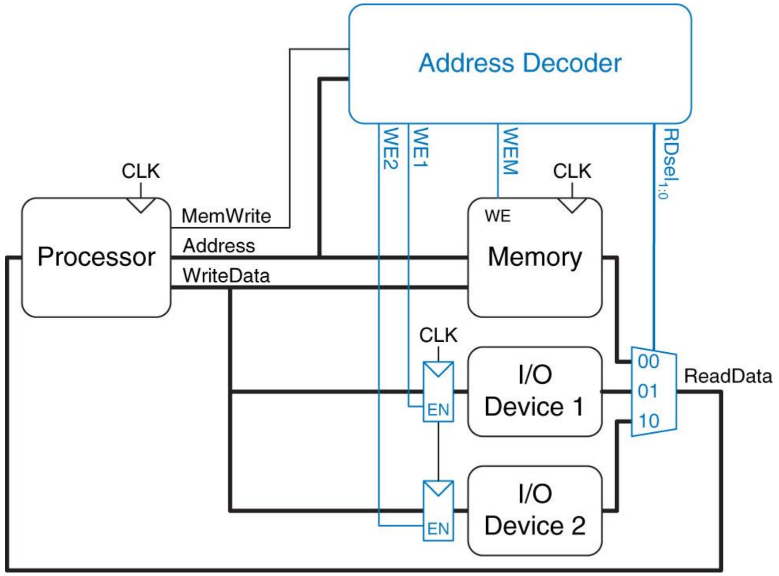


Memory-Mapped I/O Hardware



Memory-Mapped I/O Example

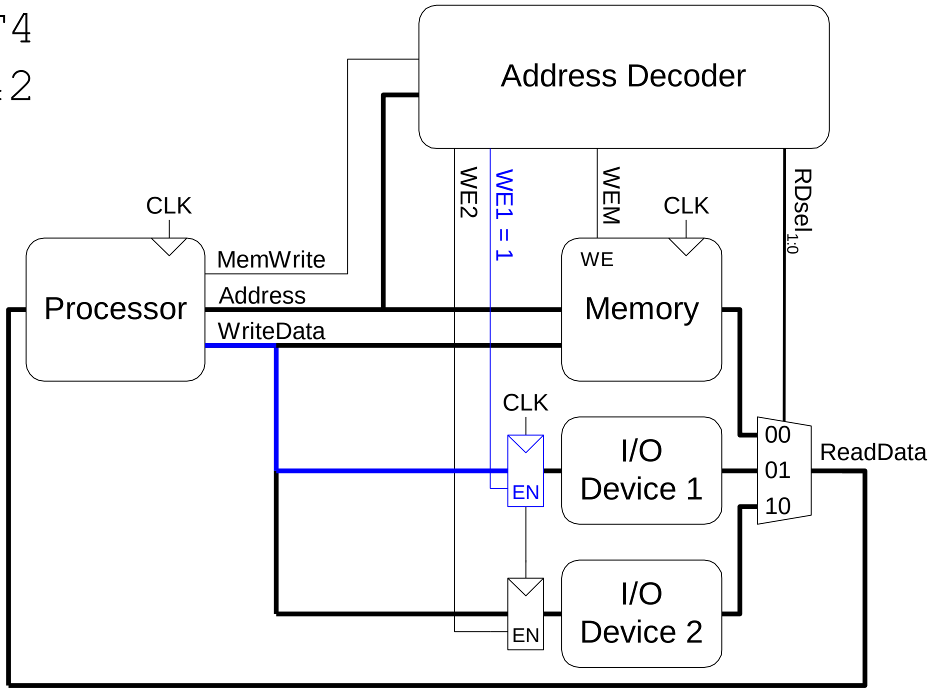
- I/O Device 1 is assigned the address 0xFFFFFFF4
  - Write the value 42 to I/O Device 1
  - Read value from I/O Device 1 and place in `r1`



Memory-Mapped I/O Example

- Write the value 42 to I/O Device 1 (0xFFFFFFF4)

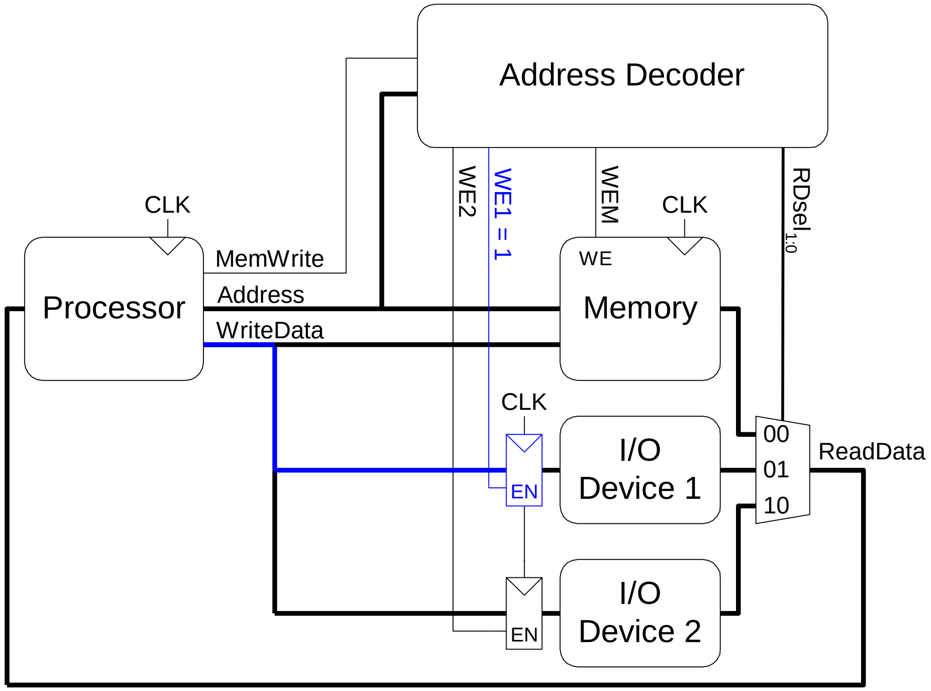
```
lui $t1,0xFFFF
ori $t1,0xFFF4
addi $t0,$0,42
sw $t0,0($t1)
```



Memory-Mapped I/O Example

- Read value from I/O Device 1 and place in `$t0`

```
lw $t0,0($t1)
```



Isolated I/O

- Dedicated I/O instructions
  - Part of ISA
- Output
  - values are written to an output register
  - on the output pins of an external port
- Input
  - values are read from an input register
  - From the input pins of an external port
- Example: Intel x86 IN/OUT instructions

I/O-Mapped I/O (cont.)

