Unit-5

INPUT-OUTPUT (I/O) ORGANIZATION AND INTERFACING

PERIPHERAL DEVICES

Input Devices

- Keyboard
- Optical input devices
  - Card Reader
  - Paper Tape Reader
  - Bar code reader
  - Digitizer
  - Optical Mark Reader
- Magnetic Input Devices
  - Magnetic Stripe Reader
- Screen Input Devices
  - Touch Screen
  - Light Pen
  - Mouse
- Analog Input Devices

Output Devices

- Card Puncher, Paper Tape Puncher
- CRT
- Printer (Impact, Ink Jet, Laser, Dot Matrix)
- Plotter
- Analog
- Voice

INPUT/OUTPUT INTERFACES

* Provides a method for transferring information between internal storage (such as memory and CPU registers) and external I/O devices.

* Resolves the *differences* between the computer and peripheral devices

- Peripherals - Electromechanical Devices
  CPU or Memory - Electronic Device

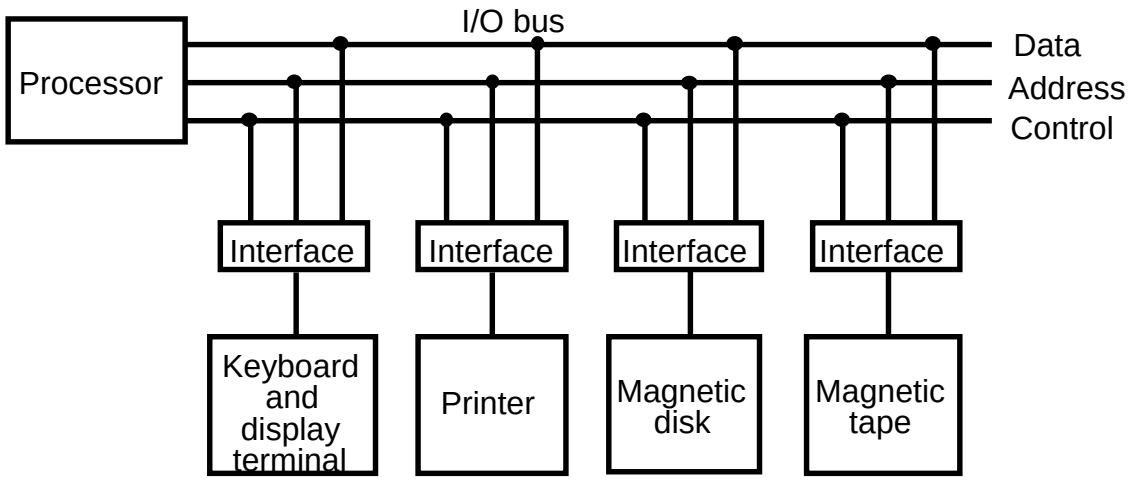
- Data Transfer Rate
  Peripherals - Usually slower
  CPU or Memory - Usually faster than peripherals

  • Some kinds of Synchronization mechanism may be needed

- Unit of Information
  Peripherals - Byte
  CPU or Memory - Word

- Operating Modes
  Peripherals - Autonomous, Asynchronous
  CPU or Memory - Synchronous

I/O BUS AND INTERFACE MODULES

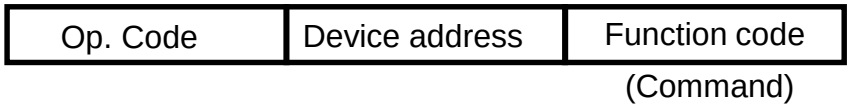


Each peripheral has an interface module associated with it.

Interface can

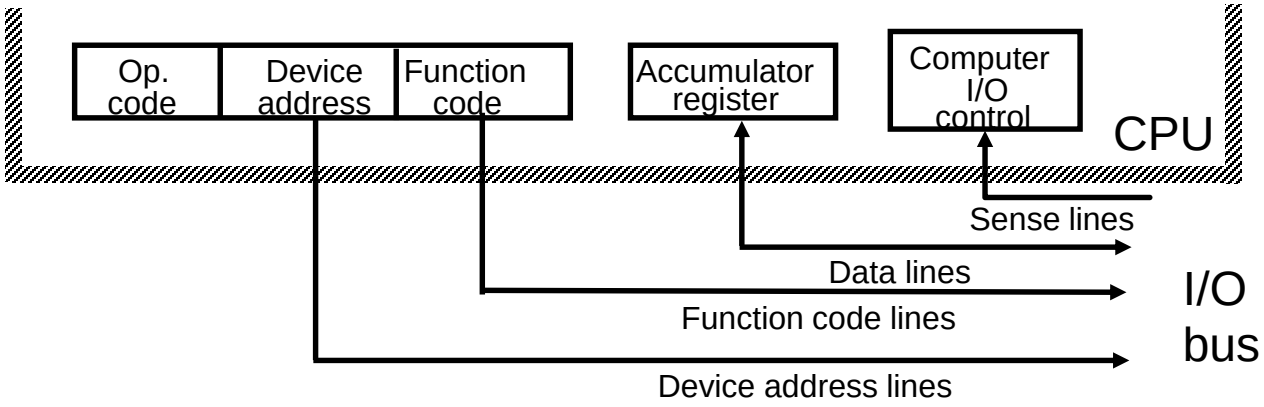
- Decode the device address (device code)
- Decode the commands (operation)
- Provide signals for the peripheral controller
- Synchronize and Supervise the data flow
- Synchronize the transfer rate between peripheral and CPU or Memory

Typical I/O instruction

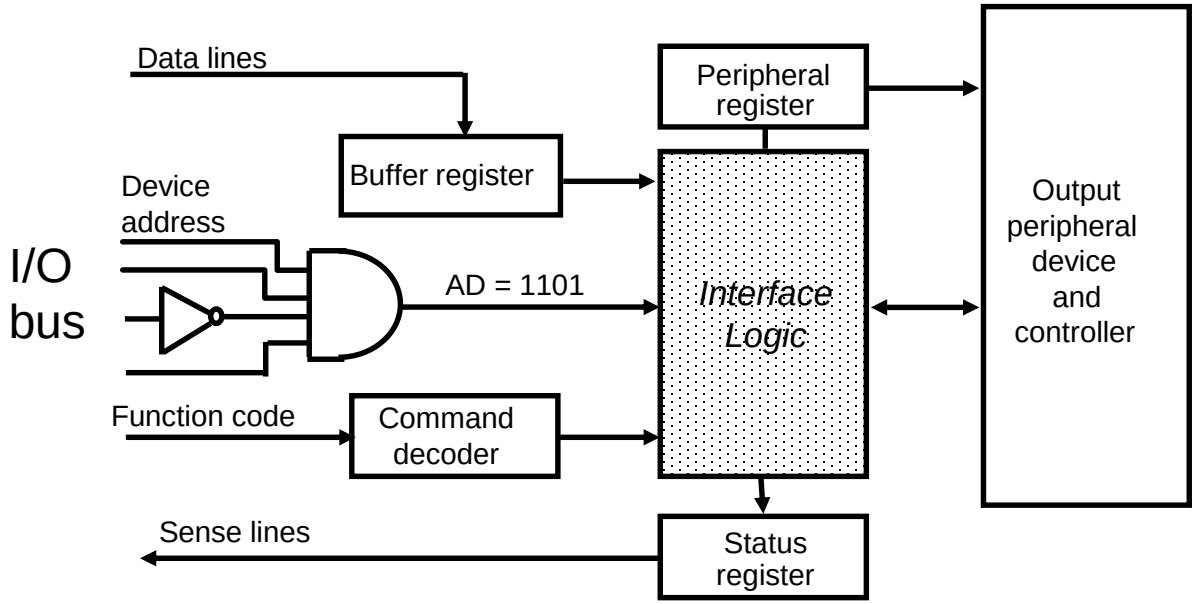


CONNECTION OF I/O BUS

Connection of I/O Bus to CPU



Connection of I/O Bus to One Interface



I/O BUS AND MEMORY BUS

Functions of Buses

* *MEMORY BUS* is for information transfer between CPU and the MM

* *I/O BUS* is for information transfer between CPU and I/O devices through their I/O interface

Physical Organizations

* Many computers use a common single bus system for both memory and I/O interface units

  - Use one common bus but separate control lines for each function
  - Use one common bus with common control lines for both functions

* Some computer systems use two separate buses, one to communicate with memory and the other with I/O interfaces

I/O Bus

- Communication between CPU and all interface units is via a common I/O Bus
- An interface connected to a peripheral device may have a number of *data registers* , a *control register*, and a *status register*
- A command is passed to the peripheral by sending to the appropriate interface register
- Function code and sense lines are not needed (Transfer of data, control, and status information is always via the common I/O Bus)

ISOLATED vs MEMORY MAPPED I/O

As a CPU needs to communicate with the various memory and input-output devices (I/O) and data flow will be occurred between the processor and these devices with the help of the system bus.

Isolated I/O

- Separate I/O read/write control lines in addition to memory read/write control lines
- Separate (isolated) memory and I/O address spaces
- Distinct input and output instructions

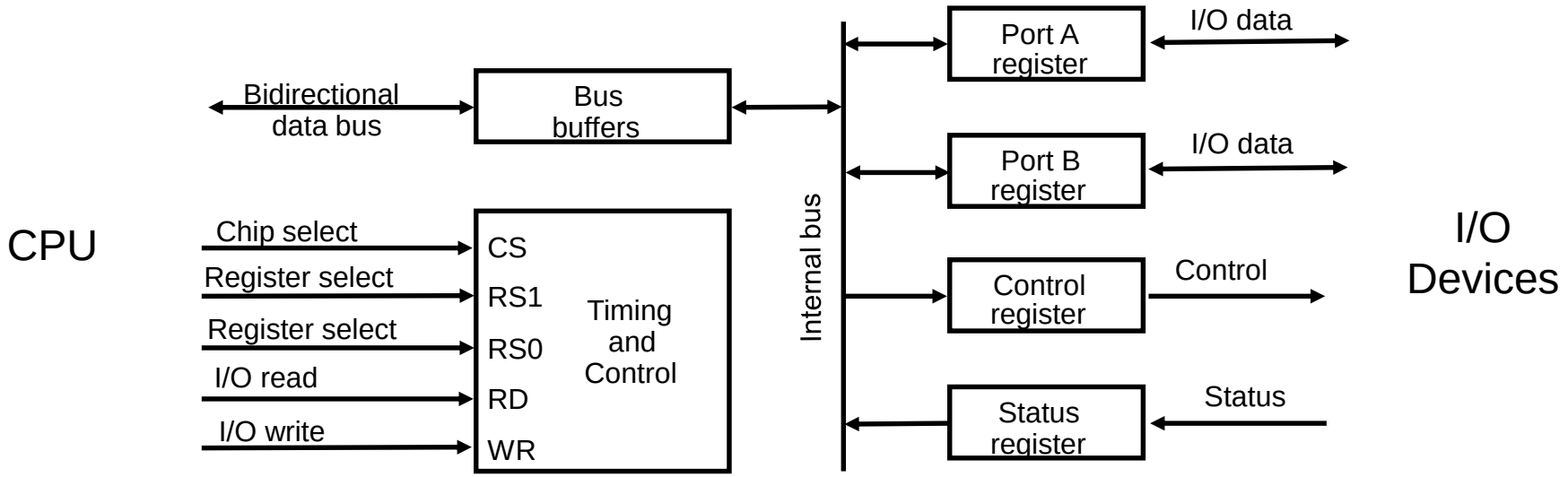
Memory-mapped I/O

- A single set of read/write control lines
  (no distinction between memory and I/O transfer)
- Memory and I/O addresses share the common address space
  -> reduces memory address range available
- No specific input or output instruction
  -> The same memory reference instructions can be used for I/O transfers
- Considerable flexibility in handling I/O operations

Difference Between ISOLATED vs MEMORY MAPPED I/O

ISOLATED I/O	MEMORY MAPPED I/O
Memory and I/O have separate address space	Both have same address space
All address can be used by the memory	Due to addition of I/O addressable memory become less for memory
Separate instruction control read and write operation in I/O and Memory	Same instructions can control both I/O and Memory
In this I/O address are called ports.	Normal memory address are for both
More efficient due to separate buses	Lesser efficient
Larger in size due to more number of buses	Smaller in size
It is complex due to separate separate logic is used to control both.	Simpler logic is used as I/O is also treated as memory only.

I/O INTERFACE (8255 PPI)



CS	RS1	RS0	Register selected
0	x	x	None - data bus in high-impedence
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

Programmable Interface

- Information in each port can be assigned a meaning depending on the mode of operation of the I/O device
  -> Port A = Data; Port B = Command; Port C = Status

- CPU initializes(loads) each port by transferring a byte to the Control Register
  -> Allows CPU can define the mode of operation of each port
  -> *Programmable Port*: By changing the bits in the control register, it is possible to change the interface characteristics

ASYNCHRONOUS DATA TRANSFER

Synchronous and Asynchronous Operations

Synchronous - All devices derive the timing information from common clock line

Asynchronous - No common clock

Asynchronous Data Transfer

Asynchronous data transfer between two independent units requires that *control signals* be transmitted between the communicating units *to indicate the time at which data is being transmitted*

Two Asynchronous Data Transfer Methods

Strobe pulse

- A strobe pulse is supplied by one unit to indicate the other unit when the transfer has to occur

Handshaking

- A control signal is accompanied with each data being transmitted to indicate the presence of data
- The receiving unit responds with another control signal to acknowledge receipt of the data

Recap of I/O Organization

- I/O devices are very different (i.e. keyboard and HDD performs totally different functions, yet they are both part of the I/O subsystem).

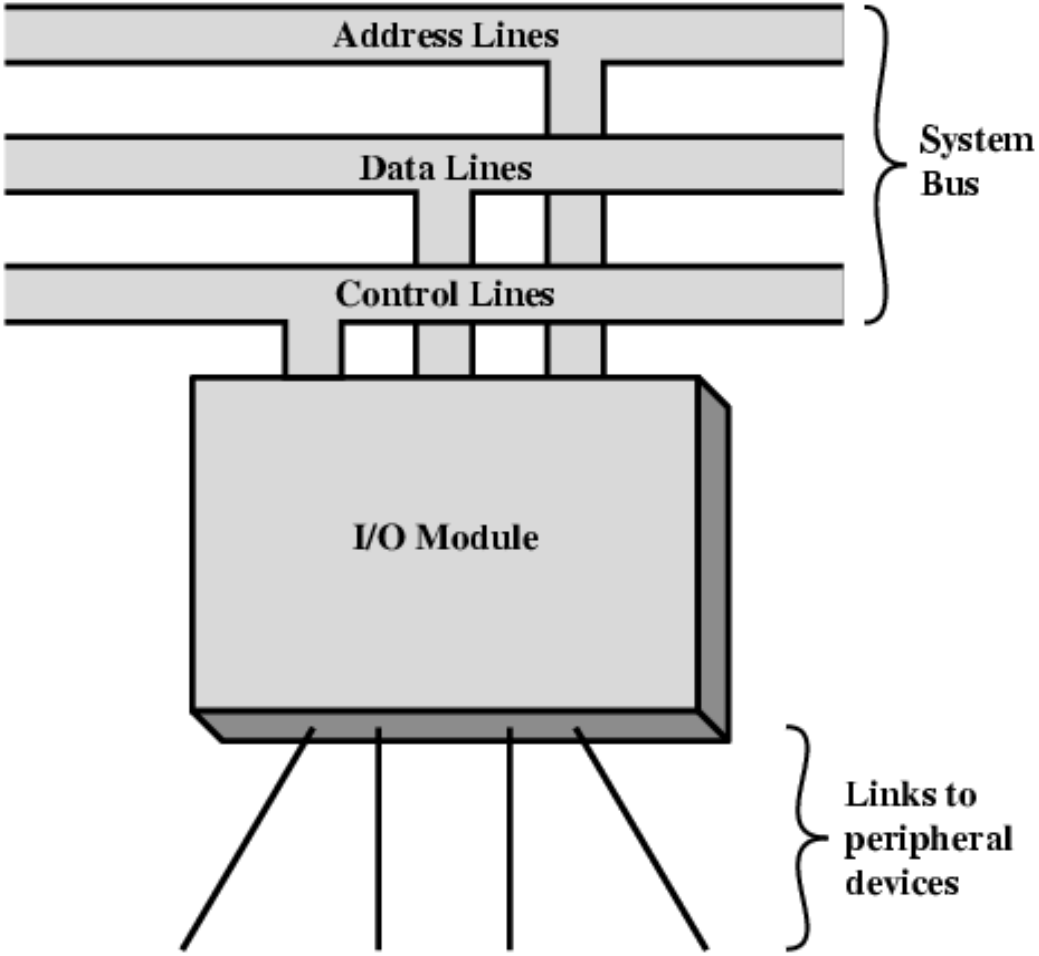
- Each I/O device needs to be connected to:
  - Address bus – to pass address to peripheral
  - Data bus – to pass data to and from peripheral
  - Control bus – to control signals to peripherals

