Designing the Control Units for the Single Cycle Data Path

CS 365

Step 4: Given Datapath: RTL -> Control

![Diagram showing the control units for a single cycle data path.]
Control

- Selecting the operations to perform (ALU, read/write, etc.)
  
  *Design the ALU Control Unit*

- Controlling the flow of data (multiplexor inputs)
  
  *Design the Main Control Unit*

- Information comes from the 32 bits of the instruction

- Example:

  add $8, $17, $18

  Instruction Format:

  
  \[
  \begin{array}{cccccccc}
  000000 & 10001 & 10010 & 01000 & 00000 & 100000 \\
  op & rs & rt & rd & shamt & funct \\
  \end{array}
  \]

  - ALU's operation based on instruction type and function code

ALU Control

- e.g., what should the ALU do with this instruction

- Example: lw $1, 100($2)

<table>
<thead>
<tr>
<th>35</th>
<th>2</th>
<th>1</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td>16 bit offset</td>
</tr>
</tbody>
</table>

- ALU control input

<table>
<thead>
<tr>
<th>0000</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>OR</td>
</tr>
<tr>
<td>0010</td>
<td>add</td>
</tr>
<tr>
<td>0110</td>
<td>subtract</td>
</tr>
<tr>
<td>0111</td>
<td>set-on-less-than</td>
</tr>
</tbody>
</table>

Recall design of ALU from Chapter 3.
ALU Control Design

<table>
<thead>
<tr>
<th>Instruction opcode</th>
<th>ALUOp</th>
<th>Instruction operation</th>
<th>Funct field</th>
<th>Desired ALU action</th>
<th>ALU control input</th>
</tr>
</thead>
<tbody>
<tr>
<td>LW</td>
<td>00</td>
<td>Load word</td>
<td>xxxxxxx</td>
<td>Add</td>
<td>0010</td>
</tr>
<tr>
<td>SW</td>
<td>00</td>
<td>Store word</td>
<td>xxxxxxx</td>
<td>Add</td>
<td>0010</td>
</tr>
<tr>
<td>BEQ</td>
<td>01</td>
<td>Branch eq</td>
<td>xxxxxxx</td>
<td>Subtract</td>
<td>0110</td>
</tr>
<tr>
<td>R-type</td>
<td>10</td>
<td>Add</td>
<td>100000</td>
<td>Add</td>
<td>0010</td>
</tr>
<tr>
<td>R-type</td>
<td>10</td>
<td>Subtract</td>
<td>100010</td>
<td>Subtract</td>
<td>0110</td>
</tr>
<tr>
<td>R-type</td>
<td>10</td>
<td>AND</td>
<td>100100</td>
<td>And</td>
<td>0000</td>
</tr>
<tr>
<td>R-type</td>
<td>10</td>
<td>OR</td>
<td>100101</td>
<td>Or</td>
<td>0001</td>
</tr>
<tr>
<td>R-type</td>
<td>10</td>
<td>Set on less than</td>
<td>101010</td>
<td>Set on less than</td>
<td>0111</td>
</tr>
</tbody>
</table>

Control

- Must describe hardware to compute 3-bit ALU control input
  - given instruction type
    - 00 = lw, sw
    - 01 = beq
    - 10 = arithmetic
    - function code for arithmetic
  - ALUOp computed from instruction type
- Describe it using a truth table (can turn into gates):

<table>
<thead>
<tr>
<th>ALUOp</th>
<th>Funct field</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALUOp1</td>
<td>ALUOp0</td>
<td>F5 F4 F3 F2 F1 F0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 0 X X X X X   0010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X 1 X X X X X   0110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 X X X 0 0 0   0010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 X X X 0 1 0   0110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 X X X 0 1 0   0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 X X X 0 1 0 1 0001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 X X X 1 0 1 0 0111</td>
</tr>
</tbody>
</table>
Design the main control unit

- Seven control signals
  - RegDst
  - RegWrite
  - ALUSrc
  - PCSrc
  - MemRead
  - MemWrite
  - MemtoReg

Control Signals

1. **RegDst = 0** => Register destination number for the Write register comes from the rt field (bits 20-16)
   
   **RegDst = 1** => Register destination number for the Write register comes from the rd field (bits 15-11)

2. **RegWrite = 1** => The register on the Write register input is written with the data on the Write data input (at the next clock edge)

3. **ALUSrc = 0** => The second ALU operand comes from Read data 2
   
   **ALUSrc = 1** => The second ALU operand comes from the sign-extension unit

4. **PCSrc = 0** => The PC is replaced with PC+4
   
   **PCSrc = 1** => The PC is replaced with the branch target address

5. **MemtoReg = 0** => The value fed to the register write data input comes from the ALU
   
   **MemtoReg = 1** => The value fed to the register write data input comes from the data memory

6. **MemRead = 1** => Read data memory

7. **MemWrite = 1** => Write data memory
R-format instructions

- RegDst = 1
- RegWrite = 1
- ALUSrc = 0
- Branch = 0
- MemtoReg = 0
- MemRead = 0
- MemWrite = 0
- ALUOp = 10

Memory access instructions

Load word

- RegDst = 0
- RegWrite = 1
- ALUSrc = 1
- Branch = 0
- MemtoReg = 1
- MemRead = 1
- MemWrite = 0
- ALUOp = 00

Store Word

- RegDst = X
- RegWrite = 0
- ALUSrc = 1
- Branch = 0
- MemtoReg = X
- MemRead = 0
- MemWrite = 1
- ALUOp = 00
### Branch Equal

- $\text{RegDst} = X$
- $\text{RegWrite} = 0$
- $\text{ALUSrc} = 0$
- $\text{Branch} = 1$
- $\text{MemtoReg} = X$
- $\text{MemRead} = 0$
- $\text{MemWrite} = 0$
- $\text{ALUOp} = 01$

### Control

<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>ALUOp</th>
<th>ALUOp0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-format</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lw</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</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>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>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Step 5: Implementing Control

- Simple combinational logic (truth tables)

Our Simple Control Structure

- All of the logic is combinational
- We wait for everything to settle down, and the right thing to be done
  - ALU might not produce “right answer” right away
  - we use write signals along with clock to determine when to write
- Cycle time determined by length of the longest path
An Abstract View of the Critical Path

- Register file and ideal memory:
  - The CLK input is a factor ONLY during write operation
  - During read operation, behave as combinational logic:
    - Address valid => Output valid after “access time.”

Critical Path (Load Operation) =
Instruction Memory’s Access Time +
Register File’s Access Time +
ALU to Perform a 32-bit Add +
Data Memory Access Time +
Setup Time for Register File Write

Single Cycle Implementation

- Calculate cycle time assuming negligible delays except:
  - memory (2ns), ALU and adders (2ns), register file access (1ns)
Summary

- 5 steps to design a processor
  - 1. Analyze instruction set => datapath requirements
  - 2. Select set of datapath components & establish clock methodology
  - 3. Assemble datapath meeting the requirements
  - 4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
  - 5. Assemble the control logic

- MIPS makes it easier
  - Instructions same size
  - Source registers always in same place
  - Immediates same size, location
  - Operations always on registers/immediates

- Single cycle datapath => CPI=1, Clock Cycle Time => long