Virtual Memory
Input/Output

CPS 104
Lecture 21

Projects Due Next Monday April 14

Reading
I/O
Chapter 8 Input Output (primarily 8.3 and 8.5)
Appendix A.8 SPIM I/O
Finite State Machines
Appendix B and Ch 5.3
Multicycle Processor
Ch 5.4
**Review: Virtual Memory**

- Provides *illusion* of very large memory
  - Sum of the memory of many jobs greater than physical memory
  - Address space of each job larger than physical memory
- Good utilization of available (fast and expensive) physical memory.
- Simplifies memory management: code and data movement, protection, ... *(main reason today)*
- Exploits memory hierarchy to keep average access time low.
- Involves at least two storage levels: *main* and *secondary*

*Virtual Address* -- address used by the programmer

*Virtual Address Space* -- collection of such addresses

*Memory Address* -- address in physical memory also known as "physical address" or "real address"

---

**Review: Paged Virtual Memory: Main Idea**

- Divide memory (virtual and physical) into fixed size blocks (*Pages, Frames*).
  - Pages in Virtual space.
  - Frames in Physical space.
- Make page size a power of 2: *(page size = $2^k$)*
- All pages in the virtual address space are contiguous.
- *Pages* can be mapped into any physical *Frame*
- Some pages in *main memory* (DRAM), some pages on *secondary memory* (disk).
Review: Paged Virtual Memory: Main Idea (Cont)

• All programs are written using Virtual Memory Address Space.
• The hardware does on-the-fly translation between virtual and physical address spaces.
• Use a Page Table to translate between Virtual and Physical addresses.
• Translation Lookaside Buffer (TLB) expedites address translation.
• Must select “good” page size to minimize fragmentation.

Review: Paged Virtual Memory

• Virtual address \(2^{32}, 2^{64}\) to Physical Address mapping \(2^{28}\)
  ➢ virtual page to physical page frame
  
<table>
<thead>
<tr>
<th>Virtual page number</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>0x1000</td>
</tr>
<tr>
<td>0x1000</td>
<td>0x2000</td>
</tr>
</tbody>
</table>

• Fixed size units for access control & translation.
Review: Virtual to Physical Address translation

Page size: 4K

Virtual Address

<table>
<thead>
<tr>
<th>Virtual Page Number</th>
<th>Page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>11 0</td>
</tr>
</tbody>
</table>

Page Table

Physical Frame Number

<table>
<thead>
<tr>
<th>Physical Frame Number</th>
<th>Page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>11 0</td>
</tr>
</tbody>
</table>

Physical Address

Need to translate every access (instruction and data)

Review: Fast Translation: Translation Buffer

- Cache of translated addresses
- 64 entry fully associative

Virtual Address

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Page offset

<table>
<thead>
<tr>
<th>v r w tag phys frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
</tr>
</tbody>
</table>

Physical Address

<table>
<thead>
<tr>
<th>4</th>
</tr>
</thead>
</table>
Address Translation and Caches

- Where is the TLB wrt the cache?
- What are the consequences?

- Most of today’s systems have more than 1 cache
  - Digital 21164 had 3 levels
  - 2 levels on chip (8KB-data,8KB-inst,96KB-unified)
  - one level off chip (2-4MB)
- Does the OS need to worry about this?

TLBs and Caches

- Conventional Organization
- Virtually Addressed Cache
- Overlap $ access with VA translation: requires $ index to remain invariant across translation
Overview

• I/O devices
  - device controller
• Device drivers
• Memory Mapped I/O
• Programmed I/O
• Direct Memory Access (DMA)
• Rotational media (disks)
• I/O Bus technologies
• RAID (if time)
Why I/O?

