Hazards in a Pipelined CPU

Jason Mars
Dealing with Data Hazards

• In Software

• In Hardware

Data Hazards are caused by *instruction dependences*. For example, the add is data-dependent on the subtract:

```assembly
subi $5, $4, #45
add  $8, $5, $2
```
Dealing with Data Hazards

- In Software

  Nops!

- In Hardware

Data Hazards are caused by *instruction dependences*. For example, the add is data-dependent on the subtract:

```assembly
subi  $5, $4, #45
add   $8, $5, $2
```
Dealing with Data Hazards

• In Software
  
  Nops!

• In Hardware
  
  Stalls!

Data Hazards are caused by *instruction dependences*. For example, the add is data-dependent on the subtract:

```
  subi  $5, $4, #45
  add   $8, $5, $2
```
Dealing with Data Hazards

- In Software
  
  Nops!

- In Hardware

  Stalls!

  Forwarding!

Data Hazards are caused by *instruction dependences*. For example, the add is data-dependent on the subtract:

```
subi $5, $4, #45
add  $8, $5, $2
```
Software: Nop Insertion

sub $2, $1, $3

and $12, $2, $5
Hardware Stalls

sub $2, $1, $3

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)
Hardware Stalls

sub $2, $1, $3

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)
Hardware Stalls

sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
Hardware Stalls

- sub $2, $1, $3
- and $12, $2, $5
- or $13, $6, $2
- add $14, $2, $2
- sw $15, 100($2)
Hardware Stalls

sub $2, $1, $3

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)
## Step by Step

<table>
<thead>
<tr>
<th></th>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub</td>
<td>$2</td>
<td>$1</td>
<td>$3</td>
<td>IM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)
Step by Step

```
<table>
<thead>
<tr>
<th></th>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub</td>
<td>$2$, $1$, $3$</td>
<td>IM</td>
<td>Reg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>and</td>
<td>$12$, $2$, $5$</td>
<td>IM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td>$13$, $6$, $2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>$14$, $2$, $2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>$15$, $100$(</td>
<td>$2$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Step by Step

```
sub $2, $1, $3
IM
Reg

and $12, $2, $5
IM
Bubble

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)
```
Step by Step

- **sub $2, $1, $3**
  - IM
  - Reg
  - DM

- **and $12, $2, $5**
  - IM
  - Bubble
  - Bubble

- **or $13, $6, $2**

- **add $14, $2, $2**

- **sw $15, 100($2)**
Step by Step

```
<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>IM</td>
<td>Reg</td>
<td>DM</td>
<td>Reg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- sub $2, $1, $3
- and $12, $2, $5
- or $13, $6, $2
- add $14, $2, $2
- sw $15, 100($2)
Step by Step

sub $2, $1, $3

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)
Step by Step

sub $2, $1, $3

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)
Your Turn

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub $2, $1, $3</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $12, $3, $5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or $13, $6, $2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $14, $12, $2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw $14, 100($2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Your Turn

sub $2, $1, $3  
add $12, $3, $5  
or $13, $6, $2  
add $14, $12, $2  
sw $14, 100($2)

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IF           ID           EX       M       WB

IF          ID           EX       M        WB

IF

Friday, February 22, 13
Your Turn

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub $2, $1, $3</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $12, $3, $5</td>
<td></td>
<td></td>
<td>IF</td>
<td>ID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or $13, $6, $2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $14, $12, $2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw $14, 100($2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Instruction Flow Diagram]

IF           ID           EX
M         WB

IF          ID           EX
M         WB

Friday, February 22, 13
Your Turn

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub $2, $1, $3</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $12, $3, $5</td>
<td>IF</td>
<td>ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or $13, $6, $2</td>
<td>IF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $14, $12, $2</td>
<td>IF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw $14, 100($2)</td>
<td>IF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Friday, February 22, 13
### Your Turn

