Design of the MIPS Processor (contd)

First, revisit the datapath for `add, sub, lw, sw`. We will augment it to accommodate the `beq` and `j` instructions.

**Execution of branch instructions**

\[
\begin{align*}
\text{beq } & \text{ $at, $zero, L} \\
\text{add } & \text{ $v1, $v0, $zero} \\
\text{add } & \text{ $v1, $v1, $v1} \\
\text{j } & \text{ somewhere} \\
\text{L: } & \text{ add $v1, $v0, $v0} \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>000100</td>
<td>00001</td>
<td>00000</td>
<td>0000 0000 0000 0011</td>
</tr>
</tbody>
</table>

Offset = 3x4 = 12

The offset must be added to the next PC to generate the target address for branch.
The modified version of MIPS

We need a second adder, since the ALU is already doing subtraction for the beq.

- PCSrc=1 branches to PC+4+(offset×4)
- PCSrc=0 continues to PC+4.

The final datapath for single cycle MIPS. Find out which paths the signal follow for lw, sw, add and beq instructions.
Executing R-type instructions

The ALUop will be determined by the value of the opcode field and the function field of the instruction word
Executing LW instruction
Executing beq instruction

The branch may or may not be taken, depending on the ALU’s Zero output.
Control signal table

This table summarizes what control signals are needed to execute an instruction. The set of control signals vary from one instruction to another.

<table>
<thead>
<tr>
<th>Operation</th>
<th>RegDst</th>
<th>RegWrite</th>
<th>ALUSrc</th>
<th>ALUOp</th>
<th>MemWrite</th>
<th>MemRead</th>
<th>MemToReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>010</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sub</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>110</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>and</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>000</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>or</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>001</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>slt</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>111</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lw</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>010</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>sw</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>010</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>beq</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>110</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

How to implement the control unit?
The Control Unit

All control signals are not shown here
1-cycle implementation is not used

Why? Because the length of the clock cycle will always be determined by the slowest operation (lw, sw) even when the data memory is not used. Practical implementations use multiple cycles per instruction, which fixes some shortcomings of the 1-cycle implementation.

Faster instructions are not held back by the slower instructions

The clock cycle time can be decreased.

Eventually simplifies the implementation of pipelining, the universal speed-up technique

This requires some changes in the datapath