Designing a Pipelined CPU

Instruction Latencies and Throughput

- **Single-Cycle CPU**
  - Load: Ifetch, Reg/Dec, Exec, Mem, Wr

- **Multiple Cycle CPU**
  - Cycle 1: Ifetch, Reg/Dec, Exec, Mem, Wr
  - Cycle 2: Ifetch, Reg/Dec, Exec, Mem, Wr
  - Cycle 3: Ifetch, Reg/Dec, Exec, Mem, Wr
  - Cycle 4: Ifetch, Reg/Dec, Exec, Mem, Wr
  - Cycle 5: Ifetch, Reg/Dec, Exec, Mem, Wr

- **Pipelined CPU**
  - Cycle 1: Ifetch, Reg/Dec, Exec, Mem, Wr
  - Cycle 2: Ifetch, Reg/Dec, Exec, Mem, Wr
  - Cycle 3: Ifetch, Reg/Dec, Exec, Mem, Wr
  - Cycle 4: Ifetch, Reg/Dec, Exec, Mem, Wr
  - Cycle 5: Ifetch, Reg/Dec, Exec, Mem, Wr
  - Cycle 6: Ifetch, Reg/Dec, Exec, Mem, Wr
  - Cycle 7: Ifetch, Reg/Dec, Exec, Mem, Wr
  - Cycle 8: Ifetch, Reg/Dec, Exec, Mem, Wr

Pipelining Advantages

- Higher throughput
- Higher of CPU resources
- But, more complicated, more complex control(?)

<table>
<thead>
<tr>
<th>CPU Design Technology</th>
<th>Control Logic</th>
<th>Peak Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Cycle CPU</td>
<td>Combinational Logic</td>
<td>1</td>
</tr>
<tr>
<td>Multiple-Cycle CPU</td>
<td>FSM or Microprogram</td>
<td>1</td>
</tr>
<tr>
<td>Pipelined CPU</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
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
Mixed Instructions in the Pipeline

- **Iw** to **IM**
- **add** to **IM**

Pipeline Principles

- All instructions that share a pipeline must have the same **stages** in the same
  - therefore, **add** does nothing during Mem stage
  - **sw** does nothing during WB stage
- All intermediate values must be latched each cycle.
- There is no **reuse**

Pipelined Datapath

The Pipeline in Execution
The Pipeline in Execution

Instruction Fetch
Instruction Decode/
Register Fetch
Address Calculation

sub $15, $4, $1
lw $12, 1000($4)

The Pipeline, with controls
But….

CSE 141
Dean Tullsen

Pipelined Control

- FSM not really appropriate.
  - !
    - signals generated, but follow instruction through the pipeline

CSE 141
Dean Tullsen
Pipelined Control Signals

<table>
<thead>
<tr>
<th>Instruction</th>
<th>RegDst</th>
<th>ALUOp1</th>
<th>ALUOp0</th>
<th>ALUSrc</th>
<th>Branch</th>
<th>MemRead</th>
<th>MemWrite</th>
<th>Write</th>
<th>Write</th>
<th>MemRef</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-Format</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SW</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>BRq</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Is it really that easy?

- What happens when...
  - add $3, $10, $11
  - lw $8, 1000($3)
  - sub $11, $8, $7
  - add $3, $10, $11

The Pipeline with Control Logic

The Pipeline in Execution

- add $10, $1, $2
- sub $11, $8, $7
- lw $8, 1000($3)
- Write Back
Data Hazards

- When a result is needed in the pipeline before it is available, a “data hazard” occurs.

Pipelining Key Points

- ET = IC * CPI * CT
- We achieve high without reducing instruction latency.
- Pipelining exploits a special kind of parallelism (parallelism between functionality required in different cycles).
- Pipelining uses combinational logic to generate (and registers to propagate) control signals.
- Pipelining creates potential hazards.