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub $2, $1, $3</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $12, $3, $5</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or $13, $6, $2</td>
<td>IF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $14, $12, $2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw $14, 100($2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

```
+-----------------+     +-----------------+     +-----------------+
| IM              | ->  | Reg          | ->  | DM           | ->  | Reg          |
| IF              |     | ID           |     | EX           |     | M            |     | WB           |
```
Your Turn

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub $2, $1, $3</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $12, $3, $5</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or $13, $6, $2</td>
<td>IF</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $14, $12, $2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw $14, 100($2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Your Turn

<table>
<thead>
<tr>
<th></th>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub</td>
<td>$2</td>
<td>$1</td>
<td>$3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>$12</td>
<td>$3</td>
<td>$5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td>$13</td>
<td>$6</td>
<td>$2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>$14</td>
<td>$12</td>
<td>$2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

sw $14, 100($2)
Your Turn

sub $2, $1, $3
add $12, $3, $5
or $13, $6, $2
add $14, $12, $2
sw $14, 100($2)

IF           ID           EX
M             WB

IF          ID           EX
M         WB

IF
ID
EX
M
WB
### Your Turn

**sub $2, $1, $3**

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**add $12, $3, $5**

| IF  | ID  | EX  | M   | WB  |     |     |     |

**or $13, $6, $2**

| IF  | B   | B   | ID  |     |     |     |     |

**add $14, $12, $2**

| IF  |     |     |     |     |     |     |     |

**sw $14, 100($2)**
### Your Turn

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub $2, $1, $3</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $12, $3, $5</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>or $13, $6, $2</td>
<td>IF</td>
<td>B</td>
<td>B</td>
<td>ID</td>
<td>EX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $14, $12, $2</td>
<td>IF</td>
<td>ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw $14, 100($2)</td>
<td>IF</td>
<td>Reg</td>
<td>Reg</td>
<td>DM</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
</tr>
</tbody>
</table>
Your Turn

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub $2, $1, $3</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $12, $3, $5</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>or $13, $6, $2</td>
<td>IF</td>
<td>B</td>
<td>B</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>add $14, $12, $2</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw $14, 100($2)</td>
<td>IF</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IF          ID           EX
M             WB
### Your Turn

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub $2, $1, $3</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $12, $3, $5</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>or $13, $6, $2</td>
<td>IF</td>
<td>B</td>
<td>B</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
</tr>
<tr>
<td>add $14, $12, $2</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw $14, 100($2)</td>
<td>IF</td>
<td>B</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Pipeline Diagram]

---

Friday, February 22, 13
Your Turn

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>B</td>
<td>B</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>B</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- sub $2, $1, $3
- add $12, $3, $5
- add $14, $12, $2
- sw $14, 100($2)
Pipeline Stalls

• To insure proper pipeline execution in light of register dependences, we must:

  • **detect** the hazard

  • **stall** the pipeline
Knowing When to Stall

6 types of data hazards
two reg reads * 3 reg writes
Knowing When to Stall

6 types of data hazards  
two reg reads * 3 reg writes
Knowing When to Stall

6 types of data hazards
two reg reads * 3 reg writes
Knowing When to Stall

6 types of data hazards
two reg reads * 3 reg writes
Thought Experiment: How do we know if we need to stall?
Thought Experiment: How do we know if we need to stall?
Thought Experiment: How do we know if we need to stall?

hint

read

RegWrite

Add

Shift

Add

Add

Add

Branch

MemWrite

MemRead

1

0

dest

dest

Friday, February 22, 13
Stalling the Pipeline

• Once we detect a hazard, then we have to be able to stall the pipeline (insert a bubble).

• Stalling the pipeline is accomplished by

  • (1) preventing the IF and ID stages from making progress
    • the ID stage because it cannot proceed until the dependent instruction completes
    • the IF stage because we do not want to lose any instructions.
  
  • (2) essentially, inserting “nops” in hardware
Stalling the Pipeline

• Preventing the IF and ID stages from proceeding
  • don’t write the PC (PCWrite = 0)
  • don’t rewrite IF/ID register (IF/IDWrite = 0)

• Inserting “nops”
  • set all control signals propagating to EX/MEM/WB to zero
The Gist of Detection

Elaboration: Regarding the remark earlier about setting control lines to 0 to avoid writing registers or memory: only the signals RegWrite and MemWrite need be 0, while the other control signals can be don't cares.

