CS61C
Review of Cache/VM/TLB

Lecture 27

May 5, 1999 (Cinco de Mayo)

Dave Patterson
(http.cs.berkeley.edu/~patterson)

www-inst.eecs.berkeley.edu/~cs61c/schedule.html
Outline

° Review Pipelining
° Review Interrupt/Polling Review slides
° Administrivia, “What’s this Stuff Good for?”

° Conclusion
The Principle of Locality:

- Program access a relatively small portion of the address space at any instant of time.
  - Temporal Locality: Locality in Time
  - Spatial Locality: Locality in Space

3 Major Categories of Cache Misses:

- **Compulsory Misses**: sad facts of life. Example: cold start misses.
- **Capacity Misses**: increase cache size
- **Conflict Misses**: increase cache size and/or associativity.
Review 2/3: Cache/VM/TLB

- Caches, TLBs, Virtual Memory all understood by examining how they deal with 4 questions:
  1) Where can block be placed?
  2) How is block found?
  3) What block is replaced on miss?
  4) How are writes handled?

- Page tables map virtual address to physical address

- TLBs are important for fast translation

- TLB misses are significant in processor performance
Virtual memory was controversial at the time: can SW automatically manage 64KB across many programs?

- 1000X DRAM growth removed controversy

Today VM allows many processes to share single memory without having to swap all processes to disk; **VM protection today is more important than memory hierarchy**

Today CPU time is a function of (ops, cache misses) vs. just f(ops): What does this mean to Compilers, Data structures, Algorithms?
I/O Review Slide

° I/O gives computers their 5 senses

° I/O speed range is million to one
  • Mouse, keyboard, network, disk, display

° Processor speed means must synchronize with I/O devices before use
Problem: How CPU Synch. with I/O device?

° Polling also called Programmed I/O

° Advantage: Simple - the processor is totally in control and does all the work
Problems with Polling

° Polling overhead can consume a lot of CPU time when waiting for I/O device
  • busy wait loop not an efficient way to use the CPU unless the device is very fast!

° If not sure when need to do I/O, then lots of processor time spent when could be doing something else useful

° Solution: I/O Interrupt
Why I/O Interrupt?

- Advantage: User program progress is only halted during actual transfer

- An I/O interrupt is like exception except:
  - An I/O interrupt is asynchronous
  - Further information needs to be conveyed

- An I/O interrupt is asynchronous with respect to instruction execution:
  - I/O interrupt is not associated with any instruction
  - I/O interrupt does not prevent any instruction from completion
    - CPU picks convenient point to take interrupt
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

PC saved

Save registers...

lw $r1,20($r0)
lw $r2,0($r1)
addi $r3,$r0,#5
sw $r3,0($r1)

... Restore PC

Restore registers

Clear current Int

‘Interrupt Handler’

External Interrupt
Review: Steps in Executing MIPS (Lec. 20)

1) Ifetch: Fetch Instruction, Increment PC
   • Page fault/Access fault on Instruction fetch?

2) Decode Instruction, Read Registers
   • Undefined Opcode?

3) Execute: Perform operation
   • Overflow?

4) Memory: read or write memory
   • Page fault/Access fault on Data access?

5) Write Back: Write Data to Register
   • I/O interrupts?
Administrivia

- Everything but last 2 projects, last 2 homeworks on grade record is correct?
  - (We hope) graded last 2 homeworks, last 2 projects handed out in discussion this week; See Kelvin ASAP about disagreements

- Should have already filled out final survey to help future 61c; how many? haven’t?

- Friday 61C Summary / Your Cal heritage / Cal v. Stanford CS education / HKN Evaluation

**Wed 5/12 Final 5-8PM in 1 Pimintel**

- Bring 2 sheets, both sides, #2 pencils
- Sun 5/9 Final Review starting 2PM (1 Pimintel)
What’s it Good For: Sony Playstation 2000

- **Emotion Engine:** 6.2 GFLOPS, 75 million polygons per second (Microprocessor Report, 13:5)
  - Superscalar MIPS core + vector coprocessor
  - Claim: *Toy Story* realism brought to games!

