Review (1/2)
° Optimal Pipeline
• Each stage is executing part of an instruction each clock cycle.
• One instruction finishes during each clock cycle.
• On average, execute far more quickly.
° What makes this work?
• Similarities between instructions allow us to use same stages for all instructions (generally).
• Each stage takes about the same amount of time as all others: little wasted time.

Review (2/2)
° Pipelining a Big Idea: widely used concept
° What makes it less than perfect?
  • Structural hazards: suppose we had only one cache?
    ⇒ Need more HW resources
  • Control hazards: need to worry about branch instructions?
    ⇒ Delayed branch or branch prediction
  • Data hazards: an instruction depends on a previous instruction?

Why Caches?
° Pipelining a Big Idea: widely used concept
° What makes it less than perfect?
  • Structural hazards: suppose we had only one cache?
    ⇒ Need more HW resources
  • Control hazards: need to worry about branch instructions?
    ⇒ Delayed branch or branch prediction
  • Data hazards: an instruction depends on a previous instruction?

Why virtual memory? (1/2)
° Protection
  • regions of the address space can be read only, execute only, . . .
° Flexibility
  • portions of a program can be placed anywhere, without relocation
° Expandability
  • can leave room in virtual address space for objects to grow
° Storage management
  • allocation/deallocation of variable sized blocks is costly and leads to (external) fragmentation; paging solves this

Memory Hierarchy Pyramid

Why virtual memory? (1/2)
° Protection
  • regions of the address space can be read only, execute only, . . .
° Flexibility
  • portions of a program can be placed anywhere, without relocation
° Expandability
  • can leave room in virtual address space for objects to grow
° Storage management
  • allocation/deallocation of variable sized blocks is costly and leads to (external) fragmentation; paging solves this
Why virtual memory? (2/2)
- Generality
  - ability to run programs larger than size of physical memory
- Storage efficiency
  - retain only most important portions of the program in memory
- Concurrent I/O
  - execute other processes while loading/dumping page

Virtual Memory Review (1/4)
- User program view of memory:
  - Contiguous
  - Start from some set address
  - Infinitely large
  - Is the only running program
- Reality:
  - Non-contiguous
  - Start wherever available memory is
  - Finite size
  - Many programs running at a time

Virtual Memory Review (2/4)
- Virtual memory provides:
  - illusion of contiguous memory
  - all programs starting at same set address
  - illusion of infinite memory
  - protection

Virtual Memory Review (3/4)
- Implementation:
  - Divide memory into “chunks” (pages)
  - Operating system controls pagetable that maps virtual addresses into physical addresses
  - Think of memory as a cache for disk
  - TLB is a cache for the pagetable

Why Translation Lookaside Buffer (TLB)?
- Paging is most popular implementation of virtual memory (vs. base/bounds)
- Every paged virtual memory access must be checked against Entry of Page Table in memory to provide protection
- Cache of Page Table Entries makes address translation possible without memory access in common case to make fast

Virtual Memory Review (4/4)
- Let’s say we’re fetching some data:
  - Check TLB (input: VPN, output: PPN)
    - hit: fetch translation
    - miss: check pagetable (in memory)
      - pagetable hit: fetch translation
      - pagetable miss: page fault, fetch page from disk to memory, return translation to TLB
  - Check cache (input: PPN, output: data)
    - hit: return value
    - miss: fetch value from memory
Paging/Virtual Memory Review

Three Advantages of Virtual Memory

1) Translation:
   • Program can be given consistent view of memory, even though physical memory is scrambled
   • Makes multiple processes reasonable
   • 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

Three Advantages of Virtual Memory

2) Protection:
   • Different 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

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

4 Questions for Memory Hierarchy

• Q1: Where can a block be placed in the upper level? (Block placement)
• Q2: How is a block found if it is in the upper level? (Block identification)
• Q3: Which block should be replaced on a miss? (Block replacement)
• Q4: What happens on a write? (Write strategy)
Q2: How is a block found in upper level?
○ Direct indexing (using index and block offset), tag compares, or combination
○ Increasing associativity shrinks index, expands tag

