Pipelining

CS230 Tutorial 07
Pipeline Stages

- Consider the following table of pipelining performance for a 5GHz CPU

  - Assume time for stages is
    - 100ps for register read/write
    - 200ps for other stages

<table>
<thead>
<tr>
<th>Instr</th>
<th>Instr fetch</th>
<th>Register read</th>
<th>ALU op</th>
<th>Memory access</th>
<th>Register write</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100 ps</td>
<td>800ps</td>
</tr>
<tr>
<td>sw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td></td>
<td>700ps</td>
</tr>
<tr>
<td>R-format</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td></td>
<td>100 ps</td>
<td>600ps</td>
</tr>
<tr>
<td>beq</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td></td>
<td></td>
<td>500ps</td>
</tr>
</tbody>
</table>
Non-Pipelined Performance Example

Use the table to solve the following:

How long does it take a non-pipelined CPU to execute the following instructions assuming register $10$ starts with value $5$?

```plaintext
loop: lw $2, 0($30)
      sw $1, 0($30)
      beq $0, $10, loop
      add $10, $0, $0
      beq $0, $10, loop2
      add $10, $10, $0
loop2: add $10, $10, $10
```

- assume time for stages is
  - 100ps for register read/write
  - 200ps for other stages

<table>
<thead>
<tr>
<th>Instr</th>
<th>Instr fetch</th>
<th>Register read</th>
<th>ALU op</th>
<th>Memory access</th>
<th>Register write</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100 ps</td>
<td>800ps</td>
</tr>
<tr>
<td>sw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td></td>
<td>700ps</td>
</tr>
<tr>
<td>R-format</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td></td>
<td>100 ps</td>
<td>600ps</td>
</tr>
<tr>
<td>beq</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td></td>
<td></td>
<td>500ps</td>
</tr>
</tbody>
</table>
Non-Pipelined Performance Example

How long does it take a non-pipelined CPU to execute the following instructions assuming register $10$ starts with value $5$?

```
loop:  lw $2, 0($30)
       sw $1, 0($30)
       beq $0, $10, loop
       add $10, $0, $0
       beq $0, $10, loop2
       add $10, $10, $0
loop2: add $10, $10, $10
```

800 (lw) + 700 (sw) + 500 (beq, branch not taken $10 != $0) + 600 (add) + 500 (beq, branch is taken, $10 == $0) + 600 (add) = 3700ps
Pipelining Hazards

- **Data hazard**: What happens when one instruction needs something that isn’t ready?
  - Example: \( \text{add }$3, $1, $2 \)
    \( \text{add }$5, $3, $4 \)
  - 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: \( \text{beq }$0, $3, somelabel \)
    \( \text{add }$5, $4, $2 \)
  - 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:
- bne
- beq
- jr
- jalr

When data is available:
- everything
- sw (non mem)
- sw (mem register only)
- lw
Data Hazard Example

Draw a pipeline diagram for the following MIPS assembly:

```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:

\[
\begin{align*}
\text{addi} & \quad \text{IF} \quad \text{ID} \quad \text{EX} \quad \text{MEM} \quad \text{WB} \\
\text{add} & \quad \text{IF} \quad \text{ID} \quad \text{EX} \quad \text{MEM} \quad \text{WB} \\
\text{lw} & \quad \text{IF} \quad \text{ID} \quad \text{EX} \quad \text{MEM} \quad \text{WB} \\
\text{add} & \quad \text{IF} \quad \text{ID} \quad \text{EX} \quad \text{MEM} \quad \text{WB} \\
\end{align*}
\]

Notice the WB was writing to register $8$ (instruction 2) at the same time ID was loading register $8$ (instruction 4). This is a \textit{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.

```assembly
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)
b.eq $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
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
Cycles per Instruction

- Count up the total length of cycles and divide by instructions.
- For non-pipelined, it is also the total number of stages.
- For pipelined, you need to draw out the diagram in order to handle stalls.