Reconsider the Pipelined datapath (slide 21)

There is a problem with the above datapath!!

Corrected datapath
Pipeline Control

We have 5 stages, each having its own control signals

Pipeline Control

- Pass control signals along just like the data

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Execution/Address Calculation stage control lines</th>
<th>Memory access stage control lines</th>
<th>Write-back stage control lines</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reg Op1 ALU Op0 ALU Src</td>
<td>Branch Mem Read Mem Write</td>
<td>Reg Write Mem to Reg</td>
</tr>
<tr>
<td>R-format</td>
<td>1 1 0 0 0 0 0 0 1 0 0 1 0 0 1 0</td>
<td>0 0 1 0 0 1 0 0 1 0</td>
<td>0 0 0 1 0 0 1 0 1 0</td>
</tr>
<tr>
<td>lw</td>
<td>X 0 0 1 1 0 0 1 0 0 1 0 0 1 0 1</td>
<td>0 0 1 0 0 1 0 0 1 0</td>
<td>0 0 0 1 0 0 1 0 1 0</td>
</tr>
<tr>
<td>beq</td>
<td>X 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0</td>
<td>0 0 1 0 0 1 0 0 1 0</td>
<td>0 0 0 1 0 0 1 0 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Control</th>
<th>WB</th>
<th>M</th>
<th>EX</th>
<th>M</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IF/ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ID/EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EX/MEM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MEM/WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pipeline control

Datapath with Control
representation of pipelining

<table>
<thead>
<tr>
<th>Cycle 1</th>
<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></td>
<td>add $4$, $5$, $6$</td>
<td>add $4$, $5$, $6$</td>
<td>add $4$, $5$, $6$</td>
<td>add $4$, $5$, $6$</td>
<td>add $4$, $5$, $6$</td>
</tr>
<tr>
<td>Cycle 2</td>
<td>and $1$, $2$, $3$</td>
<td>and $1$, $2$, $3$</td>
<td>and $1$, $2$, $3$</td>
<td>and $1$, $2$, $3$</td>
<td>and $1$, $2$, $3$</td>
</tr>
<tr>
<td>Cycle 3</td>
<td>lw $3$, 300($0)$</td>
<td>lw $3$, 300($0)$</td>
<td>lw $3$, 300($0)$</td>
<td>lw $3$, 300($0)$</td>
<td>lw $3$, 300($0)$</td>
</tr>
<tr>
<td>Cycle 4</td>
<td>sub $7$, $8$, $9$</td>
<td>sub $7$, $8$, $9$</td>
<td>sub $7$, $8$, $9$</td>
<td>sub $7$, $8$, $9$</td>
<td>sub $7$, $8$, $9$</td>
</tr>
</tbody>
</table>

Can help with answering questions like:
- how many cycles does it take to execute this code?
- what is the ALU doing during cycle 4?
- At what cycle does a specific instruction exits the pipeline?

**Pipeline throughput**: Because the pipeline completes the execution of one instruction every clock cycle (after an initial delay to fill up the pipeline), then the CPI = 1.

Equivalent representations of pipelining

<table>
<thead>
<tr>
<th>Cycle 1</th>
<th>Cycle 2</th>
<th>Cycle 3</th>
<th>Cycle 4</th>
<th>Cycle 5</th>
<th>Cycle 6</th>
<th>Cycle 7</th>
<th>Cycle 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>REG</td>
<td>IF</td>
<td>REG</td>
<td>IF</td>
<td>REG</td>
<td>IF</td>
<td>REG</td>
</tr>
<tr>
<td>add $4$, $5$, $6$</td>
<td>add $4$, $5$, $6$</td>
<td>add $4$, $5$, $6$</td>
<td>add $4$, $5$, $6$</td>
<td>add $4$, $5$, $6$</td>
<td>add $4$, $5$, $6$</td>
<td>add $4$, $5$, $6$</td>
<td></td>
</tr>
<tr>
<td>and $1$, $2$, $3$</td>
<td>and $1$, $2$, $3$</td>
<td>and $1$, $2$, $3$</td>
<td>and $1$, $2$, $3$</td>
<td>and $1$, $2$, $3$</td>
<td>and $1$, $2$, $3$</td>
<td>and $1$, $2$, $3$</td>
<td></td>
</tr>
<tr>
<td>lw $3$, 300($0)$</td>
<td>lw $3$, 300($0)$</td>
<td>lw $3$, 300($0)$</td>
<td>lw $3$, 300($0)$</td>
<td>lw $3$, 300($0)$</td>
<td>lw $3$, 300($0)$</td>
<td>lw $3$, 300($0)$</td>
<td></td>
</tr>
<tr>
<td>sub $7$, $8$, $9$</td>
<td>sub $7$, $8$, $9$</td>
<td>sub $7$, $8$, $9$</td>
<td>sub $7$, $8$, $9$</td>
<td>sub $7$, $8$, $9$</td>
<td>sub $7$, $8$, $9$</td>
<td>sub $7$, $8$, $9$</td>
<td></td>
</tr>
</tbody>
</table>