Q3: Which block replaced on a miss?
○ Easy for Direct Mapped
○ Set Associative or Fully Associative:
  • Random
  • LRU (Least Recently Used)

Miss Rates
Associativity: 2-way  4-way  8-way

<table>
<thead>
<tr>
<th>Size</th>
<th>LRU</th>
<th>Ran</th>
<th>LRU</th>
<th>Ran</th>
<th>LRU</th>
<th>Ran</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 KB</td>
<td>5.2%</td>
<td>5.7%</td>
<td>4.7%</td>
<td>5.3%</td>
<td>4.4%</td>
<td>5.0%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.9%</td>
<td>2.0%</td>
<td>1.5%</td>
<td>1.7%</td>
<td>1.4%</td>
<td>1.5%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.17%</td>
<td>1.13%</td>
<td>1.13%</td>
<td>1.12%</td>
<td>1.12%</td>
</tr>
</tbody>
</table>

Q4: What happens on a write?
○ Write through—The information is written to both the block in the cache and to the block in the lower-level memory.
○ Write back—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  • is block clean or dirty?
○ Pros and Cons of each?
  • WT: read misses cannot result in writes
  • WB: no writes of repeated writes

Address Translation & 3 Exercises

Exercise:
○ 40-bit VA, 16 KB pages, 36-bit PA
○ Number of bits in Virtual Page Number?
  a) 18; b) 20; c) 22; d) 24; e) 26; f) 28
○ Number of bits in Page Offset?
  a) 8; b) 10; c) 12; d) 14; e) 16; f) 18
○ Number of bits in Physical Page Number?
  a) 18; b) 20; c) 22; d) 24; e) 26; f) 28

Address Translation Exercise (2)

Exercise:
○ 40-bit VA, 16 KB pages, 36-bit PA
○ 2-way set-assoc TLB: 256 "slots", 2 per slot
○ Number of bits in TLB Index?
  a) 18; b) 20; c) 22; d) 24; e) 26; f) 28
○ Number of bits in TLB Tag?
  a) 8; b) 10; c) 12; d) 14; e) 16; f) 18
○ Approximate Number of bits in TLB Entry?
  a) 32; b) 36; c) 40; d) 42; e) 44; f) 46
**Address Translation Exercise (3)**

- **Exercise:**
  - 40-bit VA, 16 KB pages, 36-bit PA
  - 2-way set-associ TLB: 256 "slots", 2 per slot
  - 64 KB data cache, 64 Byte blocks, 2 way S.A.

- **Number of bits in Cache Offset?**
  - a) 6; b) 8; c) 10; d) 12; e) 14; f) 16

- **Number of bits in Cache Index?**
  - a) 6; b) 8; c) 10; d) 12; e) 14; f) 16

- **Number of bits in Cache Tag?**
  - a) 18; b) 20; c) 22; d) 24; e) 26; f) 28

- **Approximate No. of bits in Cache Entry?**

---

**Impact of What Learned About Caches?**

- **1960-1985:** Speed = \( f(\text{no. operations}) \)

- **1990s**
  - Pipelined Execution & Fast Clock Rate
  - Out-of-Order execution
  - Superscalar

- **1999:** Speed = \( f(\text{non-cached memory accesses}) \)

- Superscalar, Out-of-Order machines hide L1 data cache miss (5 clocks) but not L2 cache miss (50 clocks)?

---

**Quicksort vs. Radix as vary number keys:**

- **Instructions**

---

**Cache/VM/TLB Summary: #1/3**

- **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

- **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?
Virtual Memory allows protected sharing of memory between processes with less swapping to disk, less fragmentation than always swap or base/bound

3 Problems:
1) Not enough memory: Spatial Locality means small Working Set of pages OK
2) TLB to reduce performance cost of VM
3) Need more compact representation to reduce memory size cost of simple 1-level page table, especially for 64-bit address (See CS 162)

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?