Last Time: Practical Cache Design

Cache design control is done by many loosely coupled state machines, including ...

State Machine

To CPU

Control

To Lower Level Memory

To CPU

Addr

Din

Dout

Blocks

Tags

Addr

Din

Dout

To Lower Level Memory
State machines for bus control ....

For reads, your state machine must:

1. sense REQ
2. latch Addr
3. create Wait
4. put Data Out on the bus.

An example interface ... there are other possibilities.
State machines for block fetch from DRAM

One request ...

DRAM can be set up to request an N byte region starting at an arbitrary N+k within region

Many returns ...

State machine challenges: (1) setting up correct block read mode (2) delivering correct word direct to CPU (3) putting all words in cache in right place.
State machine for writeback to DRAM

One command ...

Many bytes written

State machine challenges: (1) putting cache block into correct location (2) what if a read or write wants to use DRAM before the burst is complete? Must stall ...
State machines to manage write buffer

Solution: add a “write buffer” to cache datapath

Holds data awaiting write-through to lower level memory

Q. Why a write buffer?  
A. So CPU doesn’t stall

Q. Why a buffer, why not just one register?  
A. Bursts of writes are common.

Q. Are Read After Write (RAW) hazards an issue for write buffer?  
A. Yes! Drain buffer before next read, or check write buffers.

On reads, state machine checks cache and write buffer -- what if word was removed from cache before lower-level write? On writes, state machine stalls for full write buffer, handles write buffer duplicates.
Don’t design one big state machine!!!

Focus on the high-level state machine structure early!

State Machine

Control

Tags

Addr

Din

Dout

Blocks

Addr

Din

Dout

To CPU

To Lower Level Memory

To CPU

To Lower Level Memory
Today’s Lecture - Virtual Memory

- Virtual address spaces
- Page table layout
- TLB design options
- Virtual machines
The Limits of Physical Addressing

“Physical addresses” of memory locations

CPU

D0-D31

A0-A31

Memory

D0-D31

A0-A31

Where we are in CS 152 ...

All programs share one address space:
The physical address space

Machine language programs must be aware of the machine organization

No way to prevent a program from accessing any machine resource
Apple II: A physically addressed machine

Apple ][ (1977)

<table>
<thead>
<tr>
<th>RAM Complement</th>
<th>Apple II System</th>
</tr>
</thead>
<tbody>
<tr>
<td>4K</td>
<td>$1,298.00</td>
</tr>
<tr>
<td>48K</td>
<td>$2,638.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Organization and Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page #</td>
</tr>
<tr>
<td>Dec</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12 through 31</td>
</tr>
<tr>
<td>32 through 63</td>
</tr>
<tr>
<td>64 through 95</td>
</tr>
<tr>
<td>96 through 191</td>
</tr>
<tr>
<td>192 through 199</td>
</tr>
<tr>
<td>193 through 199</td>
</tr>
<tr>
<td>199</td>
</tr>
</tbody>
</table>
The Limits of Physical Addressing

"Physical addresses" of memory locations

All programs share one address space: The physical address space

Machine language programs must be aware of the machine organization

No way to prevent a program from accessing any machine resource
“Virtual Addresses”

User programs run in an standardized virtual address space

Address Translation hardware managed by the operating system (OS) maps virtual address to physical memory

Hardware supports “modern” OS features:
Protection, Translation, Sharing
MIPS R4000: Address Space Model

Process A

- ASID = 12
- Address Space Identifier
- Process A and B have independent address spaces
- Address Error
- 2 GB

Process B

- ASID = 13
- Address Error
- 2 GB

All address spaces use a standard memory map

May only be accessed by kernel/supervisor

When Process A writes its address 9, it writes to a different physical memory location than Process B’s address 9

To let Process A and B share memory, OS maps parts of ASID 12 and ASID 13 to the same physical memory locations.

Still works (slowly!) if a process accesses more virtual memory than the machine has physical memory
System Control Registers

Status (12): Indicates user, supervisor, or kernel mode

EntryLo0 (2): 8-bit ASID field codes virtual address space ID.

User cannot write supervisor/kernel bits. Supervisor cannot write kernel bit.

User cannot change address translation configuration
MIPS Address Translation: How it works

Translation Look-Aside Buffer (TLB)
A small fully-associative cache of mappings from virtual to physical addresses.

TLB also contains ASID and kernel/supervisor bits for virtual address.

