How Computers Work
Lecture 12

Introduction to Pipelining

A Common Chore of College Life
Propagation Times

\[ T_{pd_{\text{wash}}} = \underline{\quad} \quad T_{pd_{\text{dry}}} = \underline{\quad} \]

Doing 1 Load

Step 1:  \rightarrow

Step 2:  \rightarrow

Total Time = \underline{\quad}  = \underline{\quad}
Doing 2 Loads
Combinational (Harvard) Method

Step 1: 
Step 2: 
Step 3: 
Step 4: 

Total Time = ________

Doing 2 Loads
Pipelined (MIT) Method

Step 1: 
Step 2: 
Step 3: 

Total Time = ________

How Computers Work Lecture 12 Page 5

How Computers Work Lecture 12 Page 6
Doing N Loads

- Harvard Method: ________________
- MIT Method: ________________

A Few Definitions

Latency: Time for 1 object to pass through entire system.

\( \text{=} \quad \text{for Harvard laundry} \)
\( \text{=} \quad \text{for MIT laundry} \)

Throughput: Rate of objects going through.

\( \text{=} \quad \text{for Harvard laundry} \)
\( \text{=} \quad \text{for MIT laundry} \)
A Computational Problem

Add 4 Numbers:

A + B + C + D

As a Combinational Circuit

Throughput
1 / 2 Tpd

Latency
2 Tpd
As a Pipelined Circuit

Throughput: \( \frac{1}{T_{pd}} \)

Latency: \( 2T_{pd} \)

Simplifying Assumptions

1. Synchronous inputs
2. \( T_s = T_h = 0 \)
   \( T_{pd} c-q = 0 \)
   \( T_{cd} c-q = 0 \)
An Inhomogeneous Case
(Combinational)

Throughput
1 / 3
Latency
3

An Inhomogeneous Case
(Pipelined)

Throughput
1 / 2
Latency
4
How about this one?

How MIT Students *REALLY* do Laundry

Steady State Throughput = _________
Steady State Latency = _________
Interleaving (an alternative to Pipelining)

For \( N \) Units of delay \( T_{pd} \), steady state Throughput \( \frac{N}{T_{pd}} \)

Latency \( T_{pd} \)

Interleaving Parallel Circuits

**clk**

1-4

**sel**

1 2 3 4

\( x \) \( x \) \( x \) \( x \)
Definition of a Well-Formed Pipeline

- Same number of registers along path from any input to every computational unit
  - Insures that every computational unit sees inputs in PHASE
- Is true (non-obvious) whenever the # of registered between all inputs and all outputs is the same.

Method for Forming Well-Formed Pipelines

- Add registers to system output at will
- Propagate registers from intermediate outputs to intermediate inputs, cloning registers as necessary.
Method for Maximizing Throughput

- Pipeline around longest latency element
- Pipeline around other sections with latency as large as possible, but <= longest latency element.

\[
\begin{align*}
\text{Comb. Latency} & = 5 \\
\text{Comb. Throughput} & = \frac{1}{5} \\
\text{Pipe. Latency} & = 6 \\
\text{Pipe. Throughput} & = \frac{1}{2}
\end{align*}
\]

A Few Questions

- Assuming a circuit is pipelined for optimum throughput with 0 delay registers, is the pipelined throughput always greater than or equal to the combinational throughput?
  - A: Yes
- Is the pipelined latency ever less than combinational latency?
  - A: No
- When is the pipelined latency equal to combinational latency?
  - A: If contents of all pipeline stages have equal combinational latency
CPU Performance

MIPS = Millions of Instructions Per Second
Freq = Clock Frequency, MHz
CPI = Clocks per Instruction

\[
\text{MIPS} = \frac{\text{Freq}}{\text{CPI}}
\]

To Increase MIPS:
1. DECREASE CPI.
   - RISC reduces CPI to 1.0.
   - CPI < 0? Tough... we'll see multiple instruction issue machines at end of term.
2. INCREASE Freq.
   - Freq limited by delay along longest combinational path; hence
   - PIPELINING is the key to improved performance through fast clocks.

Review: A Top-Down View of the Beta Architecture

With st(ra,C,rc) : Nom[C+<rc>] <- <ra>
Pipeline Stages

GOAL: Maintain (nearly) 1.0 CPI, but increase clock speed.

APPROACH: structure processor as 4-stage pipeline:

Instruction Fetch stage: Maintains PC, fetches one instruction per cycle and passes it to

Register File stage: Reads source operands from register file, passes them to

ALU stage: Performs indicated operation, passes result to

Write-Back stage: writes result back into register file.

