Recap: A Three-Bus System (+ backside cache)

- A small number of backplane buses tap into the processor-memory bus
  - Processor-memory bus is only used for processor-memory traffic
  - I/O buses are connected to the backplane bus
- Advantage: loading on the processor bus is greatly reduced

Recap: Main components of Intel Chipset: Pentium II/III

- Northbridge:
  - Handles memory
  - Graphics
- Southbridge: I/O
  - PCI bus
  - Disk controllers
  - USB controllers
  - Audio
  - Serial I/O
  - Interrupt controller
  - Timers

Arbitration: Obtaining Access to the Bus

- One of the most important issues in bus design:
  - How is the bus reserved by a device that wishes to use it?
- Chaos is avoided by a master-slave arrangement:
  - Only the bus master can control access to the bus:
    - It initiates and controls all bus requests
    - A slave responds to read and write requests
- The simplest system:
  - Processor is the only bus master
  - All bus requests must be controlled by the processor
  - Major drawback: the processor is involved in every transaction
The Daisy Chain Bus Arbitrations Scheme

- **Advantage:** simple
- **Disadvantages:**
  - Cannot assure fairness: A low-priority device may be locked out indefinitely
  - The use of the daisy chain grant signal also limits the bus speed

Centralized Parallel Arbitration

- Used in essentially all processor-memory busses and in high-speed I/O busses

Increasing the Bus Bandwidth

- **Separate versus multiplexed address and data lines:**
  - Address and data can be transmitted in one bus cycle if separate address and data lines are available
  - Cost: (a) more bus lines, (b) increased complexity

- **Data bus width:**
  - By increasing the width of the data bus, transfers of multiple words require fewer bus cycles
  - Example: SPARCstation 20’s memory bus is 128 bit wide
  - Cost: more bus lines

- **Block transfers:**
  - Allow the bus to transfer multiple words in back-to-back bus cycles
  - Only one address needs to be sent at the beginning
  - The bus is not released until the last word is transferred
  - Cost: (a) increased complexity (b) decreased response time for request

Increasing Transaction Rate on Multimaster Bus

- **Overlapped arbitration**
  - perform arbitration for next transaction during current transaction

- **Bus parking**
  - master can holds onto bus and performs multiple transactions as long as no other master makes request

- **Overlapped address / data phases (prev. slide)**
  - requires one of the above techniques

- **Split-phase (or packet switched) bus**
  - completely separate address and data phases
  - arbitrate separately for each
  - address phase yield a tag which is matched with data phase

- “All of the above” in most modern buses
Recall: PCI Read Transaction

– Turn-around cycle on any signal driven by more than one agent

I/O Device Examples

<table>
<thead>
<tr>
<th>Device</th>
<th>Behavior</th>
<th>Partner</th>
<th>Data Rate (KB/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>Input</td>
<td>Human</td>
<td>0.01</td>
</tr>
<tr>
<td>Mouse</td>
<td>Input</td>
<td>Human</td>
<td>0.02</td>
</tr>
<tr>
<td>Line Printer</td>
<td>Output</td>
<td>Human</td>
<td>1.00</td>
</tr>
<tr>
<td>Floppy disk</td>
<td>Storage</td>
<td>Machine</td>
<td>50.00</td>
</tr>
<tr>
<td>Laser Printer</td>
<td>Output</td>
<td>Human</td>
<td>100.00</td>
</tr>
<tr>
<td>Optical Disk</td>
<td>Storage</td>
<td>Machine</td>
<td>500.00</td>
</tr>
<tr>
<td>Magnetic Disk</td>
<td>Storage</td>
<td>Machine</td>
<td>5,000.00</td>
</tr>
<tr>
<td>Network-LAN</td>
<td>Input or Output</td>
<td>Machine</td>
<td>20 – 1,000.00</td>
</tr>
<tr>
<td>Graphics Display</td>
<td>Output</td>
<td>Human</td>
<td>30,000.00</td>
</tr>
</tbody>
</table>