- ☺ Pros
  - Low latency
    - Simpler decoding
- ☹ Cons
  - Not implemented by all CPUs
  - Limited addressing
    - Predetermined # of I/O signals

OS Communication with I/O

- Give commands to the I/O devices:
  - read, write, disk seek, …
- Be notified when the I/O device has completed an operation or has encountered an error
  - e.g., when a disk completes a seek, it will notify OS
- Transfer data between memory and I/O device
  - e.g., the block being read from a disk must be moved from disk to memory

Programmed I/O vs. Interrupt

- Imagine
  - scrolling your mouse in screen
  - typing by your keyboard
  - accessing slow-running I/O devices
- How the processor responds?
  - Programmed I/O (aka. Polling)
  - Interrupt-driven

Programmed I/O

- Checks the status of I/O device periodically to determine the need to service
  - ☺ Simple
  - ☺ More predictable I/O overhead
  - ☹ can waste a lot of processor time
- Used in real-time applications because of predictability

Interrupt-Driven I/O

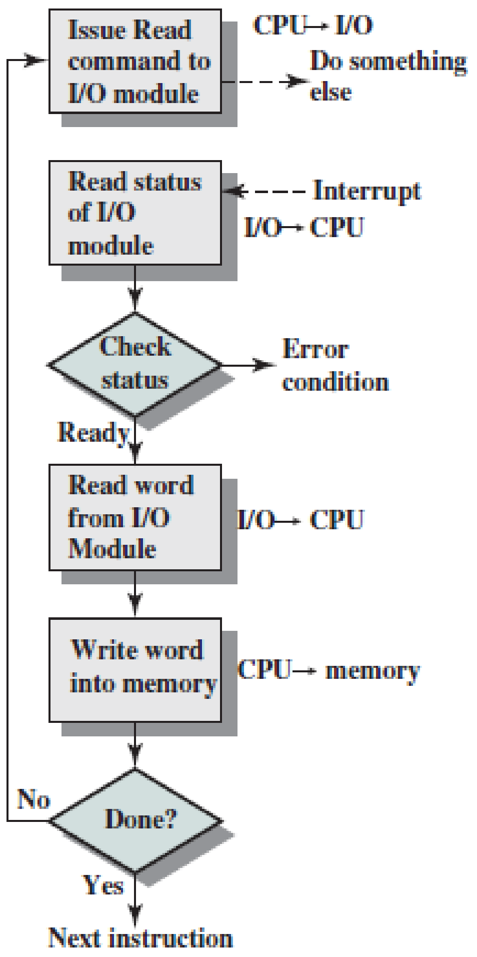
- A scheme that interrupts the processor to indicate that an I/O device needs attention

Interrupt-Driven I/O

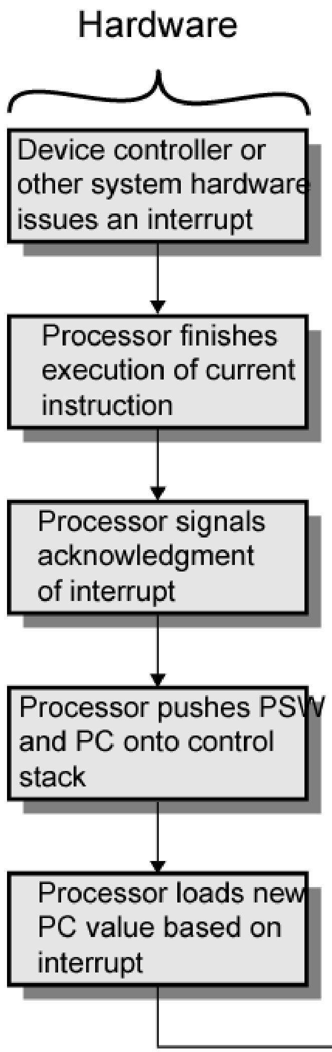
- A scheme that interrupts the processor to indicate that an I/O device needs attention
  - ☺ Overcomes CPU waiting
  - ☺ No repeated CPU checking of device
  - ☺ I/O device interrupts when ready
- I/O device
  - Raises the interrupt
  - Identifies itself
- The OS gets control and responds to the interrupt

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



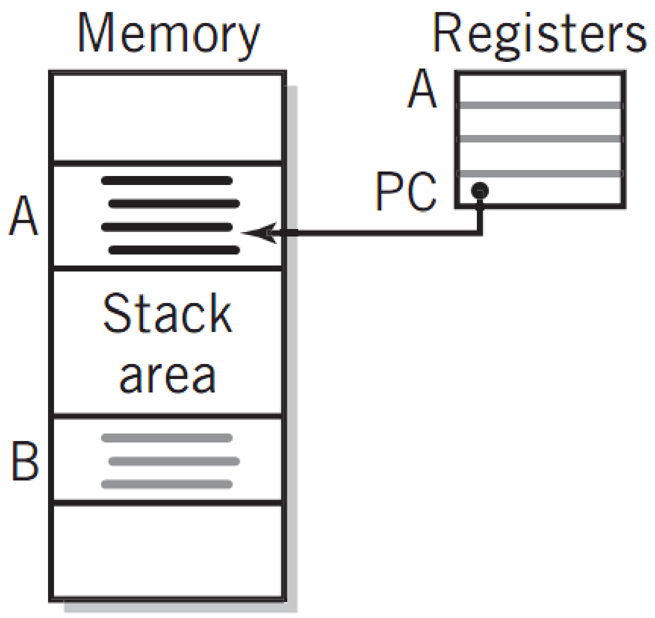
Simple Interrupt Processing


Hardware
Device controller or other system hardware issues an interrupt
Processor finishes execution of current instruction
Processor signals acknowledgment of interrupt
Processor pushes PSW and PC onto control stack
Processor loads new PC value based on interrupt
Software
Save remainder of process state information
Process interrupt
Restore process state information
Restore old PSW and PC