WHAT OTHER information do we have to pass down the pipeline?

Sketch of 4-Stage Pipeline
4-Pipeline Parallelism

Consider a sequence of instructions:

- ADDC(r1, 1, r2)
- SUBC(r1, 1, r3)
- XOR(r1, r5, r1)
- MUL(r1, r2, r0)

Executed on our 4-stage pipeline:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>IF</th>
<th>RF</th>
<th>ALU</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDC(r1,1,r2)</td>
<td>R1 Read</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBC(r1,1,r3)</td>
<td></td>
<td>R3 Read</td>
<td></td>
<td>R1 Written</td>
</tr>
<tr>
<td>XOR(r1,r5,r1)</td>
<td></td>
<td>R1,R5 Read</td>
<td></td>
<td>R3 Written</td>
</tr>
<tr>
<td>MUL(r1,1,r2)</td>
<td></td>
<td>R1,R2 Read</td>
<td>R1</td>
<td>R0 Written</td>
</tr>
</tbody>
</table>

Time
Pipeline Problems

**BUT, consider instead:**

**LOOP:**
- ADD(r1, r2, r3)
- CMPLEC(r3, 100, r0)
- BT(r0, LOOP)
- XOR(r31, r31, r3)
- MUL(r1, r2, r2)
- ...

**But, consider instead:**

- RF
- ALU
- WB
- ALU
- WB
- WBIF
- MUL(r1, r2, r2)
- RFIF
- XOR(r31, r31, r3)
- ALU
- WB
- IF
- BT(r0, LOOP)
- WB
- IF
- CMPLEC(r3, 100, r0)
- WB
- IF
- ADD(r1, r2, r3)
- ...

**Pipeline Hazards**

**PROBLEM:**
Contents of a register WRITTEN by instruction k is READ by instruction k+1... before it is stored in RF! EG:

ADD(r1, r2, r3)
CMPLEC(r3, 100, r0)
MUL(r1, 100, r4)
SUB(r1, r2, r5)

fails since CMPLEC sees "stale" <r3>.
SOLUTIONS:

1. "Program around it".
   - Document weirdo semantics, declare it a software problem.
   - Breaks sequential semantics!
   - Costs code efficiency.

EXAMPLE: Rewrite

```
ADD(r1, r2, r3)
CMPLE(r3, 100, r0)
MULC(r1, 100, r4)
SUB(r1, r2, r5)
```

R3 Written

```
ADD(r1, r2, r3)
MULC(r1, 100, r4)
SUB(r1, r2, r5)
CMPLE(r3, 100, r0)
```

```
HOW OFTEN can we do this? How Computers Work Lecture 12 Page 31
```

DRAWBACK: SLOW

```
ADD(r1, r2, r3)  IF  RF  ALU  WB
MULC(r1, 100, r4)  RF  ALU  WB
SUB(r1, r2, r5)  RF  ALU  WB
CMPLE(r3, 100, r0)  RF  ALU  WB
```

```
ADD(r1, r2, r3)  IF  RF  ALU  WB
MULC(r1, 100, r4)  IF  RF  ALU  WB
SUB(r1, r2, r5)  IF  RF  ALU  WB
CMPLE(r3, 100, r0)  IF  RF  ALU  WB
```

```
RF
ALU
WB
```

```
RF
ALU
WB
```

```
R3 Written
```

```
R3 Written
```

```
SOLUTIONS:
2. Stall the pipeline.
   Freeze IF, RF stages for 2 cycles,
   inserting NOPs into ALU IR...
```

```
RF
ALU
WB
```

```
RF
ALU
WB
```

```
R3 Written
```

```
R3 Written
```

```
DRAWBACK: SLOW
```

```
How Computers Work Lecture 12 Page 32
```
SOLUTIONS:

Add extra data paths & control logic to re-route data in problem cases.

<table>
<thead>
<tr>
<th></th>
<th>IF</th>
<th>RF</th>
<th>ALU</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD(r1, r2, r3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMPEQ(r1, r2, r3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEQ(r1, r2, r3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XOR(r1, r2, r3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL(r1, r2, r3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How Computers Work Lecture 12 Page 33
Next Time:

- Detailed Design of
  - Bypass Paths + Control Logic
- What to do when Bypass Paths Don’t Work
  - Branch Delays / Tradeoffs
  - Load/Store Delays / Tradeoffs
  - Multi-Stage Memory Pipeline