Question 1: Multiple Choice

Consider a single 2-bit predictor that starts in the "weakly taken" state. After the following sequence of branches, which state will it be in (T == Taken, N == Not-taken):
T,T,T,NT,NT,T

Correct
- Strongly not taken
- weakly not taken
- weakly taken
- strongly taken.

Unanswered

Percent Answered
- 1.613%
- 9.677%
- 87.097%
- 1.613%
- 0%
Quiz 6

Question 2: True/False

Squashing instructions and stalling the pipeline both result in increased CPI.

<table>
<thead>
<tr>
<th>Correct Answers</th>
<th>Percent Answered</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>95.161%</td>
</tr>
<tr>
<td>False</td>
<td>4.839%</td>
</tr>
<tr>
<td>Unanswered</td>
<td>0%</td>
</tr>
</tbody>
</table>

Average Score 1.90323 points
Quiz 6

Question 3: True/False

Squashing can be used to resolve control hazards.

<table>
<thead>
<tr>
<th>Correct Answers</th>
<th>Percent Answered</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>88.71%</td>
</tr>
<tr>
<td>False</td>
<td>9.677%</td>
</tr>
</tbody>
</table>

Average Score 1.77419 points
Question 4: True/False

Stalling cannot be used to resolve control hazards.

<table>
<thead>
<tr>
<th>Correct</th>
<th>Answers</th>
<th>Percent Answered</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td></td>
<td>11.29%</td>
</tr>
<tr>
<td>False</td>
<td></td>
<td>88.71%</td>
</tr>
</tbody>
</table>

Unanswered 0%
Question 5: True/False

Static branch predictors use tables of 2-bit counters.

<table>
<thead>
<tr>
<th>Correct Answers</th>
<th>Percent Answered</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>16.129%</td>
</tr>
<tr>
<td>False</td>
<td>83.871%</td>
</tr>
</tbody>
</table>

Unanswered 0%
Question 7: Calculated Numeric

If we double the number of pipeline stages in the MIPS 5-stage design by dividing each existing stage in half, what would the new branch delay penalty be (assume branches resolve at the end of decode)?

<table>
<thead>
<tr>
<th>Correct Answers</th>
<th>Percent Answered</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>53.226%</td>
</tr>
<tr>
<td>2</td>
<td>24.194%</td>
</tr>
<tr>
<td>1</td>
<td>1.613%</td>
</tr>
<tr>
<td>10</td>
<td>3.226%</td>
</tr>
<tr>
<td>3 cycles</td>
<td>3.226%</td>
</tr>
<tr>
<td>6</td>
<td>1.613%</td>
</tr>
<tr>
<td>4</td>
<td>12.903%</td>
</tr>
</tbody>
</table>

Unanswered 0%
Question 6: Calculated Numeric

On a scale of 1–10 (1 being completely unfair and 10 being completely fair), how fair was the midterm?

<table>
<thead>
<tr>
<th>Correct Answers</th>
<th>Percent Answered</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6.452%</td>
</tr>
<tr>
<td>10</td>
<td>9.677%</td>
</tr>
<tr>
<td>8.3333333333333333333333</td>
<td>1.613%</td>
</tr>
<tr>
<td>7</td>
<td>19.355%</td>
</tr>
<tr>
<td>6</td>
<td>14.516%</td>
</tr>
<tr>
<td>5</td>
<td>3.226%</td>
</tr>
<tr>
<td>4</td>
<td>8.065%</td>
</tr>
<tr>
<td>9</td>
<td>14.516%</td>
</tr>
<tr>
<td>8</td>
<td>22.581%</td>
</tr>
</tbody>
</table>

Unanswered: 0%
The Memory Hierarchy

In the book: 5.1-5.3, 5.7, 5.10
Goals for this Class

• Understand how CPUs run programs
  • How do we express the computation the CPU?
  • How does the CPU execute it?
  • How does the CPU support other system components (e.g., the OS)?
  • What techniques and technologies are involved and how do they work?

• Understand why CPU performance (and other metrics) varies
  • How does CPU design impact performance?
  • What trade-offs are involved in designing a CPU?
  • How can we meaningfully measure and compare computer systems?

• Understand why program performance varies
  • How do program characteristics affect performance?
  • How can we improve a program's performance by considering the CPU running it?
  • How do other system components impact program performance?
Memory

Abstraction: Big array of bytes

CPU

Memory
Main points for today

• What is a memory hierarchy?
• What is the CPU-DRAM gap?
• What is locality? What kinds are there?
• Learn a bunch of caching vocabulary.
• Memory is very slow compared to processors.
Memory’s impact

M = % mem ops
Mlat (cycles) = average memory latency
BCPI = base CPI with single-cycle data memory

CPI =
Memory’s impact

\[ M = \% \text{ mem ops} \]
\[ \text{Mlat (cycles)} = \text{average memory latency} \]
\[ \text{TotalCPI} = \text{BaseCPI} + M \times \text{Mlat} \]

Example:

\[
\begin{align*}
\text{BaseCPI} &= 1; \ M = 0.2; \ \text{Mlat} = 240 \text{ cycles} \\
\text{TotalCPI} &= 49 \\
\text{Speedup} &= 1/49 = 0.02 \quad \Rightarrow \quad 98\% \text{ drop in performance}
\end{align*}
\]

Remember!: Amdahl’s law does not bound the slowdown. Poor memory performance can make your program arbitrarily slow.
Memory Cache

