### Poll Q: What affects throughput?  
**Peak throughput depends on...**

<table>
<thead>
<tr>
<th></th>
<th>Single Cycle</th>
<th>Multi-Cycle</th>
<th>Pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Longest Instruction</td>
<td>Cycle Time</td>
<td>Average Instruction</td>
</tr>
<tr>
<td>B</td>
<td>Longest Instruction</td>
<td>Cycle Time</td>
<td>Longest Instruction</td>
</tr>
<tr>
<td>C</td>
<td>Longest Instruction</td>
<td>Average Instruction</td>
<td>Cycle Time</td>
</tr>
<tr>
<td>D</td>
<td>Average Instruction</td>
<td>Longest Instruction</td>
<td>Cycle Time</td>
</tr>
<tr>
<td>E</td>
<td>None of the above</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Poll Q: What affects throughput? 
Peak throughput depends on...

<table>
<thead>
<tr>
<th></th>
<th>Single Cycle</th>
<th>Multi-Cycle</th>
<th>Pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Longest Instruction</td>
<td>Average Instruction</td>
<td>Cycle Time</td>
</tr>
</tbody>
</table>

Throughput is useful work over time – one measure: insts / sec

\[
ET = \text{Inst} \times \text{CPI} \times \text{CT}
\]

- Single Cycle: \(ET = \text{Inst} \times \frac{\text{BIG}}{3..5} \times \text{CT}\)
- Multi Cycle: \(ET = \text{Inst} \times \frac{\text{BIG}}{3..5} \times \text{CT}\)
- Pipeline: \(ET = \text{Inst} \times \frac{\text{BIG}}{3..5} \times \text{CT}\)
Pipelining in Modern CPUs

- CPU Datapath
- Arithmetic Units
- System Buses
- Software (at multiple levels)
- etc...
A Pipelined Datapath

IF  Instruction fetch
ID  Instruction decode and register fetch
EX  Execution and effective address calculation
MEM Memory access
WB  Write back
Pipelined Datapath (roughly)
Execution in a Pipelined Datapath

#1 lw
#2 lw
#3 lw
#4 lw
#5 lw
Execution in a Pipelined Datapath
Mixed Instructions in the Pipeline

lw
add
Mixed Instructions in the Pipeline

\[ \text{lw} \quad \text{add} \]

- IM
- Reg
- ALU
- DM
- Reg

CC1 | CC2 | CC3 | CC4 | CC5 | CC6
Mixed Instructions in the Pipeline

lw

add
Mixed Instructions in the Pipeline

lw

add
Mixed Instructions in the Pipeline

This is called a **structural hazard** – too many instructions want to use the same resource.
In our pipeline, we can make this hazard disappear (next slide).
In more complex pipelines, structural hazards are again possible.
Pipeline Principles

• All instructions that share a pipeline should have the same *stages* in the same *order*.
  – therefore, *add* does nothing during Mem stage
  – *sw* does nothing during WB stage

• All intermediate values must be latched each cycle.
Pipeline stages

- What is the performance implication of making every instruction go through all 5 stages? (e.g., instead of 4 for add, 3 for `beq`, etc.)

<table>
<thead>
<tr>
<th></th>
<th>(Choose BEST answer)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Decreases peak throughput by 20%</td>
</tr>
<tr>
<td>B</td>
<td>Increases program latency by 20%</td>
</tr>
<tr>
<td>C</td>
<td>No significant impact on peak throughput or program latency</td>
</tr>
<tr>
<td>D</td>
<td>Depends on how many R-type instructions, <code>beq</code>, etc.</td>
</tr>
<tr>
<td>E</td>
<td>None of the above</td>
</tr>
</tbody>
</table>

(Choose BEST answer)
Pipelined Datapath

Instruction Fetch
Instruction Decode/ Register Fetch
Execute/ Address Calculation
Memory Access
Write Back

IF

IF/ID
ID/EX
EX/MEM
MEM/Write

Address
Instruction memory

PC

Add

4

Read register 1
Read register 2
Write register
Write data

Shift left 2

Add
Add result

Zero
ALU result

MUX

Address
Data memory

Read data

Write data

MUX
Pipelined Datapath

Instruction Fetch  Instruction Decode/Register Fetch  Execute/Address Calculation  Memory Access  Write Back

registers!