Computer Architecture: Lecture “6”
Multicycle MIPS Implementation
Single-Cycle CPU Summary

- Fairly straightforward
- Which instruction takes the longest? By how much? Why is that a problem?
- Execution time = insts * cpi * cycle time

→ Most machines are not single-cycle.
Evaluating Critical Path

|        | I mem | Decode, R-Read | ALU
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>1</td>
<td>1</td>
<td>.9</td>
</tr>
<tr>
<td>Load</td>
<td>1</td>
<td>1</td>
<td>.9</td>
</tr>
<tr>
<td>Store</td>
<td>1</td>
<td>1</td>
<td>.9</td>
</tr>
<tr>
<td>beq</td>
<td>1</td>
<td>1</td>
<td>.9</td>
</tr>
</tbody>
</table>

- **ALUOp**: ALU operation
- **ALUSrc**: ALU source
- **RegDst**: Register destination
- **PC**: Program counter
- **Instruction**
  - [31–0]: Instruction memory
  - [25–21]: Instruction [20–16]
  - [20–16]: Instruction [15–11]
  - [15–11]: Instruction [15–0]
  - [15–0]: Instruction memory

**Insight:** Different instructions have different critical paths!
Multicycle design

- Problem: In single-cycle design, cycle time must be long enough for longest instruction
- Solution: break execution into smaller tasks
  - each task takes a cycle;
  - different instructions require different numbers of cycles

- Another advantage: we can multiplex area-intensive datapath components (memories, ALUs, etc) and use them multiple times for a given instruction (as long as each use is on a different cycle.)

This used to be one of the key desirable features of multicycle - not so much anymore..
Idea: quantize instruction execution into smaller steps

- Five execution steps (some instructions use fewer)
  - **IF**: Instruction Fetch
  - **ID**: Instruction Decode (& register fetch & add PC+immed)
  - **EX**: Execute
  - **Mem**: Memory access
  - **WB**: Write-Back into registers

<table>
<thead>
<tr>
<th></th>
<th>I cache</th>
<th>Decode, R-Read</th>
<th>ALU</th>
<th>PC update</th>
<th>D cache</th>
<th>R-Write</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>1</td>
<td>1</td>
<td>.9</td>
<td>-</td>
<td>-</td>
<td>.8</td>
<td>3.7</td>
</tr>
<tr>
<td>Load</td>
<td>1</td>
<td>1</td>
<td>.9</td>
<td>-</td>
<td>1</td>
<td>.8</td>
<td>4.7</td>
</tr>
<tr>
<td>Store</td>
<td>1</td>
<td>1</td>
<td>.9</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>3.9</td>
</tr>
<tr>
<td>beq</td>
<td>1</td>
<td>1</td>
<td>.9</td>
<td>.1</td>
<td>-</td>
<td>-</td>
<td>3.0</td>
</tr>
</tbody>
</table>
Will multicycle be faster?
→ Depends on program + stage timings

<table>
<thead>
<tr>
<th>Instruction frequency</th>
<th>Cycles needed</th>
<th>Instruction frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>4</td>
<td>60%</td>
</tr>
<tr>
<td>Load</td>
<td>5</td>
<td>20%</td>
</tr>
<tr>
<td>Store</td>
<td>4</td>
<td>10%</td>
</tr>
<tr>
<td>beq</td>
<td>3</td>
<td>10%</td>
</tr>
</tbody>
</table>

What is CPI assuming this instruction mix???

Single cycle design:
Clock cycle time = 4.7ns
\[
time/inst = 1 \text{ cycle/inst} \times 4.7 \text{ ns/cycle} = 4.7 \text{ ns/inst}
\]

Multicycle design:
Clock cycle time = 1 ns
\[
time/inst = CPI \times 1 \text{ ns/cycle}
\]
Adding State Elements

Since execution takes place over multiple cycles, and we reuse some of the hardware, we need to capture intermediate results.

Need extra registers when:

- signal is computed in one clock cycle and used in another, AND
- the inputs to the combinational circuit can change before the signal is written into a state element.
  • We only require them to be “held” for 1 cycle
Where to add registers (more or less)

IF

- Read address
- Instruction
- Instruction memory

ID

- PC
- Registers
  - Read register 1
  - Read register 2
  - Read data 1
  - Read data 2
  - Write register
  - Write data
- ALU
  - ALUSrc
  - ALU operation
- ALU result
- Shift left 2

Ex

- Add
  - Add ALU result
  - Multiplex
  - MemRead
  - MemWrite
- Zero

Mem

- MemtoReg
- Data memory
  - Address
  - Read data
  - MemRead
  - MemWrite

WB

- RegWrite
- Sign extend
- Write data
- Mux

Add

- 4

4-bit adder
Multicycle Datapath - let's figure out basic execution model
## Summary of execution steps

<table>
<thead>
<tr>
<th>Step</th>
<th>R-type</th>
<th>Memory</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Fetch</td>
<td>IR = Mem[PC]</td>
<td>PC = PC + 4</td>
<td></td>
</tr>
<tr>
<td>Instruction Decode/</td>
<td>A = Reg[IR[25-21]]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>register fetch</td>
<td>B = Reg[IR[20-16]]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALUout = PC + (sign-extend(IR[15-0])) &lt;&lt; 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution, address</td>
<td>ALUout = A op B</td>
<td>ALUout = A + sign-extend(IR[15-0])</td>
<td>if (A==B) then PC=ALUout</td>
</tr>
<tr>
<td>computation, branch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>completion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory access or R-type</td>
<td>Reg[IR[15-11]] = ALUout</td>
<td>memory-data = Mem[ALUout] or</td>
<td></td>
</tr>
<tr>
<td>completion</td>
<td></td>
<td>Mem[ALUout] = B</td>
<td></td>
</tr>
<tr>
<td>Write-back</td>
<td></td>
<td>Reg[IR[20-16]] = memory-data</td>
<td></td>
</tr>
</tbody>
</table>

This is **Register Transfer Language (RTL)**

“High level” description of changes to state elements

We’ll go through these in exacting detail

And translate them to “low level” control signal settings

Modern design tools do this automatically
Multicycle Datapath - let's figure out control logic