Problems

- Wide variety of peripherals
  - Delivering different amounts of data
  - At different speeds
  - In different formats
- All slower than CPU and RAM

- Need I/O modules
  - Interface with the processor and memory via system buses or central switch
  - Interface to one or more peripheral devices using specific data links/interfaces

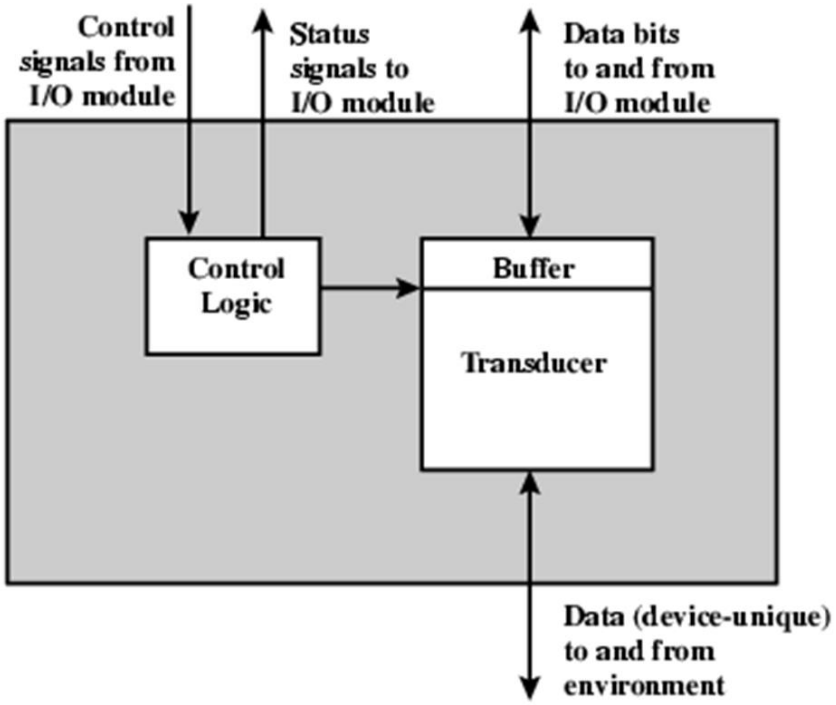
I/O Module



- Interface to CPU and Memory
- Interface to one or more peripherals

Peripheral Devices Types & Block Diagram

- Human readable
  - Screen, printer, keyboard
- Machine readable
  - Monitoring and control
- Communication
  - Modem
  - Network Interface Card (NIC)



More about I/O Modules

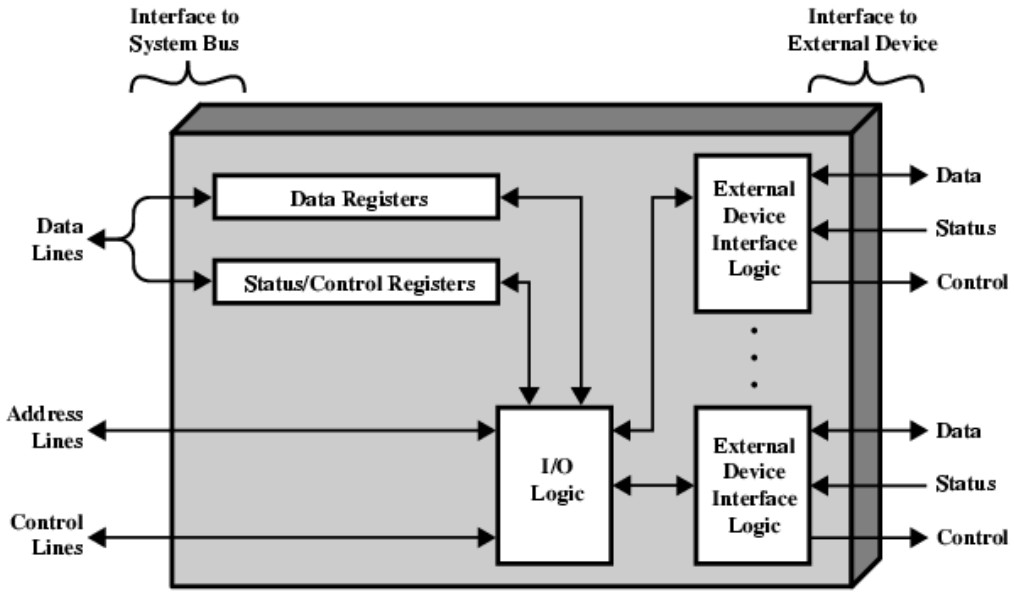
I/O Module Functions

- Control & Timing
- CPU Communication
- Device Communication
- Data Buffering
- Error Detection

I/O CPU Steps

- CPU checks I/O module device status
- I/O module returns status
- If ready, CPU requests data transfer
- I/O module gets data from device
- I/O module transfers data to CPUs

I/O Module Diagram & Design Decisions

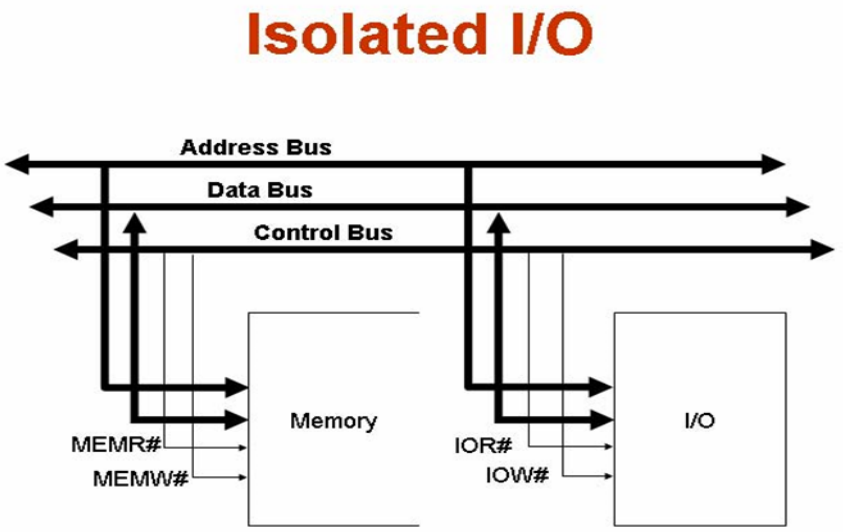


- Hide or reveal device properties to CPU
- Support multiple or single device
- Control device functions or leave for CPU
- Also O/S decisions
  - e.g. Unix treats everything it can as a file

I/O Mapping

- Memory mapped I/O
  - Devices and memory share an address space
  - I/O looks just like memory read/write
  - No special commands for I/O
    - Large selection of memory access commands available

- Isolated I/O
  - Separate address spaces
  - Need I/O or memory select lines
  - Special commands for I/O
  - Special CPU control signals
  - Devices and Memory can have overlapping addresses



Addressing I/O Devices

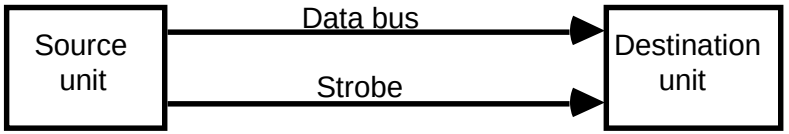
- I/O data transfer is very like memory access (CPU viewpoint)
- Each device given unique identifier
- CPU commands contain identifier (address)
- The IO Module should contain address decoding logic

STROBE CONTROL

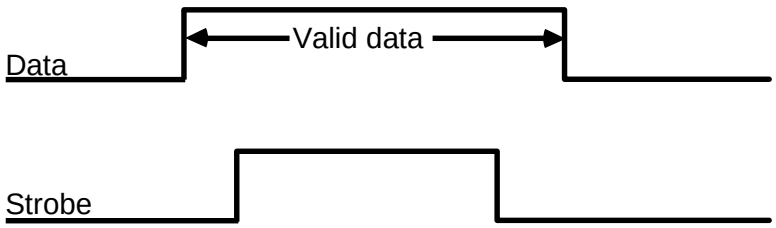
* Employs a single control line to time each transfer
* The strobe may be activated by either the source or the destination unit

Source-Initiated Strobe for Data Transfer

Block Diagram

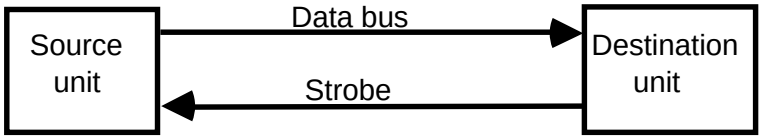


Timing Diagram

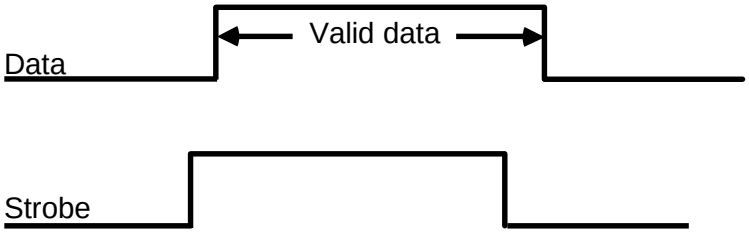


Destination-Initiated Strobe for Data Transfer

Block Diagram



Timing Diagram



HANDSHAKING

Strobe Methods

Source-Initiated

The source unit that initiates the transfer has no way of knowing whether the destination unit has actually received the data or not.

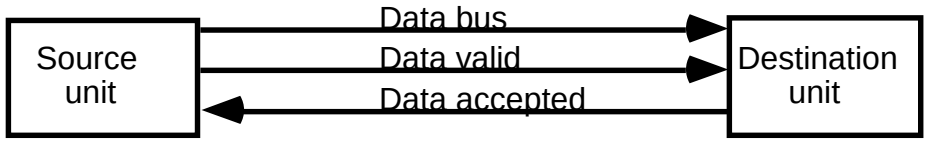
Destination-Initiated

The destination unit that initiates the transfer no way of knowing whether the source has actually placed the data on the bus or not.

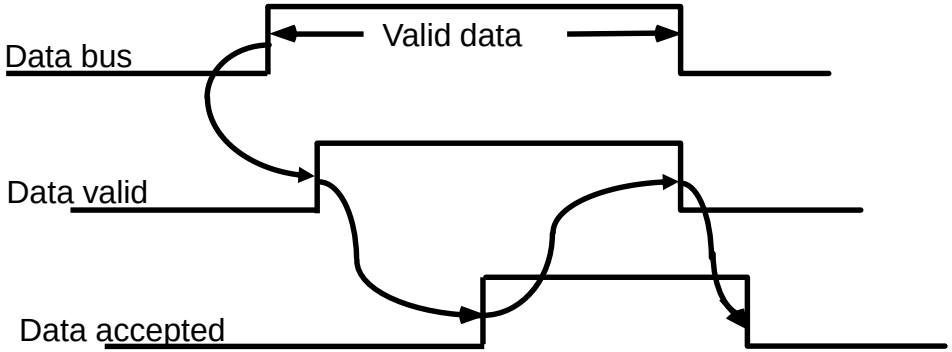
To solve this problem, the *HANDSHAKE* method introduces a second control signal to provide a *Reply* to the unit that initiates the transfer

SOURCE-INITIATED TRANSFER USING HANDSHAKE

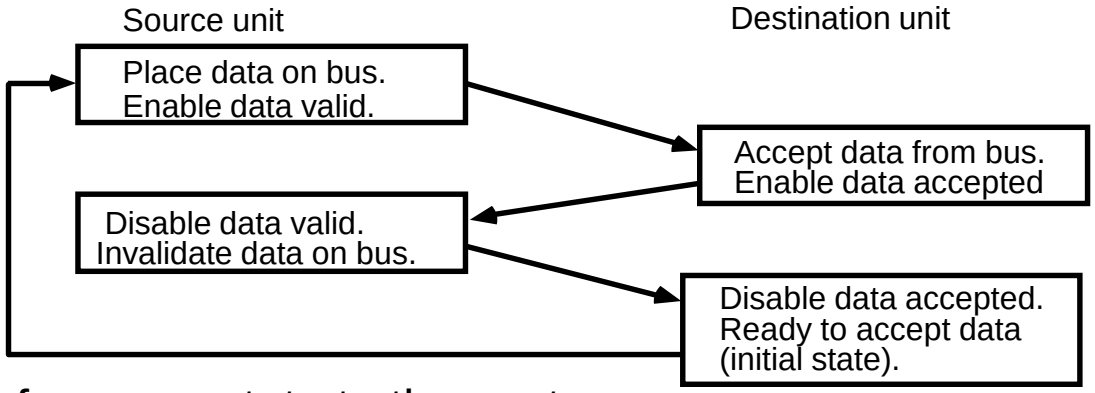
Block Diagram



Timing Diagram



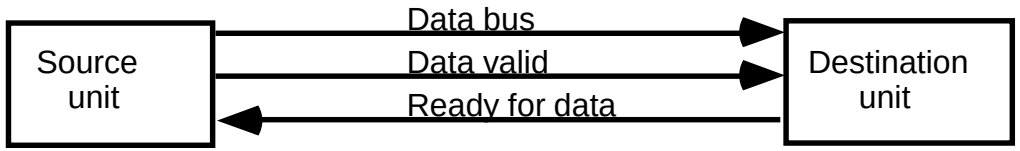
Sequence of Events



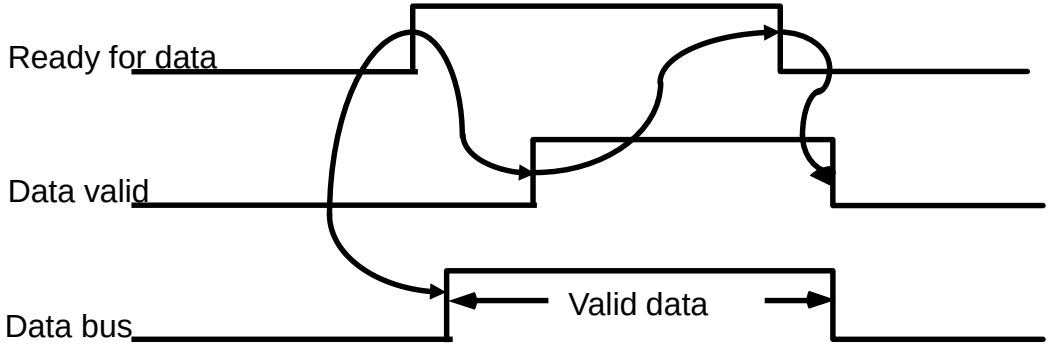
* Allows arbitrary delays from one state to the next
* Permits each unit to respond at its own data transfer rate
* The rate of transfer is determined by the slower unit

DESTINATION-INITIATED TRANSFER USING HANDSHAKE

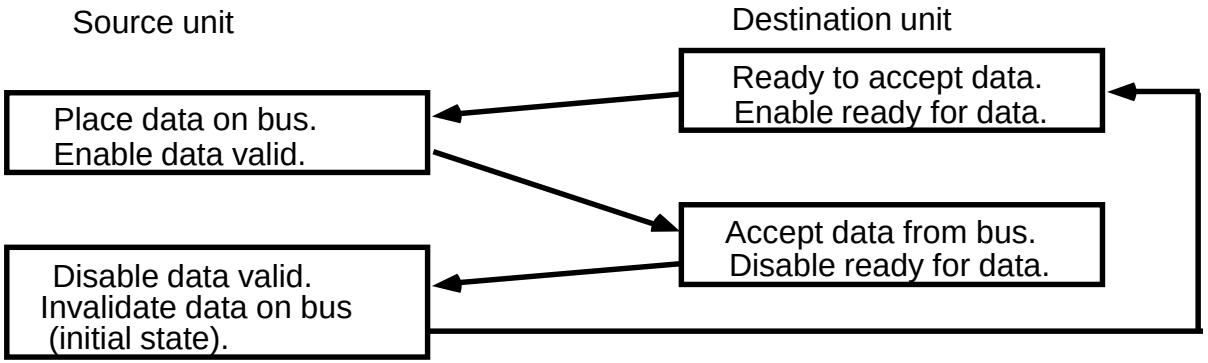
Block Diagram



Timing Diagram



Sequence of Events



* Handshaking provides a high degree of flexibility and reliability because the successful completion of a data transfer relies on active participation by both units
* If one unit is faulty, data transfer will not be completed
  -> Can be detected by means of a *timeout* mechanism

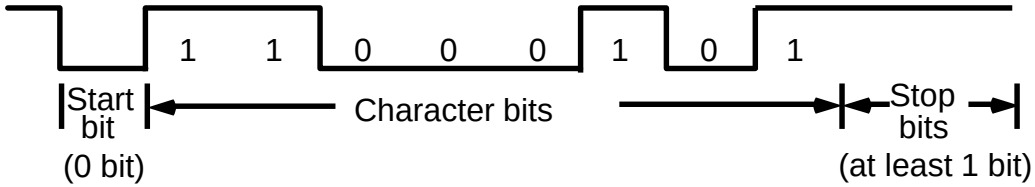
ASYNCHRONOUS SERIAL TRANSFER

Four Different Types of Transfer

Asynchronous serial transfer
Synchronous serial transfer
Asynchronous parallel transfer
Synchronous parallel transfer

Asynchronous Serial Transfer

- Employs special bits which are inserted at both ends of the character code
- Each character consists of three parts; Start bit; Data bits; Stop bits.



