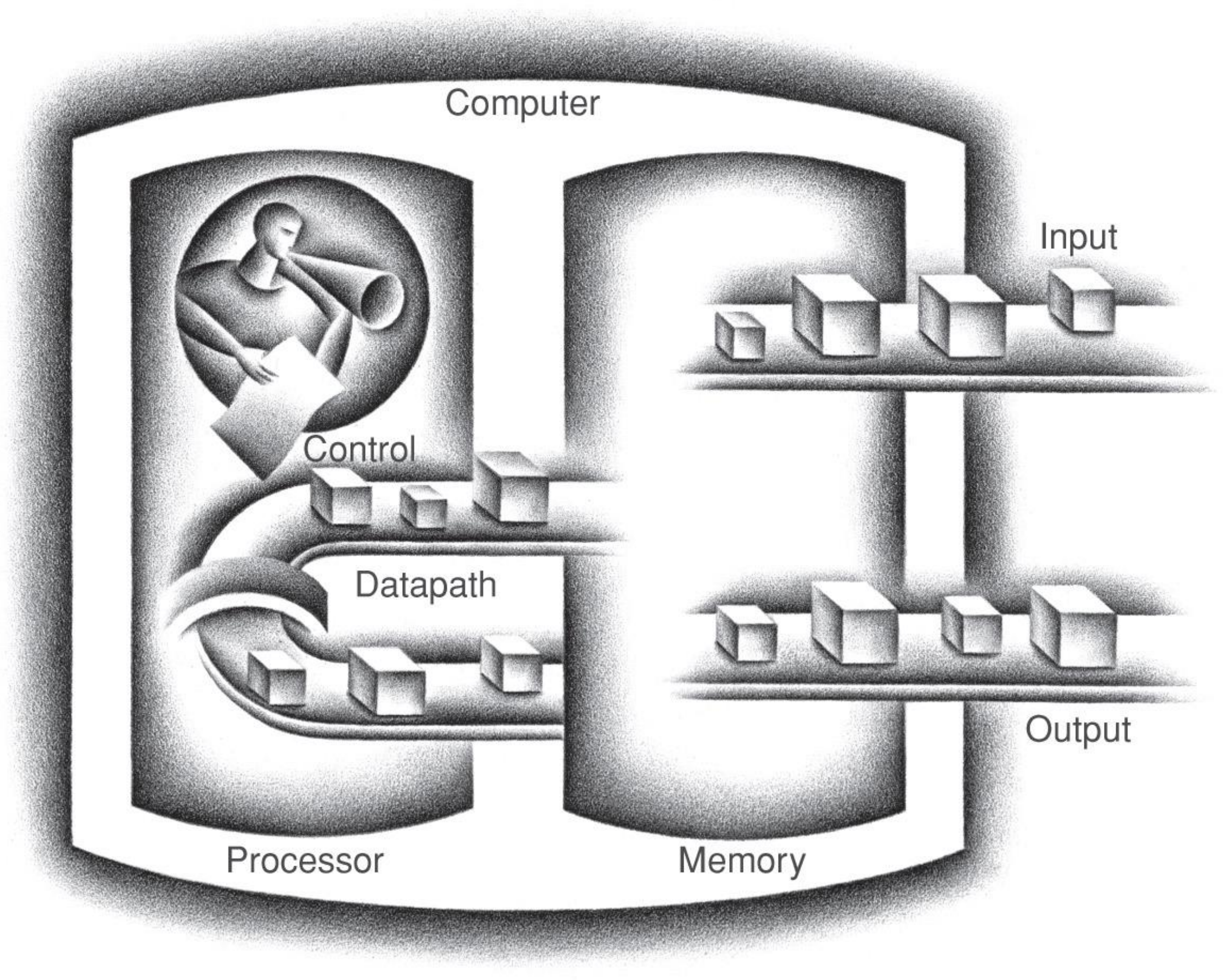
Chapter 6: Storage and I/O System

Ngo Lam Trung

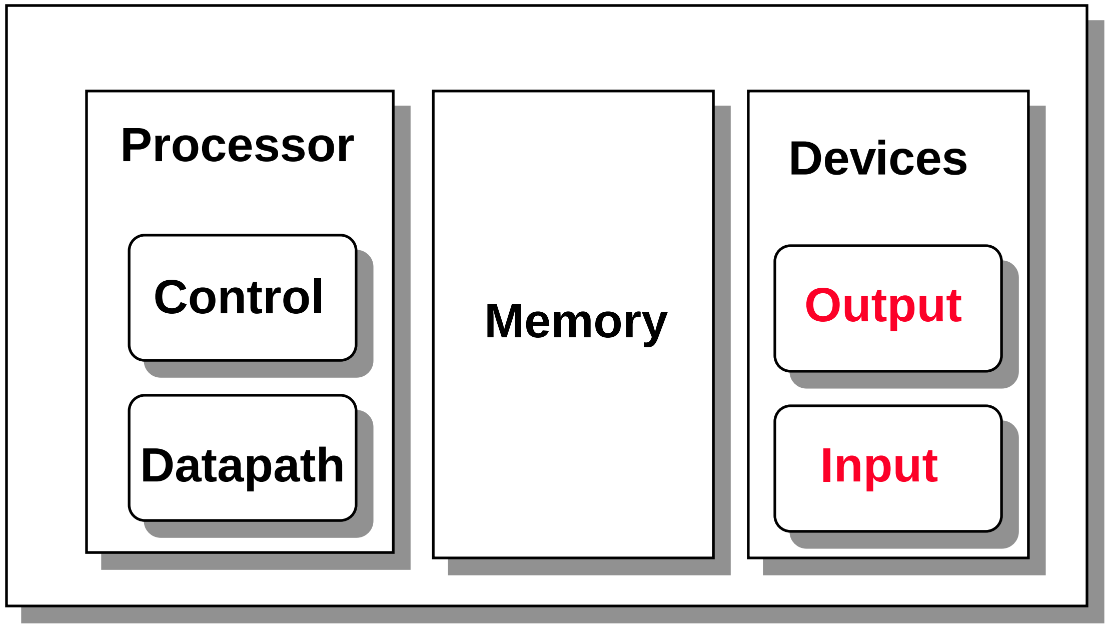
[with materials from *Computer Organization and Design, 4th Edition*, Patterson & Hennessy, © 2008, MK
and M.J. Irwin's presentation, PSU 2008]