I/O System Performance

° I/O System performance depends on many aspects of the system (“limited by weakest link in the chain”):
  • The CPU
  • The memory system:
    - Internal and external caches
    - Main Memory
  • The underlying interconnection (buses)
  • The I/O controller
  • The I/O device
  • The speed of the I/O software (Operating System)
  • The efficiency of the software’s use of the I/O devices

° Two common performance metrics:
  • Throughput: I/O bandwidth
  • Response time: Latency

Simple Producer-Server Model

° Throughput:
  • The number of tasks completed by the server in unit time
  • In order to get the highest possible throughput:
    - The server should never be idle
    - The queue should never be empty

° Response time:
  • Begins when a task is placed in the queue
  • Ends when it is completed by the server
  • In order to minimize the response time:
    - The queue should be empty
    - The server will be idle
Throughput versus Respond Time

![Graph showing throughput versus response time](image)

**Throughput Enhancement**

- In general throughput can be improved by:
  - Throwing more hardware at the problem
  - Reduces load-related latency

- Response time is much harder to reduce:
  - Ultimately it is limited by the speed of light (but we're far from it)

**Organization of a Hard Magnetic Disk**

- Typical numbers (depending on the disk size):
  - 500 to 2,000 tracks per surface
  - 32 to 128 sectors per track
  - A sector is the smallest unit that can be read or written

- Traditionally all tracks have the same number of sectors:
  - Constant bit density: record more sectors on the outer tracks

- Recently relaxed: constant bit size, speed varies with track location

**Magnetic Disk Characteristic**

- Cylinder: all the tracks under the head at a given point on all surfaces

- Read/write data is a three-stage process:
  - Seek time: position the arm over the proper track
  - Rotational latency: wait for the desired sector to rotate under the read/write head
  - Transfer time: transfer a block of bits (sector) under the read-write head

- Average seek time as reported by the industry:
  - Typically in the range of 8 ms to 12 ms
  - (Sum of the time for all possible seek) / (total # of possible seeks)

- Due to locality of disk reference, actual average seek time may:
  - Only be 25% to 33% of the advertised number
**Technology Trends**

- Today: Processing Power Doubles Every 18 months
- Today: Memory Size Doubles Every 18 months (4X/3yr)
- Today: Disk Capacity Doubles Every 18 months
- **Disk Positioning Rate (Seek + Rotate) Doubles Every Ten Years!**

**Disk History**

- **1973:** 1.7 Mbit/sq. in, 140 MBytes
- **1979:** 7.7 Mbit/sq. in, 2,300 MBytes

**Disk Capacity now doubles every 18 months; before 1990 every 36 months**

---

**Data density Mbit/sq. in.**

- 1989: 63 Mbit/sq. in, 60,000 MBytes
- 1997: 1450 Mbit/sq. in, 2300 MBytes
- 1997: 3090 Mbit/sq. in, 8100 MBytes

**MBits per square inch: DRAM as % of Disk over time**

- 470 v. 3000 Mb/si
- 9 v. 22 Mb/si
Nano-layered Disk Heads

- Special sensitivity of Disk head comes from “Giant Magneto-Resistive effect” or (GMR)
- IBM is leader in this technology
  - Same technology as TMJ-RAM breakthrough we described in earlier class.

Disk Device Terminology

- Disk Latency = Queueing Time + Controller time + Seek Time + Rotation Time + Xfer Time

Order of magnitude times for 4K byte transfers:
- Average Seek: 8 ms or less
- Rotate: 4.2 ms @ 7200 rpm
- Xfer: 1 ms @ 7200 rpm