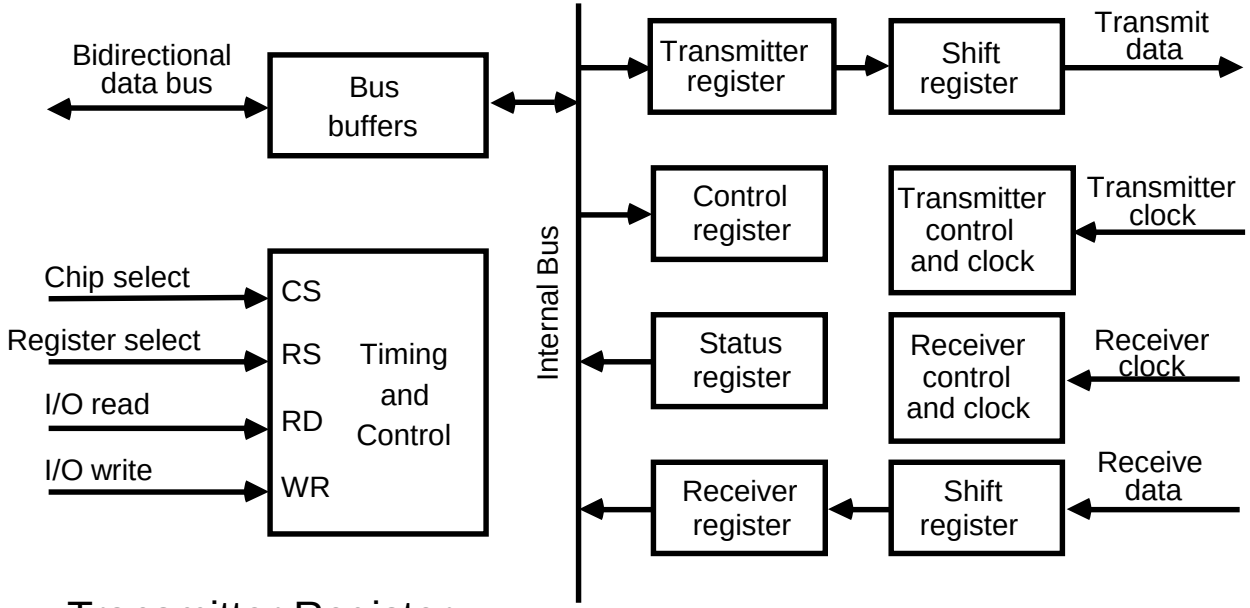
A character can be detected by the receiver from the knowledge of 4 rules;

- When data are not being sent, the line is kept in the 1-state (idle state)
- The initiation of a character transmission is detected by a *Start Bit* , which is always a 0
- The character bits always follow the *Start Bit*
- After the last character , a *Stop Bit* is detected when the line returns to the 1-state for at least 1 bit time

The receiver knows in advance the transfer rate of the bits and the number of information bits to expect

UNIVERSAL ASYNCHRONOUS RECEIVER-TRANSMITTER - USART (8251)-

A typical asynchronous communication interface available as an IC



CS	RS	Oper.	Register selected
0	x	x	None
1	0	WR	Transmitter register
1	1	WR	Control register
1	0	RD	Receiver register
1	1	RD	Status register

Transmitter Register
- Accepts a data byte(from CPU) through the data bus
- Transferred to a shift register for serial transmission

Receiver
- Receives serial information into another shift register
- Complete data byte is sent to the receiver register

Status Register Bits
- Used for I/O flags and for recording errors

Control Register Bits
- Define baud rate, no. of bits in each character, whether to generate and check parity, and no. of stop bits

Overview of I/O Organisation

- I/O devices are very different (i.e. keyboard and HDD performs totally different functions, yet they are both part of the I/O subsystem).

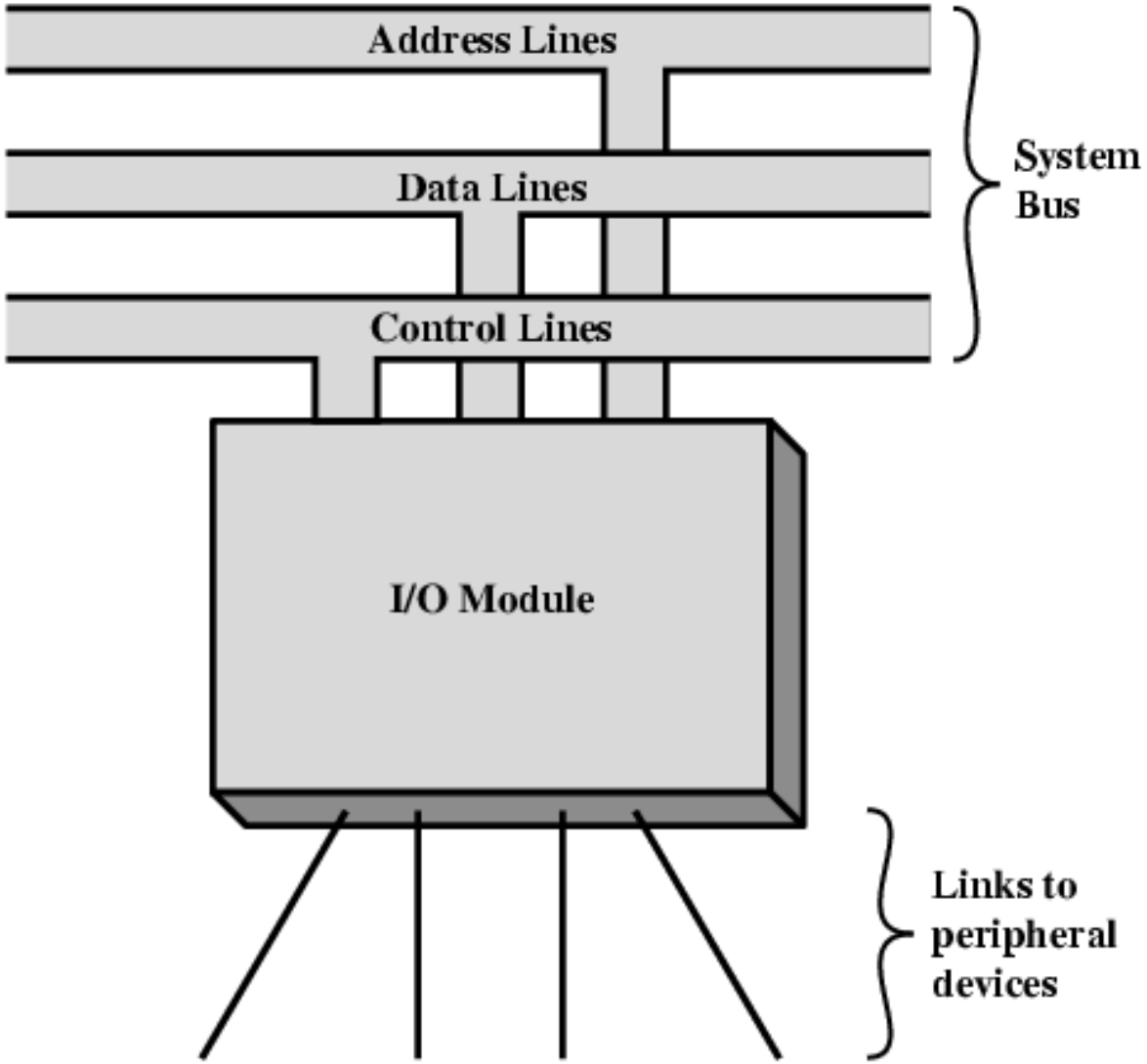
- The interfaces between the CPU and I/O devices are very similar.

- Each I/O device needs to be connected to:
  - Address bus – to pass address to peripheral
  - Data bus – to pass data to and from peripheral
  - Control bus – to control signals to peripherals

Problems

- Wide variety of peripherals
  - Delivering different amounts of data
  - At different speeds
  - In different formats
- All slower than CPU and RAM
- Need I/O modules
  - Interface with the processor and memory via system buses or central switch
  - Interface to one or more peripheral devices using specific data links/interfaces

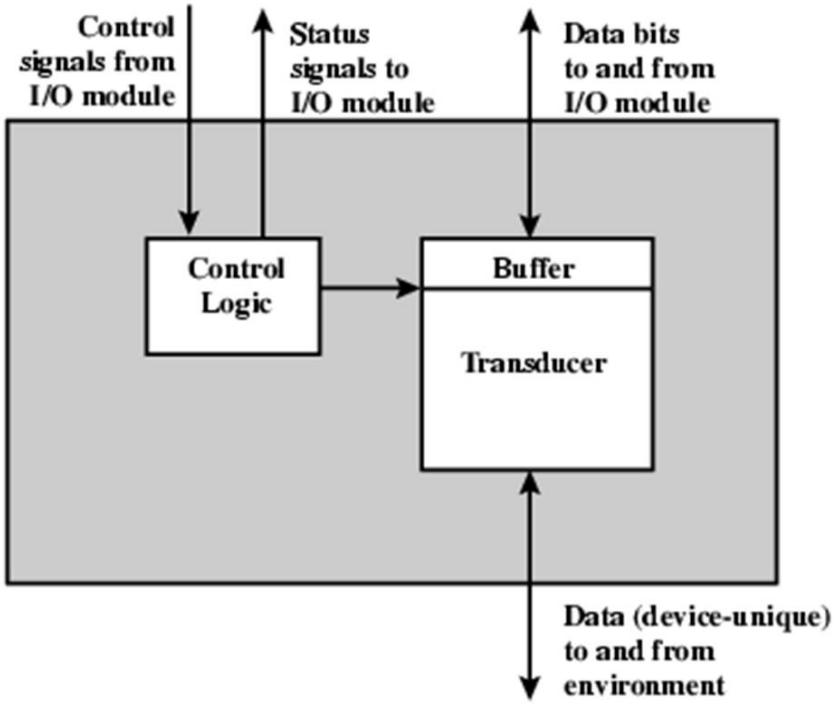
I/O Module



- Interface to CPU and Memory
- Interface to one or more peripherals

Peripheral Devices Types & Block Diagram

- Human readable
  - Screen, printer, keyboard
- Machine readable
  - Monitoring and control
- Communication
  - Modem
  - Network Interface Card (NIC)



More about I/O Modules

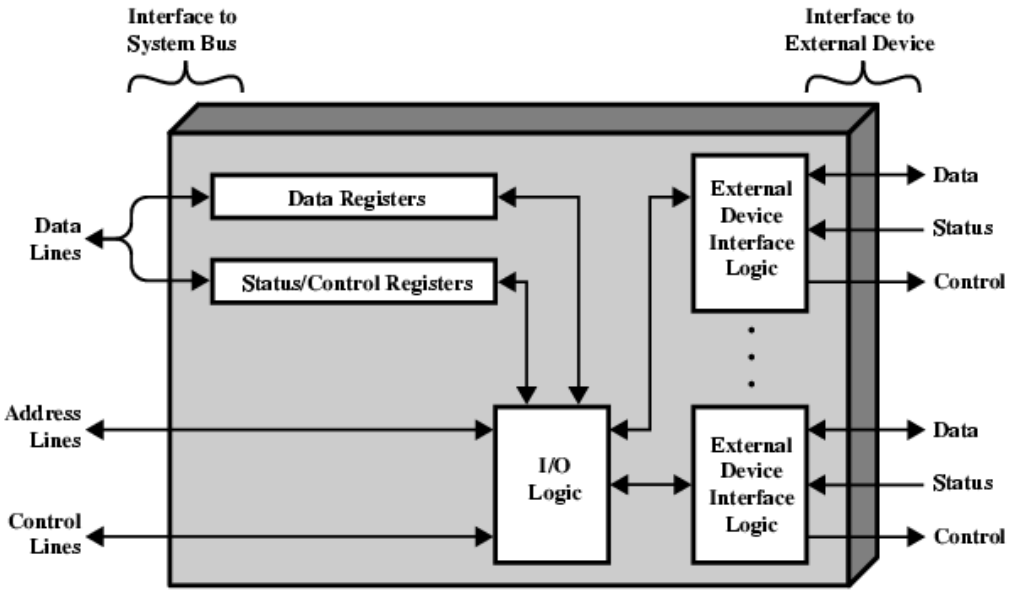
I/O Module Functions

- Control & Timing
- CPU Communication
- Device Communication
- Data Buffering
- Error Detection

I/O CPU Steps

- CPU checks I/O module device status
- I/O module returns status
- If ready, CPU requests data transfer
- I/O module gets data from device
- I/O module transfers data to CPU
- Variations for output, DMA, etc.