Thus far we have limited our concern to hazards involving arithmetic operations and data transfers. But as we saw in Section 6.1, there are also pipeline hazards involving branches. Figure 6.37 shows a sequence of instructions and indicates...

There are a thousand hacking at the branches of evil to one who is striking at the root.

Henry David Thoreau, *Walden*, 1854
What Else Can We Do?
What Else Can We Do?

Forwarding!
Forwarding

add $2, $3, $4

add $5, $3, $2
Forwarding

add $2, $3, $4
IM → Reg → ALU → DM → Reg

add $5, $3, $2
IM → Reg → ALU → DM → Reg

Registers → ALU → Data Memory → MEM/WB
Forwarding

\[ \text{add } $2, $3, $4 \]

\[ \text{add } $5, $3, $2 \]
Forwarding

add $2, $3, $4

add $5, $3, $2
FIGURE 6.30 On the top are the ALU and pipeline registers before adding forwarding. On the bottom, the multiplexors have been expanded to add the forwarding paths, and we show the forwarding unit. The new hardware is shown in color. However, this figure is a stylized drawing, leaving out details from the full datapath such as the sign extension hardware. Note that the ID/EX.RegisterRt field is shown twice, once to connect to the mux and once to the forwarding unit, but it is a single signal. As in the earlier discussion, this ignores forwarding of a store value to a store instruction.
Reducing Hazards via Forwarding

FIGURE 6.30 On the top are the ALU and pipeline registers before adding forwarding. On the bottom, the multiplexors have been expanded to add the forwarding paths, and we show the forwarding unit. The new hardware is shown in color. This figure is a stylized drawing, however, leaving out details from the full datapath such as the sign extension hardware. Note that the ID/EX.RegisterRt field is shown twice, once to connect to the mux and once to the forwarding unit, but it is a single signal. As in the earlier discussion, this ignores forwarding of a store value to a store instruction.
Reduction Hazards via Forwarding

![Diagram of pipeline stages and forwarding](image)

**Figure 6.30**: On the top are the ALU and pipeline registers before adding forwarding. On the bottom, the multiplexors have been expanded to add the forwarding paths, and we show the forwarding unit. The new hardware is shown in color. This figure is a stylized drawing, however, leaving out details from the full datapath such as the sign extension hardware. Note that the ID/EX.RegisterRt field is shown twice, once to connect to the mux and once to the forwarding unit, but it is a single signal. As in the earlier discussion, this ignores forwarding of a store value to a store instruction.
Reducing Hazards via Forwarding

**EX Hazard:**

if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA =

if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB =

(similar for the MEM stage)
Reducing Hazards via Forwarding

**EX Hazard:**

if (EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10

if (EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB =

(similar for the MEM stage)
Reducing Hazards via Forwarding

EX Hazard:
if (EX/MEM.RegWrite
    and (EX/MEM.RegisterRd != 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10
if (EX/MEM.RegWrite
    and (EX/MEM.RegisterRd != 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10
(similar for the MEM stage)
Data Forwarding

• The Previous Data Path handles two types of data hazards
  
  • EX hazard
  
  • MEM hazard

• We assume the register file handles the third (WB hazard)
  
  • if the register file is asked to read and write the same register in the same cycle, we assume that the reg file allows the write data to be forwarded to the output

• We’re still going to call that forwarding.
Eliminating Hazards via Forwarding

sub $2, $1, $3
and $6, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
Eliminating Hazards via Forwarding

```
sub $2, $1, $3
and $6, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
```
Eliminating Hazards via Forwarding

sub $2, $1, $3

and $6, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)
Eliminating Hazards via Forwarding

sub $2, $1, $3
and $6, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
Eliminating Hazards via Forwarding

