Cache Overview

● Memory Hierarchy
  ○ From fastest and most expensive to slowest and least expensive
  ○ Fastest memory is smallest and closest to CPU
  ○ Slowest memory is largest and far from the CPU
  ○ Registers -> Cache (SRAM) -> Main Memory (DRAM) -> SSD/Hard drive

● Blocks
  ○ Group words into blocks
    ■ Example: 4 words per block
  ○ A block in the cache can be called the following: (these all mean the same thing)
    ■ Cache line
    ■ Cache block
    ■ Cache slot
Cache Types: Direct-Mapped

- Each memory block maps to 1 possible cache slot, based on address
- Mapping for the cache block:
  \[ m = \left(\frac{p}{B}\right) \mod M \]
  where: 
  - \( m \) is the destination cache line
  - \( p \) is the address
  - \( M \) is the number of blocks
  - \( B \) is the block size in bytes
Cache Types: Fully Associative

- Allow a given block to go into any cache slot
- The lack of direct map from block to slot means we need to check every cache slot for the data we need

- All these checks means that this is \textit{SLOW}
- Since the entire point of caches is to be fast, this is not a good method
- More complex hardware is required
Cache Types: N-way Associative

- Group cache slots into sets of slots, each set containing N slots
- The block number determines the set:
  \[ \text{set number} = \text{(block number)} \mod \text{(number of sets)} \]
- In each set, we have fully associative caching
- But, we only need to search entries in a given set, so it’s faster
Cache Types: Summary

● What does associativity mean?

● Pro:
  ○ increased associativity means increased hit rate (but diminishing returns)

● Con:
  ○ increased associativity increases hardware cost and complexity
Write Modes

● Write-Through
  ○ Like example, writing to a block writes to memory immediately

● Write-Back
  ○ Writes go back to the cache
  ○ Each cache line has a “dirty” bit
    ■ When a block is evicted, if dirty bit is set, write block to memory

● Write-Buffered
  ○ Writes go to special buffer for blocks
  ○ When the buffer gets full, write all blocks to main memory
Consider a direct mapped, write-through cache with 8 blocks and block size 16 words. What do memory and cache look like after the following instructions?

\[
\begin{align*}
&\text{l}w \ $10, \ 0(\$1) \\
&\text{l}w \ $11, \ 16(\$1) \\
&\text{l}w \ $12, \ 64(\$1) \\
&\text{l}w \ $13, \ 0(\$2) \\
&\text{s}w \ $14, \ 0(\$2) \\
&\text{l}w \ $15, \ 0(\$3) \\
\end{align*}
\]

Assume $1 = \text{0x00003D80}$, $2 = \text{0x0000AC30}$, $3 = \text{0x0000FE24}$.

### Cache

<table>
<thead>
<tr>
<th>Block 0</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 1</td>
<td>F</td>
</tr>
<tr>
<td>Block 2</td>
<td>F</td>
</tr>
<tr>
<td>Block 3</td>
<td>F</td>
</tr>
<tr>
<td>Block 4</td>
<td>F</td>
</tr>
<tr>
<td>Block 5</td>
<td>F</td>
</tr>
<tr>
<td>Block 6</td>
<td>F</td>
</tr>
<tr>
<td>Block 7</td>
<td>F</td>
</tr>
</tbody>
</table>

### Memory

| Address 0x00003D80 | “Once upon...” | Block 246 |
| Address 0x00003DC0 | “The tree...” | Block 247 |
| Address 0x0000AC30 | “A fish...” | Block 688 |
| Address 0x0000AC40 | “My dog is...” | Block 689 |
| Address 0x0000FE24 | “Cheese is...” | Block 1016 |
Cache Types: Direct Map Example

- 16 words/block
  - block size is 64
  - 64 = 0x40
- 0x3D80 / 0x40 = 246
- 246 / 8 = 30 R 6
  - Block 246 maps to cache line 6 with tag 30

```
lw $10, 0($1)
lw $11, 16($1)
lw $12, 64($1)
lw $13, 0($2)
sw $14, 0($2)
lw $15, 0($3)
```
Assume $1 = 0x00003D80
$s2 = 0x0000AC30
$s3 = 0x0000FE24

### Cache

<table>
<thead>
<tr>
<th>Block</th>
<th>Data</th>
<th>Tag</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td></td>
<td>F</td>
</tr>
</tbody>
</table>