I/O Module Diagram & Design Decisions

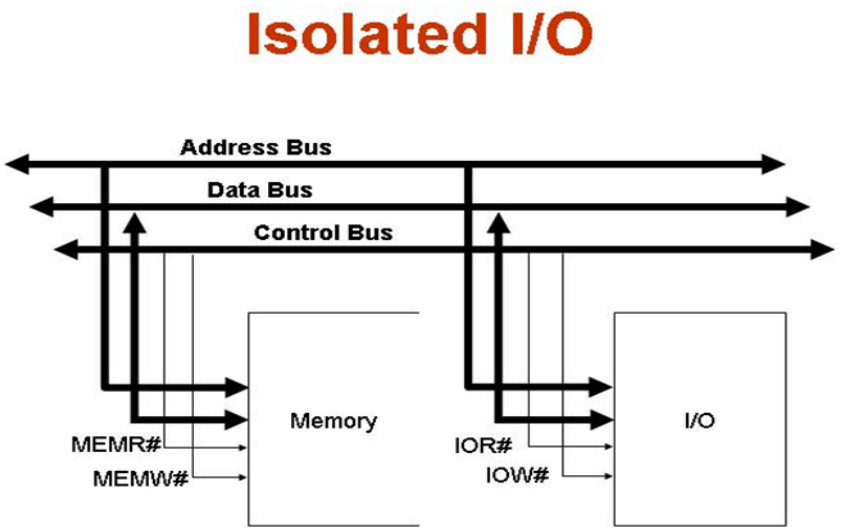


- Hide or reveal device properties to CPU
- Support multiple or single device
- Control device functions or leave for CPU
- Also O/S decisions
  - e.g. Unix treats everything it can as a file

I/O Mapping

- Memory mapped I/O
  - Devices and memory share an address space
  - I/O looks just like memory read/write
  - No special commands for I/O
    - Large selection of memory access commands available

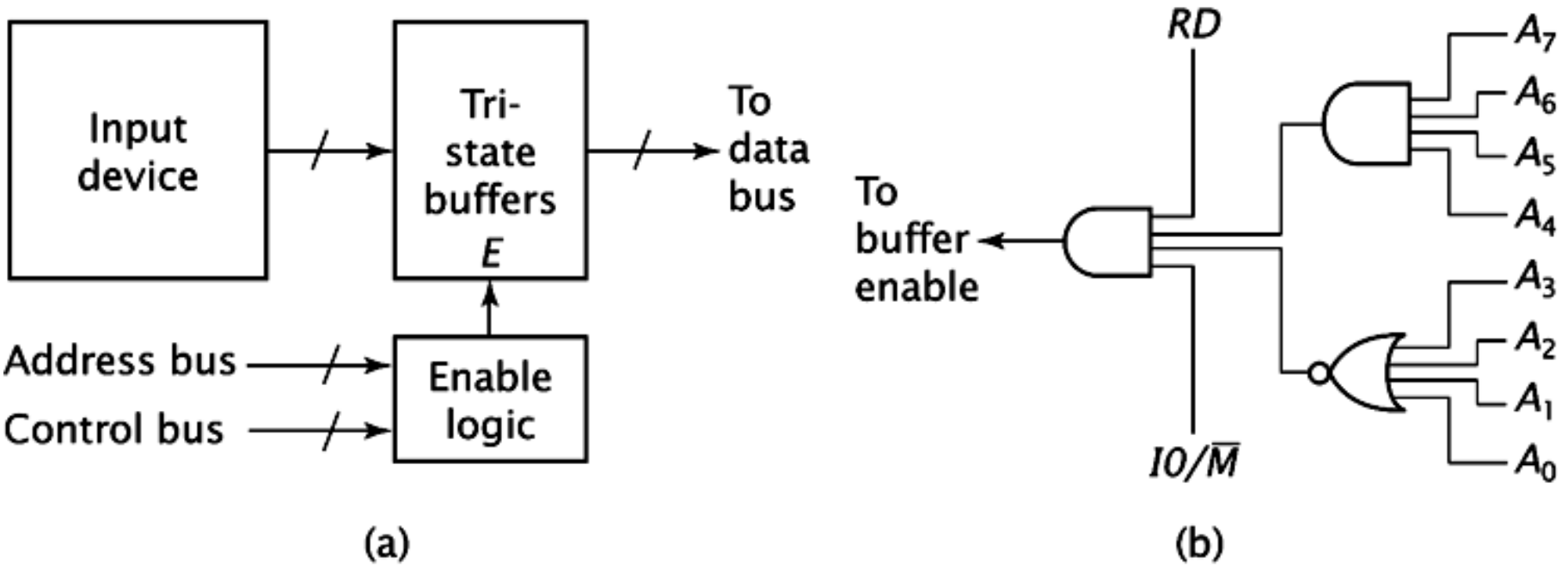
- Isolated I/O
  - Separate address spaces
  - Need I/O or memory select lines
  - Special commands for I/O
  - Special CPU control signals
  - Devices and Memory can have overlapping adresses



Addressing I/O Devices

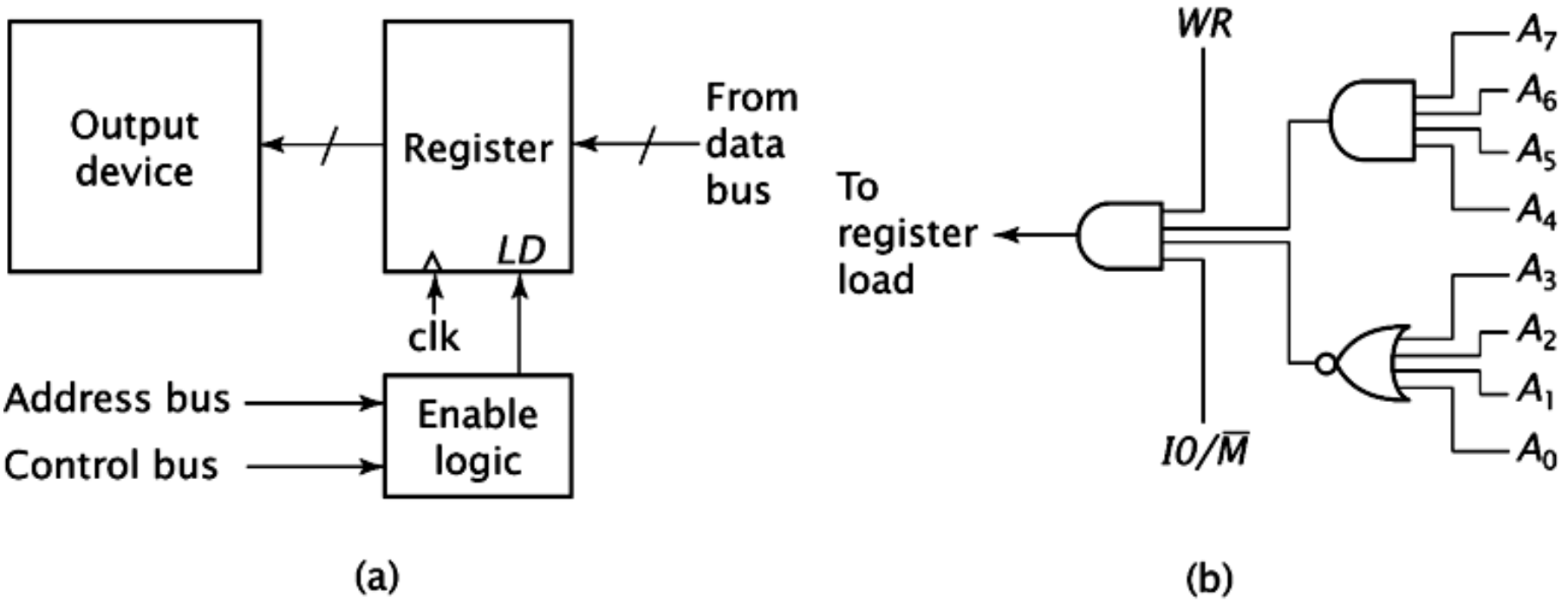
- I/O data transfer is very like memory access (CPU viewpoint)
- Each device given unique identifier
- CPU commands contain identifier (address)
- The IO Module should contain address decoding logic

Input Devices



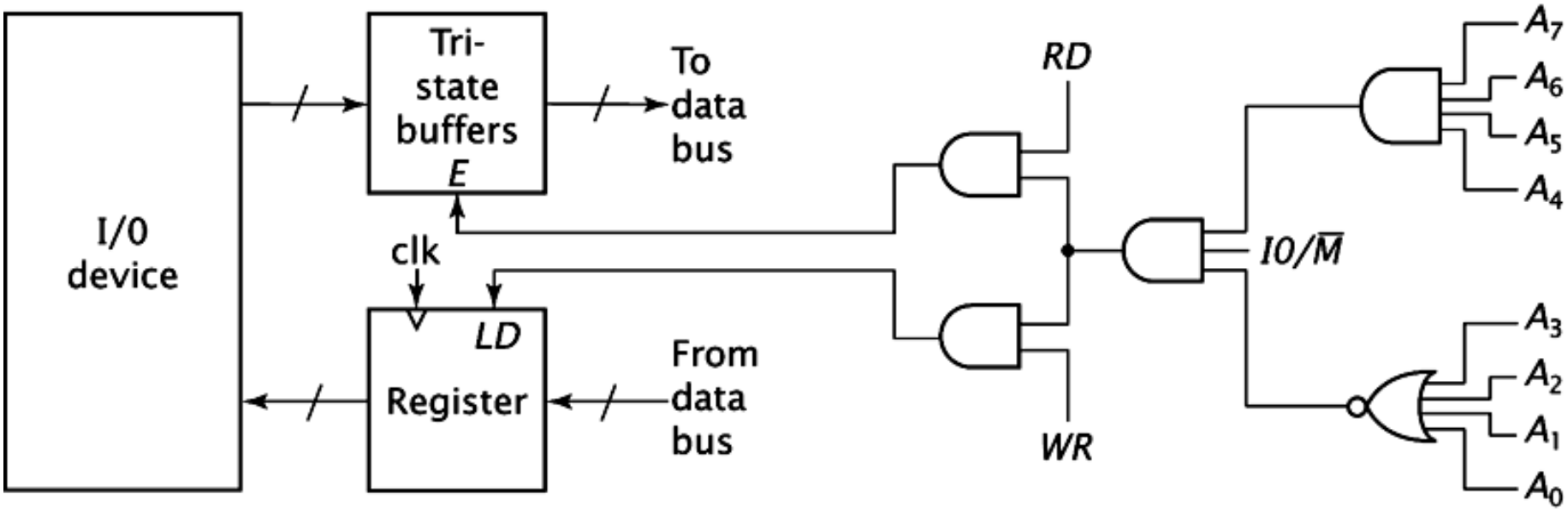
- When the values of the address/control buses are correct (the I/O device is addressed) the buffers are enabled and the data passes on to the data bus; the CPU reads this data
- When the conditions are not right, the logic bloc (enable logic) will not enable the buffers; no data on the data bus
- The example shows an I/O device mapped at address 1111 0000 on a computer with 8 bit address bus and RD and IO/M' control signals

Output Devices



- Since the output devices read data from the data bus, they don't need the buffers; data will be made available to all the devices
- Only the correctly decoded one (addressed) will read in the data
- Example shows an output device mapped at 11110000 address in a 8 bit address bus computer, with WR and IO/M' signals

Bidirectional Devices (1)

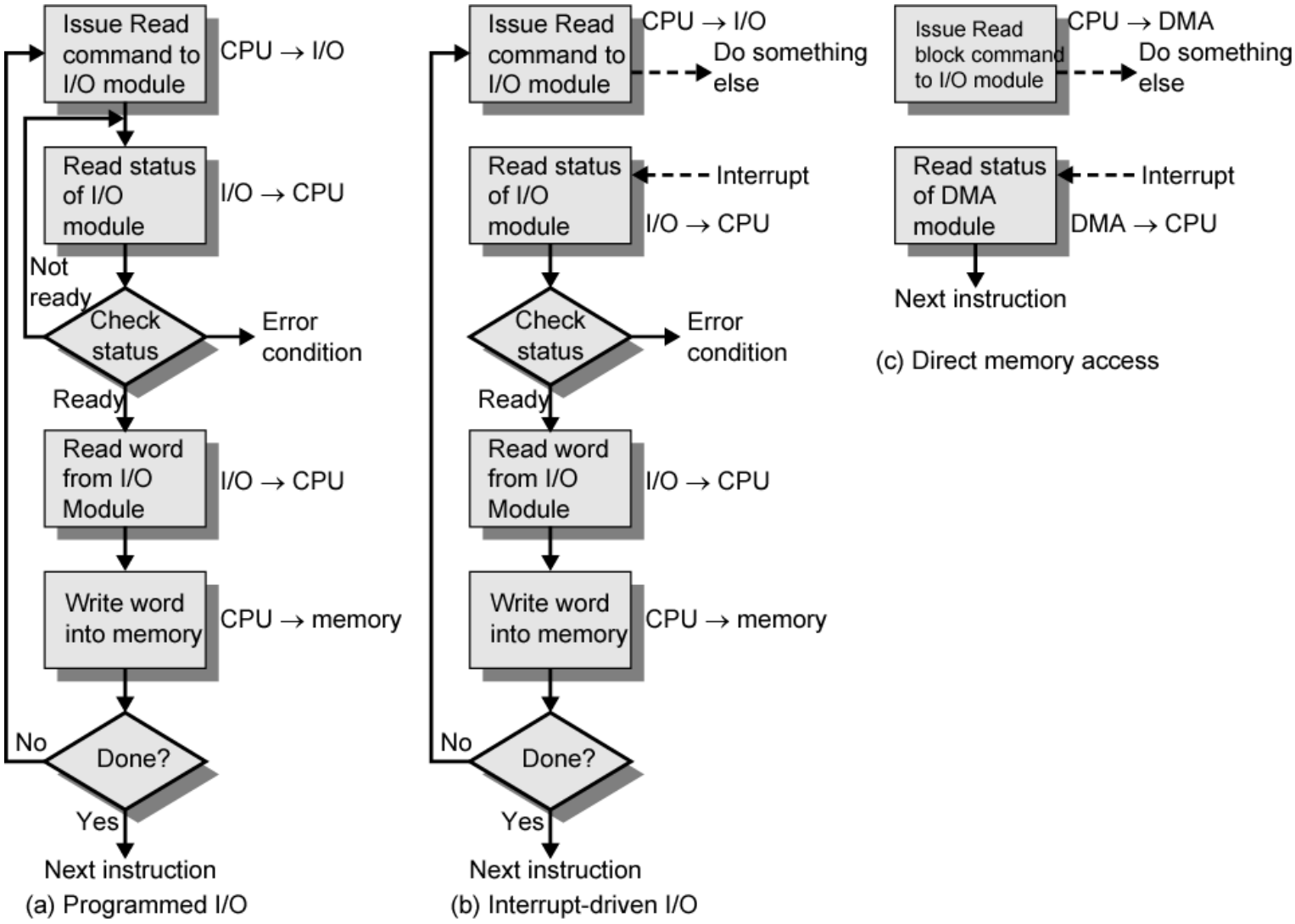


- Bidirectional devices require actually two interfaces, one for input and the other for output.
- Same gates could be used to generate the enable signal (for both the tri state buffers and the registers); the difference between read and write are made through the control signals (RD, WR)
- The example shows a combined interface for 1111 0000 address.

Bidirectional Devices (1)

- In real systems, we need to access more than just one output and one input data register
- Usually peripherals are issued with commands by the processor and they take some action and in response present data
- Up to how the processor knows if the peripheral device is ready after a command, we can have:
  - Programmed I/O (or also known as Polled I/O)
  - Interrupt driven I/O

Input / Output Techniques



Programmed I/O

Overview

- CPU has direct control over I/O
  - Sensing status
  - Read/write commands
  - Transferring data
- CPU waits for I/O module to complete operation
- Wastes CPU time

Operations

- CPU requests I/O operation
- I/O module performs operation
- I/O module sets status bits
- CPU checks status bits periodically
- I/O module does not inform CPU directly
- I/O module does not interrupt CPU
- CPU may wait or come back later

Interrupt Driven I/O

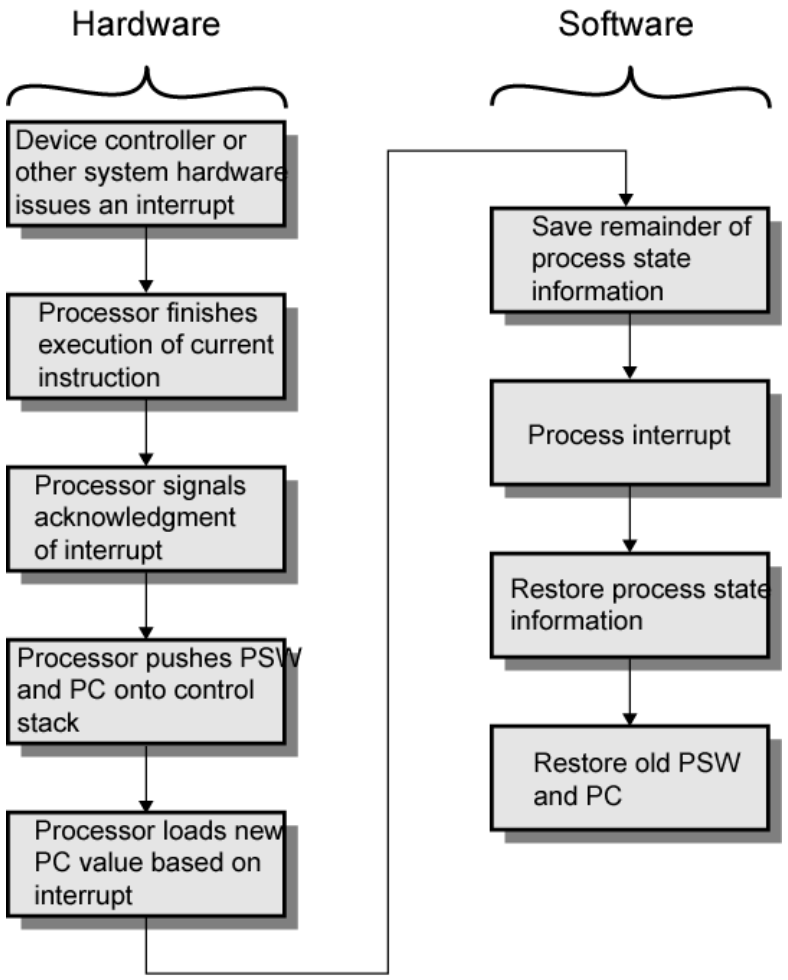
Overview

- Overcomes CPU waiting
- No repeated CPU checking of device
- I/O module interrupts when ready