*Figure 1. PlayStation 2000 employs an unprecedented level of parallelism to achieve workstation-class 3D performance.*

*Figure 2. PlayStation 2000 screenshot. (Source: Namco)*
Problems with I/O Interrupts

° I/O interrupt is more complicated than exception:
  • Needs to convey the identity of the device generating the interrupt

° 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)

° Where add special interrupt instructions, registers to instruction set?

° What prevents interrupt from occurring during interrupt handler?
Review Coprocessor Registers

° Coprocessor 0 Registers:

<table>
<thead>
<tr>
<th>name</th>
<th>number</th>
<th>usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BadVAddr</td>
<td>$8</td>
<td>Bad Virtual memory Address</td>
</tr>
<tr>
<td>Status</td>
<td>$12</td>
<td>Interrupt enable</td>
</tr>
<tr>
<td>Cause</td>
<td>$13</td>
<td>Exception type</td>
</tr>
<tr>
<td>EPC</td>
<td>$14</td>
<td>Instruction address</td>
</tr>
</tbody>
</table>

• Different registers from integer registers, just as Floating Point is another set of registers independent from integer registers
Turn off interrupts? Interrupt Enable Bit

- Bit in Status Register determines whether or not interrupts enabled:
  **Interrupt Enable bit (IE)** (0 ⇒ off, 1 ⇒ on)

- Also Kernel/User bit to support Virtual Memory modes

<table>
<thead>
<tr>
<th>(described later)</th>
<th>KU</th>
<th>IE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Register</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problems with Interrupt Enable

◦ Interrupt requests can have different urgencies

◦ Conventionally, from highest level to lowest level exception/interrupt levels:
  1) Bus error
  2) Illegal Instruction/Address trap
  3) High priority I/O Interrupt (fast response)
  4) Low priority I/O Interrupt (slow response)

◦ Alternative to blocking all interrupts?
  • Interrupt request needs to be prioritized
Prioritizing Interrupts: Interrupt Mask

- Categorize interrupts and exceptions into levels, and allow selective interruption via **Interrupt Mask (IM)** in Status Register: 5 for HW interrupts
  - Interrupt only if IE == 1 AND Mask bit == 1

- How support interruption of lower priority interrupts?
Interrupt levels

° Suppose there was an interrupt while the interrupt enable or mask bit is off: what should you do? (cannot ignore)

° **Cause** register has field--**Pending Interrupts (PI)**-- 5 bits wide (bits15:10) for each of the 5 HW interrupt levels

  • Bit becomes 1 when an interrupt at its level has occurred but not yet serviced

  • Interrupt routine checks pending interrupts ANDed with interrupt mask to decide what to service
Prioritizing Interrupts: Interrupt Mask

- To support interrupts of interrupts, have 3 deep stack in Status for IE,K/U bits:
  - Current (1:0), Previous (3:2), Old (5:4)

How is MIPS software organized to take advantage of hardware priority scheme?
Interrupt Levels in MIPS Software

° Conventionally, UNIX software system designed to have 4 to 6 Interrupt Priority Levels (IPL) that match the HW interrupt levels

° Processor always executing at one IPL, stored in a memory location and Status Register set accordingly
  • Processor at lowest IPL level, any interrupt accepted
  • Processor at highest IPL level, all interrupt ignored
  • Interrupt handlers and device drivers pick IPL to run at, faster response for some
Handling Prioritized Interrupts

° OS convention to simplify software:
  • Process cannot be preempted by interrupt at same or lower level
  • Return to interrupted code as soon as no more interrupts at a higher level
  • Any piece of code is always run at same priority level

° How write interrupt routine so that it can be interrupted?
Re-entrant Interrupt Routine?

- How allow interrupt of interrupts and safely save registers?

- Stack?
  - Resources consumed by each exception, so cannot tolerate arbitrary deep nesting of exceptions/interrupts

- With priority level system only interrupted by higher priority interrupt, so cannot be recursive

- Only need one interrupt save area ("exception frame") per priority level
Problems with CPU transferring data

° Typical I/O devices must transfer large amounts of data to memory of processor:
  • Disk must transfer complete block (4 KB? 16 KB?)
  • Large packets from network
  • Regions of frame buffer
° Can tie up processor depending on amount of I/O requests
Delegating I/O Responsibility from CPU: DMA

Direct Memory Access (DMA):
- External to the CPU
- Transfer blocks of data to or from memory without CPU intervention

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

DMA Controller (DMAC) provides signals for Peripheral Controller, and Memory Addresses and signals for Memory.
Why DMA?