- sub $2, $1, $3
- and $6, $2, $5
- or $13, $6, $2
- add $14, $2, $2
- sw $15, 100($2)
Forwarding in Action

\[ \text{add } $1, $2, $3 \quad \text{sub } $2, $3, $6 \quad \text{add } $3, $6, $4 \]
Forwarding in Action

add $1, $2, $3
sub $2, $3, $6
add $3, $6, $4
Forwarding in Action

add $1, $2, $3  sub $2, $3, $6  add $3, $6, $4
Forwarding in Action

add $1, $2, $3
sub $2, $3, $6
add $3, $6, $4
Forwarding in Action

- **add $1, $2, $3**
- **sub $2, $3, $6**

Friday, February 22, 13
Forwarding in Action

- add $1, $2, $3
- sub $2, $3, $6
Forwarding in Action

- add $1, $2, $3
- sub $2, $3, $6
Forwarding in Action

```
sub $2, $3, $6
add $1, $2, $3
```
Uh Oh... (what about lw)

lw $2, 10($1)
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
Uh Oh... (what about lw)

lw $2, 10($1)

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)
Uh Oh... (what about lw)

lw $2, 10($1)
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
Uh Oh... (what about lw)

lw $2, 10($1)

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)
Mixing Stalls and Forwarding

<table>
<thead>
<tr>
<th></th>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>$2, 10($1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>and</td>
<td>$12, $2, $5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td>$13, $6, $2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>$14, $2, $2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>$15, 100($2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Mixing Stalls and Forwarding

lw $2, 10($1)  
and $12, $2, $5  
or $13, $6, $2  
add $14, $2, $2  
sw $15, 100($2)
Mixing Stalls and Forwarding

lw $2, 10($1)

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)
Mixing Stalls and Forwarding

lw $2, 10($1) and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
Now We Need Hazard Detection

if (ID/EX.MemRead and
    ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
     (ID/EX.RegisterRt = IF/ID.RegisterRt)))
then stall the pipeline
Now We Need Hazard Detection

if (ID/EX.MemRead and 
     ((ID/EX.RegisterRt = IF/ID.RegisterRs) or 
      (ID/EX.RegisterRt = IF/ID.RegisterRt)))
then stall the pipeline
Now We Need Hazard Detection

if (ID/EX.MemRead and 
((ID/EX.RegisterRt = IF/ID.RegisterRs) or 
(ID/EX.RegisterRt = IF/ID.RegisterRt)))
then stall the pipeline
Key Points
Key Points

• Pipelining provides high throughput, but does not handle data dependences easily.
Key Points

• Pipelining provides high throughput, but does not handle data dependences easily.

• Data dependences cause *data hazards*. 
Key Points

• Pipelining provides high throughput, but does not handle data dependences easily.

• Data dependences cause *data hazards*.

• Data hazards can be solved by:
  • software (nops)
  • hardware stalling
  • hardware forwarding
Key Points

• Pipelining provides high throughput, but does not handle data dependences easily.

• Data dependences cause *data hazards*.

• Data hazards can be solved by:
  • software (nops)
  • hardware stalling
  • hardware forwarding

• Our processor, and indeed all modern processors, use a combination of forwarding and stalling.
Control (Branch) Hazards
Control Dependence

• Just as an instruction will be dependent on other instructions to provide its operands (_________ dependence), it will also be dependent on other instructions to determine whether it gets executed or not (_________ dependence or __________ dependence).

• Control dependences are particularly critical with ____________ branches.

  add $5, $3, $2
  sub $6, $5, $2
  beq $6, $7, somewhere
  and $9, $6, $1
  ...
  somewhere: or $10, $5, $2
  add $12, $11, $9
  ...

Friday, February 22, 13
Control Dependence

• Just as an instruction will be dependent on other instructions to provide its operands (data dependence), it will also be dependent on other instructions to determine whether it gets executed or not (control dependence or __________ dependence).

• Control dependences are particularly critical with __________ branches.

    add $5, $3, $2
    sub $6, $5, $2
    beq $6, $7, somewhere
    and $9, $6, $1
    ...

    somewhere: or $10, $5, $2
    add $12, $11, $9
    ...