Computer Organization

- Chapter 1-5: from processor to memory



Review: Major Components of a Computer



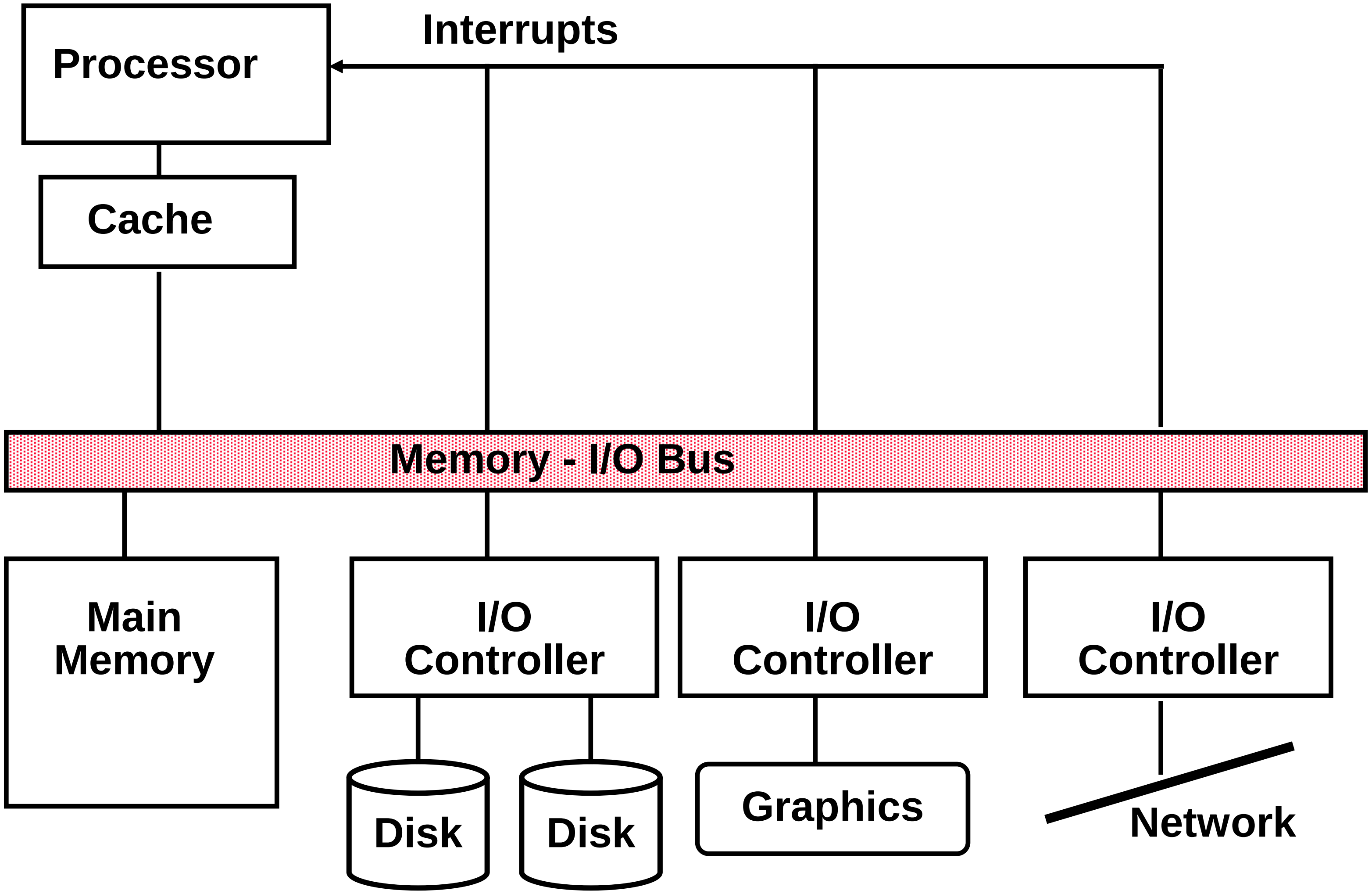
- Input + Output = I/O system
  - Hard disk
  - Network
  - USB drive

Anything else?

Important metrics

- For processor and memory: performance and cost
- For I/O system: what are the most important?
  - Performance
  - Expandability
  - Dependability
  - Cost, size, weight
  - Security
  - …

A Typical I/O System



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
Magnetic disk	storage	machine	800.0000-3000.0000
Graphics display	output	human	800.0000-8000.0000
Network/LAN	input or output	machine	100.0000-10000.0000

I/O Performance Measures

- I/O bandwidth (throughput) – amount of information that can be input/output and communicated across an interconnect between the processor/memory and 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

- Many applications require *both* high throughput and short response times

Failure

- Hardware operates in two states:
  1. *Service accomplishment*, the service is delivered as specified.
  2. *Service interruption*, the delivered service is different from the specified service.
- Changes from (1) to (2): failures
- Changes from (2) to (1): restorations
- Permanent failure: service is stopped permanently
- Intermittent failure: system oscillates between the two states → difficult to diagnose

Dependability: Reliability and Availability

- Mean Time To Failure (MTTF): average time of normal operation between two consecutive failure
- Mean Time To Repair (MTTR): average time of service interruption when failure occurs
- Reliability: measured by MTTF
- Availability:

$$\text{Availability} = \frac{\text{MTTF}}{(\text{MTTF} + \text{MTTR})}$$

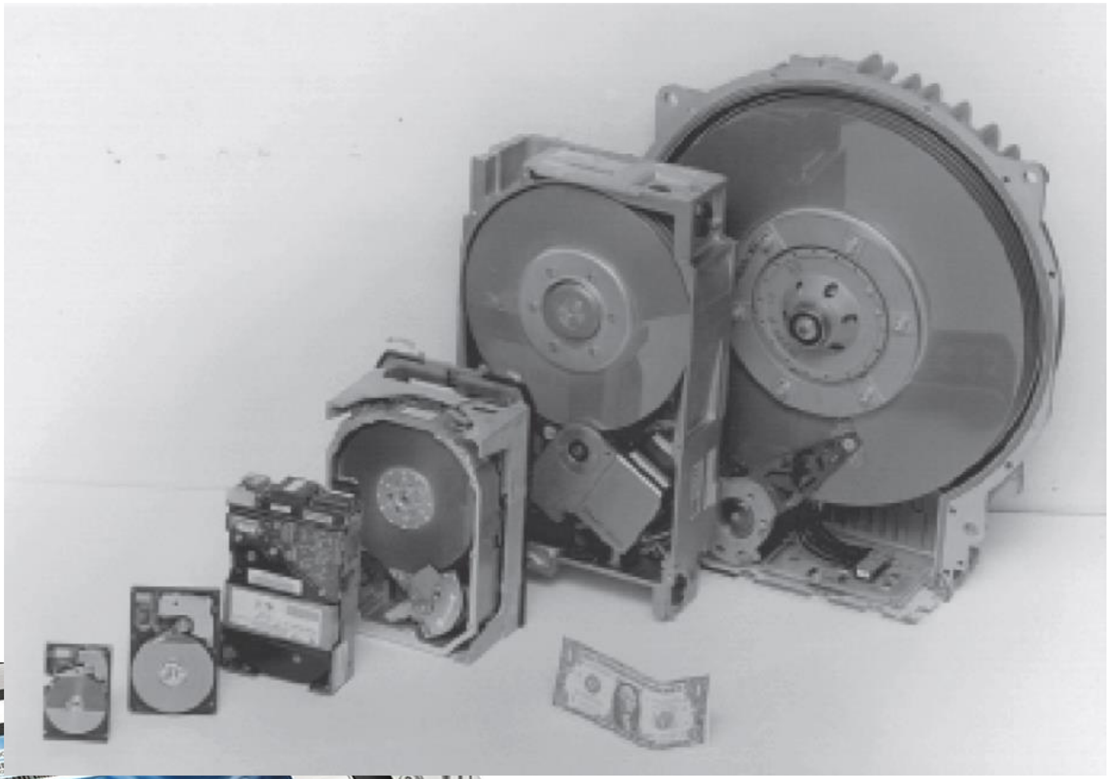
- Example MTTF:
  - Seagate ST33000655SS: 1,400,000 hours @25˚C
  - Samsung 860 EVO SSD: 1,500,000 hours

Improving MTTF and availability

- Fault avoidance: making better quality hardware
- Fault tolerance: using redundancy for back up and maintain service even in case of fault
- Fault forecasting: predicting when fault may happen to replace component before it fails → shorten MTTR
  - SMART: hardware failure prediction

Disk storage

- HDD (magnetic)
- SSD (solid-state)



HDD with rotating magnetic disks

SSD

- TRIM

System Interconnection

- Processor
- Memory
- I/O devices

- How to connect them physically?

