Pipelining

- Forecast
  - Big Picture
  - Datapath
  - Control
Motivation

- Single cycle implementation
  - CPI = 1
  - Cycle = $\text{imem} + \text{RFrd} + \text{ALU} + \text{dmem} + \text{RFwr} + \text{muxes} + \text{control}$
  - E.g. $500 + 250 + 500 + 500 + 250 + 0 + 0 = 2000\text{ps}$
  - Time/program = $P \times 2\text{ns}$
### Multicycle

- **Multicycle implementation:**

<table>
<thead>
<tr>
<th>Cycle:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr:</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td>D</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>i+4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>
Multicycle

• Multicycle implementation
  – CPI = 3, 4, 5
  – Cycle = max(memory, RF, ALU, mux, control)
  – = max(500,250,500) = 500ps
  – Time/prog = P x 4 x 500 = P x 2000ps = P x 2ns

• Would like:
  – CPI = 1 + overhead from hazards (later)
  – Cycle = 500ps + overhead
  – In practice, ~3x improvement
Big Picture

• Instruction latency = 5 cycles
• Instruction throughput = 1/5 instr/cycle
• CPI = 5 cycles per instruction
• Instead
  – Pipelining: process instructions like a lunch buffet
  – ALL microprocessors use it
    • E.g. Intel Core i7, AMD Jaguar, ARM A9
Big Picture

- Instruction Latency = 5 cycles (same)
- Instruction throughput = 1 instr/cycle
- CPI = 1 cycle per instruction
- CPI = cycle between instruction completion = 1
Ideal Pipelining

• Bandwidth increases linearly with pipeline depth
• Latency increases by latch delays
Example: Integer Multiplier

16x16 combinational multiplier
ISCAS-85 C6288 standard benchmark
Tools: Synopsys DC/LSI Logic 110nm gflxp ASIC

[Source: J. Hayes, Univ. of Michigan]
# Example: Integer Multiplier

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Delay</th>
<th>MPS</th>
<th>Area (FF/wiring)</th>
<th>Area Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combinational</td>
<td>3.52ns</td>
<td>284</td>
<td>7535 (--/1759)</td>
<td></td>
</tr>
<tr>
<td>2 Stages</td>
<td>1.87ns</td>
<td>534 (1.9x)</td>
<td>8725 (1078/1870)</td>
<td>16%</td>
</tr>
<tr>
<td>4 Stages</td>
<td>1.17ns</td>
<td>855 (3.0x)</td>
<td>11276 (3388/2112)</td>
<td>50%</td>
</tr>
<tr>
<td>8 Stages</td>
<td>0.80ns</td>
<td>1250 (4.4x)</td>
<td>17127 (8938/2612)</td>
<td>127%</td>
</tr>
</tbody>
</table>

**Pipeline efficiency**  
- 2-stage: nearly double throughput; marginal area cost  
- 4-stage: 75% efficiency; area still reasonable  
- 8-stage: 55% efficiency; area more than doubles  

**Tools:** Synopsys DC/LSI Logic 110nm gflxp ASIC
## Ideal Pipelining

<table>
<thead>
<tr>
<th>Cycle:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr:</td>
<td>i</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>i+1</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>i+2</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>i+3</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>i+4</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pipelining Idealisms

• Uniform subcomputations
  – Can pipeline into stages with equal delay

• Identical computations
  – Can fill pipeline with identical work

• Independent computations
  – No relationships between work units
  – No *dependences*, hence no pipeline hazards

• Are these practical?
  – No, but can get close enough to get significant speedup
Complications

• Datapath
  – Five (or more) instructions in flight

• Control
  – Must correspond to multiple instructions

• Instructions may have
  – data and control flow *dependencies*
  – I.e. units of work are not independent
    • One may have to stall and wait for another
Datapath

IF: Instruction fetch
ID: Instruction decode/register file read
EX: Execute/address calculation
MEM: Memory access
WB: Write back
Control

• Control
  – Concurrently set by 5 different instructions
  – Divide and conquer: carry IR down the pipe
Pipelined Datapath

• Start with single-cycle datapath
• Pipelined execution
  – Assume each instruction has its own datapath
  – But each instruction uses a different part in every cycle
  – Multiplex all on to one datapath
  – Latches separate cycles (like multicycle)

• Ignore dependences and hazards for now
  – Data
  – control
Pipelined Datapath
Pipelined Datapath

• Instruction flow
  – add and load
  – Write of registers
  – Pass register specifiers

• Any info needed by a later stage gets passed down the pipeline
  – E.g. store value through EX
Pipelined Control

• IF and ID
  – None
• EX
  – ALUop, ALUsrc, RegDst
• MEM
  – Branch, MemRead, MemWrite
• WB
  – MemtoReg, RegWrite
Datapath Control Signals
Pipelined Control

Instruction

IF/ID

ID/EX

EX/MEM

MEM/WB
Pipelined Control

• Controlled by different instructions
• Decode instructions and pass the signals down the pipe
• Control sequencing is embedded in the pipeline
  – No explicit FSM
  – Instead, distributed FSM
Summary

- Big Picture
- Datapath
- Control
- Next
  - Program dependences
  - Pipeline hazards