CSE 490/590 Computer Architecture

Cache I

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last Time...

- Pipelining hazards
  - Structural hazards
  - Data hazards
  - Control hazards
- Data hazards
  - Stall
  - Bypass
- Control hazards
  - Jump
  - Conditional branch

Branch Delay Slots (expose control hazard to software)

- Change the ISA semantics so that the instruction that follows a jump or branch is always executed
  - gives compiler the flexibility to put in a useful instruction where normally a pipeline bubble would have resulted.

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>096</td>
<td>ADD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>BEQZ r1 +200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>104</td>
<td>ADD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>304</td>
<td>ADD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Delay slot instruction executed regardless of branch outcome

- Other techniques include more advanced branch prediction, which can dramatically reduce the branch penalty... to come later

Why an Instruction may not be dispatched every cycle (CPI>1)

- Full bypassing may be too expensive to implement
  - typically all frequently used paths are provided
  - some infrequently used bypass paths may increase cycle time and counteract the benefit of reducing CPI
- Loads have two-cycle latency
  - Instruction after load cannot use load result
  - MIPS-I ISA defined load delay slots, a software-visible pipeline hazard (compiler schedules independent instruction or inserts NOP to avoid hazard). Removed in MIPS-II (pipeline interlocks added in hardware)
  - MIPS: "Microprocessor without Interlocked Pipeline Stages"
- Conditional branches may cause bubbles
  - kill following instruction(s) if no delay slots

Branch Pipeline Diagrams (branch delay slot)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>096: ADD</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MA</td>
</tr>
<tr>
<td>(2)</td>
<td>100: BEQZ +200</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MA</td>
</tr>
<tr>
<td>(3)</td>
<td>104: ADD</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MA</td>
</tr>
<tr>
<td>(4)</td>
<td>304: ADD</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MA</td>
</tr>
</tbody>
</table>

Resource Usage

- IF
- ID
- EX
- MA
- WB

Early Read-Only Memory Technologies

- Punched cards. From early 1700s through Jaquard Loom, Babbage, and then IBM
- Diode Matrix, EDSAC-2 µcode store
- Punched paper tape, instruction stream in Harvard Mk I
Early Read/Write Main Memory Technologies

Babbage, 1800s: Digits stored on mechanical wheels

Williams Tube, Manchester Mark 1, 1947

Mercury Delay Line, UNIVAC 1, 1951

Also, regenerative capacitor memory on Atanasoff-Berry computer, and rotating magnetic drum memory on IBM 650

Semiconductor Memory

- Semiconductor memory began to be competitive in early 1970s
  - Intel formed to exploit market for semiconductor memory
    - Early semiconductor memory was Static RAM (SRAM). SRAM cell internals similar to a latch (cross-coupled inverters).
  - First commercial Dynamic RAM (DRAM) was Intel 1103
    - 1Kbit of storage on single chip
    - charge on a capacitor used to hold value
  - Semiconductor memory quickly replaced core in '70s

Modern DRAM Structure

- Bits stored in 2-dimensional arrays on chip
- Modern chips have around 4 logical banks on each chip
  - each logical bank physically implemented as many smaller arrays

DRAM Architecture

- Bits stored in 2-dimensional arrays on chip
- Modern chips have around 4 logical banks on each chip
  - each logical bank physically implemented as many smaller arrays

DRAM Operation

- Three steps in read/write access to a given bank:
  - Row access (RAS)
    - decode row address, enable addressed row (often multiple Kb in row)
    - bitlines share charge with storage cell
    - small change in voltage detected by sense amplifiers which latch whole row of bits
    - sense amplifiers drive bitlines full rail to recharge storage cells
  - Column access (CAS)
    - decode column address to select small number of sense amplifier latches (4, 8, 16, or 32 bits depending on DRAM package)
    - on read, send latched bits out to chip pins
    - on write, change sense amplifier latches which then charge storage cells to required value
    - can perform multiple column accesses on same row without another row access (burst mode)
  - Precharge
    - charges bitlines to known value, required before next row access

Each step has a latency of around 15-20ns in modern DRAMs

Various DRAM standards (DDR, RDRAM) have different ways of encoding the signals for transmission to the DRAM, but all share same core architecture

DRAM Packaging

- DIMM (Dual Inline Memory Module) contains multiple chips with clock/control/address signals connected in parallel (sometimes need buffers to drive signals to all chips)
- Data pins work together to return wide word (e.g., 64-bit data bus using 16x4-bit parts)

CSE 490/590, Spring 2011
CPU-Memory Bottleneck

Performance of high-speed computers is usually limited by memory bandwidth and latency.

- **Latency** (time for a single access)
  - Memory access time >> Processor cycle time
  - Problematic
- **Bandwidth** (number of accesses per unit time)
  - Increase the bus size, etc.
  - Usually OK

![Processor-DRAM Gap (latency)](image)

**CPU-Memory Bottleneck**

Number of accesses per unit time (bus size)

Latency (time for a single access)

- Memory access time >> Processor cycle time
- Problematic

Bandwidth (number of accesses per unit time)

Increase the bus size, etc.

