Designing the Processor

Datapath & Control (review of Sec. 4.1 – 4.4)

• Simplified to contain only:
  – memory-reference instructions: lw, sw
  – arithmetic-logical instructions: add, sub, and, or, slt
  – control flow instructions: beq

• Generic Implementation:
  – The program counter (PC) supplies instruction address
  – Get the instruction from the “instruction memory”
  – The op-code determines exactly what to do.

MIPS instruction format and architecture
**Fetching instruction and incrementing PC**

1) Fetch and increment PC

2) Read registers

The control unit should decode the op-code to determine what to do next
- depends on instruction type
### Executing R-type instructions (in one cycle)

1) Fetch and increment PC
2) Read registers
3) Execute operation
4) Store result

The control unit issues:
- the corresponding ALU control
- RegWrite signal

### Executing sw instructions

1) Fetch and increment PC
2) Read registers
3) Calculate memory address (issue ALU control signals – add)
4) Write data to memory (issue MemWrite signal)
Data path conflicts

1) The second input to the ALU comes from two different places
2) The data2 output goes to two different places
3) The output of the ALU goes to two different places

Reconciling data paths

1) No problems in fanning out one signal to two points

2) Cannot connect two signals to one point (should use multiplexers)
Resolving data path conflicts

```
9
```

Executing `lw` instructions (in one cycle)

```
10
```
Reconciling R-type, sw and lw data paths

Executing beq instructions

1) Fetch and increment PC
2) Read registers
3) Calculate branch condition (issue ALU control signals – subtract)
4) Calculate target address
5) Select next PC depending on the result in 3
**Control**

- Using the op-code from the instruction, the control issues signals to:
  - select the operations to perform (ALU, read/write to registers and memory)
  - control the flow of data (multiplexor inputs).

![Diagram showing control inputs and outputs](image)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>RegDst</th>
<th>ALUSrc</th>
<th>MemtoReg</th>
<th>RegWrite</th>
<th>MemRead</th>
<th>MemWrite</th>
<th>Branch</th>
<th>ALUOp1</th>
<th>ALUOp0</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>sw</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>beq</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

13
The one cycle CPU implementation

- Use combinational logic to generate control signals
- Should wait for everything to settle down, and the right thing to be done
- Cycle time determined by length of the longest path