The Pipeline

- What comparisons tell us when to stall?
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
Can we do better? How else might we deal with (some?) data hazards?
Reducing Data Hazards Through Forwarding

add $2, $3, $4

add $5, $3, $2
Reducing Data Hazards Through Forwarding
Reducing Data Hazards Through Forwarding

\[
EX\ Hazards: (similar\ for\ the\ MEM\ stage)
\]

\[
\text{if (EX/MEM.RegWrite and (EX/MEM.RegisterRd} \neq 0) \text{ and (EX/MEM.RegisterRd} = \text{ID/EX.RegisterRs)) ForwardA = 10}
\]

\[
\text{if (EX/MEM.RegWrite and (EX/MEM.RegisterRd} \neq 0) \text{ and (EX/MEM.RegisterRd} = \text{ID/EX.RegisterRt}) ForwardB = 10
\]
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 Data 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

add $1, $12, $3  
sub $12, $3, $4  
add $3, $10, $11

Memory Access  
Write Back

Diagram of a CPU pipeline with instructions and registers.
Forwarding in Action

Instruction Fetch

add $1, $12, $3

sub $12, $3, $4

add $3, $10, $11

Write Back
Forwarding in Action

Instruction Fetch

Instruction Decode

```
add $1, $12, $3
sub $12, $3, $4
add $3, $10, $11
```
Eliminating Every Data Hazard via Forwarding?

lw $2, 10($1)
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
## Eliminating Data Hazards via Forwarding and stalling

<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></td>
<td>and $12$, $2$, $5$</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>
Eliminating Data Hazards via Forwarding and stalling

```
lw $2, 10($1)
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
```
Eliminating Data Hazards via Forwarding and stalling

lw $2, 10($1)

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

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

lw $2, 10($1)

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

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

Just to be clear, let's review what we mean by “bubble” particularly in the context of this pipeline!
Eliminating Data Hazards via Forwarding and stalling

What is really happening during the bubble (for this particular pipeline)?
Eliminating Data Hazards via Forwarding and stalling

What is really happening during the bubble (for this particular pipeline)?

- While `lw` moves to the Mem stage in CC4, the `and` instruction repeats the ID stage (important because the values the `and` reads in CC4 are the ones it will carry forward).
Eliminating Data Hazards via Forwarding and stalling

What is really happening during the bubble (for this particular pipeline)?

- While `lw` moves to the Mem stage in CC4, the `and` instruction repeats the ID stage (important because the values the `and` reads in CC4 are the ones it will carry forward).
- There is now *no instruction* in the EX stage. So we better make sure that whatever is in the EX stage is safe.
Eliminating Data Hazards via Forwarding and stalling

What is really happening during the bubble (for this particular pipeline)?

• While `lw` moves to the Mem stage in CC4, the `and` instruction repeats the ID stage (important because the values the `and` reads in CC4 are the ones it will carry forward).

• There is now no instruction in the EX stage. So we better make sure that whatever is in the EX stage is safe.
  • Safe = no state changes (PC, reg, memory), now or as it moves through the pipeline.
Poll Q: Stalls & Forwards

• How many stalls occur and how many values require hardware forwarding support to avoid stalling for our MIPS 5-stage pipeline?

<table>
<thead>
<tr>
<th>Selection</th>
<th>Stalls</th>
<th>Forwarded values</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>None of the above</td>
<td></td>
</tr>
</tbody>
</table>

add $3, $2, $1
lw $4, 100($3)
and $6, $4, $3
sub $7, $6, $2
add $9, $3, $6
Try this one...

- Show bubbles and forwarding for this code

```
add $3, $2, $1
lw $4, 100($3)
and $6, $4, $3
sub $7, $6, $2
add $9, $3, $6
```
Another one...

• Show bubbles and forwarding for this code

\[\begin{align*}
&\text{lw} \quad $9, 100($6) & \text{IF} \\
&\text{addi} \quad $6, $9, #26 \\
&\text{sub} \quad $7, $6, $9 \\
&\text{add} \quad $6, $3, $6 \\
&\text{add} \quad $3, $2, $6
\end{align*}\]