Typical Numbers of a Magnetic Disk

- Average seek time as reported by the industry:
  - Typically in the range of 8 ms to 12 ms
  - Due to locality of disk reference may only be 25% to 33% of the advertised number
- Rotational Latency:
  - Most disks rotate at 3,600 to 7200 RPM (Up to 15,000 RPM or more)
  - Approximately 16 ms to 8 ms per revolution, respectively
  - An average latency to the desired information is halfway around the disk: 8 ms at 3600 RPM, 4 ms at 7200 RPM
- Transfer Time is a function of:
  - Transfer size (usually a sector): 1 KB / sector
  - Rotation speed: 3600 RPM to 10000 RPM
  - Recording density: bits per inch on a track
  - Diameter typical diameter ranges from 2.5 to 5.25 in
  - Typical values: 2 to 40 MB per second

Administrivia

- Midterm II: Wednesday 5/5 (5:30 – 8:30 in 306 Soda Hall)
  - Review session Sunday 5/4
    - 306 Soda, 7:00 – 9:00 pm
    - Pizza at LaVal’s afterwards (I’ll buy as usual!)
  - Topics everything up to Wednesday’s lecture
    - Pipelining
    - Caches/Memory systems
    - Buses and I/O
    - Power?
- Thursday 4/29 in Section (119 Cory):
  - Discuss plans for final project
  - Problem 0 of Lab 6 due by midnight as well.
- Important: Lab 6. Design for Test
  - You should be testing from the very start of your design
  - Consider adding special monitor modules at various points in design =>
    I have asked you to label trace output from these modules with the current clock cycle #
  - The time to understand how components of your design should work is while you are designing!
  - Be conscious of you clock cycle times
### Administrivia II: Final project options

° Major organizational options:
  - 2-way superscalar (18 points)
  - 2-way multithreading (20 points)
  - 2-way multiprocessor (18 points)
  - out-of-order execution (22 points)
  - Deep Pipelined (18 points)

° Test programs will include multiprocessor versions

° Both multiprocessor and multithreaded must implement synchronizing “Test and Set” instruction:
  - Memory mapped I/O, with special address range:
    - Addresses from 0xFFFFF00 to 0xFFFFFFF
    - Only need to implement 16 synchronizing locations
  - Reads and returns old value of memory location at specified address, while setting the value to one (stall memory stage for one extra cycle).
  - For multiprocessor, this instruction must make sure that all updates to this address are suspended during operation.
  - For multithreaded, switch to other processor if value is already non-zero (like a cache miss).

### Disk I/O Performance

**Metrics:**

- **Response Time Throughput**
- Latency goes as $T_{set} \times \frac{u}{1-u}$

- $u = \text{utilization}$

**Graph:**

- Response time = Queue + Device Service time

### Introduction to Queueing Theory

° Queueing Theory applies to long term, steady state behavior \( \Rightarrow \text{Arrival rate} = \text{Departure rate} \)

° Little’s Law:
  - Mean number tasks in system = arrival rate x mean response time
    - Observed by many, Little was first to prove
    - Simple interpretation: you should see the same number of tasks in queue when entering as when leaving.

° Applies to any system in equilibrium, as long as nothing in black box is creating or destroying tasks

### A Little Queuing Theory: Use of random distributions

° Server spends a variable amount of time with customers
  - Weighted mean $m_1 = (f_1 x T_1 + f_2 x T_2 + ... + f_n x T_n) / F$
  - $\sigma^2 = (f_1 x T_1^2 + f_2 x T_2^2 + ... + f_n x T_n^2) / F - m_1^2$
  - Squared coefficient of variance: $C = \sigma^2 / m_1^2$
    - Unitless measure (100 ms$^2$ vs. 0.1 s$^2$)
  - Exponential distribution $C = 1$: most short relative to average, few others long; 90% < 2.3 x average, 63% < average
  - Hyperexponential distribution $C > 1$: further from average, C=2.0 => 90% < 2.8 x average, 69% < average
A Little Queuing Theory: Variable Service Time

- Disk response times $C \approx 1.5$ (majority seeks < average)
- Yet usually pick $C = 1.0$ for simplicity
  - Memoryless, exponential dist
  - Many complex systems well described by memoryless distribution!
- Another useful value is average time must wait for server to complete current task: $m_1(z)$
  - Called "Average Residual Wait Time"
  - Not just $1/2 \times m_1$ because doesn't capture variance
  - Can derive $m_1(z) = 1/2 \times m_1 \times (1 + C)$
  - Exponential $\Rightarrow C = 1 \Rightarrow m_1(z) = m_1$

