CS 61C: Great Ideas in Computer Architecture

Input/Output

Instructor: Justin Hsia
Agenda

• VM Wrap-up
• Administrivia
• Disks
• I/O Basics
• Exceptions and Interrupts
Virtual Memory Motivation

• Memory as cache for disk (reduce disk accesses)
  – Disk is so slow it significantly affects performance
  – Paging maximizes memory usage with large, evenly-sized pages that can go anywhere

• Allows processor to run multiple processes simultaneously
  – Gives each process illusion of its own (large) VM
  – Each process uses standard set of VAs
  – Access rights, separate PTs provide protection
Paging Summary

• Paging requires address translation
  – Can run programs larger than main memory
  – Hides variable machine configurations (RAM/HDD)
  – Solves fragmentation problem

• Address mappings stored in page tables in memory
  – Additional memory access mitigated with TLB
  – Check TLB, then Page Table (if necessary), then Cache
Hardware/Software Support for Memory Protection

• Different tasks can share parts of their virtual address spaces
  – But need to protect against errant access
  – Requires OS assistance

• Hardware support for OS protection
  – Privileged supervisor mode (a.k.a. *kernel mode*)
  – Privileged instructions
  – Page tables and other state information only accessible in supervisor mode
  – System call exception (e.g. `syscall` in MIPS)
Protection + Indirection = Virtual Address Space
Protection + Indirection = Dynamic Memory Allocation

malloc(4097)
Protection + Indirection = Dynamic Memory Allocation

Application 1
Virtual Memory

Application 2
Virtual Memory

malloc(4097)

Recursive function call
Protection + Indirection = Controlled Sharing
Protection + Indirection = Controlled Sharing

~ $FFFF\text{ }FFFF_{\text{hex}}$

stack

heap

static data

code

$\sim 0_{\text{hex}}$

Application 1
Virtual Memory

Shared Globals
“RW” Protection Bits

~ $FFFF\text{ }FFFF_{\text{hex}}$

Page Table

7
6
5
4
3
2
1
0

Stack 2
Heap 2
Stack 1
Heap 1
Static
Code

Physical Memory

~ $0_{\text{hex}}$

Application 2
Virtual Memory

Shared Code Page
“X” Protection Bit

7/31/2012

Summer 2012 -- Lecture #25
Virtual Memory Summary

• User program view:
  – Contiguous memory
  – Start from some set VA
  – “Infinitely” large
  – Is the only running program

• Reality:
  – Non-contiguous memory
  – Start wherever available memory is
  – Finite size
  – Many programs running simultaneously

• Virtual memory provides:
  – Illusion of contiguous memory
  – All programs starting at same set address
  – Illusion of ~ infinite memory (2^{32} or 2^{64} bytes)
  – Protection, Sharing

• Implementation:
  – Divide memory into chunks (pages)
  – OS controls page table that maps virtual into physical addresses
  – memory as a cache for disk
  – TLB is a cache for the page table
Virtual Memory Terminology

- Virtual Address (VA)
  - Virtual Memory (VM)
  - Virtual Page Number (VPN)
  - Page Offset (PO)
  - TLB Tag
  - TLB Index

- Physical Address (PA)
  - Physical Memory (PM)
  - Physical Page Number (PPN)
  - Page Offset (PO)
  - Tag, Index, Offset

- Page Table (PT) and Translation Lookaside Buffer (TLB)
  - Valid (V), Dirty (D), Ref (R), Access Rights (AR)
  - TLB Hit/Miss
  - PT Hit, Page Fault
  - TLB/PT Entries

- OS Tasks:
  - Swap Space
  - Page Table Base Register
  - Context Switching
Agenda

- VM Wrap-up
- Administrivia
- Disks
- I/O Basics
- Exceptions and Interrupts
Administrivia

• Project 3 (individual) due Sunday 8/5
• Final Review – Friday 8/3, 3-6pm in 306 Soda
• Final – Thurs 8/9, 9am-12pm, 245 Li Ka Shing
  – Focus on 2nd half material, though midterm material still fair game
  – MIPS Green Sheet provided again
  – Two-sided handwritten cheat sheet
    • Can use the back side of your midterm cheat sheet!
• Lecture tomorrow by Raphael
Agenda

- VM Wrap-up
- Administrivia
- Disks
- I/O Basics
- Exceptions and Interrupts
Magnetic Disks

• **Nonvolatile storage**
  – Information stored by magnetizing ferrite material on surface of rotating disk
  – Retains its value without applying power to disk, unlike main memory, which only stores data when power is applied

• **Two Types:**
  – Floppy disks – slower, less dense, removable
  – Hard Disk Drives (HDD) – faster, more dense, non-removable