<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 1</td>
<td>add $4$, $5$, $6$</td>
<td>add $4$, $5$, $6$</td>
<td>add $4$, $5$, $6$</td>
<td>add $4$, $5$, $6$</td>
</tr>
<tr>
<td>Cycle 2</td>
<td>and $1$, $2$, $3$</td>
<td>and $1$, $2$, $3$</td>
<td>and $1$, $2$, $3$</td>
<td>and $1$, $2$, $3$</td>
</tr>
<tr>
<td>Cycle 3</td>
<td>lw $3$, 300($0)$</td>
<td>lw $3$, 300($0)$</td>
<td>lw $3$, 300($0)$</td>
<td>lw $3$, 300($0)$</td>
</tr>
<tr>
<td>Cycle 4</td>
<td>sub $7$, $8$, $9$</td>
<td>sub $7$, $8$, $9$</td>
<td>sub $7$, $8$, $9$</td>
<td>sub $7$, $8$, $9$</td>
</tr>
</tbody>
</table>
Pipeline hazards

What makes pipelining easy for MIPS instructions
- all instructions have the same length
- just a few instruction formats
- memory operands appear only in loads and stores

What can go wrong?
- structural hazards: would occur if we had only one memory
- data hazards: may occur if an instruction depends on a previous instruction
- control hazards: may occur when executing branch instructions

We'll look at these hazards in the context of our simple pipeline

---

Structural hazards

Example: imagine a pipelined car wash with one hose for both soaking and rinsing.

Soak  Brush  Rinse  Dry

How do you solve the problem?
(share hardware, replicate hardware, compete for hardware)

Are there similar problems in our MIPS architecture?
**Structural hazards in MIPS**

**Potential problem:** Both IF and MEM use memory  
**Solution:** use separate memories (caches)

**Potential problem:** Both REG and WB use the register file  
**Solution:** Read from register file during the first half of a cycle and write to register file during the second half of a cycle.

---

**Data hazards (computing with wrong data)**

Assume that registers 4, 5 and 6 contain the values 100, 200 and 300, respectively

**Expected execution:**

<table>
<thead>
<tr>
<th>$4$ contains 100</th>
<th>$4$ should contain 500</th>
<th>$4$ should contain 500</th>
<th>$4$ should contain 500</th>
</tr>
</thead>
<tbody>
<tr>
<td>add $4$, $5$, $6$</td>
<td>sub $1$, $2$, $4$</td>
<td>lw $3$, 300($4)</td>
<td>and $7$, $8$, $4$</td>
</tr>
<tr>
<td>$// write 500 into register 4$</td>
<td>$// reads the value 500 from register 4$</td>
<td>$// reads the value 500 from register 4$</td>
<td>$// reads the value 500 from register 4$</td>
</tr>
</tbody>
</table>

**Pipelined execution:**

<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>sub $1$, $2$, $4$</td>
<td>add $4$, $5$, $6$; lw $3$, 300($4)</td>
<td>add $4$, $5$, $6$; sub $1$, $2$, $4$; lw $3$, 300($4)</td>
<td>add $4$, $5$, $6$; sub $1$, $2$, $4$</td>
<td>add $4$, $5$, $6$; sub $1$, $2$, $4$</td>
</tr>
<tr>
<td>$// sub$ reads 100 from register 4 (was supposed to read 500)</td>
<td>$// sub$ uses 100 in calculation leading to wrong result</td>
<td>$// add$ had computed the value 500, which is to be written into $4$.</td>
<td>$\text{The value 500 is written into }$</td>
<td>$\text{The value 500 is written into }$</td>
</tr>
</tbody>
</table>

---

**What can be done to mitigate data hazards?**
Data Hazards

- **Definition**: two instructions are data-dependent if the first writes into a register, R, and the other follows it and reads from R.
- To reduce the possibility of data hazards, writing into the register file should be performed in the first half of a cycle, and reading from the register file in the second half.
- If data-dependent instructions are not separated by at least two instructions, they will cause data hazards when executing on the pipeline.

**Example:**

```
add $3, $1, $2
lw $1, 0($3)
add $2, $4, $3
lw $4, $5, $1
sw $2, 20($4)
add $2, $1, $1
```

**A software solution:**

```
add $3, $1, $2
no-op
```

```
add $2, $4, $3
no-op
```

```
lw $1, 0($3)
add $2, $4, $3
```

```
lw $4, $5, $1
sw $2, 20($4)
```

```
add $2, $1, $1
```

**A compiler optimization**

The compiler can also reorder instructions to eliminate or reduce the number of inserted no-ops.

```
lw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
lw $t2, 4($t0)
add $t2, $t2, $s1
sw $t2, 4($t0)
```

```
lw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
lw $t2, 4($t0)
add $t2, $t2, $s1
sw $t2, 4($t0)
```

```
lw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
lw $t2, 4($t0)
add $t2, $t2, $s1
sw $t2, 4($t0)
```

```
lw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
add $t2, $t2, $s1
sw $t2, 4($t0)
```

```
lw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
add $t2, $t2, $s1
sw $t2, 4($t0)
```

```
lw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
add $t2, $t2, $s1
sw $t2, 4($t0)
```

```
lw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
add $t2, $t2, $s1
sw $t2, 4($t0)
```

```
lw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
add $t2, $t2, $s1
sw $t2, 4($t0)
```

```
lw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
add $t2, $t2, $s1
sw $t2, 4($t0)
```

```
lw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
add $t2, $t2, $s1
sw $t2, 4($t0)
```

```
lw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
add $t1, $t1, $s1
sw $t1, 0($t0)
add $t2, $t2, $s1
sw $t2, 4($t0)
```