Pipelining

CS 281 Lecture Notes
12/5/2016
Pipelining

1) Start with Examples

- Tag Makings
  - Cut
  - Sand
  - Add Nails/Decorations
  - Paint

- Laundry
  - Wash
  - Dry
  - Fold

Start to Finish Time -- **latency**

Did it improve with pipelining?

No!

What did improve?

Suppose 50 people to do laundry: Total Time?

Non-pipelined: \( 50 \times (1 + \frac{1}{4} + \frac{1}{2}) \)

Pipelined: \( 92 \)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jill</td>
<td>W</td>
<td>D</td>
</tr>
<tr>
<td>2</td>
<td>Joe</td>
<td>W</td>
<td>D</td>
</tr>
<tr>
<td>3</td>
<td>Bob</td>
<td>W</td>
<td>D</td>
</tr>
<tr>
<td>4</td>
<td>Sue</td>
<td>W</td>
<td>D</td>
</tr>
<tr>
<td>50</td>
<td>Total</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IF
320 ps shortest time for cycle
1 unit work/320 ps, so throughput rate is \( \frac{1}{320 \cdot 10^{-12}} = 3.125 \cdot 10^9 \text{ units/sec} \)
Suppose can divide into 3 stages of 100 ps

Need able to do "hold" info that is input to
1 stage \( \Rightarrow \) mem or registers

Notes different regs

Pipeline Diagram

\( \text{New latency: } 360 \text{ ps} \) 
1 unit work/120 ps = \( \frac{1}{120 \cdot 10^{-12}} = 8.33 \cdot 10^9 \text{ units/sec} \)
What happens when we cannot evenly divide?

```plaintext
300

50

150

100

Clock determined by longest less

150 + 20 = shortest cycle

Latency:

# stages x cycle = 170 - 3 = 570

Throughput:

1 instruction/unit / 170 ps

= \frac{1}{170 \cdot 10^{-12}} = 5.89 \cdot 10^9 \text{ ops/sec}

Challenge: Balance
```

What is limit? How deeply can we pipeline?

20 ps latches & cogs
So we do 20 6-stages 30 + 20 500 10 stages 30 + 20

420

Problem in dependencies

Why Grail? No

Data Hazard

Goal: SPu Pipelining

Fetch

Decode

Execute

Write-back
Fetch
- Select current PC
- Read instruction
- Compute incremented PC

Decode
- Read program registers

Execute
- Operate ALU

Memory
- Read or write data memory

Write Back
- Update register file
irmovq $1,%rax  #I1
irmovq $2,%rbx  #I2
irmovq $3,%rcx  #I3
irmovq $4,%rdx  #I4
halt          #I5