Recall: Cache Performance

Execution Time = Instruction Count x Cycle Time x (ideal CPI + Memory Stalls/Inst + Other Stalls/Inst)

Memory Stalls/Inst = Instruction Miss Rate x Instruction Miss Penalty + Loads/Inst x Load Miss Rate x Load Miss Penalty + Stores/Inst x Store Miss Rate x Store Miss Penalty

Average Memory Access time (AMAT) = Hit Time L1 + (Miss Rate L1 x Miss Penalty L1) = (Hit Rate L1 x Hit Time L1) + (Miss Rate L1 x Miss Time L1)

Average Memory Access time = Hit Time + (Miss Rate x Miss Penalty)

Recall: Cache techniques

° 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?

° Techniques people use to improve the miss rate of caches:
  
<table>
<thead>
<tr>
<th>Technique</th>
<th>MR</th>
<th>MP</th>
<th>HT</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Larger Block Size</td>
<td>+</td>
<td>–</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Higher Associativity</td>
<td>+</td>
<td>–</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Victim Caches</td>
<td>+</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pseudo-Associative Caches</td>
<td>+</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW Prefetching of Inst/Data</td>
<td>+</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler Controlled Prefetching</td>
<td>+</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Compiler Reduce Misses</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Recall: Reducing Misses via a “Victim Cache”

° How to combine fast hit time of direct mapped yet still avoid conflict misses?
° Add buffer to place data discarded from cache
° Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
° Used in Alpha, HP machines
Recall: Second-Level Cache

- **L2 Equations**
  \[
  AMAT = Hit Time_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1}\\
  \text{Miss Penalty}_{L1} = Hit Time_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}\\
  AMAT = Hit Time_{L1} + \text{Miss Rate}_{L1} \times (Hit Time_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})
  \]

- **Definitions:**
  - **Local miss rate**—misses in this cache divided by the total number of memory accesses to this cache (Miss Rate L2)
  - **Global miss rate**—misses in this cache divided by the total number of memory accesses generated by the CPU (Miss Rate L1 x Miss Rate L2)
  - Global Miss Rate is what matters

Recall: Harvard Architecture

- **Sample Statistics:**
  - 16KB I&D: Inst miss rate=0.64%, Data miss rate=6.47%
  - 32KB unified: Aggregate miss rate=1.99%

- Which is better (ignore L2 cache)?
  - Assume 33% loads/store, hit time=1, miss time=50
  - Note: data hit has 1 stall for unified cache (only one port)

\[
AMAT_{filed}=\\(\frac{1/1.33}{1+0.64\%\times50}+\\(0.33/1.33)\times(1+6.47\%\times50) = 2.05 \\
AMAT_{unified}=\\(\frac{1/1.33}{1+1.99\%\times50}+(0.33/1.33)\times(1+1+1.99\%\times50) = 2.24
\]

Recall: Levels of the Memory Hierarchy

- **Virtual memory** => treat memory as a cache for the disk
- **Terminology:** blocks in this cache are called “Pages”
  - Typical size of a page: 1K — 8K
  - Page table maps virtual page numbers to physical frames
    - “PTE” = Page Table Entry

- **Upper Level**
  - Virtual Address Space
  - Physical Address Space
  - Virtual Address
    - V page no.
    - offset
  - Page Table
    - V page no.
    - offset
    - table located in physical memory
    - P page no.
    - offset
  - Physical Address

- **Lower Level**
  - Registers
  - Cache
    - Cache
    - Blocks
    - Memory
      - Pages
      - Disk
        - Files
        - Tape
  - CPU Registers 100s Bytes <10s ns
  - Cache K Bytes 10-100 ns $.01-.001/bit
  - Main Memory M Bytes 100ns-1us $.01-.001
  - Disk G Bytes ms 10^3 - 10 cents
  - Tape infinite sec-min 10^-6
Three Advantages of Virtual Memory

Translation:
- Program can be given consistent view of memory, even though physical memory is scrambled.
- Makes multithreading reasonable (now used a lot!)
- Only the most important part of program (“Working Set”) must be in physical memory.
- Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later.

Protection:
- Different threads (or processes) protected from each other.
- Different pages can be given special behavior (Read Only, Invisible to user programs, etc).
- Kernel data protected from User programs
- Very important for protection from malicious programs
- Far more “viruses” under Microsoft Windows

Sharing:
- Can map same physical page to multiple users (“Shared memory”)

How big is the translation (page) table?