Operations

- 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

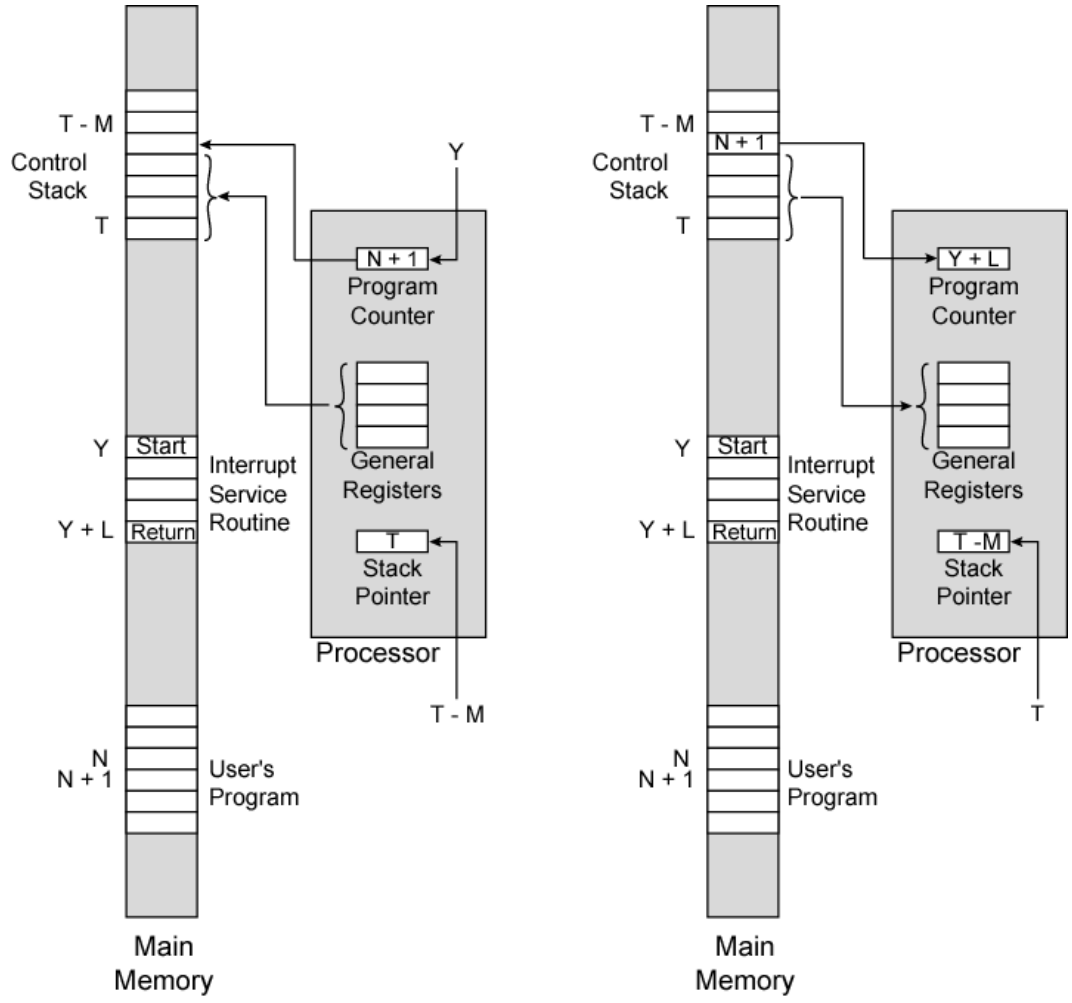
Simple Interrupt Processing



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
  - Restore context (registers)

Changes in Memory and Registers for an Interrupt



(a) Interrupt occurs after instruction at location N

(b) Return from interrupt

Design Issues

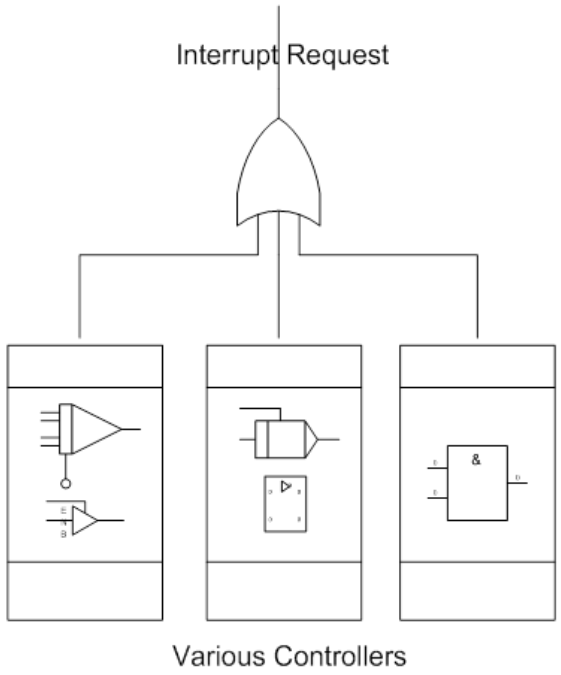
- How do you identify the module issuing the interrupt?
- How do you deal with multiple interrupts?
  - i.e. an interrupt handler being interrupted

Identifying Interrupting Module

- Different line for each module
  - Limits number of devices
- Software poll
  - CPU asks each module in turn
  - Slow
- Daisy Chain or Hardware poll
  - Interrupt Acknowledge sent down a chain
  - Module responsible places vector on bus
  - CPU uses vector to identify handler routine
- Bus Arbitration (e.g. PCI & SCSI)
  - Module must claim the bus before it can raise interrupt, thus only one module can rise the interrupt at a time
  - When processor detects interrupt, processor issues an interrupt acknowledge
  - Device places its vector on the data bus

Multiple Interrupts

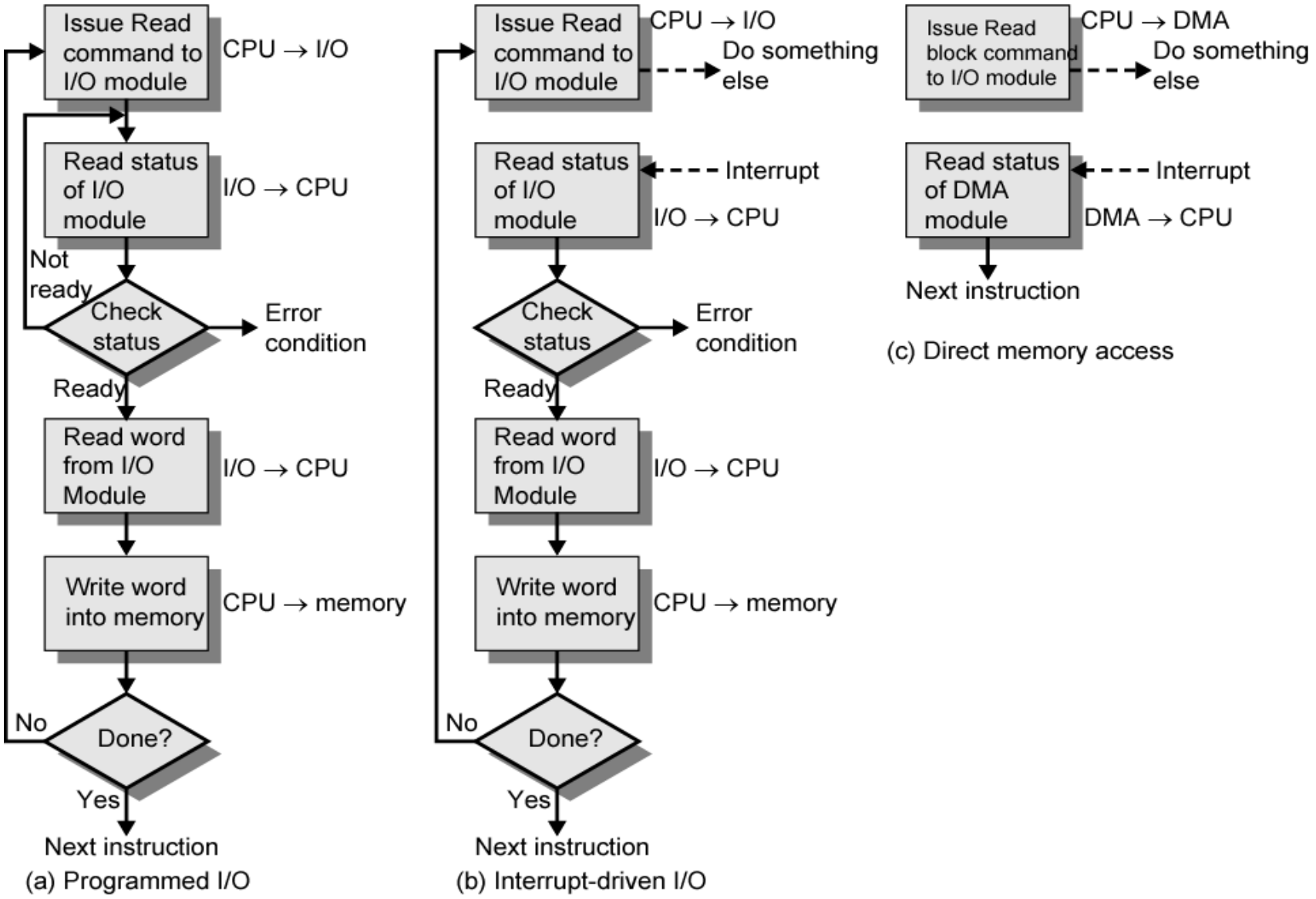
- Each interrupt line has a priority
- Higher priority lines can interrupt lower priority lines



Unified Interrupt Handling Example

```
Interrupt_Handler
    saveProcessorState();
    for (i=0; i<Number_of_devices; i++)
        if (device[i].done ==1) goto
      device_handler(i);
/* If here, then something went wrong*/
```

Input / Output Techniques

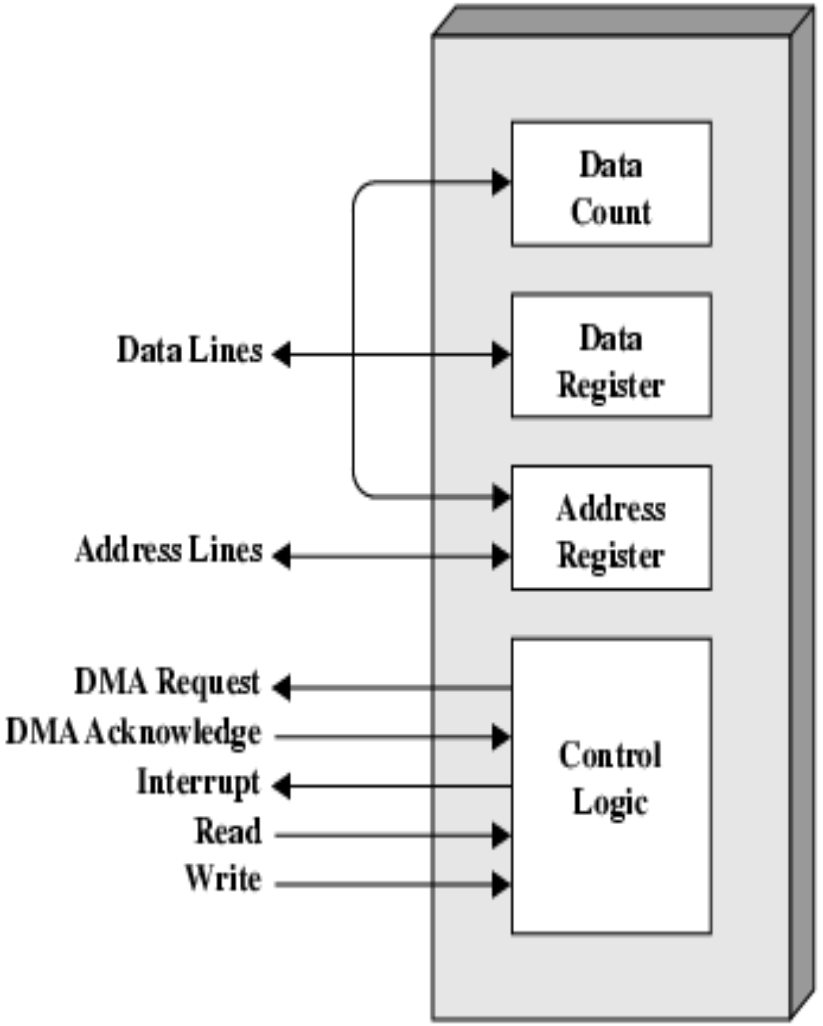


Direct Memory Access

- Interrupt driven and programmed I/O require active CPU intervention
  - Transfer rate is limited by the speed of processor testing and servicing a device
  - CPU is tied up in managing an I/O transfer. A number of instructions must be executed for each I/O transfer.

- DMA is the answer when large amounts of data need to be transferred.

DMA Function and Module



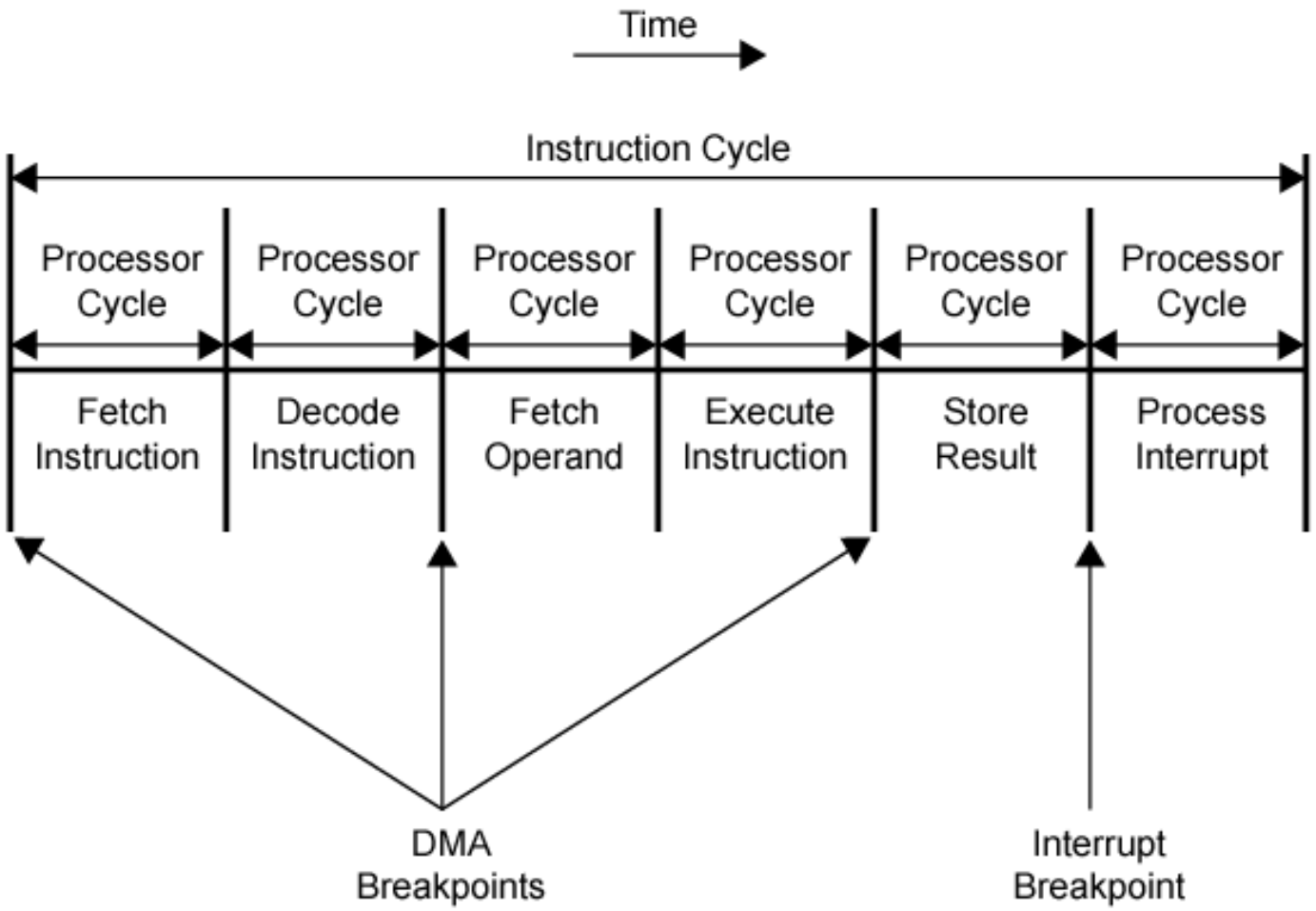
DMA controller able to mimic the CPU and take over for I/O transfers