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

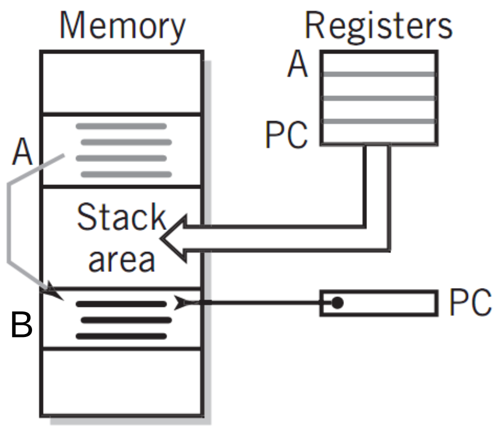
Servicing Interrupts - 1



Before interrupt arrives, program A is executing

The PC (program counter) points to the current instruction

Servicing Interrupts - 2

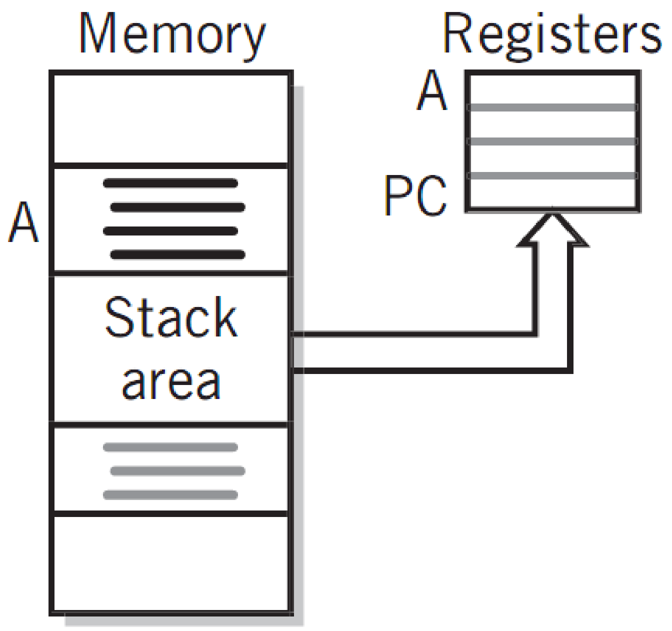


When CPU receives the interrupt, the current instruction is completed and all the registers are saved in the stack area (or in a special area known as a process control block)

The PC is loaded with the starting location of program B, the interrupt handler program

This causes a jump to B, which becomes the executing program

Servicing Interrupts - 3



When the interrupt routine is complete, the registers are restored, including the PC and the original program resumes exactly where it left off

Design Issues

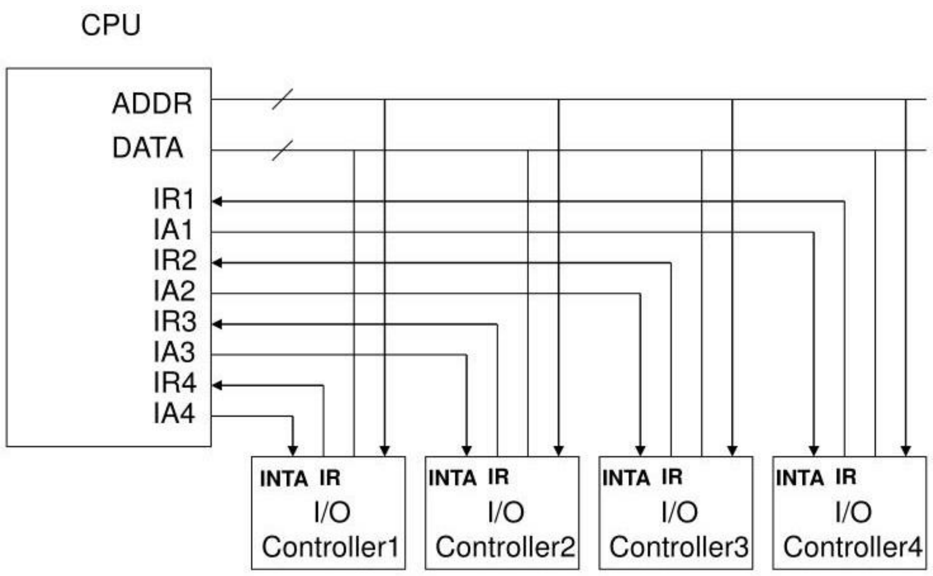
- How do you identify the module issuing the interrupt?
- How do you deal with multiple interrupts?
  - i.e. an interrupt handler being interrupted

Identifying Interrupting Module

- Multiple interrupt lines
- Software poll
- Daisy Chain or Hardware poll
- Parallel Priority Hardware
- Bus arbitration