• **Purpose in computer systems (Hard Drive):**
  – Long-term, inexpensive storage for files
  – Layer in the memory hierarchy beyond main memory
Photo of Disk Head, Arm, Actuator
Disk Device Terminology

- Several platters, with information recorded magnetically on both surfaces (usually)
- Bits recorded in *tracks*, which in turn divided into *sectors* (usually 512 Bytes)
- *Actuator* moves *head* (end of *arm*) over track (“seek”), wait for sector to rotate under head, then read or write
• **Disk Latency** = Seek Time + Rotation Time + Transfer Time + Controller Overhead
  – **Seek Time** depends on number of tracks to move arm and speed of actuator
Disk Device Performance (1/2)

- **Disk Latency** = Seek Time + Rotation Time + Transfer Time + Controller Overhead
  - **Rotation Time** depends on speed of disk rotation and how far sector is from head
Disk Latency = Seek Time + Rotation Time + Transfer Time + Controller Overhead

– Transfer Time depends on size of request and data rate (bandwidth) of disk, which is a function of bit density and RPM
Disk Device Performance (2/2)

• Average distance of sector from head?
• 1/2 time of a rotation
  – 7200 revolutions per minute ⇒ 1 rev/8.33 ms
  – 1/2 rotation (revolution) ⇒ 4.17 ms

• Average no. tracks to move arm?
  – Disk industry standard benchmark:
    • Sum all time for all possible seek distances from all possible tracks / # possible
    • Assumes average seek distance is random

• Size of disk cache can strongly affect performance
  – Cache built into disk system, OS knows nothing
Disk Drive Performance Example

• 7200 RPM drive, 4 ms seek time, 20 MiB/sec transfer rate. Negligible controller overhead. Latency to read 100 KiB file?
  – Rotation time = 4.17 ms (from last slide)
  – Transfer time = 0.1 MiB / 20 (MiB/sec) = 5 ms
  – Latency = 4 + 4.17 + 5 = 13.17 ms
  – Throughput = 100 KiB/13.17 ms = 7.59 MiB/sec

• How do numbers change when reading bigger/smaller file? File fragmented across multiple locations?
Flash Memory

• Microdrives and Flash memory (e.g. CompactFlash) are going head-to-head
  – Both non-volatile (no power, data ok)
  – Flash benefits: durable & lower power
    (no moving parts vs. need to spin μdrives up/down)
  – Flash limitations: finite number of write cycles (wear on the insulating oxide layer around the charge storage mechanism). Most ≥ 100K, some ≥ 1M W/erase cycles.

• How does Flash memory work?
  – NMOS transistor with an additional conductor between gate and source/drain which “traps” electrons. The presence/absence is a 1 or 0.

[en.wikipedia.org/wiki/Flash_memory](en.wikipedia.org/wiki/Flash_memory)
What does Apple put in its iPods?

- Toshiba flash 1, 2GB
- Samsung flash 4, 8GB
- Toshiba 1.8-inch HDD 80, 160GB
- Toshiba flash 8, 16, 32GB

Shuffle  
Nano  
Classic  
Touch
Solid-State Drives

- Data storage devices with same electronic interfaces as HDD, but implemented (usually) with flash

<table>
<thead>
<tr>
<th></th>
<th>HDD</th>
<th>Flash-based SSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Time</td>
<td>~ 12 ms</td>
<td>~ 0.1 ms</td>
</tr>
<tr>
<td></td>
<td>≈ 30M clock cycles</td>
<td>≈ 250K clock cycles</td>
</tr>
<tr>
<td>Relative Power</td>
<td>1</td>
<td>1/3</td>
</tr>
<tr>
<td>Cost</td>
<td>~ $0.05-$0.10 / GB</td>
<td>~ $0.65 / GB</td>
</tr>
</tbody>
</table>

- SSDs are also quieter, lighter, unsusceptible to magnetic fields and fragmentation, and start up faster
Agenda

• VM Wrap-up
• Administrivia
• Disks
• I/O Basics
• Exceptions and Interrupts
Five Components of a Computer

- Control
- Datapath
- Memory
- Input
- Output
Motivation for Input/Output

• I/O is how humans interact with computers
• I/O gives computers long-term memory.
• I/O lets computers do amazing things:

MIT Media Lab “Sixth Sense”

• Computer without I/O like a car without wheels; great technology, but gets you nowhere
I/O Device Examples and Speeds

- I/O speeds: 7 orders of magnitude between mouse and LAN

<table>
<thead>
<tr>
<th>Device</th>
<th>Behavior</th>
<th>Partner</th>
<th>Data Rate (KB/s)</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>Voice output</td>
<td>Output</td>
<td>Human</td>
<td>5.00</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>Magnetic disk</td>
<td>Storage</td>
<td>Machine</td>
<td>10,000.00</td>
</tr>
<tr>
<td>Wireless network</td>
<td>Input or Output</td>
<td>Machine</td>
<td>10,000.00</td>
</tr>
<tr>
<td>Graphics display</td>
<td>Output</td>
<td>Human</td>
<td>30,000.00</td>
</tr>
<tr>
<td>Wired LAN network</td>
<td>Input or Output</td>
<td>Machine</td>
<td>125,000.00</td>
</tr>
</tbody>
</table>

- When discussing transfer rates, use SI prefixes ($10^x$)
What do we need for I/O to work?