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

Types of Buses

- Processor-memory bus ("Front Side 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
  - Use either the processor-memory bus or a backplane bus to connect to memory
- Backplane bus (industry standard, e.g., ATA, PCIexpress)
  - Allow processor, memory and I/O devices to coexist on a single bus
  - Used as an intermediary bus connecting I/O busses to the processor-memory bus

I/O Transactions

- An I/O transaction is a sequence of operations over the interconnect that includes a request and may include a response either of which may carry data.
- An I/O transaction typically includes two parts
  1. Sending the address
  2. Receiving or sending the data

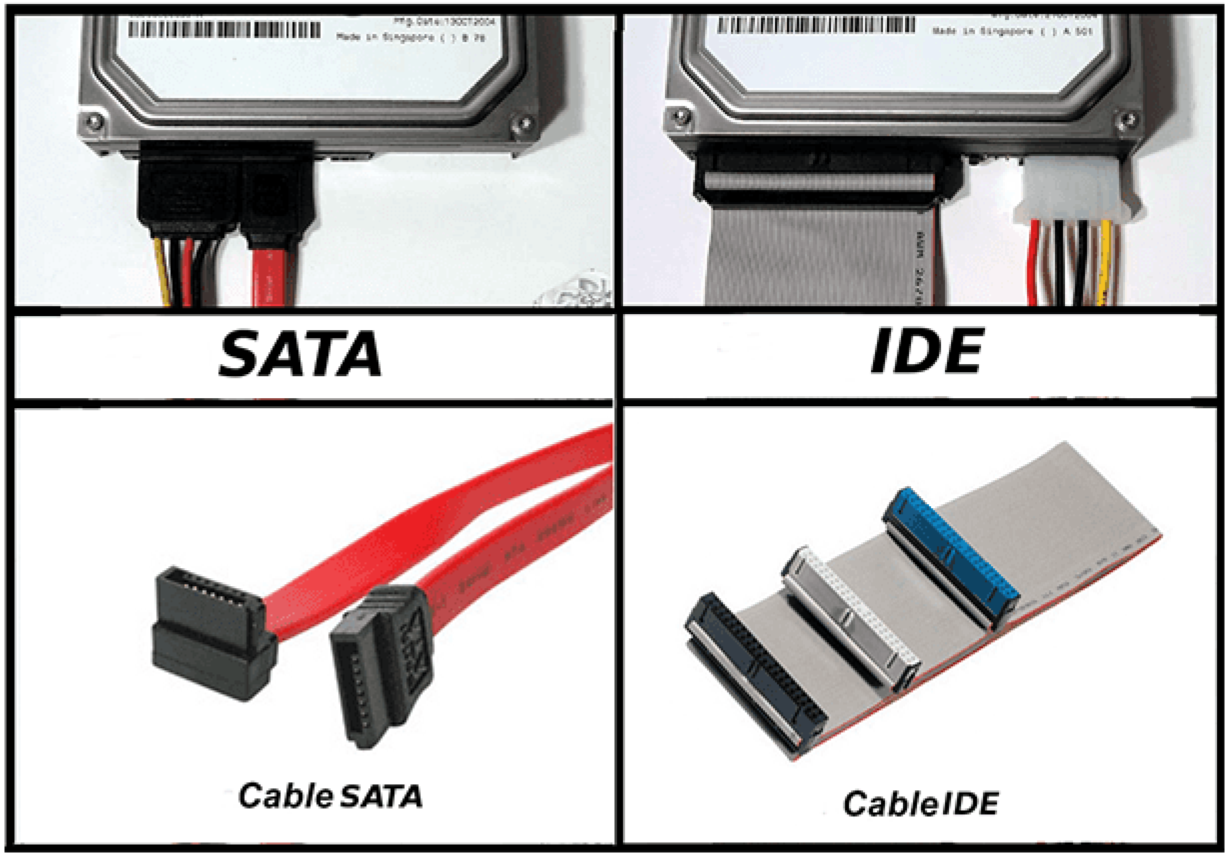
- Bus transactions are defined by what they do to memory
  - output A read transaction reads data from memory (to either the processor or an I/O device)
  - input A write transaction writes data to the memory (from either the processor or an I/O device)

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
    - Short distance
- 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)

Example: Synchronous to asynchronous

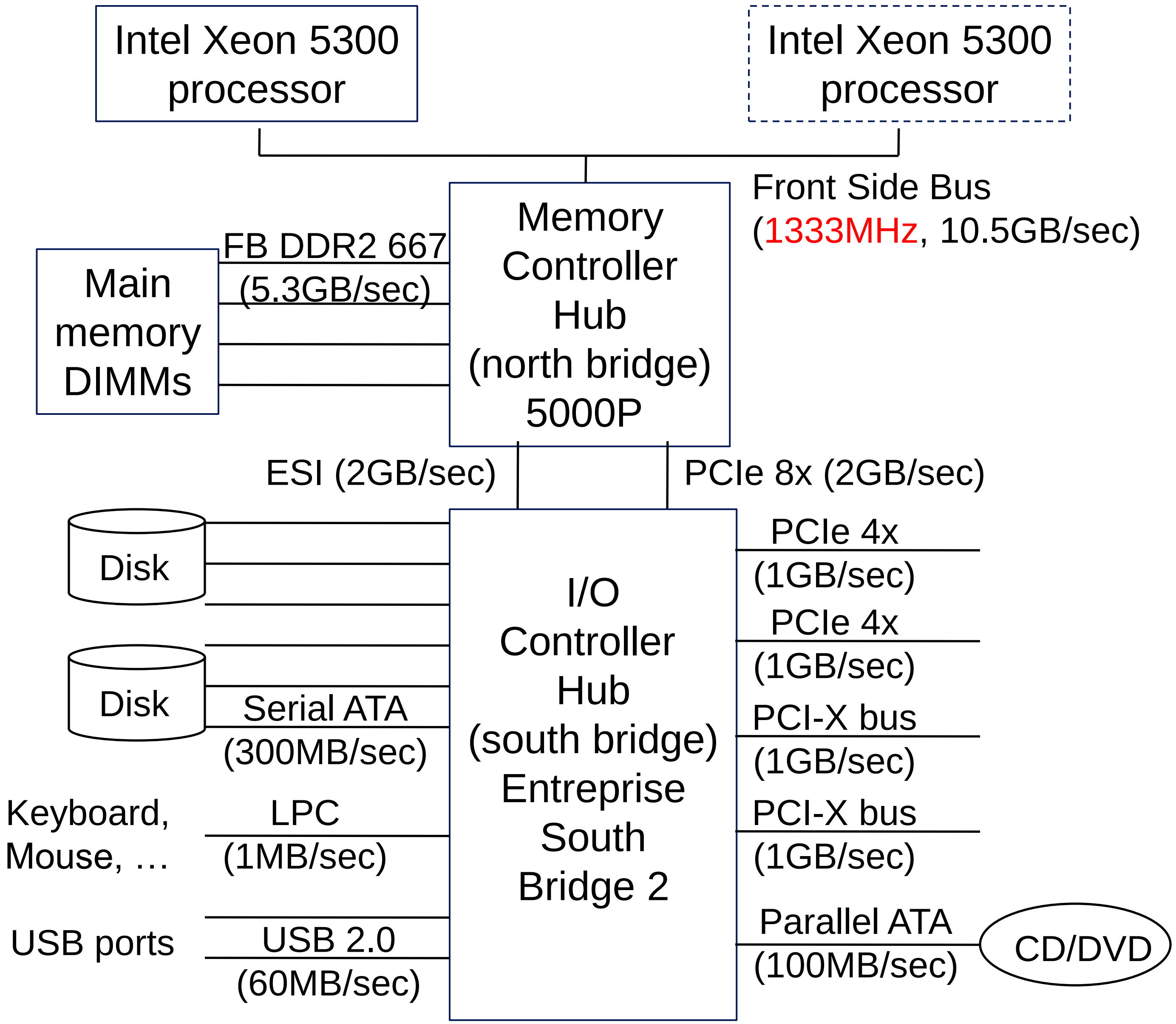
- It is difficult to use parallel wires running at a high clock rate ➔ a few one-way wires running at a very high “clock” rate (~2GHz)



