Pipelining and Caching

CS230 Tutorial 09
Pipelining Hazards

- **Data hazard:** What happens when one instruction needs something that isn’t ready?
  - Example: `add $3, $1, $2`  
    `add $5, $3, $4`
  - This is solved by **forwarding** - an extra connection from one stage to the next
  - Sometimes forwarding isn’t enough, and the next instruction needs to **stall**

- **Control hazard:** How do we know what to do after a branch?
  - Example: `beq $0, $3, somelabel`  
    `add $5, $4, $2`
  - This is solved by **branch prediction** where we guess a branch and **stall** if we were wrong

- **Structural hazard:** Intrinsic resource contention, solved in hardware.
Forwarding “Connection Points” Diagram

When data is needed:

When data is available:
Data Hazard Example

Draw a pipeline diagram for the following MIPS assembly:

```
addi $7, $0, 16
add $8, $7, $0
lw $9, 0($30)
add $8, $9, $8
```
Data Hazard Example

Draw a pipeline diagram for the following MIPS assembly:

```
addi $7, $0, 16       IF   ID   EX   MEM   WB
add $8, $7, $0        IF   ID   EX   MEM   WB
lw $9, 0($30)         IF   ID   EX   MEM   WB
add $8, $9, $8        IF   ID   EX   MEM   WB
```

Notice the WB was writing to register $8 (instruction 2) at the same time ID was loading register $8 (instruction 4). This is a **structural hazard** that was mitigated by making sure WB always happens before ID in the same cycle.
Hazard Practice 1

Draw a pipeline diagram for the following MIPS assembly. Assume address 0x110 contains the value 256.

```
addi $7, $0, 0x100
lw $8, 16($7)
beq $7, $8, end
addi $8, $8, 17
end: jr $31
```
Hazard Practice 1

Draw a pipeline diagram for the following MIPS assembly. Assume address 0x110 contains the value 256.

```
addi $7, $0, 0x100  
lw $8, 16($7)       
beq $7, $8, end    
addi $8, $8, 17    
end: jr $31
```
Hazard Practice 2

Draw a pipeline diagram for the following MIPS assembly. Assume address 0x2A8 contains the value 15.

```
addi $7, $0, 0x154
sub $8, $0, $7
sub $7, $7, $8
lw $6, 0($7)
slt $5, $8, $6
beq $5, $0, e
addi $5, $5, 13
e:  jr $31
```
Hazard Practice 2 Draw a pipeline diagram for the following MIPS assembly. Assume address 0x2A8 contains the value 15.

```
addi $7, $0, 0x154  IF ID  EX  MEM  WB
sub $8, $0, $7      IF ID  EX  MEM  WB
sub $7, $7, $8      IF ID  EX  MEM  WB
lw $6, 0($7)        IF ID  EX  MEM  WB
slt $5, $8, $6      IF ID  EX  MEM  WB
stall               IF ID  EX  MEM  WB
beq $5, $0, e       IF ID  EX  MEM  WB
stall               IF ID  EX  MEM  WB
addi $5, $5, 13     IF ID  EX  MEM
```

e: jr $31
Cache Types

● Associativity:
  ○ Direct Mapped: Each memory block maps to 1 possible cache slot, based on address
  ○ Fully Associative: Allow a given block to go into any cache slot
  ○ N-way Associative: Group cache slots into sets of slots, each set containing N slots

● Write Mode:
  ○ Write-Through: writing to a block writes to memory immediately
  ○ Write-Back: writes go back to the cache, each cache line has a “dirty” bit
  ○ Write-Buffered: writes go to special buffer for blocks
Cache Types: Direct Map Example

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?

\[
\text{lw} \ $10, \ 0(\$1) \\
\text{lw} \ $11, \ 16(\$1) \\
\text{lw} \ $12, \ 64(\$1) \\
\text{lw} \ $13, \ 0(\$2) \\
\text{sw} \ $14, \ 0(\$2) \\
\text{lw} \ $15, \ 0(\$3)
\]