- DMA gives external device ability to write memory directly: much lower overhead than having processor request one word at a time
Problems with DMA

° What if I/O devices write data that is currently in processor Cache?
  • The processor may never see new data!
  • Called “Cache coherence” problem

° Solutions:
  • Flush cache on every I/O operation (expensive)
  • Have hardware invalidate cache lines of potential address conflicts
Problems with DMA

° Virtual Address or Physical Address?
  1) If virtual address, how do address translation, since memory uses physical addresses?
  2) If physical address, what happens if when cross a page boundary, as virtual memory may not be contiguous in physical memory?

° Solutions:
  1) Give DMA a small number of address translations, done by OS when start DMA
  2) Have a list of blocks, each no larger than a page, chained together
Why use OS for I/O?

° The operating system acts as the interface between:
  • The I/O hardware and the program that requests I/O

° The Operating System must be able to prevent:
  • The user program from communicating with the I/O device directly

° If user programs could perform I/O directly:
  • Protection to the shared I/O resources could not be provided
Responsibilities of the Operating System

° Three characteristics of the I/O systems:

• The I/O system is shared by multiple programs using the processor

• I/O systems often use interrupts to communicate information about I/O operations.
  - Interrupts must be handled by the OS because they cause a transfer to supervisor mode

• The low-level control of an I/O device is complex:
  - Managing a set of concurrent events
  - The requirements for correct device control are very detailed
Operating System Requirements 1/2

° Provide protection to shared I/O resources
  • Guarantees that a user’s program can only access the portions of an I/O device to which the user has rights

° Provides abstraction for accessing devices:
  • Supply routines that handle low-level device operation

° Handles the interrupts generated by I/O devices
Operating System Requirements 2/2

- Provide equitable access to the shared I/O resources
  - All user programs must have equal access to the I/O resources

- Schedule accesses in order to enhance system throughput
How Protect I/O?

- MIPS memory maps I/O devices to allow load-store access to send commands, receive status and data

- To prevent user program from accessing data despite having a 32-bit virtual address, need protection

- (See above) MIPS CPU runs in 2 privilege levels: user mode and kernel mode
  - User mode: limited to bottom half of 32-bit virtual address
  - Kernel mode: can access full 32-bit virtual address; special areas to enable booting machine before TLB valid
Drawing of MIPS Process Memory Allocation

Address
\[ \infty \left(2^{32}-1\right) \]

I/O Regs
I/O device registers

OS code/data space

\[ \infty \left(2^{31}\right) \]

Exception Handlers

\[ \infty \left(2^{31}-1\right) \]

Stack

User code/data space

\[ \leftarrow \]

Heap

\[ \leftarrow \]

Static

\[ \leftarrow \]

Code

• OS restricts I/O Registers, Exception Handlers to OS
In More Depth: Actual MIPS address names

- Virtual address divided into 4 areas:

1) **kuseg** (low 2 GB) - for user mode, always translated via TLB and through cache

2) **kseg0** (next 0.5 GB) - translated by striping off top 1 bit (kernel mode); maps to low 0.5GB of physical memory *via caches*

3) **kseg1** (next 0.5 GB) - translated by striping off top 3 bits (kernel mode); maps to low 0.5GB of physical memory, **not via caches**

4) **kseg2** (top 1 GB) - kernel mode, always translated via TLB and through cache
How User safely invoke Operating System?

° 2 instructions

• **break**: intended to implement break point debugging feature

• **syscall**: intended to ask OS for specific services by passing argument in register
Summary 1/2

° Wide range of devices
  • multimedia and high speed networking poise important challenges

° Delegating data transfer responsibility from the CPU: DMA

° I/O performance limited by weakest link in chain between OS and device

° Operating System started as shared I/O library
Summary 2/2

° 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

° MIPS OS support / Interrupt control:
  • Interrupt Enable bit, stacked IE bits, Interrupt Priority Levels, Interrupt Mask
  • Support for OS abstraction: Kernel/User bit, stacked KU bits, syscall, rfe
  • MIPS follows coprocessor abstraction to add resources, instructions for OS
  • OS Re-entrant via restricting interrupt to higher priority