- Simplest way to implement “fully associative” lookup policy is with large lookup table.
- Each entry in table is some number of bytes, say 4
- With 4K pages, 32-bit address space, need: $2^{32}/4K = 2^{20} = 1$ Meg entries x 4 bytes = 4MB
- With 4K pages, 64-bit address space, need: $2^{64}/4K = 2^{52}$ entries = BIG!
- Can’t keep whole page table in memory!

Issues in Virtual Memory System Design

What is the size of information blocks that are transferred from secondary to main storage (M)? ⇒ page size
(Contrast with physical block size on disk, i.e. sector size)

Which region of M is to hold the new block ⇒ placement policy

How do we find a page when we look for it? ⇒ block identification

Block of information brought into M, and M is full, then some region of M must be released to make room for the new block ⇒ replacement policy

What do we do on a write? ⇒ write policy

Missing item fetched from secondary memory only on the occurrence of a fault ⇒ demand load policy

Large Address Spaces

Two-level Page Tables

32-bit address:

<table>
<thead>
<tr>
<th>Virtual Page Number</th>
<th>Page Offset</th>
</tr>
</thead>
</table>

- 2 GB virtual address space
- 4 MB of PTE2
  - paged, holes
- 4 KB of PTE1

What about a 48-64 bit address space?
Inverted Page Tables

IBM System 38 (AS400) implements 64-bit addresses. 48 bits translated start of object contains a 12-bit tag

=> TLBs or virtually addressed caches are critical

Virtual Address and a Cache: Step backward???

° Virtual memory seems to be really slow:
  • Must access memory on load/store -- even cache hits!
  • Worse, if translation not completely in memory, may need to go to disk before hitting in cache!

° Solution: Caching! (surprise!)
  • Keep track of most common translations and place them in a “Translation Lookaside Buffer” (TLB)

Making address translation practical: TLB

° Virtual memory => memory acts like a cache for the disk
° Page table maps virtual page numbers to physical frames
° Translation Look-aside Buffer (TLB) is a cache translations

TLB organization: include protection

° TLB usually organized as fully-associative cache
  • Lookup is by Virtual Address
  • Returns Physical Address + other info
° Dirty => Page modified (Y/N)?
Ref => Page touched (Y/N)?
Valid => TLB entry valid (Y/N)?
Access => Read? Write?
ASID => Which User?
Example: R3000 pipeline includes TLB stages

MIPS R3000 Pipeline

<table>
<thead>
<tr>
<th>Inst Fetch</th>
<th>Dcd/ Reg</th>
<th>ALU / E.A</th>
<th>Memory</th>
<th>Write Reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLB</td>
<td>I-Cache</td>
<td>RF</td>
<td>Operation</td>
<td>WB</td>
</tr>
<tr>
<td>E.A.</td>
<td>TLB</td>
<td>D-Cache</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TLB
- 64 entry, on-chip, fully associative, software TLB fault handler

Virtual Address Space

<table>
<thead>
<tr>
<th>ASID</th>
<th>V. Page Number</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>20</td>
<td>12</td>
</tr>
</tbody>
</table>

0x User segment (caching based on PT/TLB entry)
- 100 Kernel physical space, cached
- 101 Kernel physical space, uncached
- 11x Kernel virtual space

Allows context switching among 64 user processes without TLB flush

Administrivia

° Midterm II: Monday 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?
° Lab 6 due tomorrow night at Midnight
  - Make sure to demo to your TA
° Thursday 5/1 in Section (119 Cory):
  - Discuss plans for final project
  - Problem 0 of Lab 7 due by midnight as well.
° Important: Lab 7. 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 0xFFFFFFFFE0 to 0xFFFFFFFFF
    - 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).

What is the replacement policy for TLBs?

° On a TLB miss, we check the page table for an entry. Two architectural possibilities:
  - Hardware “table-walk” (Sparc, among others)
    - Structure of page table must be known to hardware
  - Software “table-walk” (MIPS was one of the first)
    - Lots of flexibility
    - Can be expensive with modern operating systems.

° What if missing Entry is not in page table?
  - This is called a “Page Fault”
    ⇒ requested virtual page is not in memory
  - Operating system must take over (CS162)
    - pick a page to discard (possibly writing it to disk)
    - start loading the page in from disk
    - schedule some other process to run

° Note: possible that parts of page table are not even in memory (i.e. paged out!)
  - The root of the page table always “pegged” in memory
Page Replacement: Not Recently Used (1-bit LRU, Clock)

Associated with each page is a “used” flag such that:
- used flag = 1 if the page has been referenced in recent past
- = 0 otherwise

- If replacement is necessary, choose any page frame such that its reference bit is 0. This is a page that has not been referenced in the recent past.

Architecture part: support dirty and used bits in the page table
- May need to update PTE on any instruction fetch, load, store

How does TLB affect this design problem? Software TLB miss?

