**Forwarding: a hardware solution**

<table>
<thead>
<tr>
<th>IF stage</th>
<th>ID stage</th>
<th>EX stage</th>
<th>MEM stage</th>
<th>WB stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle 3</td>
<td>lw $3, 300($4) and $7, $8, $4</td>
<td>sub $1, $2, $4</td>
<td>add $4, $5, $6</td>
<td>add $4, $5, $6</td>
</tr>
<tr>
<td>Cycle 4</td>
<td>lw $3, 300($4) and $7, $8, $4</td>
<td>sub $1, $2, $4</td>
<td>sub $1, $2, $4</td>
<td>lw $3, 300($4)</td>
</tr>
<tr>
<td>Cycle 5</td>
<td>lw $3, 300($4) and $7, $8, $4</td>
<td>add $4, $5, $6</td>
<td>sub $1, $2, $4</td>
<td>add $4, $5, $6</td>
</tr>
</tbody>
</table>

*Forwarding:* Allow "add" to pass to the "sub" the data that it will write to $4:

- At the end of cycle 3, “add” stores the correct value of $4 in the EX/MEM buffer while “sub” stores the wrong value of $4 in the ID/EX buffer.
- At the start of cycle 4, forwarding replaces the incorrect data stored in the ID/EX buffer by the correct data stored in the ID/EX buffer.

**Forwarding from EX/MEM to ID/EX**

- The *sub* did read the wrong value from $4 (stored in ID/EX).
- The *add* has the correct value to be written in $4 (stored in EX/MEM).
- The *sub* can replace the wrong value read from $4 with the correct one (from EX/MEM) before feeding it to the ALU.
- Why two muxes??
More forwarding paths

Data dependences:
- add $4, $5, $6
- sub $1, $2, $4
- lw $3, 300($4)
- and $7, $8, $4
- sub $8, $9, $4

Solution: Data forwarding

<table>
<thead>
<tr>
<th>IF stage</th>
<th>ID stage</th>
<th>EX stage</th>
<th>MEM stage</th>
<th>WB stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle 3</td>
<td>lw $3, 300($4)</td>
<td>sub $1, $2, $4</td>
<td>add $4, $5, $6</td>
<td></td>
</tr>
<tr>
<td>Cycle 4</td>
<td>and $7, $8, $4</td>
<td>lw $3, 300($4)</td>
<td>sub $1, $2, $4</td>
<td>add $4, $5, $6</td>
</tr>
<tr>
<td>Cycle 5</td>
<td>sub $8, $9, $4</td>
<td>and $7, $8, $4</td>
<td>sub $1, $2, $4</td>
<td>add $4, $5, $6</td>
</tr>
</tbody>
</table>

Forwarding from MEM/WB to ID/EX

- The and did read the wrong value from $3 (stored in ID/EX)
- The add has the correct value to be written in $3 (stored in MEM/WB)

- The and can use the correct value that will be written in $3 (from MEM/WB)
Combining the two forwarding datapaths

The default is to select the value from ID/EX buffer

Select the value from the MEM/WB buffer if
- the instruction in MEM/WB will write into a register $X$
- the instruction in ID/EX did read from register $X$

Select the value from the EX/MEM buffer if
- the instruction in EX/MEM will write into a register $X$
- the instruction in ID/EX did read from register $X$

Setting the forwarding datapaths

- Set forward control signal, A, to 10 if
  - EX/MEM.RegisterWrite, and
  - EX/MEM.Rd = ID/EX.Rs, and
  - ID/EX.Rs!= 0

- Set forward control signal, B, to 10 if
  - EX/MEM.RegisterWrite, and
  - EX/MEM. Rd = ID/EX.Rt, and
  - ID/EX.Rt!= 0
Setting the forwarding datapaths

- Set forward control signal, A, to 01 if
  - MEM/WB.RegisterWrite, and
  - MEM/WB.Rd = ID/EX.Rs, and
  - ID/EX.Rs $\neq$ 0

- Set forward control signal, B, to 01 if
  - MEM/WB.RegisterWrite, and
  - MEM/WB.Rd = ID/EX.Rt, and
  - ID/EX.Rt $\neq$ 0