A Little Queuing Theory: Average Wait Time

- Calculating average wait time in queue $T_q$:
  - All customers in line must complete; avg time: $m_1 T_{ser} = 1/\mu$
  - If something at server, it takes to complete on average $m_1(z)$
    - Chance server is busy $= u = \lambda / \mu$; average delay is $u \times m_1(z)$
    - Little’s Law
    - Defn of utilization ($u$)
    - $T_q = u \times m_1(z) + L_q \times T_{ser}$
    - $T_q = u \times m_1(z) + \lambda \times T_q \times T_{ser}$
    - $T_q \times (1 - u) = m_1(z) \times u$
    - $T_q = m_1(z) \times u/(1-u) = T_{ser} \times \{1/2 \times (1+C)\} \times u/(1-u)$

Notation:
- $\lambda$: average number of arriving customers/second
- $T_{ser}$: average time to service a customer
- $u$: server utilization (0..1): $u = \lambda \times T_{ser}$
- $T_q$: average time/customer in queue
- $L_q$: average length of queue: $L_q = \lambda \times T_q$
- $m_1(z)$: average residual wait time = $T_{ser} \times \{1/2 \times (1+C)\}$

A Little Queuing Theory: M/G/1 and M/M/1

- Assumptions so far:
  - System in equilibrium
  - Time between two successive arrivals in line are random
  - Server can start on next customer immediately after prior finishes
  - No limit to the queue: works First-In-First-Out
  - Afterward, all customers in line must complete; each avg $T_{ser}$
- Described “memoryless” or Markovian request arrival (M for C=1 exponentially random), General service distribution (no restrictions), 1 server: M/G/1 queue
- When Service times have C = 1, M/M/1 queue
  - $T_q = T_{ser} \times u / (1 - u)$
  - $T_{ser}$: average time to service a customer
  - $u$: server utilization (0..1): $u = \lambda \times T_{ser}$
  - $T_q$: average time/customer in queue

A Little Queuing Theory: An Example

- Processor sends 10 x 8KB disk I/Os per second, requests & service exponentially distrib., avg disk service = 20 ms
  - This number comes from disk equation:
    - Service time = Ave seek + ave rot delay + transfer time + ctrl overhead
  - On average, how utilized is the disk?
    - What is the number of requests in the queue?
    - What is the average time spent in the queue?
    - What is the average response time for a disk request?
- Notation:
  - $\lambda$: average number of arriving customers/second = 10
  - $T_{ser}$: average time to service a customer = 20 ms (0.02s)
  - $u$: server utilization (0..1): $u = \lambda \times T_{ser} = 10/s \times 0.02s = 0.2$
  - $T_q$: average time/customer in queue = $T_{ser} \times u / (1 - u) = 20 \times 0.2/(1-0.2) = 20 \times 0.25 = 5$ ms (0.005s)
  - $T_{sys}$: average time/customer in system: $T_{sys} = T_q + T_{ser} = 25$ ms
  - $L_q$: average length of queue: $L_q = \lambda \times T_q = 10/s \times 0.05s = 0.05$ requests in queue
  - $L_{sys}$: average # tasks in system: $L_{sys} = \lambda \times T_{sys} = 10/s \times 0.025s = 0.25$
Memory System I/O Performance

- Pipelined Bus with queue at controller?
  - Time to transfer request
    - $T_{queue} = \text{Queueing Delay} + \text{service time}$
  - Time to transfer data
- DRAM has DETERMINISTIC service time
  - $T_{ser} = t_{RAC} + (n-1) * t_{PC} + t_{precharge}$
  - $T_q = m1(z) \times \frac{u}{1-u} = T_{ser} \times \left(\frac{1}{2} \times (1+C)\right) \times \frac{u}{1-u}$ with $C=0$