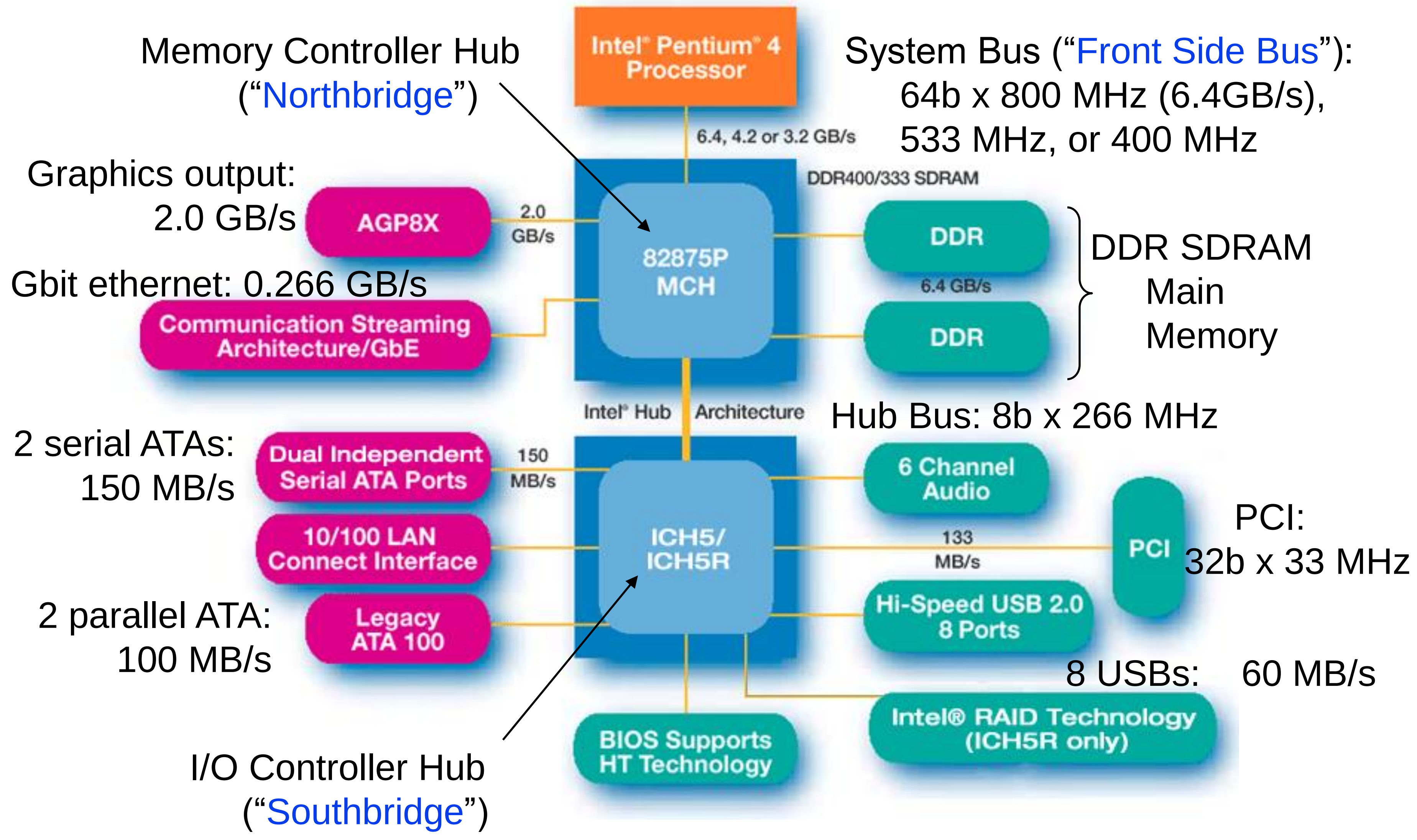
Modern I/O standards

	Firewire	USB 2.0	PCIe	Serial ATA	SA SCSI
Use	External	External	Internal	Internal	External
Devices per channel	63	127	1	1	4
Max length	4.5 meters	5 meters	0.5 meters	1 meter	8 meters
Data Width	4	2	2 per lane	4	4
Peak Bandwidth	50MB/sec (400) 100MB/sec (800)	0.2MB/sec (low) 1.5MB/sec (full) 60MB/sec (high)	250MB/sec per lane (1x) Come as 1x, 2x, 4x, 8x, 16x, 32x	300MB/sec	300MB/sec
Hot pluggable?	Yes	Yes	Depends	Yes	Yes