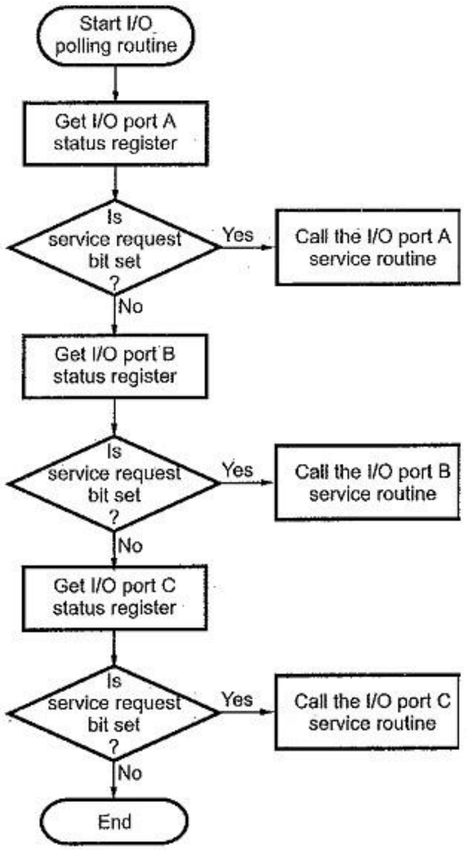
Multiple Interrupt Lines

- Different line for each module
- Limits number of devices



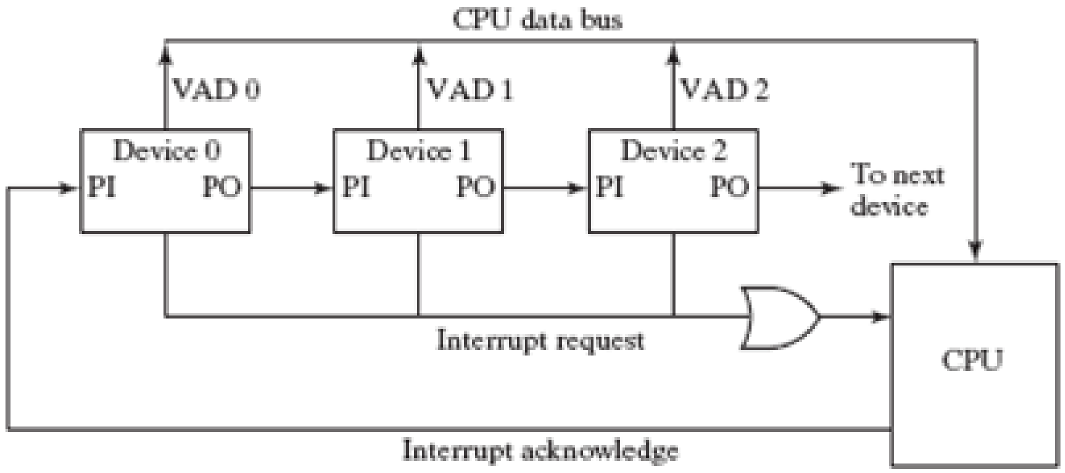
Software Poll

- CPU asks each module in turn
- Time consuming

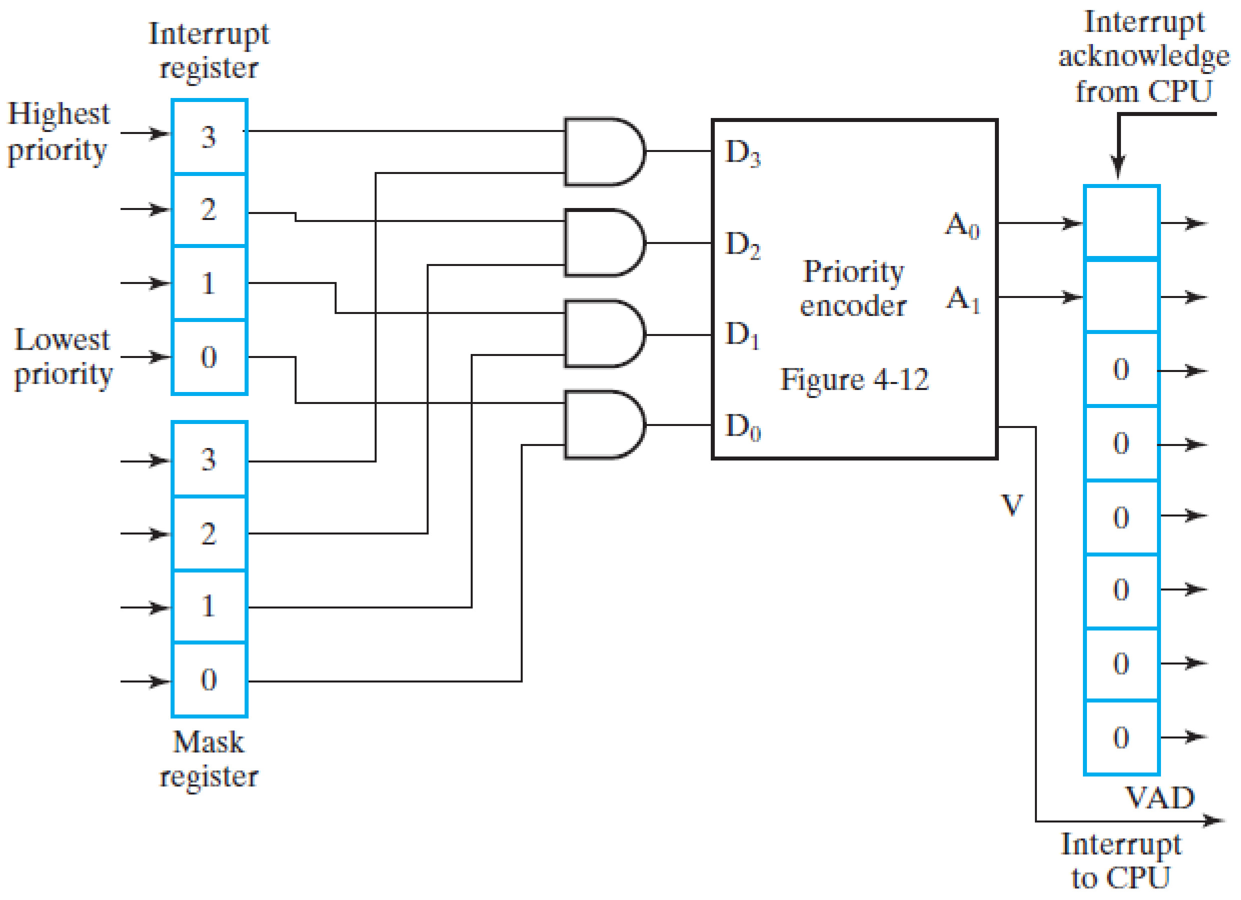

Start I/O polling routine
Get I/O port A status register
Is service request bit set ?
Yes
Call the I/O port A service routine
No
Get I/O port B status register
Is service request bit set ?
Yes
Call the I/O port B service routine
No
Get I/O port C status register
Is service request bit set ?
Yes
Call the I/O port C service routine
No
End