Forwarding may not be enough

<table>
<thead>
<tr>
<th>IF stage</th>
<th>ID stage</th>
<th>EX stage</th>
<th>MEM stage</th>
<th>WB stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle 3</td>
<td></td>
<td>lw $3$, $300($4)</td>
<td>add $1$, $6$, $7$</td>
<td>lw $3$, $300($4)</td>
</tr>
<tr>
<td>Cycle 4</td>
<td></td>
<td>&quot;and&quot; reads the wrong value of $S3$ in cycle 3</td>
<td>&quot;and&quot; uses the wrong value of $S3$ in cycle 4</td>
<td>lw $3$, $300($4)</td>
</tr>
<tr>
<td>Cycle 5</td>
<td></td>
<td>&quot;and&quot; writes to $S3$ in cycle 4</td>
<td>&quot;lw&quot; obtains the value to be written to $S3$ in cycle 4</td>
<td>lw $3$, $300($4)</td>
</tr>
<tr>
<td>Cycle 6</td>
<td></td>
<td></td>
<td></td>
<td>lw $3$, $300($4)</td>
</tr>
</tbody>
</table>

**Problem:** can’t use forwarding at the beginning of cycle 4 since "lw" produces the data to be written in $S3$ at the end of cycle 4

**Solution:** need to combine forwarding with stalling the pipe.

<table>
<thead>
<tr>
<th>IF stage</th>
<th>ID stage</th>
<th>EX stage</th>
<th>MEM stage</th>
<th>WB stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle 3</td>
<td></td>
<td>lw $3$, $300($4)</td>
<td>lw $3$, $300($4)</td>
<td>lw $3$, $300($4)</td>
</tr>
<tr>
<td>Cycle 4</td>
<td></td>
<td>and $7$, $8$, $3$</td>
<td>and $7$, $8$, $3$</td>
<td>lw $3$, $300($4)</td>
</tr>
<tr>
<td>Cycle 5</td>
<td></td>
<td>&quot;and&quot; stalls during cycle 4 and picks up the correct value of $S3$ from MEM/WB at the beginning of cycle 5</td>
<td>&quot;lw&quot; has the correct value of $S3$ in MEM/WB buffer at the end of cycle 4</td>
<td>lw $3$, $300($4)</td>
</tr>
</tbody>
</table>
Stalling the pipeline

• The `and` did read the wrong value from $3 (stored in ID/EX)
• The `lw` does not have the value to be written in $3
• We have to make sure that the `lw` and the `and` are separated by at least one instruction

• To separate `and` and `lw` by one instruction, we need to insert a bubble at run time (stall the pipeline).

Inserting a bubble (stalling the pipe)

Actually may detect the hazard and stall the pipe when `lw` is in ID/EX

The hazard occurs if:
- the instruction in ID/EX is a `lw` instruction (will read from memory)
- the instruction in ID/EX will write into a register $X$
- the instruction in IF/ID will read from register $X$

Stall the pipeline when a hazard is detected:
- freeze the contents of the IF/ID buffer and the PC
- insert a no-op into the ID/EX buffer
Hazard Detection (stallling) Unit

Hazard detection unit detects that:

\[ \text{ID/EX. MemRead} = 1 \text{ and } \]
\[ (\text{ID/EX. Rt} = \text{IF/ID. Rs} \text{ or } \text{ID/EX. Rt} = \text{IF/ID. Rt}) \]

Insert no-op:

- \( \text{ID/EX. MemWrite} \)
- \( \text{EX/MEM. RegWrite} \)
- \( \text{MEM/WB. RegWrite} \)
- \( \text{EX/MEM. Rd} \)
- \( \text{MEM/WB. Rd} \)

Freeze PC and IF/ID:

- \( \text{PCWrite} \)
- \( \text{IF/IDWrite} \)
- \( \text{ID/EX.Rt} \)
- \( \text{IF/ID.Rd} \)
- \( \text{ID/EX.Rt} \)
- \( \text{IF/ID.Rs} \)