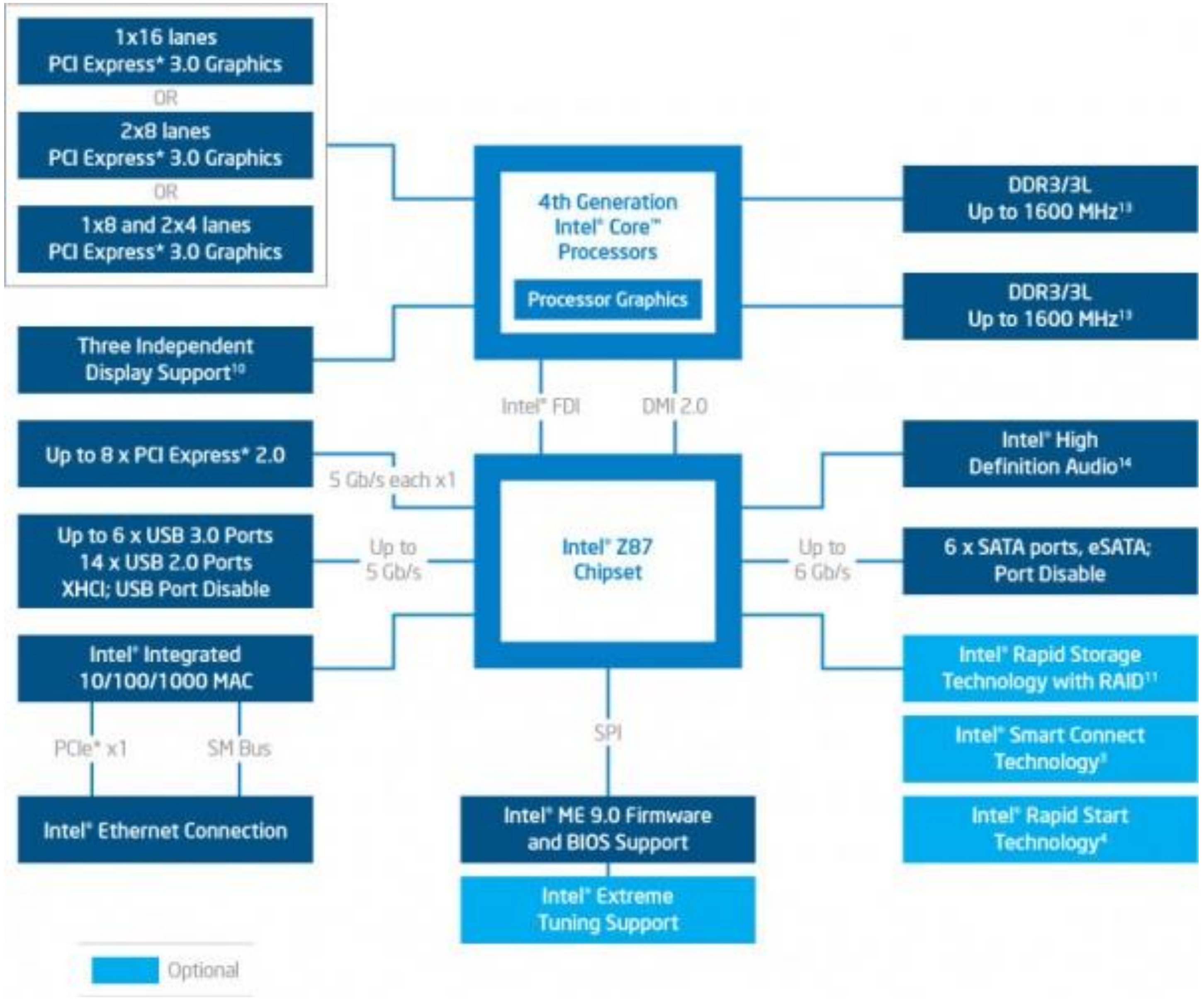
A Typical I/O System



Example: The Pentium 4's Buses


Memory Controller Hub ("Northbridge")
Intel® Pentium® 4 Processor
System Bus ("Front Side Bus"): 64b x 800 MHz (6.4GB/s), 533 MHz, or 400 MHz
6.4, 4.2 or 3.2 GB/s
Graphics output: 2.0 GB/s
DDR400/333 SDRAM
AGP8X
2.0 GB/s
82875P MCH
DDR
6.4 GB/s
DDR
DDR SDRAM Main Memory
Gbit ethernet: 0.266 GB/s
Communication Streaming Architecture/GbE
Intel® Hub Architecture
Hub Bus: 8b x 266 MHz
2 serial ATAs: 150 MB/s
Dual Independent Serial ATA Ports
150 MB/s
6 Channel Audio
ICH5/ ICH5R
10/100 LAN Connect Interface
133 MB/s
PCI
PCI: 32b x 33 MHz
2 parallel ATA: 100 MB/s
Legacy ATA 100
Hi-Speed USB 2.0 8 Ports
8 USBs: 60 MB/s
Intel® RAID Technology (ICH5R only)
BIOS Supports HT Technology
I/O Controller Hub ("Southbridge")

Intel Core i7 with Z87 chipset


1x16 lanes PCI Express* 3.0 Graphics
OR
2x8 lanes PCI Express* 3.0 Graphics
OR
1x8 and 2x4 lanes PCI Express* 3.0 Graphics
Three Independent Display Support10
Up to 8 x PCI Express* 2.0
5 Gb/s each x1
Up to 6 x USB 3.0 Ports 14 x USB 2.0 Ports XHCI; USB Port Disable
Up to 5 Gb/s
Intel® Integrated 10/100/1000 MAC
PCIe* x1
SM Bus
Intel® Ethernet Connection
4th Generation Intel® Core™ Processors
Processor Graphics
Intel® FDI
DMI 2.0
Intel® Z87 Chipset
SPI
Intel® ME 9.0 Firmware and BIOS Support
Intel® Extreme Tuning Support
DDR3/3L Up to 1600 MHz13
DDR3/3L Up to 1600 MHz13
Intel® High Definition Audio14
Up to 6 Gb/s
6 x SATA ports, eSATA; Port Disable
Intel® Rapid Storage Technology with RAID11
Intel® Smart Connect Technology3
Intel® Rapid Start Technology4
Optional

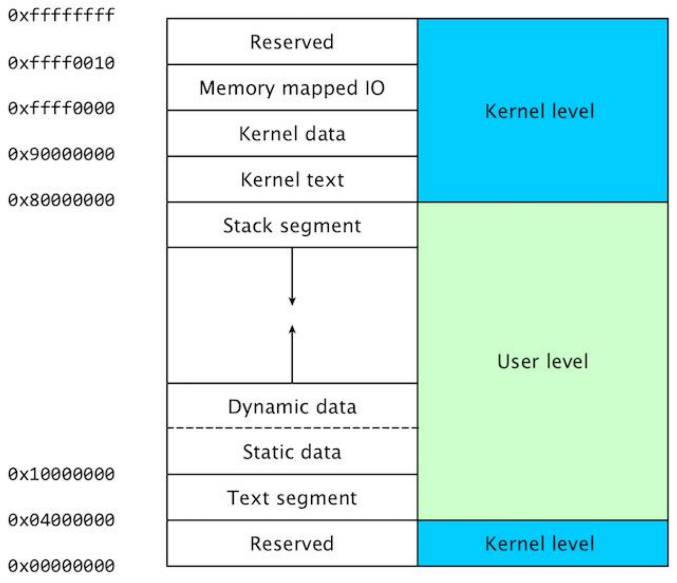
Interfacing I/O Devices

- Physical connection is done, now how about data transfer?

- How is a user I/O request transformed into a device command and communicated to the device?
- How is data actually transferred to or from a memory location?
- What is the role of the operating system?

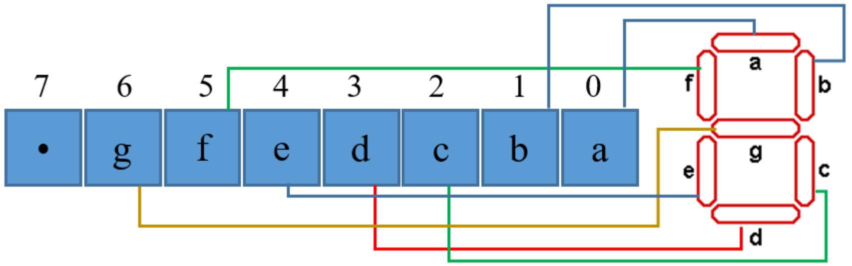
Communication of I/O Devices and Processor

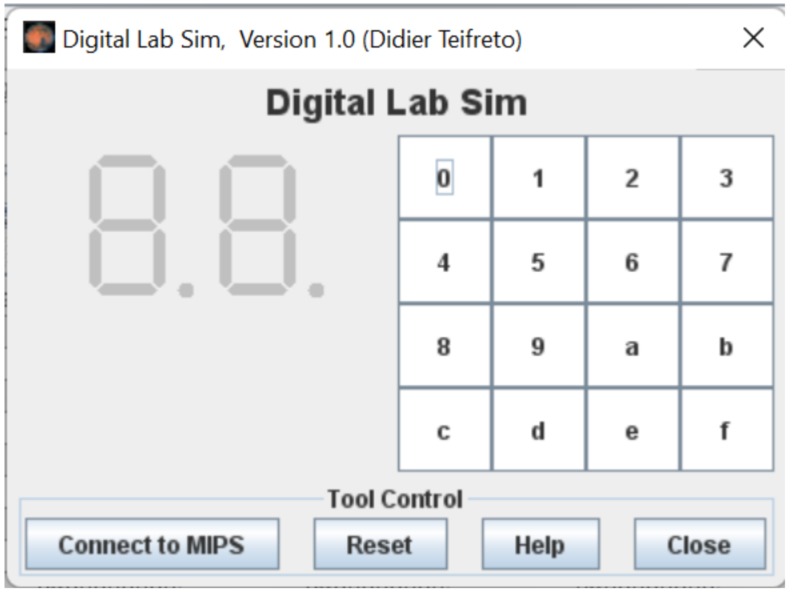
- How the processor directs (find) the I/O devices
  - Special I/O instructions
    - Must specify both the device and the command
  - Memory-mapped I/O
    - I/O devices are mapped to memory addresses
    - Read and writes to those memory addresses are interpreted as commands to the I/O devices
    - Load/stores to the I/O address space can *only* be done by the OS
- MIPS
  - Memory-mapped I/O
  - load/store instructions



Example: controlling 7-seg LED in MARS

- Tools→Digital Lab Sim: 2x 7-seg LEDs display
  - Byte value at address 0xFFFF0010 : command right seven segment display
  - Byte value at address 0xFFFF0011 : command left seven segment display