    ...
Control Dependence

• Just as an instruction will be dependent on other instructions to provide its operands (data dependence), it will also be dependent on other instructions to determine whether it gets executed or not (branch dependence or ________ dependence).

• Control dependences are particularly critical with ____________ branches.

    add $5, $3, $2
    sub $6, $5, $2
    beq $6, $7, somewhere
    and $9, $6, $1
    ...

    somewhere: or $10, $5, $2
    add $12, $11, $9
    ...

Control Dependence

• Just as an instruction will be dependent on other instructions to provide its operands (data dependence), it will also be dependent on other instructions to determine whether it gets executed or not (branch dependence or control dependence).

• Control dependences are particularly critical with _____________ branches.

```
add $5, $3, $2
sub $6, $5, $2
beq $6, $7, somewhere
and $9, $6, $1
...
somewhere: or $10, $5, $2
add $12, $11, $9
...
```
Control Dependence

• Just as an instruction will be dependent on other instructions to provide its operands (data dependence), it will also be dependent on other instructions to determine whether it gets executed or not (branch dependence or control dependence).

• Control dependences are particularly critical with conditional branches.

```
add $5, $3, $2
sub $6, $5, $2
beq $6, $7, somewhere
and $9, $6, $1
...
somewhere: or $10, $5, $2
      add $12, $11, $9
...```
Branch Hazards

beq $2, $1, here

add ...

sub ...

lw ...

here: lw ...

Friday, February 22, 13
Branch Hazards

beq $2, $1, here

add ...

sub ...

lw ...

here: lw ...
Branch Hazards
Branch Hazards
Dealing with Branch Hazards

• Hardware
  • stall until you know which direction
  • reduce hazard through earlier computation of branch direction
  • guess which direction
    • assume not taken (easiest)
    • more educated guess based on history (requires that you know it is a branch before it is even decoded!)

• Hardware/Software
  • nops, or instructions that get executed either way (delayed branch).
Stalling for Branch Hazards

beq $4, $0, there

and $12, $2, $5

or ...

add ...

sw ...
Stalling for Branch Hazards

• Seems wasteful, particularly when the branch isn’t taken.

• Makes all branches cost 4 cycles.
Assume Branch *Not Taken*

```
beq $4, $0, there
and $12, $2, $5
or ...
add ...
sw ...
```

works pretty well when you’re right
Assume Branch *Not Taken*

beq $4, $0, there
and $12, $2, $5
or ...
add ...
there: sub $12, $4, $2

same performance as stalling when you’re wrong
Assume Branch *Not Taken*

- Performance depends on percentage of time you guess right.

- Flushing an instruction means to prevent it from changing any permanent state (registers, memory, PC).

  - sounds a lot like a bubble...

- But notice that we need to be able to insert those bubbles later in the pipeline
Reducing the Branch Delay
Reducing the Branch Delay

can easily get to 2-cycle stall
Reducing the Branch Delay can easily get to 2-cycle stall
Reducing the Branch Delay can easily get to 2-cycle stall
Stalling for Branch Delay

beq $4, $0, there

and $12, $2, $5

or ...

add ...

sw ...
Reducing the Branch Delay
Reducing the Branch Delay

Harder but possible to get to 1-cycle stall
Reducing the Branch Delay

Harder but possible to get to 1-cycle stall
Reducing the Branch Delay

Harder but possible to get to 1-cycle stall
Reducing the Branch Delay

FIGURE 6.38 The ID stage of clock cycle 3 determines that a branch must be taken, so it selects 72 as the next PC address and zeros the instruction fetched for the next clock cycle. Clock cycle 4 shows the instruction at location 72 being fetched and the single bubble or nop instruction in the pipeline as a result of the taken branch. (Since the nop is really sll $0, $0, 0, it's arguable whether or not the ID stage in clock 4 should be highlighted.)
Stalling for the Branch Delay

beq $4, $0, there

and $12, $2, $5

or ...

add ...

sw ...
Eliminating the Branch Stall

• There’s no rule that says we have to see the effect of the branch immediately. Why not wait an extra instruction before branching?

• The original SPARC and MIPS processors each used a single branch delay slot to eliminate single-cycle stalls after branches.

• The instruction after a conditional branch is always executed in those machines, regardless of whether the branch is taken or not.
Branch Delay Slot

beq $4, $0, there
and $12, $2, $5
there: or ...
add ...
sw ...

Branch delay slot instruction (next instruction after a branch) is executed even if the branch is taken.
Filling the Branch Delay Slot

• The branch delay slot is only useful if you can find something to put there.

• If you can’t find anything, you must put a \texttt{nop} to insure correctness.

• Where do we find instructions to fill the branch delay slot?
Filling the Branch Delay Slot

• Which instruction can we move into the branch delay slot?

add $5, $3, $7
sub $6, $1, $4
and $7, $8, $2
beq $6, $7, there
nop /* branch delay slot */
add $9, $1, $2
sub $2, $9, $5
...
there:
mult $2, $10, $11
Filling the Branch Delay Slot

• Which instruction can we move into the branch delay slot?