Hardware Poll (Daisy Chain)

- Interrupt Acknowledge sent down a chain
- Module responsible places vector on bus
- CPU uses vector to identify interrupt

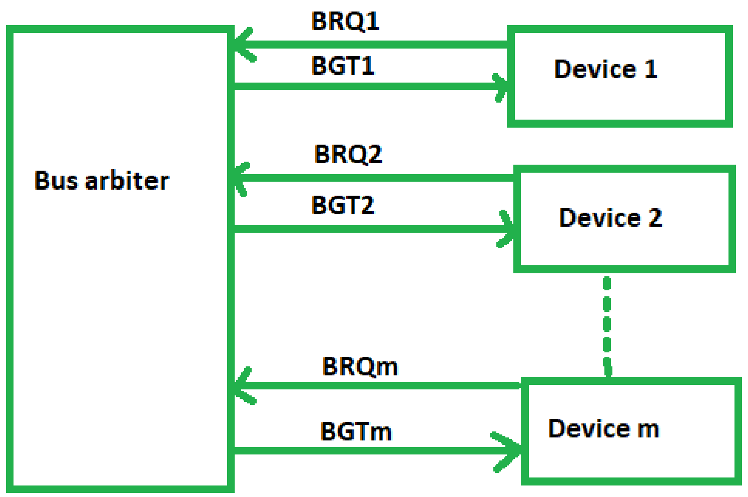


Parallel Priority Hardware



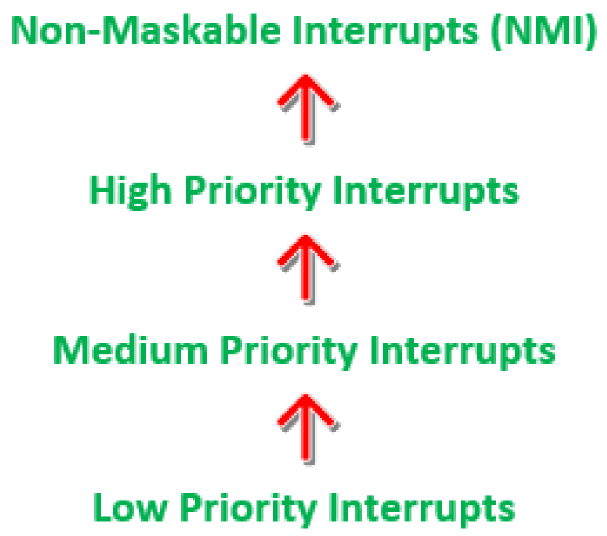
Bus Arbitration

- Module claims the bus before it can raise interrupt
- The processor detects the interrupt and responds on the interrupt acknowledge line
- The requesting module places its vector on data lines



Multiple Interrupts

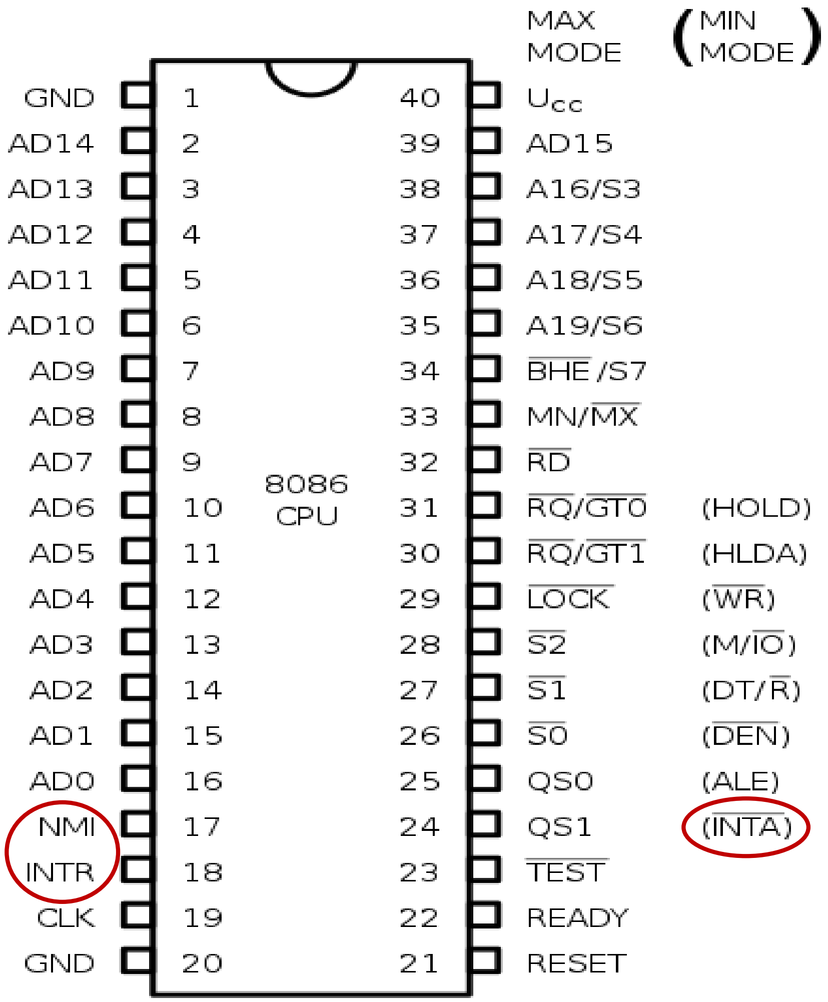
- Each interrupt has a priority
- Higher priority lines can interrupt lower priority lines