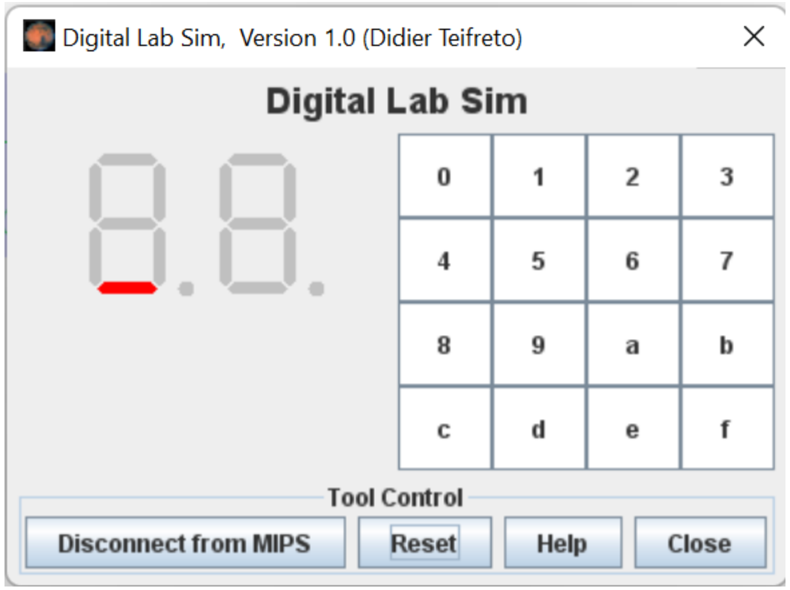




Example: controlling 7-seg LED in MARS

- Tools→Digital Lab Sim: 2x 7-seg LEDs display
  - Byte value at address 0xFFFF0010 : command right seven segment display
  - Byte value at address 0xFFFF0011 : command left seven segment display

```
li    $a0,   0x8          #value
li    $t0,   0xFFFF0011 #address
sb    $a0,   0($t0)      #turn-on
```



Exercise

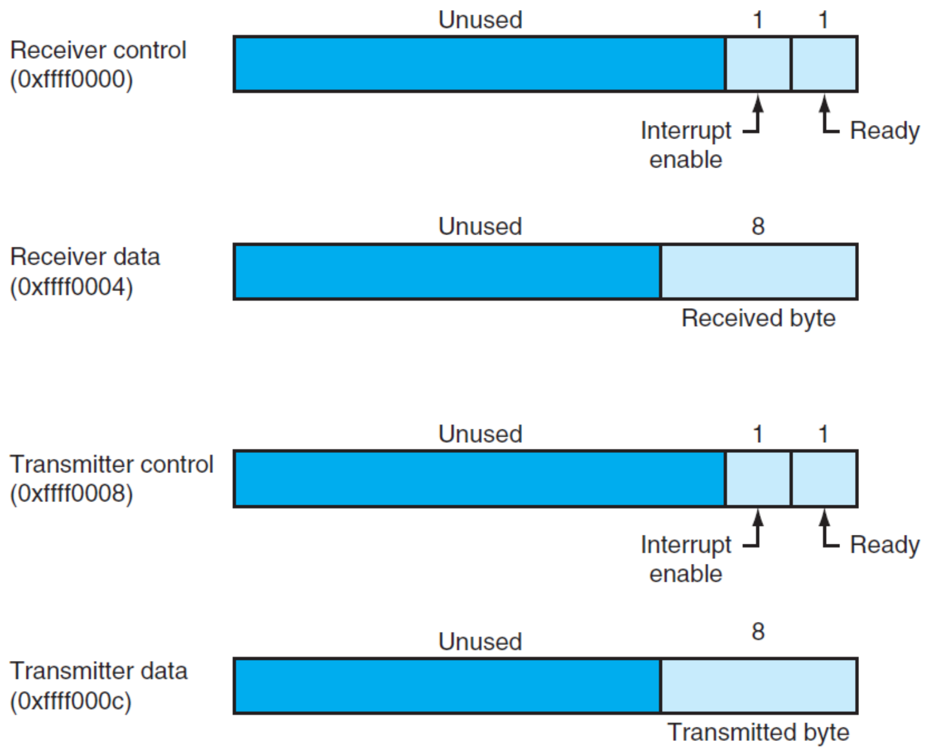
- Write program to display the value 27 to Digital Lab Sim

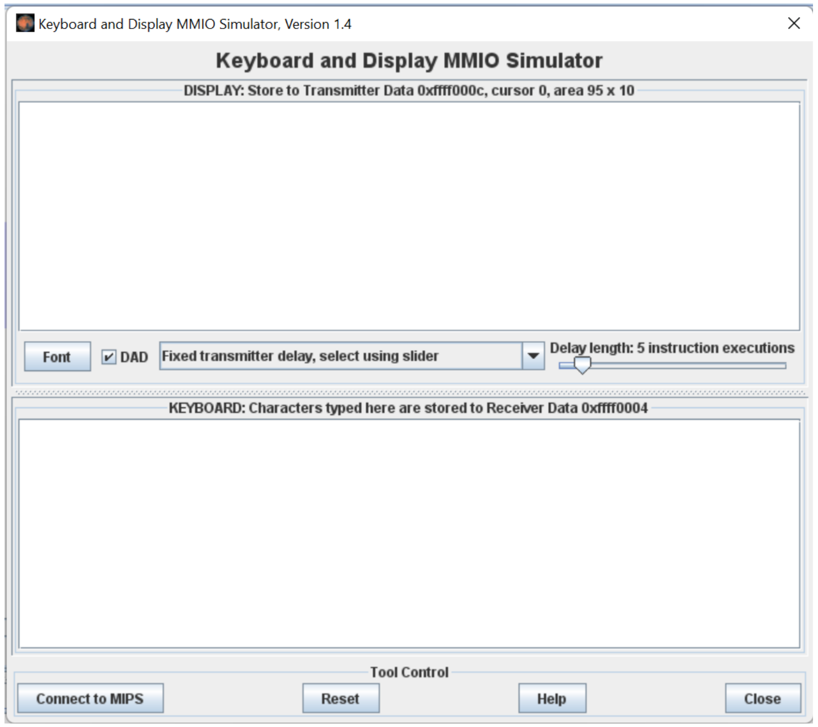
Communication of I/O Devices and Processor

- How I/O devices communicate with the processor
  - Polling
  - Interrupt driven I/O
  - Direct memory access
- 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

Example: polling the terminal

- Terminal:
  - Input: receiver control (0xffff0000) and data (0xffff0004)
  - Output: transmitter control (0xffff0008) and data (0xffff000c)


Receiver control (0xffff0000)
Unused
1
1
Interrupt enable
Ready
Receiver data (0xffff0004)
Unused
8
Received byte
Transmitter control (0xffff0008)
Unused
1
1
Interrupt enable
Ready
Transmitter data (0xffff000c)
Unused
8
Transmitted byte


Keyboard and Display MMIO Simulator, Version 1.4
Keyboard and Display MMIO Simulator
Font
DAD
Tool Control
Connect to MIPS
Reset
Help
Close

DISPLAY: Store to Transmitter Data 0xffff000c, cursor 0, area 95 x 10

[✓] DAD — Fixed transmitter delay, select using slider — Delay length: 5 instruction executions

