Review Problem 28

- Given what we know about pipelining, assume in a widget factory it takes 40 minutes to make 1 widget. If we pipeline the process into S stages, how long will it take to make N widgets?

Ideal case:

\[ \text{Cycle Time} = \frac{40\text{ min}}{S} \]

\[ \text{Exec (N)} = 40 + (N \times \frac{40}{S}) \]

\[ = (N + \text{Drain}) \times \text{Cycle Time} = (N + (S-1)) \times \frac{40}{S} \]

Problems: "chain by" stages, time to move between stages
Pipelined Datapath

Divide datapath into multiple pipeline stages

IF Instruction Fetch

RF Register Fetch

EX Execute

MEM Data Memory

WB Writeback

PC → Instr. Memory → Register File → Register → Data Memory → Register File
Pipelined Control

The Main Control generates the control signals during Reg/Dec
Control signals for Exec (ALUOp, ALUSrc, ...) are used 1 cycle later
Control signals for Mem (MemWE, Mem2Reg, ...) are used 2 cycles later
Control signals for Wr (RegWE, ...) are used 3 cycles later
Can pipelining get us into trouble?

Yes: **Pipeline Hazards**

**structural hazards**: attempt to use the same resource two different ways at the same time

E.g., combined washer/dryer would be a structural hazard or folder busy doing something else (watching TV)

**data hazards**: attempt to use item before it is ready

E.g., one sock of pair in dryer and one in washer; can’t fold until get sock from washer through dryer

instruction depends on result of prior instruction still in the pipeline

**control hazards**: attempt to make decision before condition evaluated

E.g., washing football uniforms and need to get proper detergent level; need to see after dryer before next load in branch instructions

Can always resolve hazards by **waiting**

pipeline control must detect the hazard
take action (or delay action) to resolve hazards
Pipelining the Load Instruction

The five independent functional units in the pipeline datapath are:

- Instruction Memory for the *Ifetch* stage
- Register File’s Read ports (bus A and busB) for the *Reg/Dec* stage
- ALU for the *Exec* stage
- Data Memory for the *Mem* stage
- Register File’s Write port (bus W) for the *Wr* stage
The Four Stages of R-type

Ifetch: Fetch the instruction from the Instruction Memory
Reg/Dec: Register Fetch and Instruction Decode
Exec: ALU operates on the two register operands
Wr: Write the ALU output back to the register file
Structural Hazard

Interaction between R-type and loads causes structural hazard on writeback.
Important Observation

Each functional unit can only be used once per instruction.
Each functional unit must be used at the same stage for all instructions:

Load uses Register File’s Write Port during its 5th stage

<table>
<thead>
<tr>
<th>Load</th>
<th>Ifetch</th>
<th>Reg/Dec</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

R-type uses Register File’s Write Port during its 4th stage

<table>
<thead>
<tr>
<th>R-type</th>
<th>Ifetch</th>
<th>Reg/Dec</th>
<th>Exec</th>
<th>Wr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Solution: Delay R-type’s register write by one cycle:

Now R-type instructions also use Reg File’s write port at Stage 5
Mem stage is a NOOP stage: nothing is being done.

<table>
<thead>
<tr>
<th>R-type</th>
<th>Ifetch</th>
<th>Reg/Dec</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Pipelining the R-type Instruction

Clock

<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>
<th>Cycle 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R-type</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R-type</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R-type</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Last #1

Read Ports

Write Port
The Four Stages of Store

Ifetch: Fetch the instruction from the Instruction Memory
Reg/Dec: Register Fetch and Instruction Decode
Exec: Calculate the memory address
Mem: Write the data into the Data Memory
Wr: NOOP

Compatible with Load & R-type instructions
The Stages of Conditional Branch

Ifetch: Fetch the instruction from the Instruction Memory
Reg/Dec: Register Fetch and Instruction Decode, compute branch target
Exec: Test condition & update the PC
Mem: NOOP
Wr: NOOP
Control Hazard

Branch updates the PC at the end of the Exec stage.

<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>
<th>Cycle 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R-type

<table>
<thead>
<tr>
<th>Ifetch</th>
<th>Reg/Dec</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
</table>

CBZ

<table>
<thead>
<tr>
<th>Ifetch</th>
<th>Reg/Dec</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
</table>

load

<table>
<thead>
<tr>
<th>Ifetch</th>
<th>Reg/Dec</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
</table>

R-type

<table>
<thead>
<tr>
<th>Ifetch</th>
<th>Reg/Dec</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
</table>

R-type

<table>
<thead>
<tr>
<th>Ifetch</th>
<th>Reg/Dec</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
</table>

R-type

<table>
<thead>
<tr>
<th>Ifetch</th>
<th>Reg/Dec</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
</table>

PC

PC+4/PC+4*SEC
Accelerate Branches

When can we compute branch target address?
When can we compute the CBZ condition?

The only ARM branches are those that can be computed quickly.
Control Hazard 2

Branch updates the PC at the end of the Reg/Dec stage.

Cycle 1 | Cycle 2 | Cycle 3 | Cycle 4 | Cycle 5 | Cycle 6 | Cycle 7 | Cycle 8 | Cycle 9
---|---|---|---|---|---|---|---|---
Clock

R-type | Ifetch | Reg/Dec | Exec | Mem | Wr
---|---|---|---|---|---
CBZ

load

R-type | Ifetch | Reg/Dec | Exec | Mem | Wr
---|---|---|---|---|---

R-type | Ifetch | Reg/Dec | Exec | Mem | Wr
---|---|---|---|---|---

Cycle 1 | Cycle 2 | Cycle 3 | Cycle 4
---|---|---|---

Beq | Ifetch | Reg/Dec | Exec | Mem | Wr
Solution #1: Stall

Delay loading next instruction, load no-op instead

Clock

<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>
<th>Cycle 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBZ</td>
<td>Ifetch</td>
<td>Reg/Dec</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
<td>Stall</td>
<td>Bubble</td>
<td>Bubble</td>
</tr>
</tbody>
</table>

Stall

PC+4: Branch

R-type | Ifetch | Reg/Dec | Exec | Mem | Wr

R-type | Ifetch | Reg/Dec | Exec | Mem | Wr

CPI if all other instructions take 1 cycle, and branches are 20% of instructions?

$$CPI = 0.8 \times 1 + 0.2 \times 2 = 0.8 + 0.4 = 1.2$$

20% slower
Solution #2: Branch Prediction

Guess all branches not taken, squash if wrong

<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>
<th>Cycle 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R-type

<table>
<thead>
<tr>
<th>Ifetch</th>
<th>Reg/Dec</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
</table>

CBZ

<table>
<thead>
<tr>
<th>Ifetch</th>
<th>Reg/Dec</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Ifetch</th>
<th>Reg/Dec</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
</table>

Guess the branch PC+4: load is not taken

R-type

<table>
<thead>
<tr>
<th>Ifetch</th>
<th>Reg/Dec</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
</table>

R-type

<table>
<thead>
<tr>
<th>Ifetch</th>
<th>Reg/Dec</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
</table>

R-type

<table>
<thead>
<tr>
<th>Ifetch</th>
<th>Reg/Dec</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
</table>

CPI if 50% of branches actually not taken, and branch frequency 20%?

$$CPI = 0.8 \times 1 + 2(0.5 \times 1 + 0.5 \times 2) = 1.1$$

10% slowdown