Fast common case: Virtual address is in TLB, process has permission to read/write it.
Page tables encode virtual address spaces

OS manages the page table for each ASID

Page Table (One per ASID)

Virtual address

Physical Memory Space

frame

frame

frame

frame

A virtual address space is divided into blocks of memory called **pages**

A machine usually supports pages of a few sizes (MIPS R4000):

<table>
<thead>
<tr>
<th>Page Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Kbytes</td>
</tr>
<tr>
<td>16 Kbytes</td>
</tr>
<tr>
<td>64 Kbytes</td>
</tr>
<tr>
<td>256 Kbytes</td>
</tr>
<tr>
<td>1 Mbyte</td>
</tr>
<tr>
<td>4 Mbytes</td>
</tr>
<tr>
<td>16 Mbytes</td>
</tr>
</tbody>
</table>

A page table is indexed by a virtual address

A valid page table entry codes **physical memory “frame”** address for the page
The TLB caches page table entries

In this example, physical and virtual pages must be the same size!

MIPS handles TLB misses in software (random replacement). Other machines use hardware.
Page tables may not fit in memory!

A table for 4KB pages for a 32-bit address space has 1M entries

Each process needs its own address space!

Two-level Page Tables

32 bit virtual address

Top-level table wired in main memory

Subset of 1024 second-level tables in main memory; rest are on disk or unallocated
What if a page resides on disk?

TLB caches page table entries. Virtual Address

V page no.

offset

Page Table for ASID

index into page table

table located in physical memory

P page no.

offset

Physical frame address

Physical Address

V=0 pages either reside on disk or have not yet been allocated. OS handles V=0

"Page fault"

Question: What to do when a TLB miss causes an access to a page table entry with V=0?
VM and Disk: Page replacement policy

Set of all pages in Memory

Head pointer
Place pages on free list if used bit is still clear.
Schedule pages with dirty bit set to be written to disk.

Tail pointer:
Clear the used bit in the page table

Page Table

<table>
<thead>
<tr>
<th>dirty</th>
<th>used</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Dirty bit: page written.
Used bit: set to 1 on any reference

On page fault: deallocate page table entry of a page on the free list.

Freelists

Architect’s role: support setting dirty and used bits
Define the timing diagrams and signal names for the IM, DM, IC, DC buses.

List the bugs you will target in test benches.

Other items ...

Also: Lab 3 Peer Evaluations ...
TLB Design Concepts
MIPS R4000 TLB: A closer look …

“Virtual Addresses”

CPU

A0-A31

D0-D31

“Physical Addresses”

Translation Look-Aside Buffer (TLB)

Virtual Physical

Memory System

A0-A31

D0-D31

Data

Checked against CPO ASID

Physical space larger than virtual space!

Virtual Address with 1M (2^20) 4-Kbyte pages

39 32 31 29 28 20 bits = 1M pages 12 11 0

VPN

Offset

36-bit Physical Address

tlb

35 PFN Offset

0

Bits 31, 30 and 29 of the virtual address select user, supervisor, or kernel address spaces.
Can TLB and caching be overlapped?

This works, but ...

Q. What is the downside?

A. Inflexibility. VPN size locked to cache tag size.
Can we cache virtual addresses?

“Virtual Addresses”

CPU

A0-A31

D0-D31

Cache

Virtual

D0-D31

“Physical Addresses”

Translation

Look-Aside

Buffer

(TLB)

Main Memory

A0-A31

D0-D31

Virtual

Physical

Only use TLB on a cache miss!

Downside: a subtle, fatal problem. What is it?

A. Synonym problem. If two address spaces share a physical frame, data may be in cache twice. Maintaining consistency is a nightmare.
Virtual Memory Recap

- VM: Uniform memory models, protection, sharing.
- A TLB acts as a fast cache for recent address translations.
- Operating systems manage the page table and (often) the TLB.
Running Windows on a Mac

Depends on the meaning of the word “run” ...
Method #1: Boot Camp

Basic Idea: New Macs use Intel CPU and support chips. So, set up boot ROM to let you choose Win or OS X.

+++ Great compatibility. Just add device drivers.

+++ No performance hit: full-speed, use all RAM, etc.

--- Must reboot to change OS.

--- Sharing files between OS partitions securely is tricky.
Method #2: Run WINE on OS X

Basic Idea: Emulate the Windows API in software running under OS X. Lets you run Windows apps in a “compatibility box” without running Windows.

+++ Do not need to buy Windows.
+++ No reboot to run Win-app.
--- Slow.
--- Chances are, the app you want to run has compatibility woes.
Method 3: Virtual PC

**Basic Idea:** Make a software emulation of PC hardware, that runs as a process under OS X. Boot Windows and run apps on the emulator.