KEYBOARD: Characters typed here are stored to Receiver Data 0xffff0004

Example: reading 1 byte from terminal

- Polling for data, then read when data is available

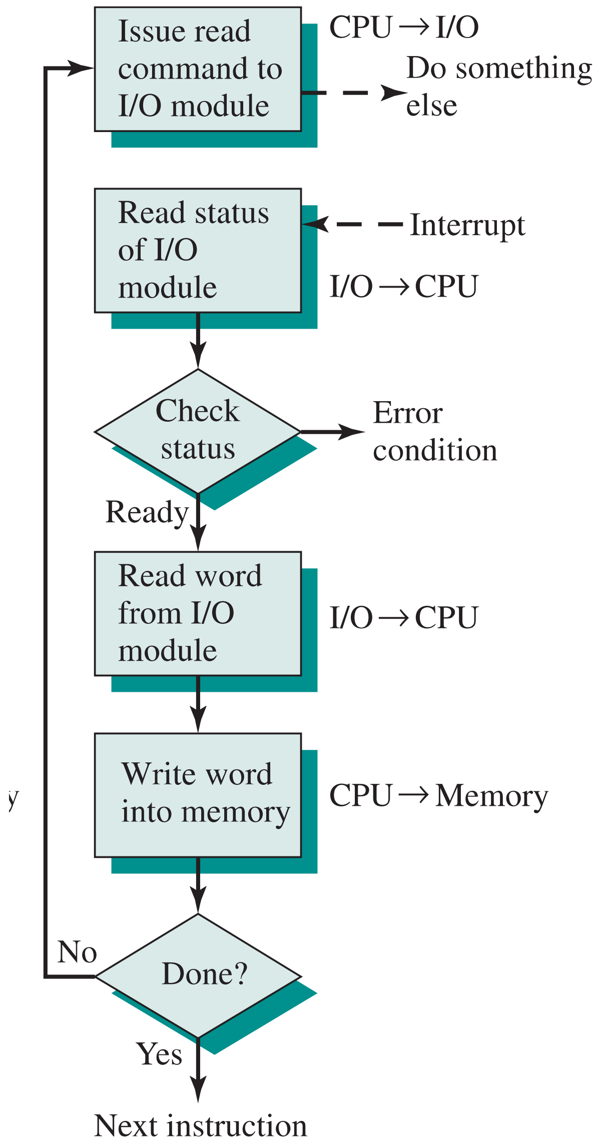
```
        li   $s0,  KEY_CODE
        li   $s1,  KEY_READY
WaitForKey:  lw   $t1, 0($k1)   # check data available
        beq  $t1, $zero, WaitForKey # if $t1 == 0 then Polling
ReadKey:     lw   $t0, 0($k0)
```

Exercise

- Write a program to continuously read data from the terminal, encode the data by shifting it 3 position in the ASCII table, then write the encoded data to the terminal.
- Remember to check for terminal input and output ready before read/write.

Interrupt driven I/O

- The I/O device issues an interrupt to indicate that it needs attention.
- The processor detects and "serves" the interrupt by executing a handler (aka. Interrupt service routine).

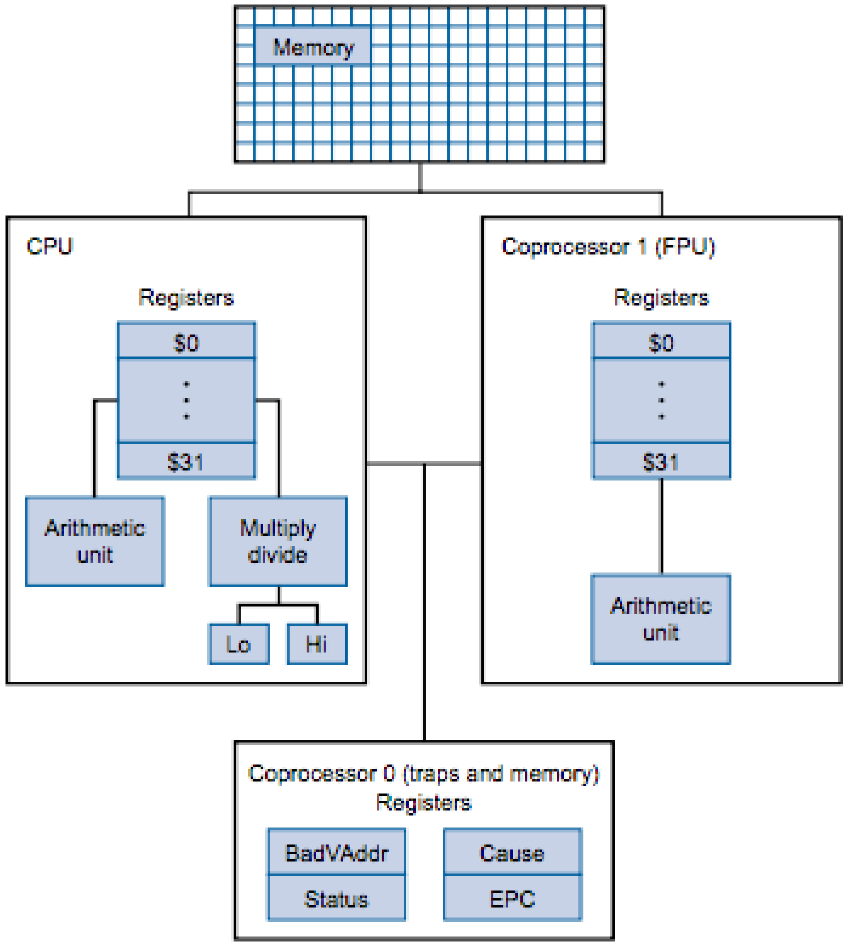


Interrupt Driven I/O

- 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
  - Indicate the I/O device causing the interrupt and to save the necessary information prior to servicing the interrupt and to resume normal processing after servicing the interrupt

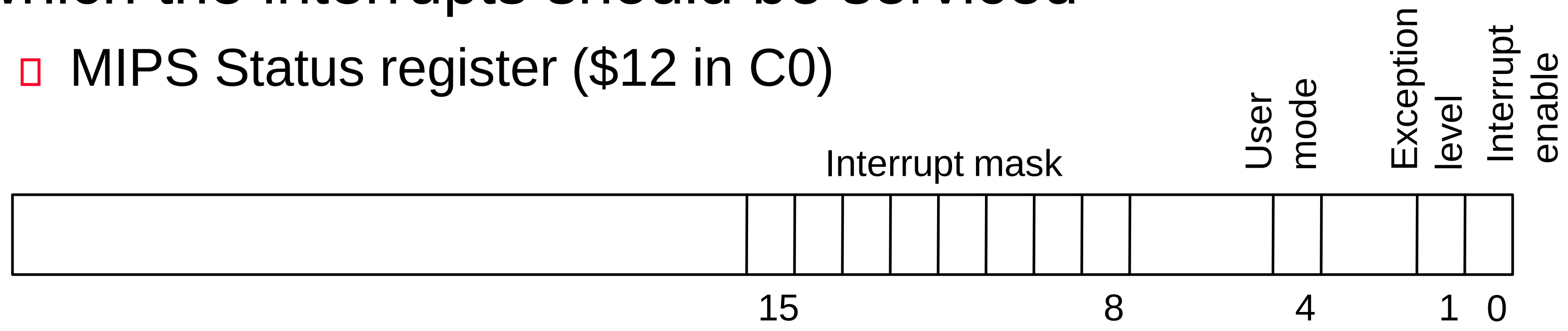
MIPS Coprocessor 0

- Support exception handling
  - Exception, interrupt, trap
- Status register ($12)
- Cause register ($13)
- EPC register ($14)
  - ➔ return address

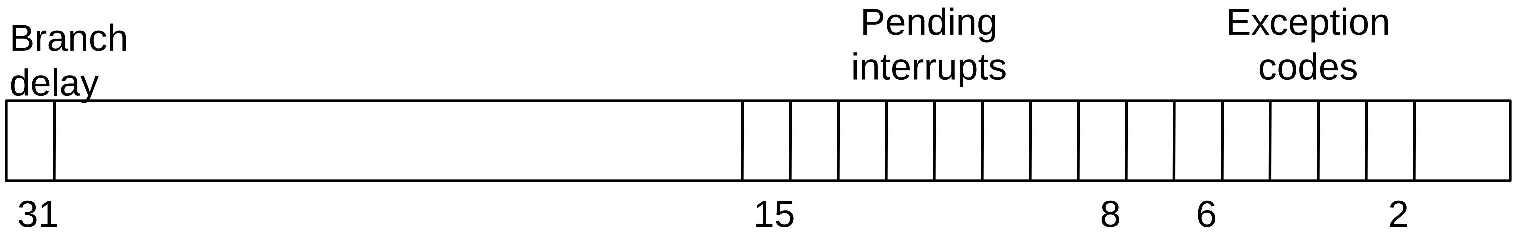


Interrupt Priority Levels

- Priority levels can be used to direct the OS the order in which the interrupts should be serviced
  - MIPS Status register ($12 in C0)



    - Determines who can interrupt the processor (if Interrupt enable is 0, none can interrupt)
  - MIPS Cause register ($13 in C0)



    - To enable a Pending interrupt, the correspond bit in the Interrupt mask must be 1
    - Once an interrupt occurs, the OS can find the reason in the Exception codes field

Interrupt handling

- When an interrupt occurred: MIPS branch to interrupt service routine located at *0x80000180* → use directive **`.ktext`** for interrupt service routine (ISR).
- Inside ISR
  - Check for interrupt source in Cause[6..2]
  - EPC ($14) stores return address
  - Exit from ISR with instruction **`eret`** (exception return). This basically restores PC with value in EPC.