### Memory

```
“Once upon...”
Block 246 Address 0x00003D80

“The tree...”
Block 247 Address 0x00003DC0

“A fish...”
Block 688 Address 0x0000AC30

“My dog is...”
Block 689 Address 0x0000AC40

“Cheese is...”
Block 1016 Address 0x0000FE24
```
Cache Types: Direct Map Example

- \((0x3D80 + 64) / 0x40 = 247\)
- \(247 / 8 = 30 R 7\)
  - Block 247 maps to cache line 7 with tag 30

lw $10, 0($1)
lw $11, 16($1)
**lw $12, 64($1)**
lw $13, 0($2)
sw $14, 0($2)
lw $15, 0($3)
Assume $1 = 0x00003D80
$2 = 0x0000AC30
$3 = 0x0000FE24

<table>
<thead>
<tr>
<th>Cache</th>
<th>Data</th>
<th>Tag</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 0</td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>Block 1</td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>Block 2</td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>Block 3</td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>Block 4</td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>Block 5</td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>“Once upon...” Block 6</td>
<td>30</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>“The tree...” Block 7</td>
<td>30</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory</th>
<th>“Once upon...”</th>
<th>Address 0x00003D80</th>
</tr>
</thead>
<tbody>
<tr>
<td>“The tree...”</td>
<td>Block 247</td>
<td>Address 0x00003DC0</td>
</tr>
<tr>
<td>“A fish...”</td>
<td>Block 688</td>
<td>Address 0x0000AC30</td>
</tr>
<tr>
<td>“My dog is...”</td>
<td>Block 689</td>
<td>Address 0x0000AC40</td>
</tr>
<tr>
<td>“Cheese is...”</td>
<td>Block 1016</td>
<td>Address 0x0000FE24</td>
</tr>
</tbody>
</table>
Cache Types: Direct Map Example

- 0xAC30 / 0x40 = 688 R 48
- 688 / 8 = 86 R 0
  - Block 688 maps to cache line 0 with tag 86

lw $10, 0($1)
lw $11, 16($1)
lw $12, 64($1)
lw $13, 0($2)
sw $14, 0($2)
lw $15, 0($3)
Assume $1 = 0x00003D80
$2 = 0x0000AC30
$3 = 0x0000FE24

Data                  Tag      Valid
---                   ---       ---
“Once upon...”       Block 0   86      T

Memory

“Once upon...”
Block 246  Address 0x00003D80
“The tree...”
Block 247  Address 0x00003DC0
“A fish...”
Block 688  Address 0x0000AC30
“My dog is...”
Block 689  Address 0x0000AC40
“Cheese is...”
Block 1016 Address 0x0000FE24

“Once upon...”
Block 246
“The tree...”
Block 247
“A fish...”
Block 688
“My dog is...”
Block 689
“Cheese is...”
Block 1016

0xAC30 / 0x40 = 688 R 48
688 / 8 = 86 R 0
Block 688 maps to cache line 0 with tag 86

lw $10, 0($1)
lw $11, 16($1)
lw $12, 64($1)
lw $13, 0($2)
sw $14, 0($2)
lw $15, 0($3)
Assume $1 = 0x00003D80
$2 = 0x0000AC30
$3 = 0x0000FE24

“Once upon...”
Block 0
“The tree...”
Block 1
“A fish...”
Block 2
“My dog is...”
Block 3
“Cheese is...”
Block 4

“Once upon...”
Block 5
“The tree...”
Block 6
“A fish...”
Block 7

“Once upon...”
Block 246  Address 0x00003D80
“The tree...”
Block 247  Address 0x00003DC0
“A fish...”
Block 688  Address 0x0000AC30
“My dog is...”
Block 689  Address 0x0000AC40
“Cheese is...”
Block 1016 Address 0x0000FE24
## Cache Types: Direct Map Example

- **0xAC30 / 0x40 = 688 R 48**
- **688 / 8 = 86 R 0**
  - Block 688 maps to cache line 0 with tag 86

### lw, sw

\[
\begin{align*}
\text{lw} & \quad 10, 0(1) \\
\text{lw} & \quad 11, 16(1) \\
\text{lw} & \quad 12, 64(1) \\
\text{lw} & \quad 13, 0(2) \\
\text{sw} & \quad 14, 0(2) \\
\text{lw} & \quad 15, 0(3)
\end{align*}
\]