+++ Runs on PowerPC Macs.
+++ Good compatibility.
Easier to emulate CPU than Win API.

--- Must buy Windows.
--- Slow
Emulating a PC --> emulating everything!

- Windows expects to see raw disks, so VirtualPC has **Virtual Disks**.
- Windows expects to set up a graphics card, so VirtualPC has a **Virtual GPU**.
- Windows expects to manipulate page tables, so VirtualPC has **Virtual TLB**.
- Windows expects to configure network: **Virtual Ethernet Card**.

Like the movie “The Truman Show” ... no wonder it's slow.
Method 4: Parallels, a Virtual Machine

Basic Idea: Like emulating a PC, but different. Use an Intel-based Mac, runs on top of OS X. Uses hardware support to create a fast virtual PC that boots Windows.

+++ Reasonable performance.

Virtual CPU runs 33% slower than running on physical CPU.

2 GB physical memory for a 512 MB virtual PC to run w/o disk swaps.

Source: http://www.atpm.com/12.10/parallels.shtml
“Hardware assist?” What do we mean?

In an emulator, we run Windows code by simulating the CPU in software.

In a virtual machine, we let “safe” instructions (ex: ADD R3 R2 R1) run on the actual hardware.

We use hardware features to deny direct execution of instructions that could “break out” of the virtual machine.

We have seen an example of this sort of hardware feature earlier today ...
Recall: A LW that misses TLB launches a program!

Virtual Address

\[ \text{V page no.} \quad \text{offset} \]

Page Table

\[ \text{Page Table} \]

\[ \text{Page Table for ASID} \]

\[ \text{index into page table} \]

Physical Address

\[ \text{P page no.} \quad \text{offset} \]

Physical frame address

MIPS handles TLB misses in software (random replacement). Other machines use hardware.

TLB caches page table entries. virtual address

Page Table

\[ \text{Page Table} \]

\[ \text{page} \quad \text{off} \]

Physical Address

\[ \text{V=0 pages either reside on disk or have not yet been allocated. OS handles V=0 "Page fault"} \]

In this example, physical and virtual pages must be the same size!

TLB

\[ \text{frame} \quad \text{page} \]

\[ \text{2} \quad \text{2} \]

\[ \text{0} \quad \text{5} \]
General Mechanism: Instruction Trap

Conceptually, we set CPU up to rewrite “unsafe” instructions with a function call.

Sample Program

<table>
<thead>
<tr>
<th>Instruction</th>
<th>CPU Does</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND R2, R2, R1</td>
<td>AND R2, R2, R1</td>
</tr>
<tr>
<td>ADD R4, R3, R2</td>
<td>ADD R4, R3, R2</td>
</tr>
<tr>
<td>UNSAFE</td>
<td>JAL UNSAFE_STUB</td>
</tr>
<tr>
<td>ADD R5, R2, R2</td>
<td>NOP</td>
</tr>
</tbody>
</table>

CPU rewrite instructions??? We have already done this for pipelining....
Recall: “Muxing in” NOPS to do stalls

Sample program
ADD R4, R3, R2
OR R5, R4, R2

Keep executing OR instruction until R4 is ready. Until then, send NOPS to IR 2/3.

Freeze PC and IR until stall is over.

New datapath hardware
(1) Mux into IR 2/3 to feed in NOP.
(2) Write enable on PC and IR 1/2
Parallels is extra special.

Why? It runs “on top of” Mac OS X, and was written by a third-party company (Parallels). Uses recent Intel hardware features to make this happen.

To explain the art of virtual machines, we look at

OS-like program that turns a hardware PC into many virtual PCs.
Guest mini-lecture by CS 152 alumni Fred Jiang

Being videotaped for Dave Patterson’s education course.
Xiaofan Jiang  
Electrical Engineering and Computer Sciences  
University of California, Berkeley  

http://www.eecs.berkeley.edu/~fxjiang/  
* Slides borrowed heavily from David Patterson
Introduction to Virtual Machines

- VMs developed in late 1960s
  - Important in mainframe computing
  - Largely ignored in single user computers of 1980s and 1990s

- Recently regained popularity due to
  - Isolation and security
  - Failures in security and reliability of OS
  - Multiple users
  - Increases in raw speed of processors
**Virtual Machines** provide a complete system level environment at binary ISA

- Assume ISAs match the native hardware ISA

  - E.g., IBM VM/370, VMware ESX Server, and Xen

- Present illusion that VM users have entire computer to themselves, including a copy of OS

- Single computer runs multiple VMs, and can support a multiple, different OSes

- **Host vs guest**
Virtual Machine Monitors (VMMs)

- Virtual machine monitor (VMM) / hypervisor
- VMM maps virtual resources to physical resources
- Physical resource may be time-shared, partitioned, or emulated in software
- VMM is much smaller than a traditional OS
VMM Overhead?

