Schedule

- 4/18 No class
- 4/21 Midterm exam
- 4/25 지정좌석제 갱신
Exam #2 (Midterm)

- April 21 (Thursday), 2011
- 13:30 – 14:45
- #400102 (even student IDs)
- #400112 (odd student IDs)

Scope
- Chap. 1
- Chap. 2 (except 2.11)
- Chap. 3 (except 3.7)
- Chap. 4.1 – 4.4

Closed-book exam
Pipelining

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
A Real Pipelining Example
Pipelining Analogy

- Pipelined laundry: overlapping execution
  - Parallelism improves performance

  - Four loads:
    - Speedup
      - $= 8/3.5 = 2.3$
  - Non-stop:
    - Speedup
      - $= 2n/(0.5n+1.5) \approx 4$
      - = number of stages
MIPS Pipeline

Five stages, one step per stage

- **IF**: Instruction fetch from memory
- **ID**: Instruction decode & register read
- **EX**: Execute operation or calculate address
- **MEM**: Access memory operand
- **WB**: Write result back to register
Pipeline Performance (1)

- **Assume time for stages is**
  - 100ps for register read or 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>
Pipeline Performance (2)

- Single-cycle datapath vs. Pipelined datapath

**Single-cycle**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Time (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $1, 100(0)$</td>
<td>800</td>
</tr>
<tr>
<td>lw $2, 200(0)$</td>
<td>200</td>
</tr>
<tr>
<td>lw $3, 300(0)$</td>
<td>200</td>
</tr>
</tbody>
</table>

**Pipelined**

<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>200</td>
</tr>
<tr>
<td>lw $3, 300(0)$</td>
<td>200</td>
</tr>
</tbody>
</table>

Single-cycle: $T_c = 800\text{ps}$
Pipelined: $T_c = 200\text{ps}$
Pipeline Performance (3)

- **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
Pipelining and ISA Design

- MIPS ISA designed for pipelining
  - All instructions are 32-bits
    - Easier to fetch and decode in one cycle
    - (cf.) x86: 1- to 17-byte instructions
  - Few and regular instruction formats
    - Can decode and read registers in one step
  - Load/store addressing
    - Can calculate address in 3rd stage, access memory in 4th stage
  - Alignment of memory operands
    - Memory access takes only one cycle
MIPS Pipelined Datapath

Right-to-left flow leads to hazards
Pipeline Registers

- Need registers between stages
  - To hold information produced in previous cycle
IF for Load
ID for Load
EX for Load
WB for Load

Wrong register number
Correct Datapath for Load
IF for Store
ID for Store
EX for Store

Diagram showing the execution stage in computer architecture.
MEM for Store
WB for Store
Multi-Cycle Pipeline Diag. (1)

- Form showing resource usage

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
Multi-Cycle Pipeline Diagram (2)

- Traditional form

<table>
<thead>
<tr>
<th>Time (in clock cycles)</th>
<th>CC 1</th>
<th>CC 2</th>
<th>CC 3</th>
<th>CC 4</th>
<th>CC 5</th>
<th>CC 6</th>
<th>CC 7</th>
<th>CC 8</th>
<th>CC 9</th>
</tr>
</thead>
</table>

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

- Instruction fetch
- Instruction decode
- Execution
- Data access
- Write back

- Instruction fetch
- Instruction decode
- Execution
- Data access
- Write back

- Instruction fetch
- Instruction decode
- Execution
- Data access
- Write back

- Instruction fetch
- Instruction decode
- Execution
- Data access
- Write back
Single-Cycle Pipeline Diagram

- State of pipeline in a given cycle

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Cycle 1</th>
<th>Cycle 2</th>
<th>Cycle 3</th>
<th>Cycle 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add $14, $5, $6</td>
<td>Instruction fetch</td>
<td>Instruction decode</td>
<td>Execution</td>
<td>Memory</td>
</tr>
<tr>
<td>Lw $13, 24 ($1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lw $10, 20($1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pipelined Control (Simplified)
Pipelined Control (1)

- Control signals derived from instruction
  - As in single-cycle implementation
Pipelined Control (2)