- Memory cost
  - \(\gg\) capacity -> more $$
  - \(\gg\) speed/bw -> more $$
  - \(\gg\) speed -> larger (less dense)
- Build several memories with different trade-offs
- How do you use it? Build a “memory hierarchy”
- What should it mean for the memory abstraction?
Memory Cache

- Memory cost
  - $\gg$ capacity $\rightarrow$ more $$$
  - $\gg$ speed/bw $\rightarrow$ more $$$
  - $\gg$ speed $\rightarrow$ larger (less dense)

- Build several memories with different trade-offs
- How do you use it? Build a “memory hierarchy”
- What should it mean for the memory abstraction?
A typical memory hierarchy

- **on-chip cache**
  - KBs
  - Cost: 2.5 $/MB
  - Access time: < 1ns

- **off-chip cache**
  - MBs
  - Cost: 0.07 $/MB
  - Access time: 5ns

- **main memory**
  - GBs
  - Cost: 0.0004 $/MB
  - Access time: 60ns

- **Disk**
  - TBs
  - Cost: 0.0004 $/MB
  - Access time: 10,000,000ns
How far away is the data?

- $10^9$ Tape/Optical Robot: 2,000 Years
- $10^6$ Disk: 2 Years
- 100 Memory: 1.5 hr
- 10 On Board Cache: 10 min
- 2 On Chip Cache: 1 min
- 1 Registers: 1 min
Why should we expect caching to work?

- Why did branch prediction work?
Why should we expect caching to work?

- Why did branch prediction work?
- Where is memory access predictable
  - Predictably accessing the same data
    - In loops: for(i = 0; i < 10; i++) {s += foo[i];}
    - foo = bar[4 + configuration_parameter];
  - Predictably accessing different data
    - In linked lists: while(l != NULL) {l = l->next;}
    - In arrays: for(i = 0; i < 10000; i++) {s += data[i];}
    - structure access: foo(some_struct.a, some_struct.b);
The Principle of Locality

• "Locality" is the tendency of data access to be predictable. There are two kinds:

  • Spatial locality: The program is likely to access data that is close to data it has accessed recently

  • Temporal locality: The program is likely to access the same data repeatedly.
Locality in Action

• Label each access with whether it has temporal or spatial locality or neither

  • 1
  • 2
  • 3
  • 10
  • 4
  • 1800
  • 11
  • 30

  • 1
  • 2
  • 3
  • 4
  • 10
  • 190
  • 11
  • 30
  • 12
  • 13
  • 182
  • 1004
Locality in Action

- Label each access with whether it has temporal or spatial locality or neither
  - 1 n
  - 2 s
  - 3 s
  - 10 n
  - 4 s
  - 1800 n
  - 11 s
  - 30 n
  - 1 t
  - 2 s, t
  - 3 s, t
  - 4 s, t
  - 10 s, t
  - 190 n
  - 11 s, t
  - 30 s
  - 12 s
  - 13 s
  - 182 n?
  - 1004 n
Locality in Action

- Label each access with whether it has temporal or spatial locality or neither
  - 1 n
  - 2 s
  - 3 s
  - 10 n
  - 4 s
  - 1800 n
  - 11 s
  - 30 n

- 1 t
- 2 s, t
- 3 s, t
- 4 s, t
- 10 s, t
- 190 n
- 11 s, t
- 30 s
- 12 s
- 13 s
- 182 n?
- 1004 n

There is no hard and fast rule here. In practice, locality exists for an access if the cache performs well.
Cache Vocabulary

- **Hit** - The data was found in the cache
- **Miss** - The data was not found in the cache
- **Hit rate** - hits/total accesses
- **Miss rate** = 1 - Hit rate
- **Locality** - see previous slides
- **Cache line** - the basic unit of data in a cache. generally several words.
- **Tag** - the high order address bits stored along with the data to identify the actual address of the cache line.
- **Hit time** -- time to service a hit
- **Miss time** -- time to service a miss (this is a function of the lower level caches.)
Cache Vocabulary

• There can be many caches stacked on top of each other
• If you miss in one you try in the “lower level cache”
  Lower level, mean higher number
• There can also be separate caches for data and instructions. Or the cache can be “unified”
• In the 5-stage MIPS pipeline
  • the L1 data cache (d-cache) is the one nearest processor. It corresponds to the “data memory” block in our pipeline diagrams
  • the L1 instruction cache (i-cache) corresponds to the “instruction memory” block in our pipeline diagrams.
  • The L2 sits underneath the L1s.
  • There is often an L3 in modern systems.
Typical Cache Hierarchy

- Fetch/ L1 Icache 16KB
- Decode
- EX
- Mem L1 Dcache 16KB
- Write back

Unified L2
8MB

Unified L3
32MB

DRAM
Many GBs

Many GBs
Data vs Instruction Caches

• Why have different I and D caches?
Data vs Instruction Caches

- Why have different I and D caches?
  - Different areas of memory
  - Different access patterns
    - I-cache accesses have lots of spatial locality. Mostly sequential accesses.
    - I-cache accesses are also predictable to the extent that branches are predictable
    - D-cache accesses are typically less predictable
  - Not just different, but often across purposes.
    - Sequential I-cache accesses may interfere with the data the D-cache has collected.
    - This is “interference” just as we saw with branch predictors
  - At the L1 level it avoids a structural hazard in the pipeline
  - Writes to the I cache by the program are rare enough that they can be slow (i.e., self modifying code)