- Depends on the workload
- **Processor-bound** programs (e.g., SPEC)
- **I/O-intensive** workloads $\Rightarrow$ **OS-intensive** $\Rightarrow$ Many system calls and privileged instructions
- What if I/O-intensive workload is also **I/O-bound**?
Other Uses of VMs

- Protection
- Managing Software
- Managing Hardware
  - VMs allow separate SW stacks to run independently yet share HW, thereby consolidating number of servers
  - Migrate running VM to a different computer (why?)
Requirements of a VMM

- A VM Monitor
  - Isolates state of guests from each other, and
  - Protects itself from guest software
- Guest software should behave on a VM exactly as if running on the native HW (except?)
- Guest software should not be able to change allocation of real system resources directly
- VMM must control ≈ everything
  - Access to privileged state, address translation, I/O, exceptions and Interrupts, ...
Requirements of a VMM

- Privilege level of VMM > guest VM (user mode)
- Execution of privileged instructions handled by VMM
- At least 2 processor modes, system and user
- Privileged subset of instructions available only in system mode, trap if executed in user mode
- All system resources controllable only via these instructions
ISA Support for Virtual Machines

- Add VM support into ISA?
  - E.g. Intel’s VTx, AMD’s Pacifica

- VMM must ensure that guest system only interacts with virtual resources
  - If guest OS attempts to access or modify information related to HW resources via a privileged instruction—e.g., reading or writing the page table pointer—it will trap to the VMM

- If not, VMM must intercept instruction and support a virtual version of the sensitive information as the guest OS expects
Virtualization of virtual memory

VMM separates **physical** and **machine** memory

- Guest OS maps virtual memory to physical memory via its page tables, and VMM page tables map physical memory to machine memory

- **Shadow page table** maps directly from the guest virtual address space to HW physical address space

- VMM must trap any attempt by guest OS to change its page table or to access the page table pointer
Memory Allocation Policy*

What is fair?

*VMware (ESX Server)
Memory Sharing Among VMs*

- Multiple VMs running same OS, apps
- Collapse redundant copies of code, data, zeros

*VMware (ESX Server)
Impact of I/O on VM

Most difficult part of virtualization

Why?

Give each VM generic versions of each type of I/O device driver, and let VMM to handle real I/O

Method for mapping virtual to physical I/O device depends on the type of device:

- Disks?
- Network interfaces?
Example: Xen VM

- Xen: Open-source System VMM for 80x86 ISA
- “Paravirtualization” – small modifications to guest OS to simplify virtualization
- Examples of paravirtualization in Xen:
  - Guest OS allowed to allocate pages, just check that didn’t violate protection restrictions
  - Xen takes advantage of 4 protection levels available in 80x86
Xen changes for paravirtualization

- Port of Linux to Xen changed ≈ 3000 lines, or ≈ 1% of 80x86-specific code
- Does not affect application-binary interfaces of guest OS
- OSes supported in Xen 2.0

<table>
<thead>
<tr>
<th>OS</th>
<th>Runs as host OS</th>
<th>Runs as guest OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux 2.4</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linux 2.6</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>NetBSD 2.0</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>NetBSD 3.0</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Plan 9</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>FreeBSD 5</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Xen Performance

Performance relative to native Linux for Xen for 6 benchmarks from Xen developers

- SPEC INT2000
- PostgreSQL OLTP
- SPEC WEB99
- Linux build time
- dbench
- PostgreSQL Inf. Retrieval
Conclusion

- VM Monitor presents the illusion that VM users have entire computer to themselves, isolates state of guests, and protects itself from guest software (including OS)
- Virtual machine revival - {overcome security flaws of large OSes, manage software, manage hardware, processor performance no longer highest priority}
- Virtualization challenges for processor, virtual memory, and I/O
- Xen as example VMM using paravirtualization
Learn whatever lesson you need to learn from your grade, and then move on.

Mid-Term Grades

When driving, do not focus most of your attention on the rear view mirror.

Most of the time, there is nothing interesting behind you.
Mid-Term 1: Grade Distribution

I handle all regrade requests.
Regrade deadline: Thursday Oct 26, 1PM
Detecting and correcting RAM bit errors

Replacing lost network packets, recovering from disk drive failure

Detecting arbitrary bit errors in network packets