- CPU tells DMA controller:
  - Operation to execute
  - Device address involved in the I/O operation (sent on data lines)
  - Starting address of memory block for data (sent on data lines) and stored in the DMA address register
  - Amount of data to be transferred (sent on data lines) and stored into the data count
- CPU carries on with other work
- DMA controller deals with transfer
- DMA controller sends interrupt when finished

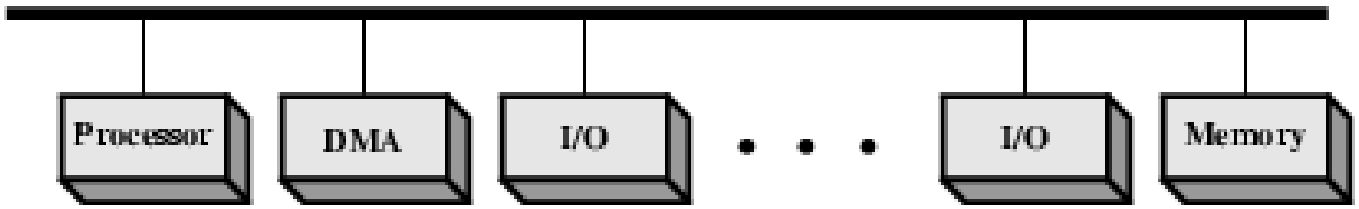
DMA Transfer Cycle Stealing

- DMA controller takes over bus for a cycle
- Transfer of one word of data
- Not an interrupt
  - CPU does not switch context
- CPU suspended just before it accesses bus
  - i.e. before an operand or data fetch or a data write
- Slows down CPU but not as much as CPU doing transfer

DMA and Interrupt Breakpoints During an Instruction Cycle

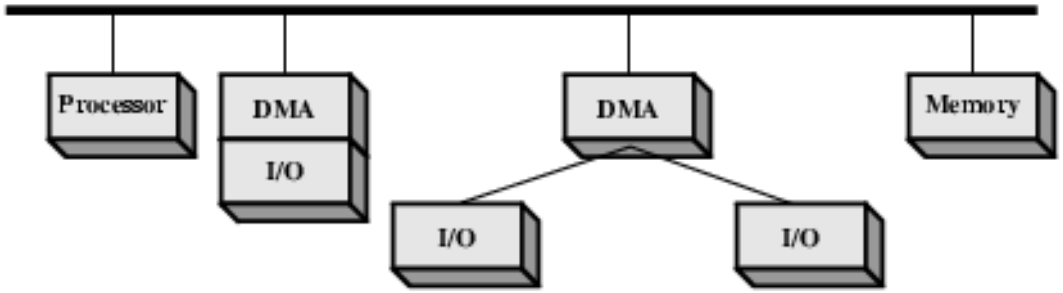


DMA Configurations (1)



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

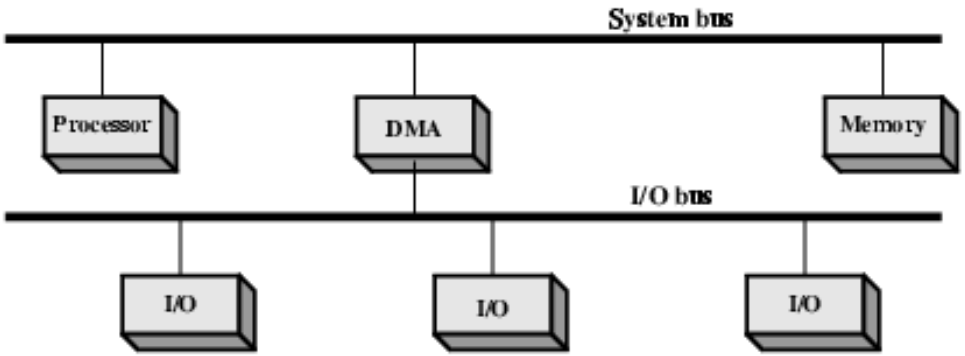
DMA Configurations (2)



(b) Single-bus, Integrated DMA-I/O

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

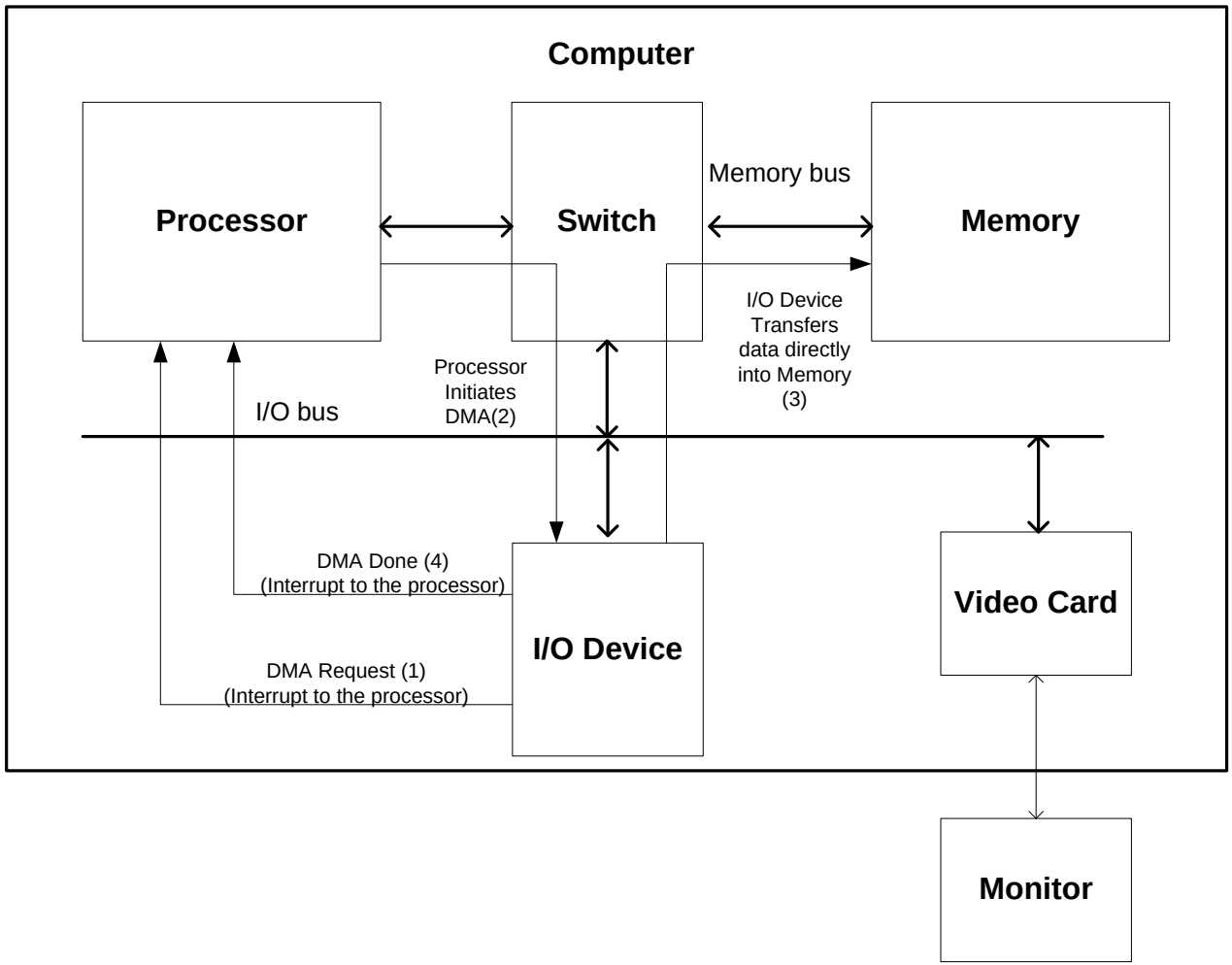
DMA Configurations (3)



(c) I/O bus

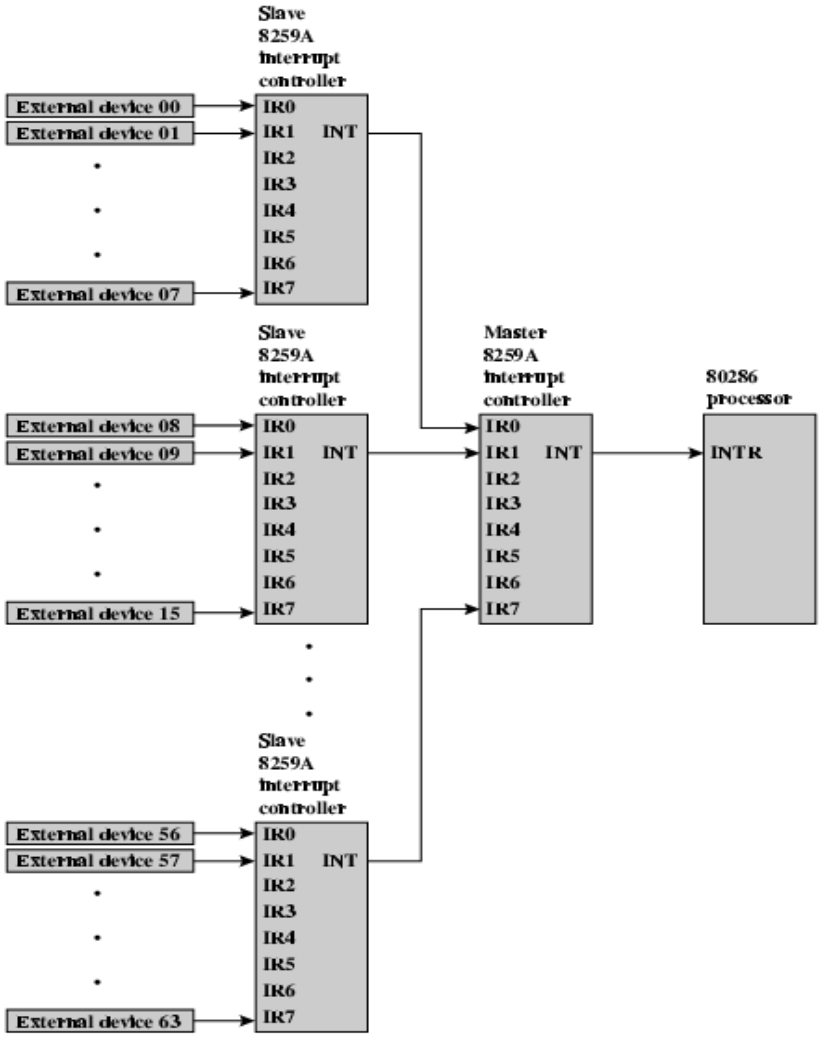
- Separate I/O Bus
- Bus supports all DMA enabled devices
- Each transfer uses bus once
  - DMA to memory
- CPU may be suspended once

DMA Operation Example



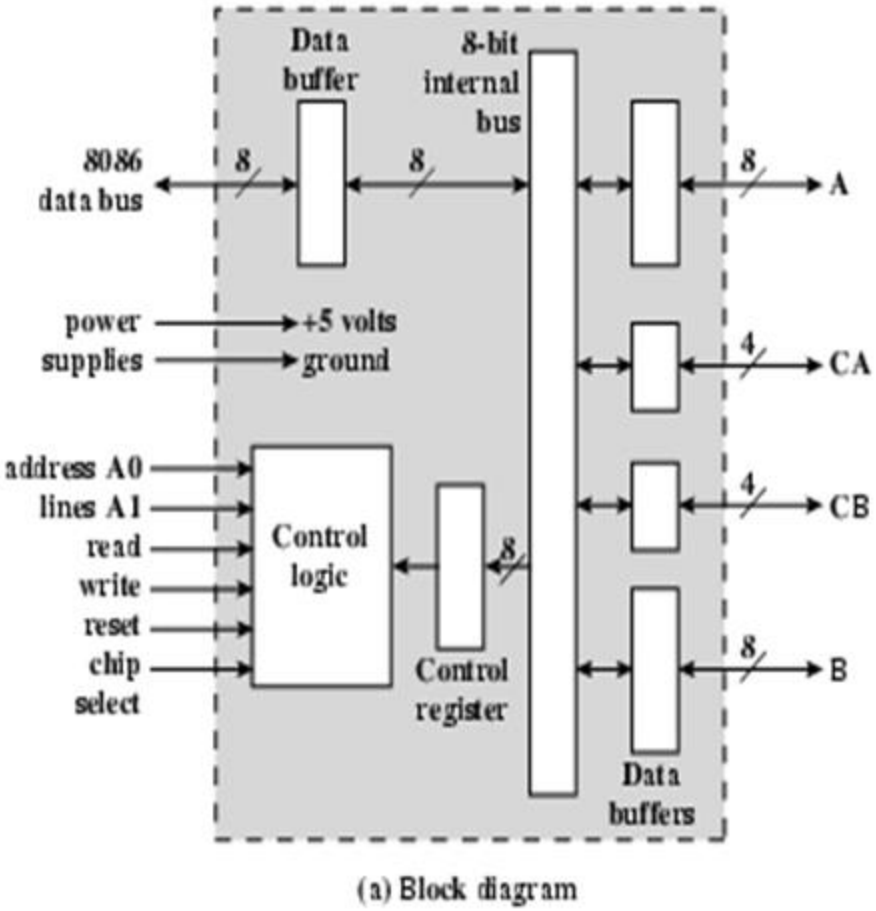
- Separate I/O Bus

82C59A Interrupt Controller



- Intel x86 processors have only one interrupt request line and one interrupt acknowledge line, thus they require an external interrupt controller
- 82C59A supports 8 sources of interrupt and can be configured as stand alone or in a master/slave configuration
- Sequence of events:
  - 82C59A accepts interrupt requests from modules, determines which has priority, signals the processor (INTR).
  - Processor acknowledges (INTA).
  - 82C59A will place vector information on data bus as a response to INTA.
  - Processor proceeds to handle the interrupt and communicates directly with the I/O module that generated the interrupt
- 82C59A is programmable by the processor. The processor decides what is the interrupt schema out of few possible:
  - Fully nested – interrupts are served according to priority from 0 (INT0) to 7 (INT7)
  - Rotating – after being serviced, a device is placed into the lowest priority in the group
  - Special masks – processor can inhibit certain priorities from certain devices