1) A way to connect many types of devices
2) A way to control these devices, respond to them, and transfer data
3) A way to present them to user programs so they are useful
Instruction Set Architecture for I/O

• What must the processor do for I/O?
  – Input: reads a sequence of bytes
  – Output: writes a sequence of bytes

• Some processors have special input and output instructions

• Alternative model (used by MIPS):
  – Use loads for input, stores for output (in small pieces)
  – Called *Memory Mapped Input/Output*
  – A portion of the address space dedicated to communication paths to Input or Output devices (no memory there)
Memory Mapped I/O

- Certain addresses are not regular memory
- Instead, they correspond to registers in I/O devices

```
address
0xFFFFFFFF
0xFFFF0000
```

```
control reg.
data reg.
```

0
Processor-I/O Speed Mismatch

• 1 GHz microprocessor can execute 1 billion load or store instr/sec (4,000,000 KB/s data rate)
  – Recall: I/O devices data rates range from 0.01 KB/s to 125,000 KB/s

• Input: Device may not be ready to send data as fast as the processor loads it
  – Also, might be waiting for human to act

• Output: Device not be ready to accept data as fast as processor stores it

• What can we do?
Processor Checks Status Before Acting

• Path to a device generally has 2 registers:
  • *Control Register* says it’s OK to read/write (I/O ready)
  • *Data Register* contains data

1) Processor reads from control register in a loop, waiting for device to set *Ready bit* (0 → 1)

2) Processor then loads from (input) or writes to (output) data register
   – Resets Ready bit of control register (1 → 0)

• This process is called “*Polling*”
I/O Example (Polling in MIPS)

• **Input:** Read from keyboard into $v0

```
lui $t0, 0xffff # ffff0000
Waitloop:  lw $t1, 0($t0) # control reg
          andi $t1,$t1,0x1
          beq $t1,$zero, Waitloop
lw $v0, 4($t0) # data reg
```

• **Output:** Write to display from $a0

```
lui $t0, 0xffff # ffff0000
Waitloop:  lw $t1, 8($t0) # control reg
          andi $t1,$t1,0x1
          beq $t1,$zero, Waitloop
          sw $a0,12($t0) # data reg
```

• “Ready” bit is from processor’s point of view!

7/31/2012  

Summer 2012 -- Lecture #25
Cost of Polling?

• Processor specs: 1 GHz clock, 400 clock cycles for a polling operation (call polling routine, accessing the device, and returning)

• Determine % of processor time for polling:
  – **Mouse**: Polled 30 times/sec so as not to miss user movement
  – **Floppy disk**: Transferred data in 2-Byte units with data rate of 50 KB/sec. No data transfer can be missed.
  – **Hard disk**: Transfers data in 16-Byte chunks and can transfer at 16 MB/second. Again, no transfer can be missed.
% Processor time to poll

• Mouse polling:
  – *Time taken:* 30 [polls/s] × 400 [clocks/poll] = 12K [clocks/s]
  – *% Time:* \(1.2 \times 10^4\) [clocks/s] / \(10^9\) [clocks/s] = 0.0012%
  – Polling mouse little impact on processor

• Disk polling:
  – *Time taken:* 1M [polls/s] × 400 [clocks/poll] = 400M [clocks/s]
  – *% Time:* \(4 \times 10^8\) [clocks/s] / \(10^9\) [clocks/s] = 40%
  – Unacceptable!

• **Problems:** polling, accessing small chunks
Alternatives to Polling?

• Wasteful to have processor spend most of its time “spin-waiting” for I/O to be ready

• Would like an unplanned procedure call that would be invoked only when I/O device is ready

• **Solution:** Use *exception* mechanism to help trigger I/O, then *interrupt* program when I/O is done with data transfer
  
  — This method is discussed next
Get To Know Your Instructor
Agenda

• VM Wrap-up
• Administrivia
• Disks
• I/O Basics
• Exceptions and Interrupts
Exceptions and Interrupts

• “Unexpected” events requiring change in flow of control
  – Different ISAs use the terms differently

• **Exception**
  – Arises within the CPU
    (e.g. undefined opcode, overflow, syscall, TLB Miss)

• **Interrupt**
  – From an external I/O controller

• Dealing with these without sacrificing performance is difficult!
Handling Exceptions (1/2)

