Agenda

- Data Hazard – Forwarding
- Control Hazard
Data Hazard
Consider a 5-stage pipeline:

Add $x3, x1, x2$

Sub $x5, x4, x3$

Data Hazard - forwarding

<table>
<thead>
<tr>
<th>cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

No Forwarding

With Forwarding
Data Hazard – extra hardware for forwarding

Consider a 5-stage pipeline:

\[
\text{add } x_3, x_1, x_2 \\
\text{sub } x_5, x_4, x_3
\]

<table>
<thead>
<tr>
<th>cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{PC} \\
\text{rs1} \\
\text{imm} \\
\text{rs2}
\]
Pipeline Exercise

Below is part of a pipelined datapath with synchronous reads and writes RegFile. Assuming no potential data hazards, this design still does not function correctly for r-type instructions (ignore immediate instructions for now). **What caused the error?** Add any extra components necessary to correct the design (Hint: which stage is we and rd being used? How about rs?)

These control signals are decoded in the ID stage.
Pipeline Exercise - Forwarding

Now this RegFile is modified and it has **synchronous writes** with **asynchronous reads**. Add appropriate forwarding to eliminate all data hazards (ignore hardware for immediate instructions for now).

(Hint: Assume the output of ALU will be used immediately in the next cycle)
Control Hazard
Control Hazard
Control Hazard – case 1

Branches are **not** taken by default

\[ x_1 = x_2 \]

```
beq x1, x2, imm
add x3, x1, x2
sub x4, x1, x2
xor x5, x1, x2
or x6, x1, x2
...
imm: and x3, x1, x2
nop
```
Control Hazard – case 2

Branches are **taken** by default
- with forwarding – assume no data hazard
- Assume x1 = x2

\[
\begin{align*}
\text{beq} & \quad x1, \ x2, \ \text{imm} \\
\text{add} & \quad x3, \ x1, \ x2 \\
\text{sub} & \quad x4, \ x1, \ x2 \\
\text{xor} & \quad x5, \ x1, \ x2 \\
\text{or} & \quad x6, \ x1, \ x2 \\
\ldots \\
\text{imm:} & \quad \text{and} \ x3, \ x1, \ x2 \\
\text{nop} & \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>#</th>
<th>IF</th>
<th>D</th>
<th>EX</th>
<th>M</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>beq</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>and</td>
<td>beq</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>nop</td>
<td>and</td>
<td>beq</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>nop</td>
<td>and</td>
<td>beq</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>nop</td>
<td>and</td>
<td>beq</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>nop</td>
<td>and</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>nop</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Control Hazard

- How do we modify the datapath to make sure branch address is available at next cycle? (Hint: what should we forward in this case?)
Control Hazard – Branch Prediction

- Base on last choice is a naïve but useful strategy in many cases
- Consider the following (very common case) code. If we predict based on previous branch result, what’s the success vs failure rate?

```c
int s = 0;
for (int i=0; i<100; i++)
{
    s += i;
}
```

```
addi x1 , x0 , 0
addi x2 , x0 , 1
addi x10, x0 , 100
add x1 , x1 , x2
addi x2 , x2 , 1
blt x2 , x10 , -8
nop
```

99% success

1% fail

1st: i = 0 predict taken 

i = 1 again

i = 99 again

i = 100 again

\( \checkmark \)

\( \times \)

\( \checkmark \)

\( \checkmark \)