Giving Commands to I/O Devices

- Two methods are used to address the device:
  - Special I/O instructions
  - Memory-mapped I/O
- Special I/O instructions specify:
  - Both the device number and the command word
    - Device number: the processor communicates this via a set of wires normally included as part of the I/O bus
    - Command word: this is usually send on the bus’s data lines
- Memory-mapped I/O:
  - Portions of the address space are assigned to I/O device
  - Read and writes to those addresses are interpreted as commands to the I/O devices

Memory Mapped I/O

- Single Memory & I/O Bus
  - No Separate I/O Instructions
- Issues:
  - Real implementations usually “below” the cache, rather than in parallel with the cache (what you have for Labs 5 & 6)
    - Requires cache invalidation!
  - User programs are prevented from issuing I/O operations directly:
    - The I/O address space is protected by the address translation

I/O Device Notifying the OS

- The OS needs to know when:
  - The I/O device has completed an operation
  - The I/O operation has encountered an error
- This can be accomplished in two different ways
  - I/O Interrupt:
    - Whenever an I/O device needs attention from the processor, it interrupts the processor from what it is currently doing.
  - Polling:
    - The I/O device put information in a status register
    - The OS periodically check the status register
**Example: Device Interrupt**

```
add $r1,$r2,$r3
subi $r4,$r1,#4
slli $r4,$r4,#2
Hiccup (!)
lw $r2,0($r4)
lw $r3,4($r4)
add $r2,$r2,$r3
sw 8($r4),$r2
```

- **Advantage:**
  - User program progress is only halted during actual transfer
- **Disadvantage:**
  - Special hardware is needed to:
    - Cause an interrupt (I/O device)
    - Detect an interrupt (processor)
    - Save the proper states to resume after the interrupt (processor)

**Alternative: Polling**

```
Disable Network Intr

subi $r4,$r1,#4
slli $r4,$r4,#2
lw $r2,0($r4)
add $r3,$r0,#5
sw $r3,0($r1)
```

- **Polling Point (check device register)**
  - **“Handler”**
    - Beq $r1,no_mess
    - Lw $r1,20($zro)
    - Lw $r2,0($r1)
    - Addi $r3,$r0,#5
    - Sw 0($r1),$r3

**Polling: Programmed I/O**

```
Polling is faster/slower than Interrupts

Polling is faster than interrupts because
- Compiler knows which registers in use at polling point. Hence, do not need to save and restore registers (or not as many).
- Other interrupt overhead avoided (pipeline flush, trap priorities, etc).

Polling is slower than interrupts because
- Overhead of polling instructions is incurred regardless of whether or not handler is run. This could add to inner-loop delay.
- Device may have to wait for service for a long time.

When to use one or the other?
- Multi-axis tradeoff
  - Frequent/regular events good for polling, as long as device can be controlled at user level.
  - Interrupts good for infrequent/irregular events
  - Interrupts good for ensuring regular/predictable service of events.
```

- **Advantage:**
  - Simple: the processor is totally in control and does all the work
  - Your memory-mapped I/O from Lab 5/6 could poll on input!

- **Disadvantage:**
  - Polling overhead can consume a lot of CPU time
**Delegating I/O Responsibility from the CPU: DMA**

CPU sends a starting address, direction, and length count to DMAC. Then issues "start".

- Direct Memory Access (DMA):
  - External to the CPU
  - Act as a maser on the bus
  - Transfer blocks of data to or from memory without CPU intervention

**DMAC provides handshake signals for Peripheral Controller, and Memory Addresses and handshake signals for Memory.**

**Delegating I/O Responsibility from the CPU: IOP**

CPU sends a starting address, direction, and length count to DMAC. Then issues "start".

**DMAC provides handshake signals for Peripheral Controller, and Memory Addresses and handshake signals for Memory.**

**CPU** sends an instruction to IOP.

1. **CPU** issues an instruction to **IOP**.
2. **IOP** looks in memory for commands.
3. **IOP** transfers data to/from memory.
4. **IOP** interrupts CPU when done.