Usually OK

**Physical Size Affects Latency**

- Signals have further to travel
- Fan out to more locations

CSE 490/590 Administrivia

- Very important to attend
  - Recitations next week & the week after
- Guest lectures
  - There will be a couple guest lectures late Feb/early Mar.
- Quiz 1
  - Rescheduled
  - Fri, 2/11
  - Closed book, in-class
  - Includes lectures until last Monday (1/31)
  - Review: next Wed (2/9)

**Memory Hierarchy**

- **Small, Fast Memory** (RF, SRAM)
  - **capacity**: Register << SRAM << DRAM
  - **latency**: Register << SRAM << DRAM
  - **bandwidth**: on-chip >> off-chip

- **Big, Slow Memory** (DRAM)

On a data access:

- If data ∈ fast memory → low latency access (SRAM)
- If data ∉ fast memory → long latency access (DRAM)

Relative Memory Cell Sizes

[Table: Memory cell sizes for various DRAM technologies]

[Figure: Relative memory cell sizes]
Levels of the Memory Hierarchy

- CPU Registers
  - 100s Bytes
  - <10s ns
  - 1-0.1 cents/bit
- Cache
  - K Bytes
  - 10-100 ns
  - $.0001-.00001 cents/bit
- Main Memory
  - M Bytes
  - 200ns-500ns
  - $.0001-.00001 cents/bit
- Disk
  - G Bytes
  - 10ms (10,000,000 ns)
  - 10^-5-10^-6 cents/bit

<table>
<thead>
<tr>
<th>Capacity</th>
<th>Access Time</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>1-0.1 cents/bit</td>
<td>1-0.1 ns</td>
</tr>
<tr>
<td>Cache</td>
<td>$.0001-.00001 cents/bit</td>
<td>10-100 ns</td>
</tr>
<tr>
<td>Main Memory</td>
<td>$.0001-.00001 cents/bit</td>
<td>200ns-500ns</td>
</tr>
<tr>
<td>Disk</td>
<td>10^-5-10^-6 cents/bit</td>
<td>10ms</td>
</tr>
</tbody>
</table>

Management of Memory Hierarchy

- Small/fast storage, e.g., registers
  - Address usually specified in instruction
  - Generally implemented directly as a register file
    - but hardware might do things behind software’s back, e.g., stack management, register renaming
- Larger/slower storage, e.g., main memory
  - Address usually computed from values in register
  - Generally implemented as a hardware-managed cache hierarchy
    - hardware decides what is kept in fast memory
    - but software may provide “hints”, e.g., don’t cache or prefetch

Real Memory Reference Patterns


Typical Memory Reference Patterns

- Instruction fetches
- Stack accesses
- Data accesses

Common Predictable Patterns

- Temporal Locality: If a location is referenced it is likely to be referenced again in the near future.
- Spatial Locality: If a location is referenced it is likely that locations near it will be referenced in the near future.
Caches

Caches exploit both types of predictability:

- Exploit temporal locality by remembering the contents of recently accessed locations.
- Exploit spatial locality by fetching blocks of data around recently accessed locations.

Inside a Cache

Cache Algorithm (Read)

Look at Processor Address, search cache tags to find match. Then either

- Found in cache a.k.a. HIT
  - Return copy of data from cache
- Not in cache a.k.a. MISS
  - Read block of data from Main Memory
  - Wait ...
  - Return data to processor and update cache

Q: Which line do we replace?

Placement Policy

Direct-Mapped Cache

Tag Index Block Offset

V1 Tag k Data Block b

HIT

Data Word or Byte

Direct-Mapped Cache

Tag Index Block Offset

0 1 2 3 4 5 6 7

0 1 2 3 4 5 6 7

0 1 2 3 4 5 6 7

0 1 2 3 4 5 6 7

Fully Associative anywhere

(2-way) Set Associative anywhere in set 0 (12 mod 4)

Direct Mapped only into block 4 (12 mod 8)
Direct Map Address Selection
higher-order vs. lower-order address bits

Tag
Data Block
V = Block Offset

Index
Tag
Block Offset

HIT
Data Word or Byte

2-Way Set-Associative Cache

Tag
Index
Block Offset

Data Word or Byte
HIT

Fully Associative Cache

Tag
Index
Block Offset

Data Word or Byte
HIT

Replacement Policy
In an associative cache, which block from a set should be evicted when the set becomes full?

- Random
- Least Recently Used (LRU)
  - true implementation only feasible for small sets (2-way)
  - pseudo-LRU binary tree often used for 4-8 way
- First In, First Out (FIFO) a.k.a. Round-Robin
  - used in highly associative caches
- Not Least Recently Used (NLRU)
  - FIFO with exception for most recently used block or blocks

This is a second-order effect. Why?

Replacement only happens on misses

Acknowledgements
- These slides heavily contain material developed and copyright by
  - Krste Asanovic (MIT/UCB)
  - David Patterson (UCB)
- And also by:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
- MIT material derived from course 6.823
- UCB material derived from course CS252