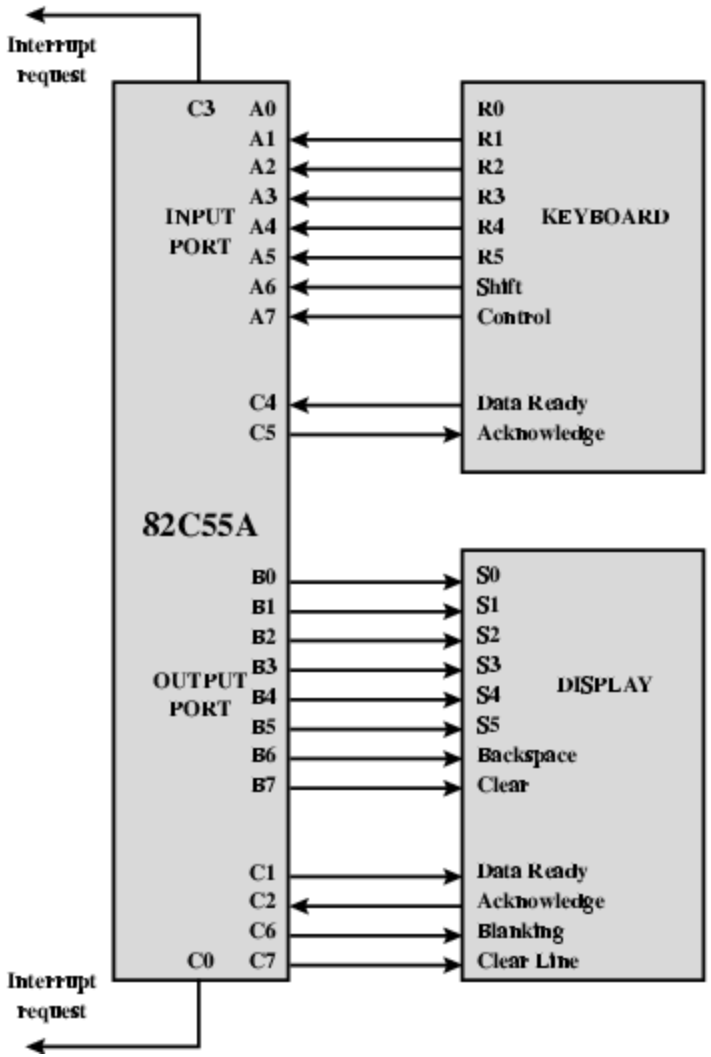
Intel 82C55A I/O Module



(a) Block diagram

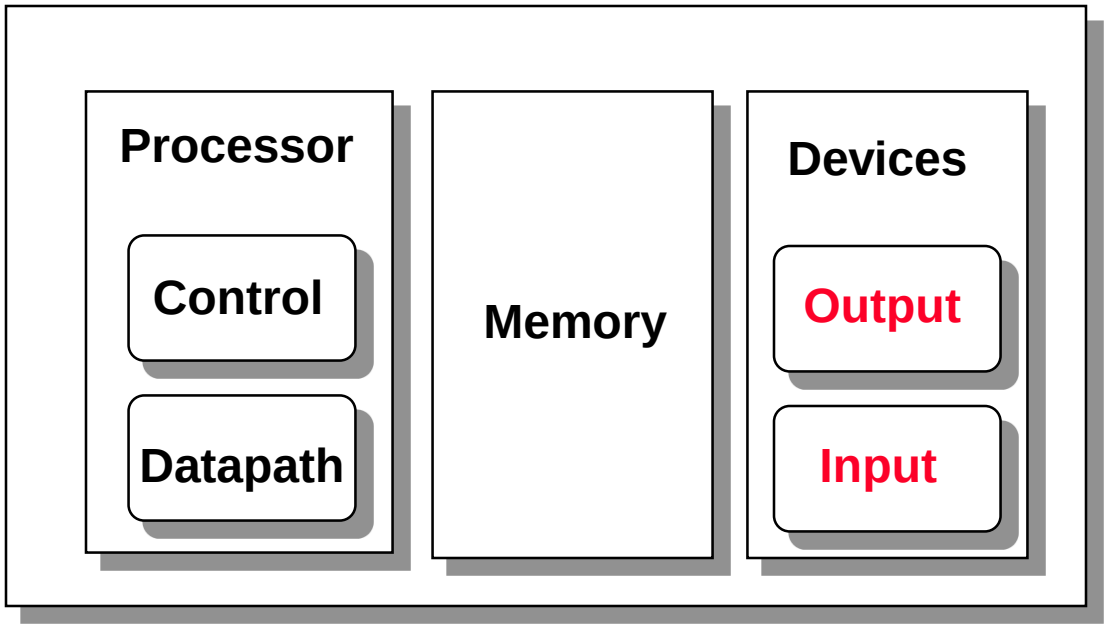
- 24 I/O programmable lines by means of control registers
  - 3 8 bit groups ABC
  - Group C is further divided into Ca and Cb subgroups that can be used in conjunction with A and B ports
- D0-D7 bidirectional data I/F with x86 processor
- A0,A1 specify one of the three I/O ports or control register for data transfer
- A transfer takes place only when CS is enabeld together with either Read or Write

Keyboard/Display Interfaces to 82C55A



- Group C signals are used for interrupt request and handshacking
  - Data Ready Line used to indicate that the data is reay on the I/O lines
  - Acknowledge is used to indicate to the device that it can reuse the I/O lines (clear and/or place new data on them)
  - Interrupt request tied to the system interrupt controller
- Note that two of the 8 bit inputs from the keyboard/display are special purpose pins. However, they will be treated as normal signals by the I/O module. They will only be interpreted by the Keyboard/Display driver.

Review: Major Components of a Computer

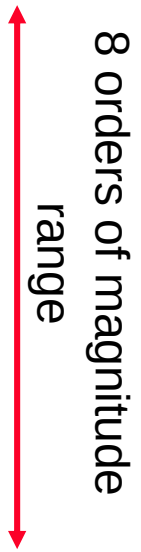


- Important metrics for an I/O system
  - Performance
  - Expandability
  - Dependability
  - Cost, size, weight

Input and Output Devices

- I/O devices are incredibly diverse with respect to
  - Behavior – input, output or storage
  - Partner – human or machine
  - Data rate – the peak rate at which data can be transferred between the I/O device and the main memory or processor

Device	Behavior	Partner	Data rate (Mb/s)
Keyboard	input	human	0.0001
Mouse	input	human	0.0038
Laser printer	output	human	3.2000
Graphics display	output	human	800.0000-8000.0000
Network/LAN	input or output	machine	100.0000-1000.0000
Magnetic disk	storage	machine	240.0000-2560.0000



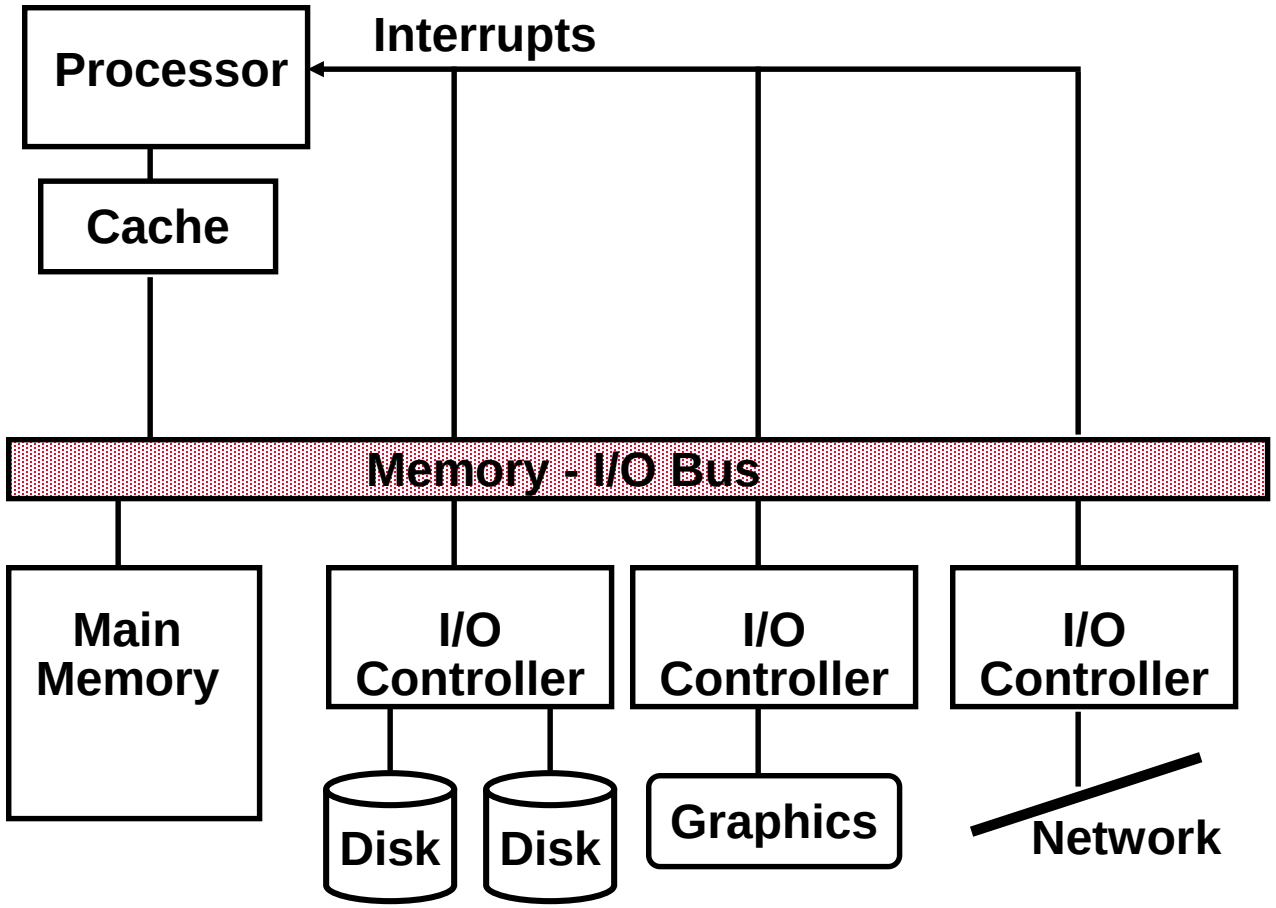
I/O Performance Measures

- I/O bandwidth (throughput) – amount of information that can be input (output) and communicated across an interconnect (e.g., a bus) to the processor/memory (I/O device) per unit time
  1. How much data can we move through the system in a certain time?
  2. How many I/O operations can we do per unit time?

- I/O response time (latency) – the total elapsed time to accomplish an input or output operation
  - An especially important performance metric in real-time systems

- Many applications require both high throughput and short response times

A Typical I/O System



I/O System Performance

- Designing an I/O system to meet a set of bandwidth and/or latency constraints means

1. Finding the weakest link in the I/O system – the component that constrains the design
   - The processor and memory system ?
   - The underlying interconnection (e.g., bus) ?
   - The I/O controllers ?
   - The I/O devices themselves ?
2. (Re)configuring the weakest link to meet the bandwidth and/or latency requirements
3. Determining requirements for the rest of the components and (re)configuring them to support this latency and/or bandwidth

I/O System Performance Example

- A disk workload consisting of 64KB reads and writes where the user program executes 200,000 instructions per disk I/O operation and
  - a processor that sustains 3 billion instr/s and averages 100,000 OS instructions to handle a disk I/O operation

The maximum disk I/O rate (# I/O's/s) of the processor is

$$\frac{\text{Instr execution rate}}{\text{Instr per I/O}} = \frac{3 \times 10^9}{(200 + 100) \times 10^3} = 10{,}000 \text{ I/O's/s}$$

  - a memory-I/O bus that sustains a transfer rate of 1000 MB/s

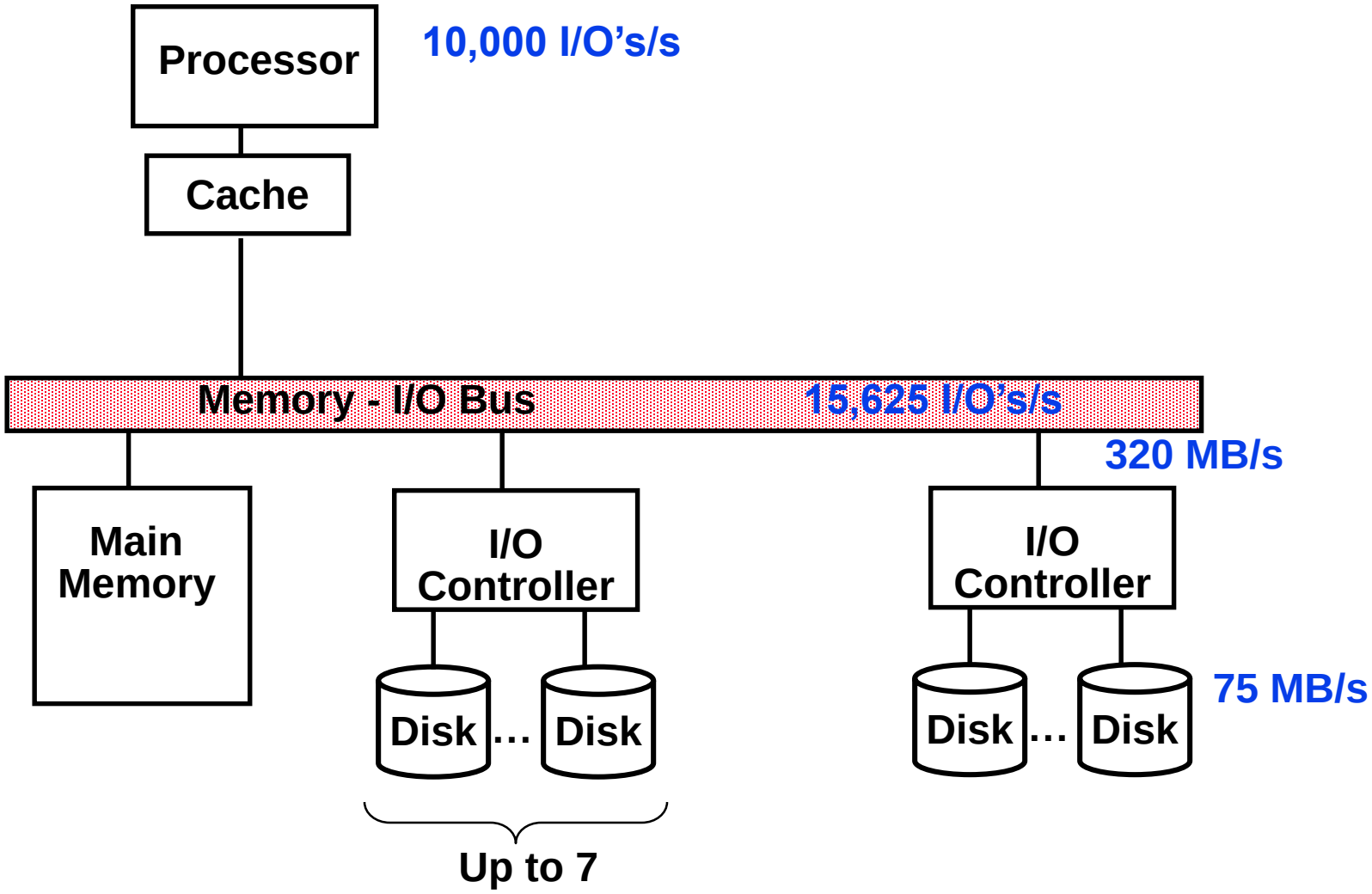
Each disk I/O reads/writes 64 KB so the maximum I/O rate of the bus is

$$\frac{\text{Bus bandwidth}}{\text{Bytes per I/O}} = \frac{1000 \times 10^6}{64 \times 10^3} = 15{,}625 \text{ I/O's/s}$$

  - SCSI disk I/O controllers with a DMA transfer rate of 320 MB/s that can accommodate up to 7 disks per controller
  - disk drives with a read/write bandwidth of 75 MB/s and an average seek plus rotational latency of 6 ms

what is the maximum sustainable I/O rate and what is the number of disks and SCSI controllers required to achieve that rate?

Disk I/O System Example



I/O System Performance Example, Con't

So the processor is the bottleneck, not the bus

- disk drives with a read/write bandwidth of 75 MB/s and an average seek plus rotational latency of 6 ms

Disk I/O read/write time = seek + rotational time + transfer time =
6ms + 64KB/(75MB/s) = 6.9ms

Thus each disk can complete 1000ms/6.9ms or 146 I/O's per second. To saturate the processor requires 10,000 I/O's per second or
10,000/146 = 69 disks

To calculate the number of SCSI disk controllers, we need to know the average transfer rate per disk to ensure we can put the maximum of 7 disks per SCSI controller and that a disk controller won't saturate the memory-I/O bus during a DMA transfer

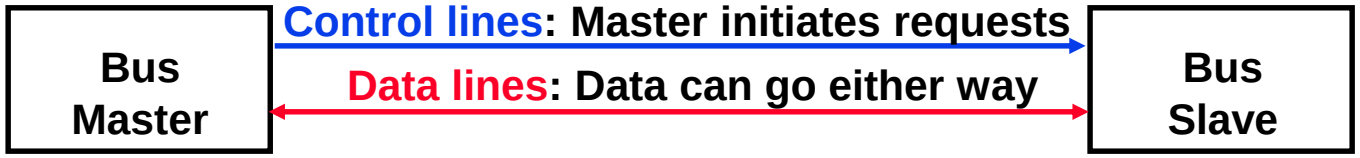
Disk transfer rate = (transfer size)/(transfer time) = 64KB/6.9ms = 9.56 MB/s

Thus 7 disks won't saturate either the SCSI controller (with a maximum transfer rate of 320 MB/s) or the memory-I/O bus (1000 MB/s). This means we will need 69/7 or 10 SCSI controllers.