- Interactive Applications (keyboard, mouse, screen)
- Long term storage (files, data repository)
- Swap for VM
- Many different devices
  - character vs. block
  - Networks are everywhere!
- $10^6$ difference CPU ($10^{-9}$) & I/O ($10^{-3}$)
- Response Time vs. Throughput
  - Not always another process to execute
- OS hides (some) differences in devices
  - same (similar) interface to many devices
- Permits many apps to share one device

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>Laser Printer</td>
<td>Output</td>
<td>Human</td>
<td>100.00</td>
</tr>
<tr>
<td>Graphics Display</td>
<td>Output</td>
<td>Human</td>
<td>30,000.00</td>
</tr>
<tr>
<td>Network-LAN</td>
<td>Input/Output</td>
<td>Machine</td>
<td>10,000.00</td>
</tr>
<tr>
<td>Floppy disk</td>
<td>Storage</td>
<td>Machine</td>
<td>50.00</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>
</tbody>
</table>
Device Drivers

- **top-half**
  - API (open, close, read, write, ioctl)
  - I/O Control (ioctl, device specific arguments)

- **bottom-half**
  - interrupt handler
  - communicates with device
  - resumes process

- **Must have access to user address space and device control registers** => runs in kernel mode.
Review: Handling an Interrupt/Exception

- Invoke specific kernel routine based on type of interrupt
  - interrupt/exception handler
- Must determine what caused interrupt
- Clear the interrupt
- Return from interrupt (RETT, MIPS = RFE)

Processor <-> Device Interface Issues

- Interconnections
  - Busses
- Processor interface
  - I/O Instructions
  - Memory mapped I/O
- I/O Control Structures
  - Device Controllers
  - Polling/Interrupts
- Data movement
  - Programmed I/O / DMA
- Capacity, Access Time, Bandwidth
Device Controllers

Controller deals with mundane control (e.g., position head, error detection/correction)
Processor communicates with Controller

I/O Instructions

Separate instructions (in, out)
Memory Mapped I/O

- Issue command through store instruction
- Check status with load instruction

Caches?

Communicating with the processor

- Polling
  - can waste time waiting for slow I/O device
  - busy wait
  - can interleave with useful work

- Interrupts
  - interrupt overhead
  - interrupt could happen anytime - asynchronous
  - no busy wait
Data Movement

- Programmed I/O
  - processor has to touch all the data
  - too much processor overhead
    - for high bandwidth devices (disk, network)

- DMA
  - processor sets up transfer(s)
  - DMA controller transfers data
  - complicates memory system

Programmed I/O & Polling

- Advantage: CPU totally in control

- Disadvantage: Overhead of polling
  - Program must perform check of device, thus can't do useful work
Programmed I/O & Interrupt Driven Data Transfer

- User program progress halted only during actual transfer
- Interrupt overhead can dominate transfer time
- Processor must touch all data...too slow for some devices

Direct Memory Access (DMA)

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

- CPU delegates responsibility for data transfer to a special controller

DMAC provides handshake signals for device controller, and memory addresses and handshake signals for memory.
I/O and Virtual Caches

I/O is accomplished with physical addresses DMA
- flush pages from cache
- need pa->va reverse translation
- coherent DMA

SPIM (Future Homework): Interrupt Handler

- MIPS/SPIM program
- Use memory-mapped I/O
- Use interrupts

Program should:
- Accept keyboard input
  - interrupts
- Echo input to terminal
  - polling
- Exit if user typed ‘q’

- Programmed I/O
Terminal Control

- **Memory mapped I/O**
  - use LW, SW
- **mapped_io command line option**
- **Receiver - input**
  - `ready=1` when data valid
- **Transmitter**
  - `ready=1` when ready to print next char

Interrupt Driven I/O

- **Set Interrupt Enable = 1**
  - generates a level 0 interrupt when Ready becomes 1
  - if interrupt is enabled in Status Register also

- **Run spim with -notrap**
  - allows you to install interrupt handler
  - So use both -mapped_io and -notrap
Status Register

- Bit 0 = interrupt enable
- Bit 8 = allow level 0 interrupts
  - terminal input generates level 0 int.
- Coprocessor 0, register 12
  - use mfc0, mtc0
- On interrupt, bits 0-5 are shifted left by 2
  - disables interrupts and enters kernel mode
- When done servicing interrupt, use rfe to restore

<table>
<thead>
<tr>
<th>Interrupt mask</th>
<th>Old</th>
<th>Previous</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 8 5 4 3 2 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cause Register

- Code 0000 = external interrupt
  - terminal interrupt

<table>
<thead>
<tr>
<th>Pending interrupts</th>
<th>Exception code</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 10 5 2</td>
<td></td>
</tr>
</tbody>
</table>

© Alvin R. Lebeck
Types of Storage Devices

• Magnetic Disks
• Magnetic Tapes
• CD ROM
• Juke Box (automated tape library, robots)