```
add   $5, $3, $7
sub   $6, $1, $4
and   $7, $8, $2
beq   $6, $7, there
nop   /* branch delay slot */
add   $9, $1, $2
sub   $2, $9, $5
...
there:
mult  $2, $10, $11
```
Branch Prediction

• Not all architectures have 1 cycle branch stalls! In fact most don’t.

• Always assuming the branch is not taken is a crude form of branch prediction.

• What about loops that are 95% of the time?

  • we would like the option of assuming not taken for some branches, and taken for others, depending on ???
Branch Prediction

- Not all architectures have 1 cycle branch stalls! In fact most don’t.

- Always assuming the branch is not taken is a crude form of branch prediction.

- What about loops that are **Taken** 95% of the time?

  - we would like the option of assuming not taken for some branches, and taken for others, depending on ???
Branch Prediction
Branch Prediction

• Historically, two broad classes of branch predictors:
Branch Prediction

• Historically, two broad classes of branch predictors:

• Static predictors – for branch B, always make the same prediction.
Branch Prediction

- Historically, two broad classes of branch predictors:

  - *Static predictors* – for branch B, always make the same prediction.

  - *Dynamic predictors* – for branch B, make a new prediction every time the branch is fetched.
Branch Prediction

• Historically, two broad classes of branch predictors:

  • Static predictors – for branch B, always make the same prediction.

  • Dynamic predictors – for branch B, make a new prediction every time the branch is fetched.

• Tradeoffs?
Branch Prediction

- Historically, two broad classes of branch predictors:
  - *Static predictors* – for branch B, always make the same prediction.
  - *Dynamic predictors* – for branch B, make a new prediction every time the branch is fetched.

- Tradeoffs?

- Modern CPUs all have sophisticated dynamic branch prediction.
Dynamic Branch Prediction

```c
for (i=0; i<10; i++) {
    ...
    ...
    add $i, $i, #1
    beq $i, #10, loop
}
```
Two Bit Predictors

This state machine also referred to as a *saturating counter* – it counts down (on *not taken*) to 00 or up (on *taken*) to 11, but does not wrap around.

```plaintext
for (i=0; i<10; i++) {
    ...
    ...
}
add $i, $i, #1
beq $i, #10, loop
```
A Million Branch Predictors

- Two-Level Adaptive
- Local Predictor
- Global Predictor
- Overriding
- Alloyed Predictor
- Agree Predictor
- Hybrid Predictor
- Tournament Predictor
Key Points
Key Points

• Control (or branch) hazards arise because we must fetch the next instruction before we know if we are branching or where we are branching.
Key Points

• Control (or branch) hazards arise because we must fetch the next instruction before we know if we are branching or where we are branching.

• Control hazards are detected in hardware.
Key Points

• Control (or branch) hazards arise because we must fetch the next instruction before we know if we are branching or where we are branching.

• Control hazards are detected in hardware.

• We can reduce the impact of control hazards through:
  
  • *early detection of branch address and condition*
  
  • *branch prediction*
  
  • *branch delay slots*