Example: detect a keypad button pressed

- If keyboard interruption is enabled, an exception is started, with cause register bit number 11 set.
- Byte value at 0xFFFF0012 : command row number of hexadecimal keyboard (bit 0 to 3) and enable keyboard interrupt (bit 7)
- Byte value at 0xFFFF0014 : receive row and column of the key pressed, 0 if not key pressed
- The MIPS program has to scan, one by one, each row (send 1,2,4,8...) and then observe value at address 0xFFFF0014
  - Row number (4 left bits)
  - Column number (4 right bits)
  - The code for each key: 0x11, 0x21, 0x41, 0x81, 0x12, 0x22, 0x42, 0x82, 0x14, 0x24, 0x44, 0x84, 0x18, 0x28, 0x48, 0x88.

Example: detect a keypad button pressed

```
.eqv IN_ADRESS_HEXA_KEYBOARD       0xFFFF0012

.data
Message: .asciiz "Oh my god. Someone's presed a button.\n"
# MAIN Procedure
.text
main:
# Enable Digital Lab Sim keyboard interrupt
        li     $t1,    IN_ADRESS_HEXA_KEYBOARD
        li     $t3,    0x80   # bit 7 for interrupt
        sb     $t3,    0($t1)

Loop:   nop
        nop
        nop
        nop
        b        Loop              # Wait for interrupt
end_main:
```

Example: detect a keypad button pressed

- Interrupt service routine

```
.ktext 0x80000180
IntSR:  addi    $v0, $zero,  4   # show message
        la      $a0, Message
        syscall

return: eret                     # Return from exception
```

- Note for MARS:
  - Add a nop between syscall and jump, branch. Otherwise PC and EPC will get incorrect values.
  - Press “Connect to MIPS” on tools before starting simulation.

Direct Memory Access (DMA)

- For high-bandwidth devices (like disks) interrupt-driven I/O would consume a *lot* of processor cycles
- With DMA, the DMA controller has the ability to transfer large blocks of 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 DMA controller manages the entire transfer (possibly thousand of bytes in length), arbitrating for the bus
  3. When the DMA transfer is complete, the DMA controller interrupts the processor to let it know that the transfer is complete
- There may be multiple DMA devices in one system
  - Processor and DMA 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 input (from disk to memory) – the processor will be using stale data if that location is also in the cache
  - For a DMA output (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 can be solved by
  1. Routing all I/O activity through the cache – expensive and a large negative performance impact
  2. Having the OS invalidate all the entries in the cache for an I/O input or force write-backs for an I/O output (called a cache flush)
  3. Providing hardware to *selectively* invalidate cache entries – i.e., need a snooping cache controller

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 (i.e., 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/sec) of the processor is

  - 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

  - 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?

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

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

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

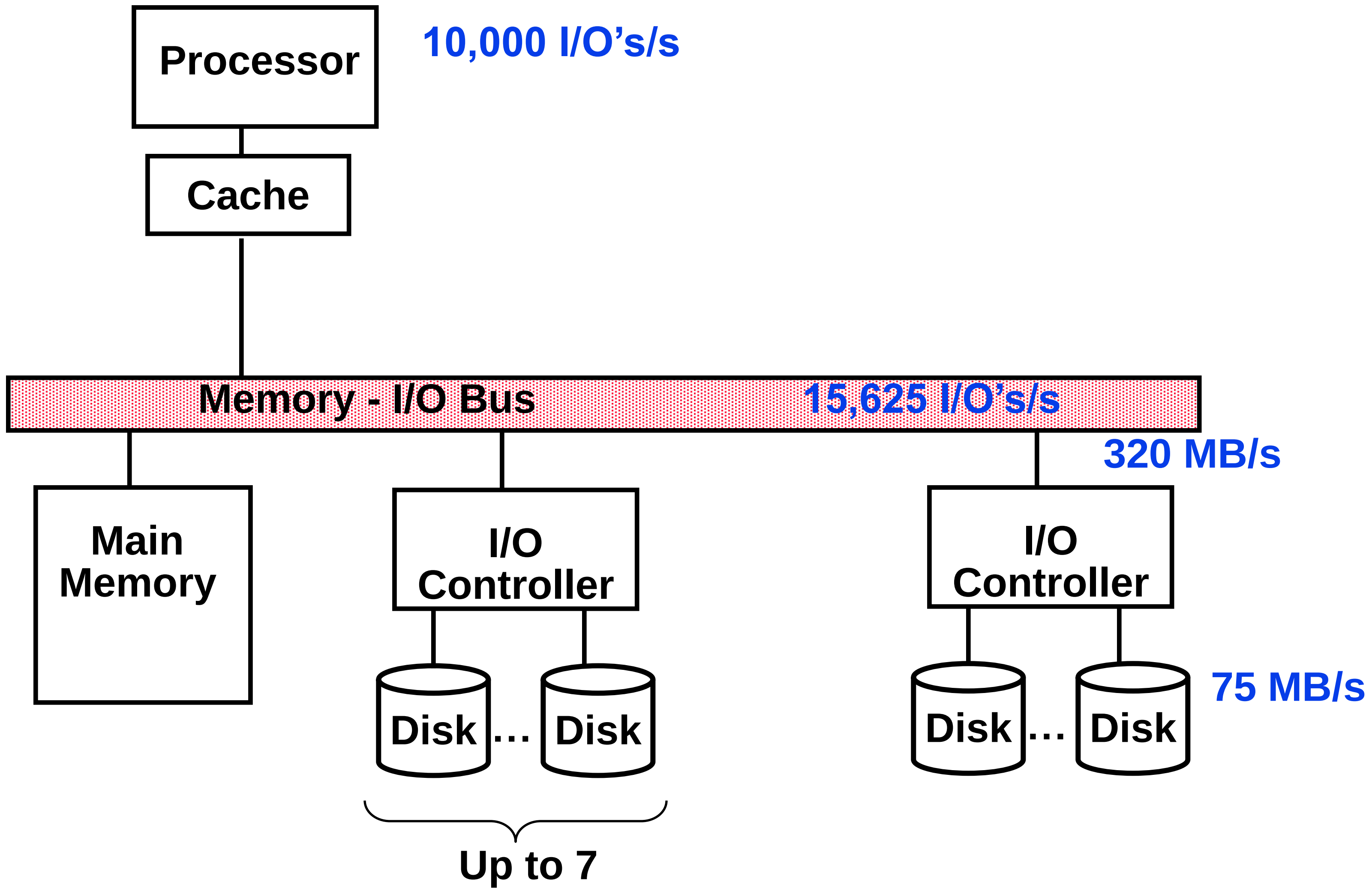
Each disk I/O reads/writes 64 KB so the maximum I/O rate of the bus is

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

  - 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.

Summary

- Characteristics of I/O system and devices
- I/O performance measures
- I/O system organization
- Methods for I/O operation and control
  - Polling
  - Interrupt
  - DMA