EN164: Design of Computing Systems
Lecture 15: Processor / Pipeline Design 1

Professor Sherief Reda
http://scale.engin.brown.edu
Electrical Sciences and Computer Engineering
School of Engineering
Brown University
Spring 2011

[ material from Patterson & Hennessy, 4th ed and Harris 1st ed ]
Pipelining analogy

- Pipelined laundry: overlapping execution
  - Parallelism improves performance

- Four loads:
  - Speedup = 16/7 = 2.3

- Non-stop:
  - Ideal Speedup = 4n/(n + 3) ≈ 4
    = number of stages
MIPS stages

Five stages, one step per stage
1. IF: Instruction fetch from memory
2. ID: Instruction decode & register read
3. EX: Execute operation or calculate address
4. MEM: Access memory operand
5. WB: Write result back to register
MIPS datapath pipeline stages

- Need registers between stages
  - To hold information produced in previous cycle
Pipeline datapath abstraction

- Form showing resource usage
Multi-cycle datapath pipeline diagram

- **Traditional form**

![Pipeline Diagram]

Program execution order (in instructions):
- `lw $10, 20($1)`
- `sub $11, $2, $3`
- `add $12, $3, $4`
- `lw $13, 24($1)`
- `add $14, $5, $6`
Tracing `lw` in its journey: 1\textsuperscript{st} cycle
Tracing \texttt{lw} in its journey: 2\textsuperscript{nd} cycle
Tracing lw in its journey: 3rd cycle
Tracing lw in its journey: 4th cycle
Tracing \texttt{lw} in its journey: 5\textsuperscript{th} cycle

![Diagram showing the 5\textsuperscript{th} cycle of processing an \texttt{lw} instruction, with a note indicating a wrong register number.]
Corrected pipeline datapath for lw
Pipeline state in 5th cycle

lw $10, 20($1)
sub $11, $2, $3
add $12, $3, $4
lw $13, 24($1)
add $14, $5, $6
Pipeline performance

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- Compare pipelined datapath with single-cycle datapath

<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>
Single-cycle versus pipeline performance

**Single-cycle (T_c = 800ps)**

```
Program execution order
(in instructions)
```

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Time (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $1, 100($0)</td>
<td>200</td>
</tr>
<tr>
<td>lw $2, 200($0)</td>
<td>600</td>
</tr>
<tr>
<td>lw $3, 300($0)</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Pipelined (T_c = 200ps)**

```
Program execution order
(in instructions)
```

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Time (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $1, 100($0)</td>
<td>200</td>
</tr>
<tr>
<td>lw $2, 200($0)</td>
<td>400</td>
</tr>
<tr>
<td>lw $3, 300($0)</td>
<td>600</td>
</tr>
</tbody>
</table>

S. Reda EN164 Sp '11
Pipeline speedup

- If all stages are balanced
  - i.e., all take the same time
  - \[\text{Time between instructions}_{\text{pipelined}} = \frac{\text{Time between instructions}_{\text{nonpipelined}}}{\text{Number of stages}}\]
- If not balanced, speedup is less
- Speedup due to increased throughput
  - Latency (time for each instruction) does not decrease