Reducing translation time further
- As described, TLB lookup is in serial with cache lookup:
  - Virtual Address
    - V page no.
    - offset
    - TLB Lookup
      - V access
      - PA
      - P page no.
      - offset
    - Physical Address
      - 10

- Machines with TLBs go one step further: they overlap TLB lookup with cache access.
  - Works because lower bits of result (offset) available early

Overlapped TLB & Cache Access
- If we do this in parallel, we have to be careful, however:
  - Machines with TLBs go one step further: they overlap TLB lookup with cache access.
    - Works because lower bits of result (offset) available early

What if cache size is increased to 8KB?
Problems With Overlapped TLB Access

Overlapped access only works as long as the address bits used to index into the cache do not change as the result of VA translation. This usually limits things to small caches, large page sizes, or high n-way set associative caches if you want a large cache.

Example: suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K:

```
 11 → 2
```

This bit is changed by VA translation, but is needed for cache lookup.

Solutions:
go to 8K byte page sizes; go to 2 way set associative cache; or SW guarantee VA[13]=PA[13]

Cache Optimization: Alpha 21064

- TLBs fully associative
- TLB updates in SW (“Priv Arch Libr”)
- Separate Instr & Data TLB & Caches
- Caches 8KB direct mapped, write thru
- Critical 8 bytes first
- Prefetch instr. stream buffer
- 4 entry write buffer between D$ & L2$
- 2 MB L2 cache, direct mapped, (off-chip)
- 256 bit path to main memory, 4 x 64-bit modules
- Victim Buffer: to give read priority over write

Another option: Virtually Addressed Cache

Only require address translation on cache miss!

- synonym problem: two different virtual addresses map to same physical address => two different cache entries holding data for the same physical address!

- nightmare for update: must update all cache entries with same physical address or memory becomes inconsistent

What is a bus?

A Bus Is:

- shared communication link
- single set of wires used to connect multiple subsystems

- A Bus is also a fundamental tool for composing large, complex systems
  - systematic means of abstraction
**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

**Advantages of Buses**

- It creates a communication bottleneck
  - The bandwidth of that bus 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
- 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
A bus transaction includes two parts:
- Issuing the command (and address)  – request
- Transferring the data  – action

Master is the one who starts the bus transaction by:
- issuing the command (and address)

Slave is the one who responds to the address by:
- Sending data to the master if the master ask for data
- Receiving data from the master if the master wants to send data

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
  - Optimized for cache block transfers
- 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 (standard or proprietary)
  - Backplane: an interconnection structure within the chassis
  - Allow processors, memory, and I/O devices to coexist
  - Cost advantage: one bus for all components

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: IBM PC - AT

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
- 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 (+ 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

### 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

### What is DMA (Direct Memory Access)?

- Typical I/O devices must transfer large amounts of data to memory of processor:
  - Disk must transfer complete block
  - Large packets from network
  - Regions of frame buffer
- DMA gives external device ability to access memory directly, much lower overhead than having processor request one word at a time.
- **Issue:** Cache coherence:
  - What if I/O devices write data that is currently in processor Cache?
    - The processor may never see new data!
  - **Solutions:**
    - Flush cache on every I/O operation (expensive)
    - Have hardware invalidate cache lines (remember “Coherence” cache misses?)

### What defines a bus?

- **Transaction Protocol**
- **Timing and Signaling Specification**
- **Bunch of Wires**
- **Electrical Specification**
- **Physical / Mechanical Characteristics** – the connectors
Summary #1 / 2: Virtual Memory

- VM allows many processes to share single memory without having to swap all processes to disk
- *Translation, Protection, and Sharing* are more important than memory hierarchy
- Page tables map virtual address to physical address
  - TLBs are a cache on translation and are extremely important for good performance
  - Special tricks necessary to keep TLB out of critical cache-access path
  - TLB misses are significant in processor performance:
    - These are funny times: most systems can’t access all of 2nd level cache without TLB misses!

Summary #2 / 2

- Buses are an important technique for building large-scale systems
  - Their speed is critically dependent on factors such as length, number of devices, etc.
  - Critically limited by capacitance
  - Tricks: esoteric drive technology such as GTL
- Important terminology:
  - Master: The device that can initiate new transactions
  - Slaves: Devices that respond to the master
- Two types of bus timing:
  - Synchronous: bus includes clock
  - Asynchronous: no clock, just REQ/ACK strobing
- Direct Memory Access (dma) allows fast, burst transfer into processor’s memory:
  - Processor’s memory acts like a slave
  - Probably requires some form of cache-coherence so that DMA’ed memory can be invalidated from cache.