• In MIPS, exceptions managed by a System Control Coprocessor (CP0)
• Save PC of offending (or interrupted) instruction
  – In MIPS: save in special register called Exception Program Counter (EPC)
• Save indication of the problem
  – In MIPS: saved in special register called Cause register
  – In simple implementation, might only need 1-bit (0 for undefined opcode, 1 for overflow)
• Jump to exception handler code at address 0x80000180
Handling Exceptions (2/2)

• Operating system is also notified
  – Can kill program (e.g. segfault)
  – For I/O device request or syscall, often switch to another process in meantime
    • This is what happens on a TLB misses and page faults
Exception Properties

• Re-startable exceptions
  – Pipeline can flush the instruction
  – Handler executes, then returns to the instruction
    • Re-fetched and executed from scratch

• PC+4 saved in EPC register
  – Identifies causing instruction
  – PC+4 because it is the available signal in a pipelined implementation
    • Handler must adjust this value to get right address
Handler Actions

• Read Cause register, and transfer to relevant handler

• OS determines action required:
  – If restartable exception, take corrective action and then use EPC to return to program
  – Otherwise, terminate program and report error using EPC, Cause register, etc. (e.g. our best friend the segfault)
Exceptions in a Pipeline

• Another kind of control hazard
• Consider overflow on add in EX stage
  add $1, $2, $1
  1) Prevent $1 from being clobbered
  2) Complete previous instructions
  3) Flush add and subsequent instructions
  4) Set Cause and EPC register values
  5) Transfer control to handler
• Similar to mispredicted branch
  – Use much of the same hardware
Exception Example

Time (clock cycles)

Instr. Order

and

or

add

slt

lw

lui
Exception Example

**Time (clock cycles)**

- **Flush add, slt, lw**
- **Save PC+4 into EPC**

1st instruction of handler
Multiple Exceptions

• Pipelining overlaps multiple instructions
  – Could have multiple exceptions at once!
  – e.g. page fault in \texttt{lw} the same clock cycle as overflow of following instruction \texttt{add}

• **Simple approach:** Deal with exception from *earliest* instruction and flush subsequent instructions
  – Called *precise exceptions*
  – In previous example, service \texttt{lw} exception first

• What about multiple issue or out-of-order execution?
  – Maintaining precise exceptions can be difficult!
Imprecise Exceptions

• Just stop pipeline and save state
  – Including exception cause(s)

• Let the software handler work out:
  – Which instruction(s) had exceptions
  – Which to complete or flush
    • May require “manual” completion

• Simplifies hardware, but more complex handler software
  – Not feasible for complex multiple-issue out-of-order pipelines to always get exact instruction

• All computers today offer precise exceptions—affects performance though
I/O Interrupt

• An I/O interrupt is like an exception except:
  – An I/O interrupt is “asynchronous”
  – More information needs to be conveyed

• “Asynchronous” with respect to instruction execution:
  – I/O interrupt is not associated with any instruction, but it can happen in the middle of any given instruction
  – I/O interrupt does not prevent any instruction from running to completion
Interrupt-Driven Data Transfer

1. I/O interrupt
2. Save PC
3. Jump to interrupt service routine
4. Perform transfer
5. Store...

Memory

User program

Interrupt service routine

add
sub
and
or
read
store
jr
Interrupt-Driven I/O Example (1/2)

• Assume the following system properties:
  – 500 clock cycle overhead for each transfer, including interrupt
  – Disk throughput of 16 MB/s
  – Disk interrupts after transferring 16 B
  – Processor running at 1 GHz

• If disk is active 5% of program, what % of processor is consumed by the disk?
  – \(5\% \times 16 \text{ [MB/s]} / 16 \text{ [B/inter]} = 50,000 \text{ [inter/s]}\)
  – \(50,000 \text{ [inter/s]} \times 500 \text{ [clocks/inter]} = 2.5\times10^7 \text{ [clocks/s]}\)
  – \(2.5\times10^7 \text{ [clocks/s]} / 10^9 \text{ [clock/s]} = 2.5\% \text{ busy}\)
Interrupt-Driven I/O Example (2/2)

- 2.5% busy (interrupts) much better than 40% (polling)
- **Real Solution:** *Direct Memory Access (DMA)* mechanism
  - Device controller transfers data directly to/from memory without involving the processor
  - Only interrupts once per page (large!) once transfer is done
Summary

• Disks work by positioning head over spinning platters
  – Very slow relative to CPU, flash memory is alternative

• I/O gives computers their 5 senses + long term memory
  – I/O speed range is 7 orders of magnitude (or more!)

• Processor speed means must synchronize with I/O devices before use:
  – Polling works, but expensive due to repeated queries

• Exceptions are “unexpected” events in processor

• Interrupts are asynchronous events that are often used for interacting with I/O devices