Handling Priority

- Multiple interrupt lines
  - processor picks the interrupt line with highest priority
- Software poll
  - order in which modules are polled determines priority
- Daisy Chain or Hardware poll
  - order of modules on a daisy chain determines priority
- Parallel Hardware priority
  - Use of priority encoder
- Bus arbitration
  - only current bus master can interrupt

8086 Chip



MAX MODE
(MIN MODE)
8086 CPU
GND 1 — 40 Ucc
AD14 2 — 39 AD15
AD13 3 — 38 A16/S3
AD12 4 — 37 A17/S4
AD11 5 — 36 A18/S5
AD10 6 — 35 A19/S6
AD9 7 — 34 BHE/S7
AD8 8 — 33 MN/MX
AD7 9 — 32 RD
AD6 10 — 31 RQ/GT0 (HOLD)
AD5 11 — 30 RQ/GT1 (HLDA)
AD4 12 — 29 LOCK (WR)
AD3 13 — 28 S2 (M/IO)
AD2 14 — 27 S1 (DT/R)
AD1 15 — 26 S0 (DEN)
AD0 16 — 25 QS0 (ALE)
NMI 17 — 24 QS1 (INTA)
INTR 18 — 23 TEST
CLK 19 — 22 READY
GND 20 — 21 RESET

OS Communication with I/O

- Give commands to the I/O devices:
  - read, write, disk seek, …
- Be notified when the I/O device has completed an operation or has encountered an error
  - e.g., when a disk completes a seek, it will notify OS
- Transfer data between memory and I/O device
  - e.g., the block being read from a disk must be moved from disk to memory

High-bandwidth I/O

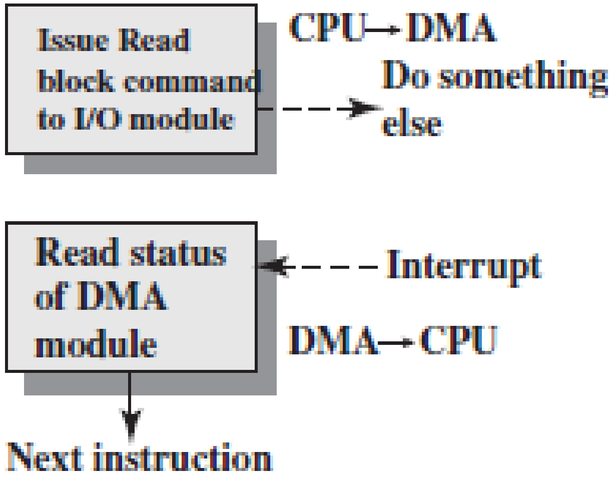
- Programmed I/O & interrupt are suitable for low bandwidth devices
  - Data transfer is done in small no of bytes
- For higher bandwidth (e.g. hard disks)
  - Programmed I/O is used in real-time applications
  - Interrupt driven approach
    - Helps OS to work on other tasks while actual I/O transfer is done
    - But the overhead of each transfer is still intolerable
- Solution: Direct Memory Access (DMA)

Direct Memory Access (DMA)

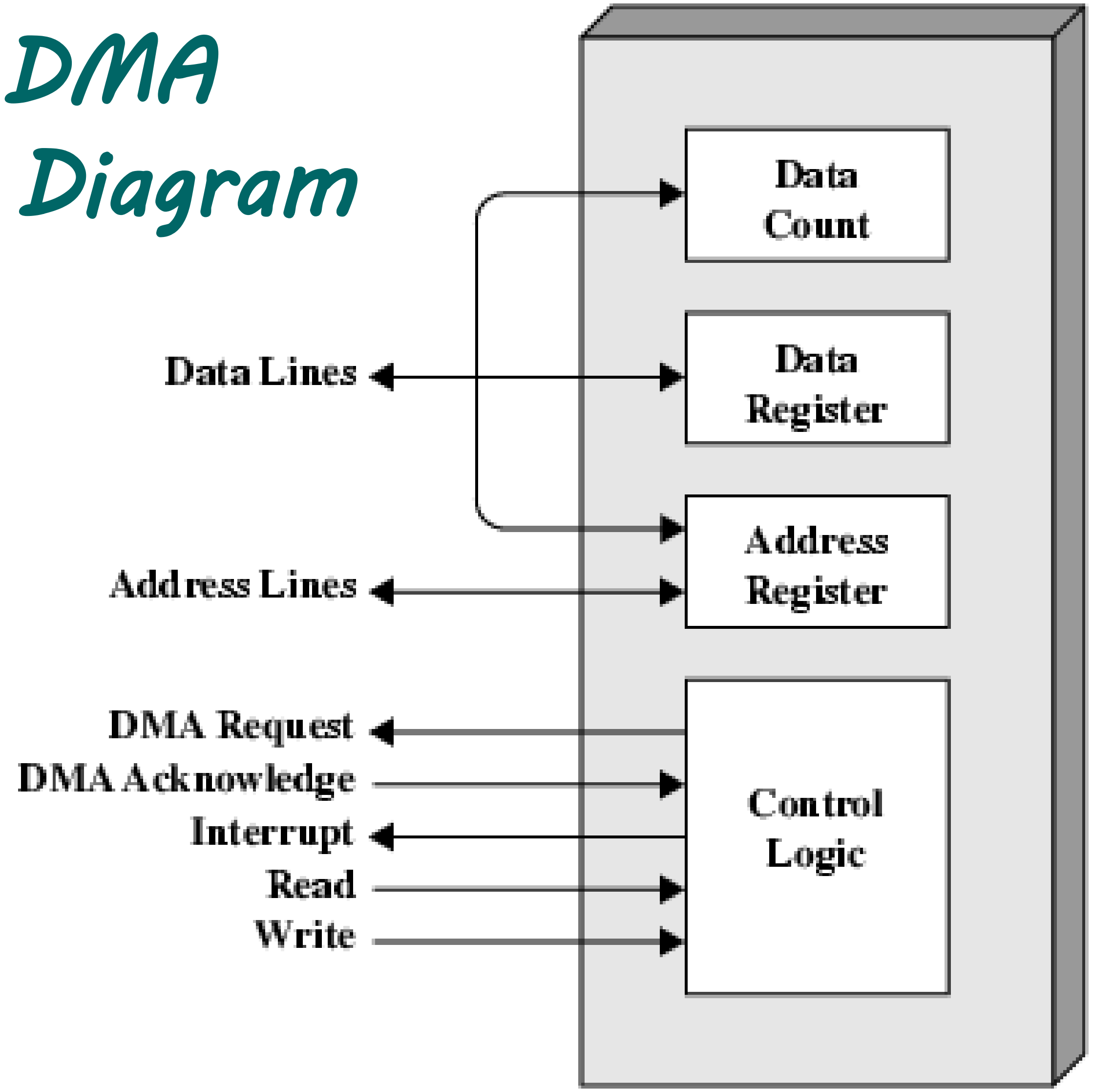
- A mechanism that provides a device controller with the ability to transfer data directly to or from the memory without involving the processor
- Implemented with a specialized controller that transfers data between an I/O device and memory independent of the processor