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

Typical Numbers of a Magnetic Disk

• Rotational Latency:
  - Most disks rotate at 3,600 to 10,000 RPM
  - Approximately 16ms to 3.5ms 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 7200 RPM
  - Recording density: bits per inch on a track
  - Diameter typical ranges from 2.5 to 5.25 in
  - Typical values: 2 to 12 MB per second
Disk Access

- **Access time** = queue + seek + rotational + transfer + overhead

- **Seek time**
  - move arm over track
  - average is confusing (startup, slowdown, locality of accesses)

- **Rotational latency**
  - wait for sector to rotate under head
  - average = 0.5/(3600 RPM) = 8.3ms

- **Transfer Time**
  - f(size, BW bytes/sec)

---

Disk Access Time Example

- **Disk Parameters:**
  - Transfer size is 8K bytes
  - Advertised average seek is 12 ms
  - Disk spins at 7200 RPM
  - Transfer rate is 4 MB/sec

- **Controller overhead is 2 ms**

- **Assume that disk is idle so no queuing delay**

- **What is Average Disk Access Time for a Sector?**
  - Ave seek + ave rot delay + transfer time + controller overhead
  - 12 ms + 0.5/(7200 RPM/60) + 8 KB/4 MB/s + 2 ms
  - 12 + 4.15 + 2 + 2 = 20 ms

- **Advertised seek time assumes no locality: typically 1/4 to 1/3 advertised seek time: 20 ms => 12 ms**
A bus is a shared communication link
It uses one set of wires to connect multiple subsystems

Advantages of Buses

- Versatility:
  - New devices can be added easily
  - Peripherals 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 of Buses

- The bus creates a communication bottleneck
  - Bus bandwidth can limit 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
  - The need to support a range of devices with:
    - Widely varying latencies
    - Widely varying data transfer rates

The General Organization of a Bus

- Control lines:
  - Signal requests and acknowledgments
  - Indicate what type of information is on the data lines
- Data lines carry information between the source and the destination:
  - Data and Addresses
  - Complex commands
Master versus Slave

- A bus transaction includes two parts:
  - Sending the address
  - Receiving or sending the data
- Master is the device that starts the bus transaction by:
  - Sending the address
- Slave is the device that responds to the address by:
  - Sending data to the master if the master asks for data
  - Receiving data from the master if the master wants to send data

Output Operation

- Output: **Processor** sending data to the I/O device:
  1. **Step 1: Request Memory**
     - Processor
     - Control (Memory Read Request)
     - Memory
     - I/O Device (Disk)
     - Data (Memory Address)
  2. **Step 2: Read Memory**
     - Processor
     - Control
     - Memory
     - I/O Device (Disk)
     - Data
  3. **Step 3: Send Data to I/O Device**
     - Processor
     - Control (Device Write Request)
     - Memory
     - I/O Device (Disk)
     - Data (I/O Device Address and then Data)
**Input Operation**

- **Input is defined as the Processor receiving data from the I/O device:**

  **Step 1: Request Memory**
  - Processor
  - Control (Memory Write Request)
  - Data (Memory Address)
  - I/O Device (Disk)
  - Memory

  **Step 2: Receive Data**
  - Processor
  - Control (I/O Read Request)
  - Data (I/O Device Address and then Data)
  - I/O Device (Disk)
  - Memory

**Types of Buses**

- **Processor-Memory Bus (design specific)**
  - Short and high speed
  - Only need to match the memory system
    - Maximize memory-to-processor bandwidth
  - Connects directly to the processor

- **External I/O Bus (industry standard)**
  - Usually is lengthy and slower
  - Need to match a wide range of I/O devices
  - Connects to the processor-memory bus or backplane bus

- **Backplane Bus (industry standard)**
  - Backplane: an interconnection structure within the chassis
  - Allow processors, memory, and I/O devices to coexist
  - Cost advantage: one single bus for all components

- **Bit-Serial Buses (New trend: USB, Firewire, .. )**
  - Use High speed unidirectional point-to-point communication
A Computer System with One Bus: Backplane Bus

- A single bus (the backplane bus) is used for:
  - Processor to memory communication
  - Communication between I/O devices and memory
- Advantages: Simple and low cost
- Disadvantages: slow and the bus can become a major bottleneck
- Example: Early IBM PC

