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
  \[ \frac{8}{3.5} = 2.3 \]

Non-stop:
- Speedup
  \[ \frac{2n}{0.5n+1.5} \approx 4 \]
  = 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

```
IF   ID   EX   MEM   WB
IF   ID   EX   MEM   WB
IF   ID   EX   MEM   WB
...  ...
```
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

Program execution order (in instructions)

**Single-cycle:** $T_c = 800\text{ps}$

- lw $1, 100(0)$
- lw $2, 200(0)$
- lw $3, 300(0)$

Program execution order (in instructions)

**Pipelined:** $T_c = 200\text{ps}$

- lw $1, 100(0)$
- lw $2, 200(0)$
- lw $3, 300(0)$
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
ID for Load
EX for Load
MEM for Load
WB for Load

Wrong register number
Correct Datapath for Load
IF for Store

Instruction fetch

IF/ID

ID/EX

EX/MEM

MEM/WB

0 Mux

PC

Add

4

0 Mux

Address

Instruction memory

Instruction

Read register 1

Read register 2

Write register

Write data

16

32

Sign-extend

Add result

Shift left 2

Add

ALU

Zero ALU result

0 Mux

Mux

Address

Read data

Data memory

Write data

Read data
ID for Store

Instruction decode
EX for Store

EX/MEM

MEM/WB

ID/EX

IF/ID

Instruction memory

Address

0 Mux

PC

Add

4

Read register 1
Read register 2
Write register
Write data

Instruction

Read data 1
Read data 2

Zero ALU result

Shift left 2

Add

sw

Execution

Data memory

Address

0 Mux

Read data

Write data

16

Sign-extend

32

ALU

0 Mux

ICE3003: Computer Architecture | Spring 2012 | Jin-Soo Kim (jinsookim@skku.edu)
MEM for Store
WB for Store

- Instruction memory
- Address
- PC
- Add
- 4
- 0 MUX
- Instruction
- IF/ID
- ID/EX
- EX/MEM
- MEM/WB
- SW
- Add result
- Shift left 2
- ALU
- Zero
- Data memory
- Address
- Read data
- Write data
- 16
- 32
- Sign-extend
- 0 MUX
- 0 MUX
Multi-Cycle Pipeline Diagram (1)

- Form showing resource usage

Time (in clock cycles)
CC 1  CC 2  CC 3  CC 4  CC 5  CC 6  CC 7  CC 8  CC 9

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>Program execution order (in instructions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $10, 20($1)</td>
</tr>
<tr>
<td>sub $11, $2, $3</td>
</tr>
<tr>
<td>add $12, $3, $4</td>
</tr>
<tr>
<td>lw $13, 24($1)</td>
</tr>
<tr>
<td>add $14, $5, $6</td>
</tr>
</tbody>
</table>

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

- **lw $10, 20($1)**
  - Instruction fetch
  - Instruction decode
  - Execution
  - Data access
  - Write back

- **sub $11, $2, $3**
  - Instruction fetch
  - Instruction decode
  - Execution
  - Data access
  - Write back

- **add $12, $3, $4**
  - Instruction fetch
  - Instruction decode
  - Execution
  - Data access
  - Write back

- **lw $13, 24($1)**
  - Instruction fetch
  - Instruction decode
  - Execution
  - Data access
  - Write back

- **add $14, $5, $6**
  - 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</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add $14, $5, $6</td>
<td>IF/ID</td>
<td>Instruction fetch</td>
</tr>
<tr>
<td>lw $13, 24 ($1)</td>
<td>ID/EX</td>
<td>Instruction decode</td>
</tr>
<tr>
<td>add $12, $3, $4</td>
<td>EX/MEM</td>
<td>Execution</td>
</tr>
<tr>
<td>sub $11, $2, $3</td>
<td>MEM/WB</td>
<td>Memory</td>
</tr>
<tr>
<td>lw $10, 20($1)</td>
<td></td>
<td>Write-back</td>
</tr>
</tbody>
</table>
Pipelined Control (Simplified)
Pipelined Control (1)

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