### Assume

\[
\begin{align*}
1 & = 0x00003D80 \\
2 & = 0x0000AC30 \\
3 & = 0x0000FE24 \\
14 & = 0x47474747
\end{align*}
\]

### Cache

<table>
<thead>
<tr>
<th>Data</th>
<th>Tag</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;GGGGsh...&quot;</td>
<td>86</td>
<td>T</td>
</tr>
<tr>
<td>Block 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 1</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>Block 2</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>Block 3</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>Block 4</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>Block 5</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>&quot;Once upon...&quot;</td>
<td>30</td>
<td>T</td>
</tr>
<tr>
<td>Block 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;The tree...&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;My dog is...&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;Cheese is...&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00003D80</td>
<td>&quot;Once upon...&quot;</td>
</tr>
<tr>
<td>0x0000AC30</td>
<td>&quot;The tree...&quot;</td>
</tr>
<tr>
<td>0x0000FE24</td>
<td>&quot;GGGGsh...&quot;</td>
</tr>
<tr>
<td>0x00003DCA</td>
<td>&quot;My dog is...&quot;</td>
</tr>
<tr>
<td>0x0000FE24</td>
<td>&quot;Cheese is...&quot;</td>
</tr>
<tr>
<td>0x00003D80</td>
<td>&quot;Once upon...&quot;</td>
</tr>
<tr>
<td>0x0000AC30</td>
<td>&quot;The tree...&quot;</td>
</tr>
<tr>
<td>0x0000FE24</td>
<td>&quot;GGGGsh...&quot;</td>
</tr>
<tr>
<td>0x00003DCA</td>
<td>&quot;My dog is...&quot;</td>
</tr>
<tr>
<td>0x0000FE24</td>
<td>&quot;Cheese is...&quot;</td>
</tr>
</tbody>
</table>
Cache Types: Direct Map Example

- \(0x\text{FE24} / 0x40 = 1016 \ R \ 36\)
- \(1016 / 8 = 127 \ R \ 0\)
  - Block 1016 maps to cache line 0 with tag 127

\[
\begin{array}{|c|c|c|}
\hline
\text{Block 0} & \text{127} & \text{T} \\
\hline
\text{Block 1} & & \\
\hline
\text{Block 2} & & \\
\hline
\text{Block 3} & & \\
\hline
\text{Block 4} & & \\
\hline
\text{Block 5} & & \\
\hline
\text{Block 6} & 30 & \\
\hline
\text{Block 7} & 30 & \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{“Cheese is...”} & \\
\hline
\text{Block 0} & \\
\hline
\text{“The tree...”} & \\
\hline
\text{Block 1} & \\
\hline
\text{“Once upon...”} & \\
\hline
\text{Block 2} & \\
\hline
\text{“GGGGsh...”} & \\
\hline
\text{Block 3} & \\
\hline
\text{“My dog is...”} & \\
\hline
\text{Block 4} & \\
\hline
\text{“Cheese is...”} & \\
\hline
\text{Block 5} & \\
\hline
\text{“Once upon...”} & \\
\hline
\text{Block 6} & \\
\hline
\text{“The tree...”} & \\
\hline
\text{Block 7} & \\
\hline
\end{array}
\]

lw $10, 0($1)
lw $11, 16($1)
lw $12, 64($1)
lw $13, 0($2)
sw $14, 0($2)
lw $15, 0($3)

Assume $1 = 0x00003D80$
$2 = 0x0000AC30$
$3 = 0x0000FE24$
$14 = 0x47474747$

Once upon...
The tree...
GGGGsh...
My dog is...
Cheese is...

Once upon...
The tree...
GGGGsh...
My dog is...
Cheese is...
**Write-Back Example**

What would have happened here if this was a write-back cache?

sw $14, 0($2)  
lw $15, 0($3)  

Assume $2 = 0x0000AC30  
$3 = 0x0000FE24  
$14 = 0x47474747

<table>
<thead>
<tr>
<th>Block 0</th>
<th>F</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 1</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 2</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 3</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 4</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 5</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 6</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 7</td>
<td>F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Data**
- **Tag**
- **Valid Dirty**

### Memory

- “Once upon...”  
  Block 246  Address 0x00003D80
- “The tree...”  
  Block 247  Address 0x00003DC0
- “A fish...”  
  Block 688  Address 0x0000AC30
- “My dog is...”  
  Block 689  Address 0x0000AC40
- “Cheese is...”  
  Block 1016  Address 0x0000FE24
Write-Back Example

First we load block 688 into cache line 0 as before, then write GGGG to just the cache.

```
sw $14, 0($2)
lw $15, 0($3)
```

Assume $2 = 0x0000AC30

$3 = 0x0000FE24

$14 = 0x47474747

Once upon...