DMA Controller

- Additional Module (hardware) on bus
- Becomes bus master and directs reads or writes between itself and memory


Issue Read block command to I/O module
CPU→DMA
Do something else
Read status of DMA module
Interrupt
DMA→CPU
Next instruction

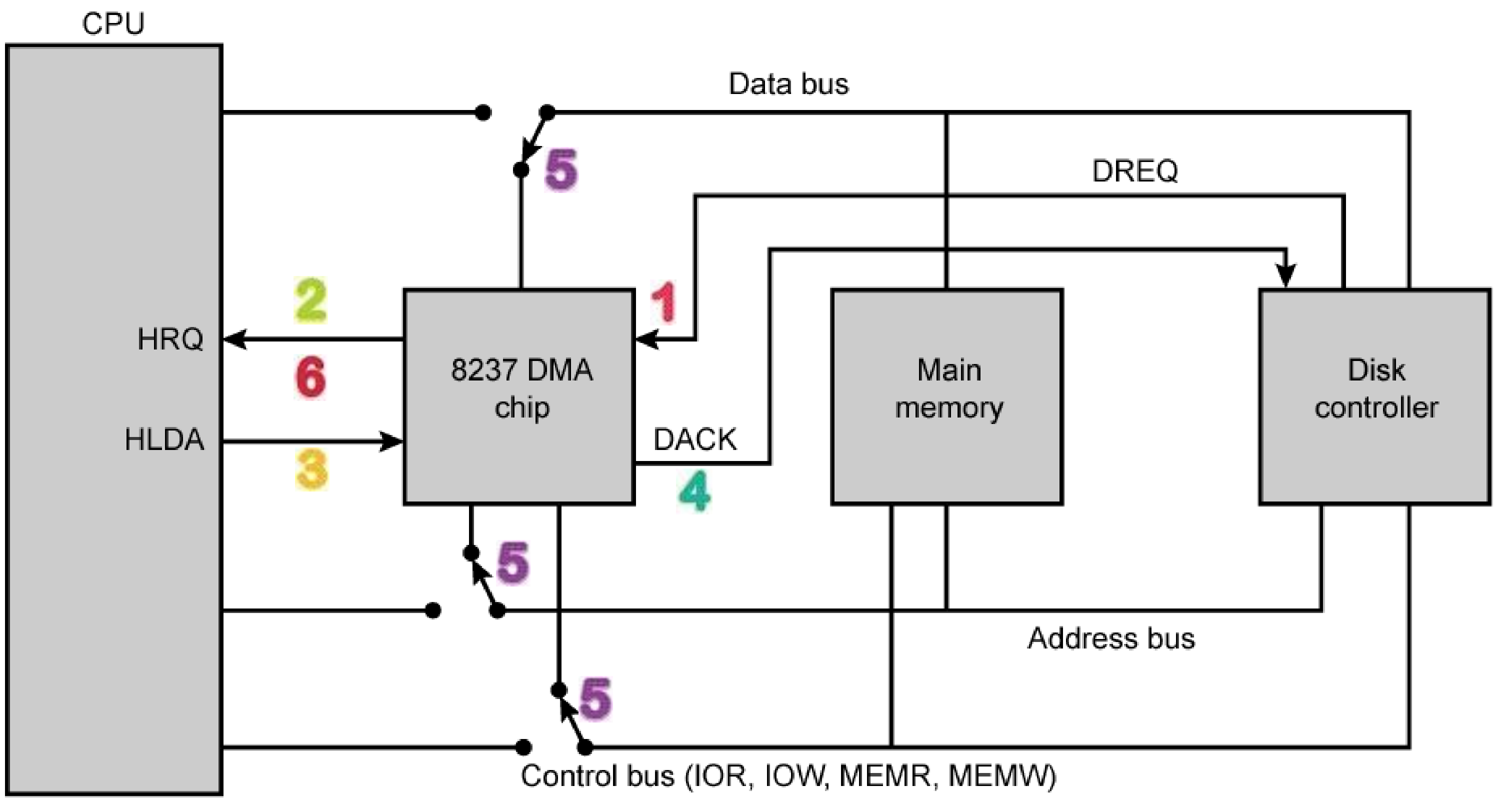
Typical DMA Module Diagram



DMA Operation

- CPU tells DMA controller
  - Read/Write
  - Device address
  - Starting address of memory block for data
  - Amount of data to be transferred
- CPU carries on with other work
- DMA controller deals with transfer
- DMA controller sends interrupt when finished