Assume \( \$1 = 0\times00003D80 \), \( \$2 = 0\times0000AC30 \), and \( \$3 = 0\times0000FE24 \).
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
$2 = 0x0000AC30
$3 = 0x0000FE24

```
<table>
<thead>
<tr>
<th>Block</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>Block 6</td>
<td></td>
<td>30</td>
<td>T</td>
</tr>
<tr>
<td>Block 7</td>
<td></td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>
```

```
“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...”
Cache Types: Direct Map Example

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

\[\text{lw } $10, 0($1)\]
\[\text{lw } $11, 16($1)\]
\[\text{lw } $12, 64($1)\]
\[\text{lw } $13, 0($2)\]
\[\text{sw } $14, 0($2)\]
\[\text{lw } $15, 0($3)\]
Assume $1 = 0x00003D80$
$2 = 0x0000AC30$
$3 = 0x0000FE24$

Data | Tag | Valid
--- | --- | ---
Block 0 | F | |
Block 1 | F | |
Block 2 | F | |
Block 3 | F | |
Block 4 | F | |
Block 5 | F | |
Block 6 | 30 | T
Block 7 | 30 | 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
Cache Types: Direct Map Example

- \(0x{AC30} / 0x{40} = 688 \div 8 = 86\ R\ 0\)
  - Block 688 maps to cache line 0 with tag 86

<table>
<thead>
<tr>
<th>Block</th>
<th>Address</th>
<th>Data</th>
<th>Tag</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x{3D80}</td>
<td>“A fish...”</td>
<td>86</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>6</td>
<td>0x{AC30}</td>
<td></td>
<td>30</td>
<td>T</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>30</td>
<td>T</td>
</tr>
</tbody>
</table>

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 $1 = 0x{3D80}$
$2 = 0x{AC30}$
$3 = 0x{FE24}$

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

- $0x\text{AC30} / 0x\text{40} = 688 \text{ R 48}$
- $688 / 8 = 86 \text{ R 0}$
  - 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 $1 = 0x\text{0003D80}$
$2 = 0x\text{0000AC30}$
$3 = 0x\text{0000FE24}$
$14 = 0x\text{47474747}$

<table>
<thead>
<tr>
<th>Cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>Tag</td>
<td></td>
</tr>
<tr>
<td>Valid</td>
<td></td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>Address</th>
<th>“Once upon…”</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3D80</td>
<td>Block 246</td>
</tr>
<tr>
<td>0xAC30</td>
<td>Block 247</td>
</tr>
<tr>
<td>0xFE24</td>
<td>Block 689</td>
</tr>
<tr>
<td>----------</td>
<td>----------------</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Address</th>
<th>“GGGGsh…”</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3DC0</td>
<td>Block 688</td>
</tr>
<tr>
<td>0xAC40</td>
<td>Block 689</td>
</tr>
<tr>
<td>0xFE24</td>
<td>Block 1016</td>
</tr>
<tr>
<td>----------</td>
<td>----------------</td>
</tr>
</tbody>
</table>
```

```
| “Once upon…” |
| “The tree…”  |
| “GGGGsh…”    |
| “My dog is…” |
| “Cheese is…” |
```

```
| “Once upon…” |
| “The tree…”  |
| “GGGGsh…”    |
```
# Cache Types: Direct Map Example

- **0xFE24 / 0x40 = 1016 R 36**
- **1016 / 8 = 127 R 0**
  - Block 1016 maps to cache line 0 with tag 127

- 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

<table>
<thead>
<tr>
<th>Cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Tag</td>
</tr>
<tr>
<td>&quot;Once upon...&quot;</td>
<td>Block 246</td>
</tr>
<tr>
<td>&quot;The tree...&quot;</td>
<td>Block 247</td>
</tr>
<tr>
<td>&quot;GGGGsh...&quot;</td>
<td>Block 688</td>
</tr>
<tr>
<td>&quot;My dog is...&quot;</td>
<td>Block 689</td>
</tr>
<tr>
<td>&quot;Cheese is...&quot;</td>
<td>Block 1016</td>
</tr>
</tbody>
</table>

- 0xFE24 / 0x40 = 1016 R 36
- 1016 / 8 = 127 R 0
- Block 1016 maps to cache line 0 with tag 127
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

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

<table>
<thead>
<tr>
<th>Block</th>
<th>Data</th>
<th>Tag</th>
<th>Valid</th>
<th>Dirty</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>“GGGGsh...”</td>
<td>86</td>
<td>T</td>
<td>Y</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>246</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>247</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>688</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>689</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1016</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory

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

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

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

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

```
<table>
<thead>
<tr>
<th>Block 0</th>
<th>127</th>
<th>T</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 1</td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Block 2</td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Block 3</td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Block 4</td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Block 5</td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Block 6</td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Block 7</td>
<td>F</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>
```

```
Once upon...
Block 246 Address 0x00003D80
The tree...
Block 247 Address 0x00003DC0
GGGGsh...
Block 688 Address 0x0000AC30
My dog is...
Block 689 Address 0x0000AC40
```

```
Block 1016 Address 0x0000FE24
```

"Cheese is..."
Cache Exercise Solutions

Consider a 2-way associative write back cache. It 8 blocks split into 4 sets and block size 16 words. What do memory and cache look like after the following instructions?

```
lw $10, 0($2)
sw $14, 0($3)
lw $15, 0($1)
```

Assume $1 = 0x00003D80
$s2 = 0x0000AC30
$s3 = 0x0000FE24
$s14 = 0x47474747
Cache Exercise Solutions

- block size is 64 = 0x40
- block 688 maps to set 0 with tag 172
- block 1016 maps to set 0 with tag 254
- block 246 maps to set 2 with tag 61

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

lw $10, 0($2)
sw $14, 0($3)
lw $15, 0($1)
Assignment and midterm reminders

- Always test your code on the university servers before you hand it in
- Hand in the .asm file (*not the .mips file*)