The tree...

A fish...

My dog is...

Cheese is...

"Once upon..."

Block 246 Address 0x00003D80

"The tree..."

Block 247 Address 0x00003DC0

"A fish..."

Block 688 Address 0x0000AC30

"My dog is..."

Block 689 Address 0x0000AC40

"Cheese is..."

Block 1016 Address 0x0000FE24
Write-Back Example

Now when we load block 1016 into line 0 we have to save the old contents of line 0 back to memory because the dirty bit was set. Then we reset the dirty bit. If we missed saving this back to memory, the sw would be lost.

\[
\text{sw } \$14, 0(\$2) \\
\text{lw } \$15, 0(\$3)
\]

Assume $2 = 0x0000AC30$

$3 = 0x0000FE24$

$14 = 0x47474747$

```
<table>
<thead>
<tr>
<th>Block</th>
<th>Address</th>
<th>Data</th>
<th>Tag</th>
<th>Valid</th>
<th>Dirty</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x00003D80</td>
<td>Cheese</td>
<td>127</td>
<td>T</td>
<td>N</td>
</tr>
<tr>
<td>1</td>
<td>0x00003DC0</td>
<td></td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0x0000AC30</td>
<td></td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0x0000AC40</td>
<td></td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0x0000FE24</td>
<td></td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0x0000AC3C</td>
<td></td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0x0000AC48</td>
<td></td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0x0000AC54</td>
<td></td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>
```

Memory

```
<table>
<thead>
<tr>
<th>Block</th>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>246</td>
<td>0x00003D80</td>
<td>Once upon...</td>
</tr>
<tr>
<td>247</td>
<td>0x00003DC0</td>
<td>The tree...</td>
</tr>
<tr>
<td>688</td>
<td>0x0000AC30</td>
<td>GGGGsh...</td>
</tr>
<tr>
<td>689</td>
<td>0x0000AC40</td>
<td>My dog is...</td>
</tr>
<tr>
<td>1016</td>
<td>0x0000FE24</td>
<td>Cheese is...</td>
</tr>
</tbody>
</table>
```

```
sw $14, 0($2)
lw $15, 0($3)
```
CPU time - including memory access time

- We need to update our previous formula for calculating CPU time, now that we know the memory accesses take time

  \[ \text{CPU time} = (\text{CPU execution cycles} + \text{mem stall cycles}) \times (\text{cycle time}) \]

- To calculate the mem stall cycles, we use the following:

  \[ \text{mem stall cycles} = \left( \frac{\text{accesses}}{\text{program}} \right) \times (\text{miss rate}) \times (\text{miss penalty}) \]

- The miss rate is misses / instruction
CPU time - Example

- Compute CPI given the following:

  - Instruction cache miss rate: 3%
  - Data cache miss rate: 6%
  - Miss penalty: 100 cycles
  - Instructions are 25% loads and stores
  - Base CPI: 2 cycles
Solution:

Instruction cache miss cycles: we know we have 3% miss on instruction cache: this means 3% of the time our instructions take an extra 100 cycles.
Miss cycles / instruction = 0.03 * 100 cycles = 3 cycles / instruction

Data cache miss cycles: 25% of the instructions involve data access, and 6% of the time we have a data cache miss, so:
Miss cycles / instruction = 0.25 * 0.06 * 100 cycles = 1.5 cycles / instruction

So then the actual CPI can be computed as follows:
Actual CPI = Base CPI + IF stall cycles + data cache stall cycles
= 2 + 3 + 1.5
= 6.5
Assignment reminders

● Submit a .txt XOR a .pdf for each question
  ○ Do not submit both for the same question!
  ○ You may submit a .pdf for one question and a .txt for a different question

● Make sure your diagrams and tables are clear and easy to read
  ○ Make sure to leave enough space