Review Problem 44

- Which is the best L1 cache for this system?
  - Direct Mapped: 1 cycle, 80% hit rate
  - 2-way Set Associative: 2 cycle, 90% hit rate
  - Fully Associative: 3 cycle, 95% hit rate

<table>
<thead>
<tr>
<th>Level</th>
<th>Hit Time</th>
<th>Hit Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2</td>
<td>10 cycles</td>
<td>90%</td>
</tr>
<tr>
<td>Main Memory</td>
<td>40 cycles</td>
<td>99%</td>
</tr>
<tr>
<td>Disk</td>
<td>4,000 cycles</td>
<td>100%</td>
</tr>
</tbody>
</table>

---

Direct Mapped: \[1 + 0.2 \times 18 = 4.6\]

2-way: \[2 + 0.1 \times 18 = 3.8\]

Fully Associative: \[3 + 0.05 \times 18 = 3.9\]
Writing & Caches

Direct-mapped cache with 16-byte blocks, initially **empty**

```
STUR X0, [X31, #0]
```

---

**Cache Line:**

```
Write-back
```

**Main Memory:**

---
Writing & Caches (cont.)

Write-back:
- Just save to the cache
- Remember to write the data to MM on eviction
- "dirty" bit: the cache value is newer than lower levels

Write-through:
- Write to both MM + cache

Write buffer:
- Another unit that writes for you

Flush-on-write:
- Fill empty parts of cache block from main memory
- Alternative: per-byte valid bits
Replacement Methods

If we need to load a new cache line, where does it go?

Direct-mapped

Only one possible location

Set Associative

N choices: temporal locality
LRU.

Fully Associative

All locations possible:
Replacement Strategies

When needed, pick a location

Approach #1: Random
   Just arbitrarily pick from possible locations

Approach #2: Least Recently Used (LRU)
   Use temporal locality
   Must track somehow – extra cache bits to indicate how recently used

In practice, Random typically only 12% worse than LRU

Victim cache: Small FA queue of last M evictions
**Split Caches**

**Instruction vs. Data accesses**

How do the two compare in usage?

*Instruction cache: heavy locality. Fixed sized access
Don't write the instructions.*

*Data: less locality. Variable size*

How many accesses/cycle do we need for our pipelined CPU?

1 instruction read/cycle

*Sometimes 1 data read/write in cycle*

Typically split the caches into separate instruction, data caches

*Higher bandwidth
Optimize to usage
Slightly higher miss rate because each cache is smaller.*
Multi-level Caches

Instead of just having an on-chip (L1) cache, an off-chip (L2) cache is helpful.

Ex. Base machine with \( \text{CPI} = 1.0 \) if all references hit the L1, 2 GHz

Main memory access delay of 50 ns. L1 miss rate of 5%

How much faster would the machine be if we added a L2 with a miss rate of 10%, and an access time of 20 ns.

\[
\text{L1+MM} = 1 + 0.05 \times 100 = 6 \text{ cycles}
\]

\[
\text{L1+L2+MM} = 1 + 0.05 \times (40 + 1 \times 100)
= 1 + 2 + 0.5 = 3.5 \text{ cycles}
\]
Cache Summary

Illusion of big fast memory via locality

Small SRAM upper levels, DRAM MM

Locality: temporal, spatial

Types of cache misses
- Cold
- Conflict
- Capacity
Review Problem 47

- Here is a graph of runtime vs. $N$, on a log-log plot, for the following code. Explain

```c
int x[N];
for (j = 0; j < 1000; j++)
    for (int i = 0; i < N; i++)
        x[i]++;
```
Virtual Memory

<table>
<thead>
<tr>
<th>Technology</th>
<th>Access Time</th>
<th>Cost/Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM</td>
<td>1-7 cycles</td>
<td>10,000x</td>
</tr>
<tr>
<td>DRAM</td>
<td>100 cycles</td>
<td>200x</td>
</tr>
<tr>
<td>Flash</td>
<td>10,000 cycle</td>
<td>15x</td>
</tr>
<tr>
<td>Disk</td>
<td>10,000,000 cycles</td>
<td>1x</td>
</tr>
</tbody>
</table>

Disk more cost effective than even DRAM
Use Disk as memory?

Virtual Memory: View disk as the lowest level in the memory hierarchy
"Page" memory to disk when information won't fit in main memory
Virtual Addresses

Thought experiment: What happens when you run two programs at once? How do they share the address space?

Solution: Virtual addresses
Each address the processor generates is a Virtual Address
Virtual Addresses are mapped to Physical Addresses
Virtual address may correspond to address in memory, or to disk

Other important terminology

Page – the block for main memory, moved as a group to/from disk
Page fault – “miss” on main memory. Handled as a processor exception
Memory mapping/address translation – conversion process from virtual to physical addresses
Virtual to Physical Addresses

Virtual Addresses

<table>
<thead>
<tr>
<th>Address Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Addresses</td>
</tr>
</tbody>
</table>

Disk Addresses

Virtual Address:
- Virtual Page Number
- Page Offset

Physical Address:
- Physical Page Number
- Page Offset

Virtual Addresses: 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

Physical Address: 169
Virtual Memory

Main Memory is a cache
  Large cache lines/page size – temporal locality
  Huge page fault/cache miss penalty – fully associative, software managed
  Translation complex – cache the misses themselves

Thrashing
Dynamic Branch Prediction

Branches introduce control hazards
   Determine the right next instruction in time for instruction fetch

Previous solutions:
   Stall
   Statically predict not taken
   Branch Delay slot

Better:
   Branch-prediction buffers (caches)
Problems With Static Branch Predictors

Are all conditional branches created equal?

Branch in a loop
  Branch @ top -> almost always T
  Branch @ bottom -> almost always T

if-then-else
  if (error) exit(1) -> always T
  if (normal condition) -> almost always T
  if (variable is odd) -> ???
Branch Prediction Buffer

Direct-mapped cache w/1-bit history
Predict taken/not taken by previous execution
If incorrect prediction, annul instructions incorrectly started

Tags?
Valid bits?

Too much overhead, you won't go to MM anyway.
Thought Experiment

Consider the following code segment:

```c
while (1) {
    <code 1>
    for(int i=0; i<9; i++) {
        <code 2>
    }
    <code 3>
}
```

1-bit prediction accuracy?

\[
\begin{array}{c|c|c}
\text{Result} & \text{Prediction} & \text{PT} \\
\hline
T & T & T \\
T & T & T \\
T & T & T \\
T & T & T \\
T & T & T \\
T & T & T \\
T & T & T \\
T & T & T \\
T & T & T \\
\end{array}
\]

\[\text{Screw-up}\]
2-bit Predictor

82% - 99% accurate
while (1) {
    if (normal_condition) {
        <code1>
    }
    for(int i=0; i<9; i++) {
        if (exception) {
            return (FALSE);
        }
        <code 2>
    }
    if (random 50/50 chance) {
        <code 3>
    }
}