I/O System Interconnect Issues

- A bus is a shared communication link (a single set of wires used to connect multiple subsystems) that needs to support a range of devices with widely varying latencies and data transfer rates
  - Advantages
    - Versatile – new devices can be added easily and can be moved between computer systems that use the same bus standard
    - Low cost – a single set of wires is shared in multiple ways
  - Disadvantages
    - Creates a communication bottleneck – bus bandwidth limits the maximum I/O throughput
- The maximum bus speed is largely limited by
  - The length of the bus
  - The number of devices on the bus

Bus Characteristics



- Control lines
  - Signal requests and acknowledgments
  - Indicate what type of information is on the data lines
- Data lines
  - Data, addresses, and complex commands
- Bus transaction consists of
  - Master issuing the command (and address) – request
  - Slave receiving (or sending) the data – action
  - Defined by what the transaction does to memory
    - Input – inputs data from the I/O device to the memory
    - Output – outputs data from the memory to the I/O device

Types of Buses

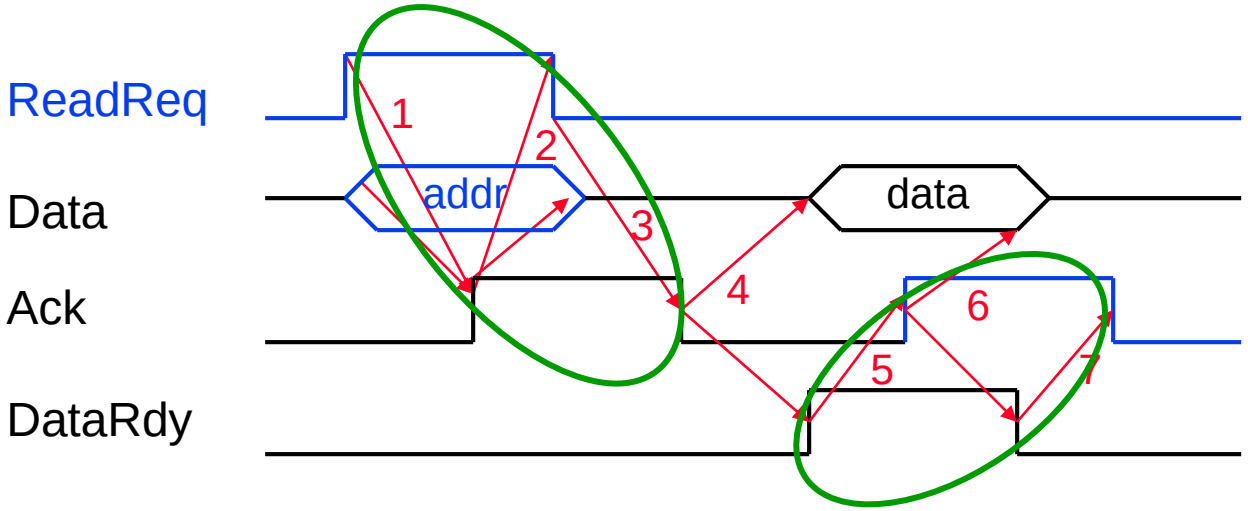
- Processor-memory bus (proprietary)
  - Short and high speed
  - Matched to the memory system to maximize the memory-processor bandwidth
  - Optimized for cache block transfers
- I/O bus (industry standard, e.g., SCSI, USB, Firewire)
  - Usually is lengthy and slower
  - Needs to accommodate a wide range of I/O devices
  - Connects to the processor-memory bus or backplane bus
- Backplane bus (industry standard, e.g., ATA, PCIexpress)
  - The backplane is an interconnection structure within the chassis
  - Used as an intermediary bus connecting I/O busses to the processor-memory bus

Synchronous and Asynchronous Buses

- Synchronous bus (e.g., processor-memory buses)
  - Includes a clock in the control lines and has a fixed protocol for communication that is relative to the clock
  - Advantage: involves very little logic and can run very fast
  - Disadvantages:
    - Every device communicating on the bus must use same clock rate
    - To avoid clock skew, they cannot be long if they are fast
- Asynchronous bus (e.g., I/O buses)
  - It is not clocked, so requires a handshaking protocol and additional control lines (ReadReq, Ack, DataRdy)
  - Advantages:
    - Can accommodate a wide range of devices and device speeds
    - Can be lengthened without worrying about clock skew or synchronization problems
  - Disadvantage: slow(er)

Asynchronous Bus Handshaking Protocol

- Output (read) data from memory to an I/O device



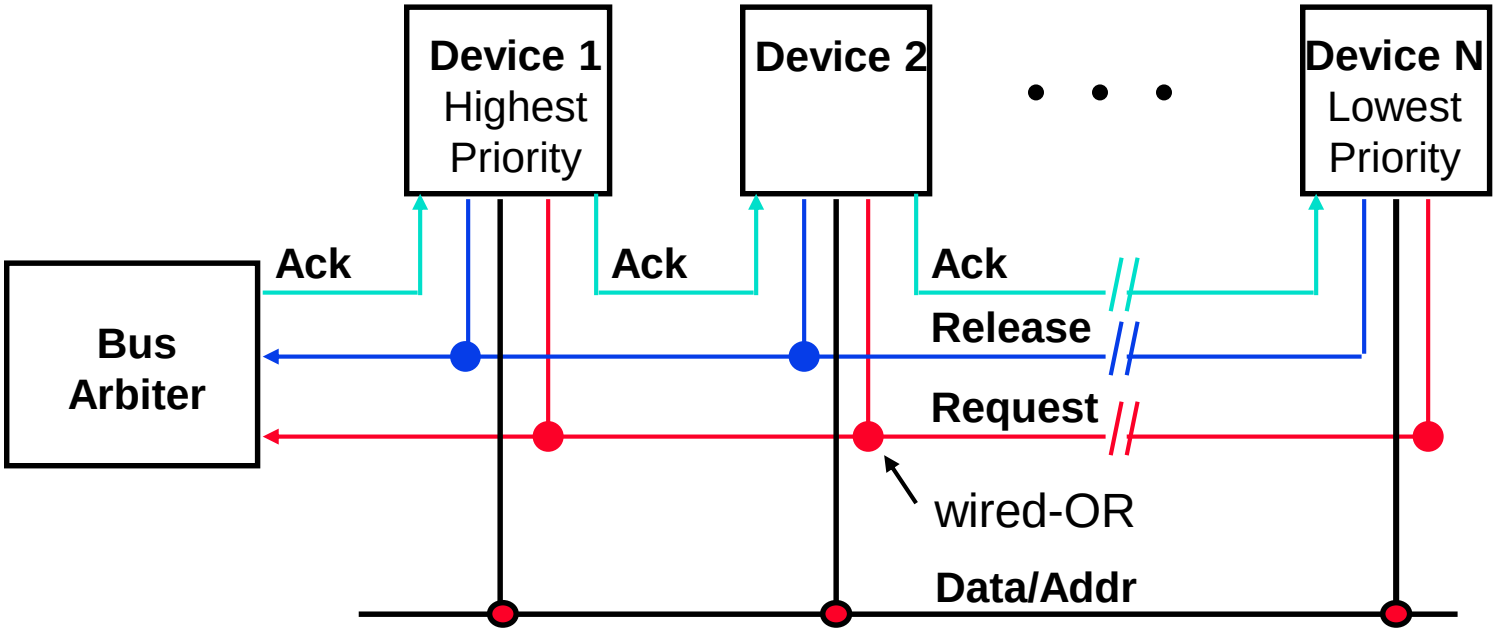
I/O device signals a request by raising ReadReq and putting the addr on the data lines

1. Memory sees ReadReq, reads addr from data lines, and raises Ack
2. I/O device sees Ack and releases the ReadReq and data lines
3. Memory sees ReadReq go low and drops Ack
4. When memory has data ready, it places it on data lines and raises DataRdy
5. I/O device sees DataRdy, reads the data from data lines, and raises Ack
6. Memory sees Ack, releases the data lines, and drops DataRdy
7. I/O device sees DataRdy go low and drops Ack

The Need for Bus Arbitration

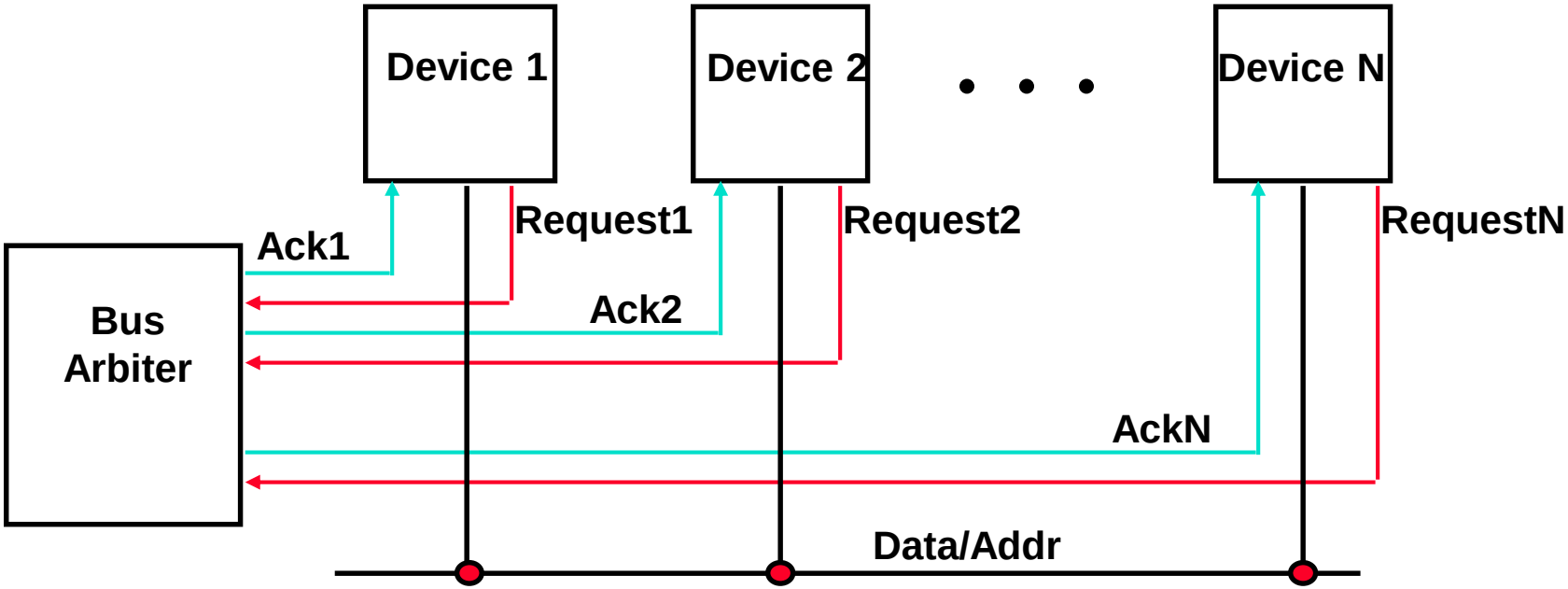
- Multiple devices may need to use the bus at the same time so must have a way to arbitrate multiple requests
- Bus arbitration schemes usually try to balance:
  - Bus priority – the highest priority device should be serviced first
  - Fairness – even the lowest priority device should never be completely locked out from the bus
- Bus arbitration schemes can be divided into four classes
  - Daisy chain arbitration – see next slide
  - Centralized, parallel arbitration – see next-next slide
  - Distributed arbitration by self-selection – each device wanting the bus places a code indicating its identity on the bus
  - Distributed arbitration by collision detection – device uses the bus when its not busy and if a collision happens (because some other device also decides to use the bus) then the device tries again later (Ethernet)

Daisy Chain Bus Arbitration



- Advantage: simple
- Disadvantages:
  - Cannot assure fairness – a low-priority device may be locked out indefinitely
  - Slower – the daisy chain grant signal limits the bus speed

Centralized Parallel Arbitration



- Advantages: flexible, can assure fairness
- Disadvantages: more complicated arbiter hardware
- Used in essentially all processor-memory buses and in high-speed I/O buses

Bus Bandwidth Determinates

- The bandwidth of a bus is determined by
  - Whether its is synchronous or asynchronous and the timing characteristics of the protocol used
  - The data bus width
  - Whether the bus supports block transfers or only word at a time transfers

	Firewire	USB 2.0
Type	I/O	I/O
Data lines	4	2
Clocking	Asynchronous	Synchronous
Max # devices	63	127
Max length	4.5 meters	5 meters
Peak bandwidth	50 MB/s (400 Mbps) 100 MB/s (800 Mbps)	0.2 MB/s (low) 1.5 MB/s (full) 60 MB/s (high)

Buses in Transition

- Companies are transitioning from synchronous, parallel, *wide* buses to asynchronous *narrow* buses
  - Reflection on wires and clock skew makes it difficult to use 16 to 64 parallel wires running at a high clock rate (e.g., ~400 MHz) so companies are transitioning to buses with a few one-way wires running at a very high “clock” rate (~2 GHz)

	PCI	PCIexpress	ATA	Serial ATA
Total # wires	120	36	80	7
# data wires	32 – 64 (2-way)	2 x 4 (1-way)	16 (2-way)	2 x 2 (1-way)
Clock (MHz)	33 – 133	635	50	150
Peak BW (MB/s)	128 – 1064	300	100	375 (3 Gbps)

Communication of I/O Devices and Processor

- How the processor directs the I/O devices
  - Special I/O instructions
    - Must specify both the device and the command
  - Memory-mapped I/O
    - Portions of the high-order memory address space are assigned to each I/O device
    - Read and writes to those memory addresses are interpreted as commands to the I/O devices
    - Load/stores to the I/O address space can only be done by the OS
- How the I/O device communicates with the processor
  - Polling – the processor periodically checks the status of an I/O device to determine its need for service
    - Processor is totally in control – but does all the work
    - Can waste a lot of processor time due to speed differences
  - Interrupt-driven I/O – the I/O device issues an interrupts to the processor to indicate that it needs attention

Interrupt-Driven I/O

- An I/O interrupt is asynchronous wrt instruction execution
  - Is not associated with any instruction so doesn't prevent any instruction from completing
    - You can pick your own convenient point to handle the interrupt
- With I/O interrupts
  - Need a way to identify the device generating the interrupt
  - Can have different urgencies (so may need to be prioritized)
- Advantages of using interrupts
  - Relieves the processor from having to continuously poll for an I/O event; user program progress is only suspended during the actual transfer of I/O data to/from user memory space
- Disadvantage – special hardware is needed to
  - Cause an interrupt (I/O device) and detect an interrupt and save the necessary information to resume normal processing after servicing the interrupt (processor)

Direct Memory Access (DMA)

- For high-bandwidth devices (like disks) interrupt-driven I/O would consume a *lot* of processor cycles
- DMA – the I/O controller has the ability to transfer data directly to/from the memory without involving the processor
  1. The processor initiates the DMA transfer by supplying the I/O device address, the operation to be performed, the memory address destination/source, the number of bytes to transfer
  2. The I/O DMA controller manages the entire transfer (possibly thousand of bytes in length), arbitrating for the bus
  3. When the DMA transfer is complete, the I/O controller interrupts the processor to let it know that the transfer is complete
- There may be multiple DMA devices in one system
  - Processor and I/O controllers contend for bus cycles and for memory

The DMA Stale Data Problem

- In systems with caches, there can be two copies of a data item, one in the cache and one in the main memory
  - For a DMA read (from disk to memory) – the processor will be using stale data if that location is also in the cache
  - For a DMA write (from memory to disk) and a write-back cache – the I/O device will receive stale data if the data is in the cache and has not yet been written back to the memory
- The coherency problem is solved by
  1. Routing all I/O activity through the cache – expensive and a large negative performance impact
  2. Having the OS selectively invalidate the cache for an I/O read or force write-backs for an I/O write (flushing)
  3. Providing hardware to selectively invalidate or flush the cache – need a hardware snooper (details upcoming)

I/O and the Operating System

- The operating system acts as the interface between the I/O hardware and the program requesting I/O
  - To protect the shared I/O resources, the user program is not allowed to communicate directly with the I/O device
- Thus OS must be able to give commands to I/O devices, handle interrupts generated by I/O devices, provide equitable access to the shared I/O resources, and schedule I/O requests to enhance system throughput
  - I/O interrupts result in a transfer of processor control to the supervisor (OS) process