**Device** transfers are controlled by the IOP directly.

**IOP steals memory cycles.**

**Reliability and Availability**

- Two terms that are often confused:
  - Reliability: Is anything broken?
  - Availability: Is the system still available to the user?

- Availability can be improved by adding hardware:
  - Example: adding ECC on memory

- Reliability can only be improved by:
  - Better environmental conditions
  - Building more reliable components
  - Building with fewer components
  - Improve availability may come at the cost of lower reliability

- Durability: Will the data last forever?

**Manufacturing Advantages of Disk Arrays**

**Conventional:**
- 4 disk designs
- 3.5"  5.25"  10"

**Disk Array:**
- 1 disk design
- 3.5"
Reliability of N disks = Reliability of 1 Disk ÷ N
50,000 Hours ÷ 70 disks = 700 hours
Disk system MTTF: Drops from 6 years to 1 month!

Arrays (without redundancy) too unreliable to be useful!

Hot spares support reconstruction in parallel with access: very high media availability can be achieved.

Files are "striped" across multiple spindles
Redundancy yields high data availability
Disks will fail
Contents reconstructed from data redundantly stored in the array
→ Capacity penalty to store it
→ Bandwidth penalty to update

Techniques:
- Mirroring/Shadowing (high capacity cost)
- Horizontal Hamming Codes (overkill)
- Parity & Reed-Solomon Codes
- Failure Prediction (no capacity overhead!)
VaxSimPlus — Technique is controversial

Each disk is fully duplicated onto its "shadow"
Very high availability can be achieved

Bandwidth sacrifice on write:
Logical write = two physical writes

Reads may be optimized

Most expensive solution: 100% capacity overhead

Targeted for high I/O rate, high availability environments

Parity computed across recovery group to protect against hard disk failures
33% capacity cost for parity in this configuration
wider arrays reduce capacity costs, decrease expected availability, increase reconstruction time

Arms logically synchronized, spindles rotationally synchronized
globally a single high capacity, high transfer rate disk

Targeted for high bandwidth applications: Scientific, Image Processing
**RAID 5+: High I/O Rate Parity**

- A logical write becomes four physical I/Os.
- Independent writes possible because of interleaved parity.
- Reed-Solomon Codes ("Q") for protection during reconstruction.

![Disk Address Example](image)

**Problems of Disk Arrays: Small Writes**

- **RAID-5: Small Write Algorithm**
  - 1 Logical Write = 2 Physical Reads + 2 Physical Writes

- **Hewlett-Packard (HP) AutoRAID**
  - HP has interesting solution which combines both mirroring and RAID level 5.
  - Dynamically adapts disk storage:
    - For recent or highly used data, uses mirroring.
    - For less recently used data, uses RAID 5.
  - Gets speed of mirroring when it matters and density of RAID 5 on average.

- **I/O Summary:**
  - I/O performance limited by weakest link in chain between OS and device.
  - Three Components of Disk Access Time:
    - Seek Time: advertised to be 8 to 12 ms. May be lower in real life.
    - Rotational Latency: 4.1 ms at 7200 RPM and 8.3 ms at 3600 RPM
    - Transfer Time: 2 to 12 MB per second
  - I/O device notifying the operating system:
    - Polling: it can waste a lot of processor time.
    - I/O interrupt: similar to exception except it is asynchronous.
  - Delegating I/O responsibility from the CPU:
    - DMA, or even IOP.
  - Queueing theory is important:
    - 100% utilization means very large latency.
    - Remember, for M/M/1 queue (exponential source of requests/service):
      - queue size goes as \( u/(1-u) \)
      - latency goes as \( T_{ser} u/(1-u) \)
    - For M/G/1 queue (more general server, exponential sources):
      - latency goes as \( m1(z) \times u/(1-u) = T_{ser} x (1/2 x (1+C)) \times u/(1-u) \)
  - Redundancy + Repair is key to high reliability.