A Two-Bus System

- I/O buses tap into the processor-memory bus via bus adaptors:
  - Processor-memory bus: mainly for processor-memory traffic
  - I/O buses: provide expansion slots for I/O devices
- Example: Apple Macintosh-II
  - NuBus: Processor, memory, and a few selected I/O devices
  - SCCI Bus: the rest of the I/O devices
A Three-Bus System

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

The Many Buses of Today’s Systems
Synchronous and Asynchronous Bus

- **Synchronous Bus:**
  - Includes a clock in the control lines
  - A fixed protocol for communication that is relative to the clock
  - Advantage: involves very little logic and can run very fast
  - Disadvantages:
    » Every device on the bus must run at the same clock rate
    » To avoid clock skew, bus must be short if it is fast.

- **Asynchronous Bus:**
  - It is not clocked
  - It can accommodate a wide range of devices
  - It can be lengthened without worrying about clock skew
  - It requires a handshaking protocol

A Handshaking Protocol

- **Three control lines**
  - **ReadReq:** indicates a read request for memory
    Address is put on the data lines at the same line
  - **DataRdy:** indicates the data word is now ready on the data lines
    Data is put on the data lines at the same time
  - **Ack:** acknowledge the ReadReq or the DataRdy of the other party
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: USB vs. 32-bit PCI vs. 64-bit PCI
  - 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

---

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
Multiple Potential Bus Masters: the Need for Arbitration

- **Bus arbitration scheme:**
  - A bus master wanting to use the bus asserts the bus request
  - A bus master cannot use the bus until its request is granted
  - A bus master must signal to the arbiter after finish using the bus

- **Bus arbitration schemes usually try to balance two factors:**
  - **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 broad classes:**
  - **Distributed arbitration by self-selection:** each device wanting the bus places a code indicating its identity on the bus.
  - **Distributed arbitration by collision detection:** Ethernet uses this.
  - **Daisy chain arbitration:** single device with all request lines.
  - **Centralized, parallel arbitration:** see next-next slide

---

Centralized Bus Arbitration

- **Arbitration Circuit**
  - Master A
  - Master B
  - Master C

- **Signals:**
  - Request-C
  - Grant-C
  - Request-B
  - Grant-B
  - Request-A
  - Grant-A
  - Release
The Daisy Chain Bus Arbitration 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

Summary of Bus Options

<table>
<thead>
<tr>
<th>Option</th>
<th>High performance</th>
<th>Low cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus width</td>
<td>Separate address &amp; data lines</td>
<td>Multiplex address &amp; data lines</td>
</tr>
<tr>
<td>Data width</td>
<td>Wider is faster (e.g., 64 bits)</td>
<td>Narrower is cheaper (e.g., 8 bits)</td>
</tr>
<tr>
<td>Transfer size</td>
<td>Multiple words has less bus overhead</td>
<td>Single-word transfer is simpler</td>
</tr>
<tr>
<td>Bus masters</td>
<td>Multiple (requires arbitration)</td>
<td>Single master (no arbitration)</td>
</tr>
<tr>
<td>Clocking</td>
<td>Synchronous</td>
<td>Asynchronous</td>
</tr>
</tbody>
</table>
Communication Networks

- Send/receive queues in processor memories
- Network controller copies back and forth via DMA
- No host intervention needed
- Interrupt host when message sent or received

Relationship to Processor Architecture

- Virtual memory frustrates DMA
  - page faults during DMA?
- Synchronization between controller and CPU
- Caches required for processor performance cause problems for I/O
  - Flushing is expensive, I/O pollutes cache
  - Solution is borrowed from shared memory multiprocessors "snooping" (coherent DMA)
- Caches and write buffers
  - need uncached and write buffer flush for memory mapped I/O
Summary

- Virtual memory & caches
- I/O
  - Processor interface issues
  - Memory mapped I/O
  - Interrupts and polling
- SPIM terminal device
- I/O devices
  - device controller
- Rotational media (disks)
  - Disk access time equation, etc.
- I/O bus ↔ memory bus
  - Standard interfaces, many devices from different vendors
- Buses
  - Asynchronous vs. synchronous
  - Bandwidth tradeoffs
  - Arbitration schemes
- I/O and virtual memory interaction