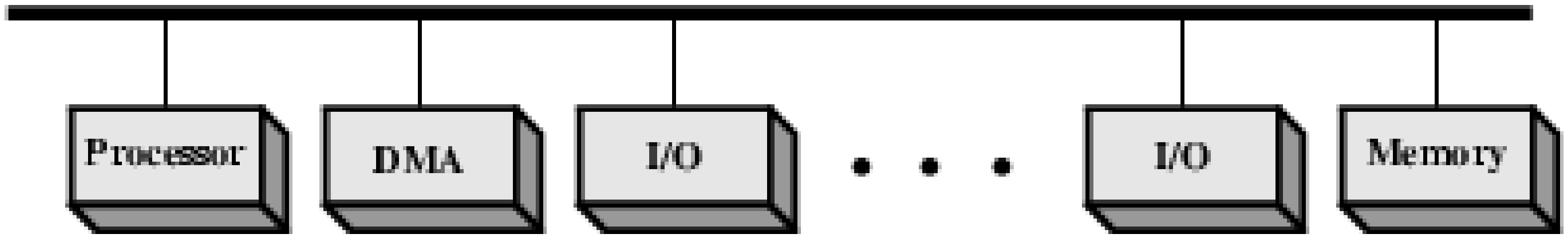
DMA Usage of Systems Bus



DACK = DMA acknowledge
DREQ = DMA request
HLDA = HOLD acknowledge
HRQ = HOLD request

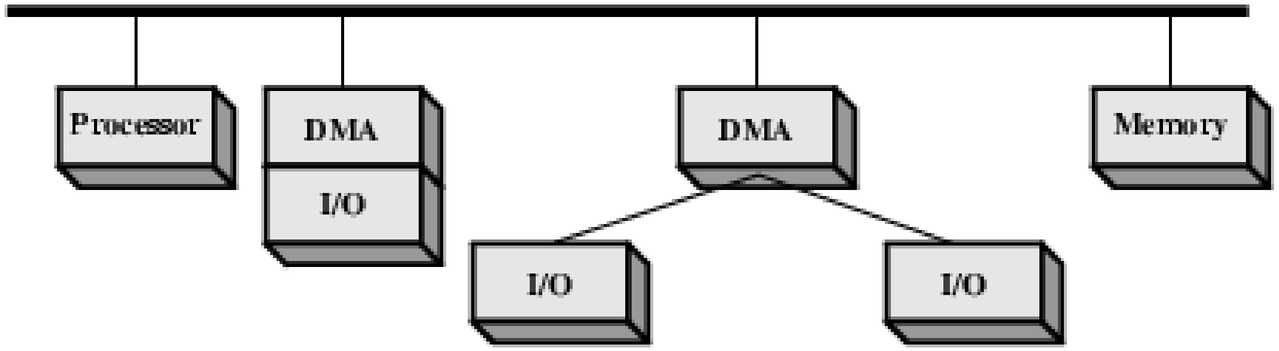
DMA Configurations (1)

- Single Bus, Detached DMA controller
- Each transfer uses bus twice
  - I/O to DMA then DMA to memory
- CPU is suspended twice



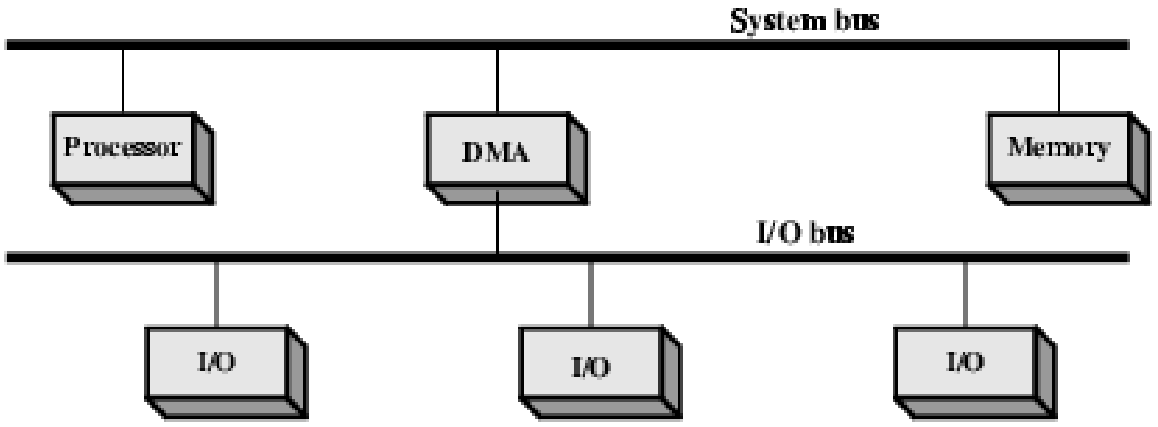
DMA Configurations (2)

- Single Bus, Integrated DMA controller
- Controller may support >1 device
- Each transfer uses bus once
  - DMA to memory
- CPU is suspended once



DMA Configurations (3)

- Separate I/O Bus
- Bus supports all DMA enabled devices
- Each transfer uses bus once
  - DMA to memory
- CPU is suspended once



Review

- I/O devices (Instances & Characteristics)
- I/O transactions
  - Sync & Async data transfer
  - Handshaking
- ISA perspective
  - Memory-mapped vs. isolated I/O
  - Interrupt vs. Programmed I/O
  - Direct Memory Access (DMA)