How to design a controller to produce signals to control the datapath

Recap: The MIPS Instruction Formats
- All MIPS instructions are 32 bits long. The three instruction formats:
  - R-type
    
    | op | rs | rt | rd | shamt | funct |
    |----|----|----|----|-------|-------|
    | 6bits | 5bits | 5bits | 5bits | 5bits | 6bits |
  - I-type
    
    | op | rt | immediate |
    |----|----|------------|
    | 6bits | 5bits | 6bits |
  - J-type
    
    | op | target address |
    |----|----------------|
    | 6bits | 26bits |

- The different fields are:
  - op: operation of the instruction
  - rs, rt, rd: the source and destination registers specifier
  - shamt: shift amount
  - funct: selects the variant of the operation in the “op” field
  - immediate: address / immediate value
  - target address: target address of the jump instruction

Recap: The MIPS Subset
- ADD and subtract
  - add rd, rs, rt
  - sub rd, rs, rt
- OR imm:
  - ori rt, rs, imm16
- LOAD and STORE
  - lw rt, rs, imm16
  - sw rt, rs, imm16
- BRANCH:
  - beq rs, rt, imm16
- JUMP:
  - j target

Recap: A Single Cycle Datapath
- We have everything except control signals (underline)
- Today’s lecture will show you how to generate the control signals

The Big Picture: Where are We Now?
- The Five Classic Components of a Computer
- Today’s Topic: Designing the Control for the Single Cycle Datapath

RTL: The ADD Instruction
- add rd, rs, rt
  - mem(PC) Fetch the instruction from memory
  - PC ← PC + 4 Calculate the next instruction’s address
- A note: 2nd step and 3rd step can be done in parallel.
**Instruction Fetch Unit at the Beginning of Add / Subtract**

- Fetch the instruction from Instruction memory: Instruction ← mem[PC]
- This is the same for all instructions

**Instruction Fetch Unit at the End of Add and Subtract**

- PC ← PC + 4
- This is the same for all instructions except: Branch and Jump

**The Single Cycle Datapath during Add and Subtract**

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c|c}
\hline
& & & & & & & & & & \\
& & & & & & & & & & \\
\hline
&&&&&&&&&& \\
\hline
up & rs & rt & shamt & funct & imm16 & 32 & 26 & 21 & 16 & immediate \\
\hline
\end{array}
\]

- \( R[rd] \) ← \( R[rs] + / - R[rt] \)
- \( RegDst = x \)

**The Single Cycle Datapath during Or Immediate**

- \( R[rt] \) ← \( R[rs] \) or \( \text{ZeroExt}[imm16] \)
- \( RegDst = x \)

**The Single Cycle Datapath during Load**

- \( R[rt] \) ← Data Memory (\( R[rs] + \text{SignExt}[imm16] \))
- \( RegDst = 0 \)

**The Single Cycle Datapath during Store**

- Data Memory (\( R[rs] + \text{SignExt}[imm16] \)) ← \( R[rt] \)
- \( MemtoReg = 1 \)
The Single Cycle Datapath during Branch

- If (R[rs] - R[rt] == 0) then Zero <- 1; else Zero <- 0
- RegDst = x
- ALUop = Add
- ALUSrc = x
- Extender
- Addr<31:2> = 0
- Addr<1:0> = 0
- Jump = 0
- Clk
- Instruction Fetch Unit

Instruction Fetch Unit at the End of Branch

- If (Zero == 1) then PC = PC + 4 + SignExt[imm16]*4; else PC = PC + 4
- Add<31:2> = 0
- Add<1:0> = 0
- Jump = 0
- Instructions<31:0>

The Single Cycle Datapath during Jump

- Nothing to do! Make sure control signals are set correctly!
- Branch = 0
- RegDst = x
- ALUop = Add
- ALUSrc = x
- Extender
- Addr<31:2> = 0
- Addr<1:0> = 0
- Jump = 0
- Clk
- Instruction Fetch Unit

Instruction Fetch Unit at the End of Jump

- PC <- PC<31:29> concat target<25:0> concat "00"
- Add<31:2> = 0
- Add<1:0> = 0
- Jump = 0
- Instructions<31:0>

A Summary of the Control Signals

- op | rs | rt | imm | funct | add, sub | or | shamt | jump
- RegWrite | 1 | 0 | 0 | 0 | x | x | x | x
- MemRead | 0 | 0 | 1 | 1 | 0 | x | x | x
- MemWrite | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0
- Branch | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0
- Jump | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1
- ExtEnd | x | x | x | 0 | 0 | x | x | x
- ALU<0-2> | Add | Sub | Add | Add | Sub | xxx
- R-type | 31 | 26 | 21 | 16 | 11 | 6 | 0 | 0
- I-type | op | rs | rt | imm | add, sub | or | shamt | funct | jump
- J-type | op | rs | rt | immediate | add, sub | or | shamt | funct | jump

The Concept of Local Decoding
In this exercise, ALUop has to be 2 bits wide to represent:
- (1) "R-type" instructions
- "I-type" instructions that require the ALU to perform:
  - (2) Or, (3) Add, and (4) Subtract

To implement the full MIPS ISA, ALUop hat to be 3 bits to represent:
- (1) "R-type" instructions
- "I-type" instructions that require the ALU to perform:
  - (2) Or, (3) Add, (4) Subtract, and (5) And (Example: andi)

The Truth Table for ALUctr

The Logic Equation for ALUctr<0>

The Logic Equation for ALUctr<1>

Recall ALU Homework (also P. 286 text):

The Logic Equation for ALUctr<2>

ALU <3:0> Instruction Operation
0000 add
10000 add
01000 subtract
10100 and
00100 or
11000 and
01100 or
11100 set-on-less-than

ALUctrl<0>  =  !ALUop<2>  &  ALUop<0>  +  
ALUop<2>  &  !func<2>  &  func<1>  &  !func<0>

This makes func<3> a don't care

ALUctrl<0> = ALUop<2> & ALUop<0> + 
ALUop<2> & func<2> & func<1> & func<0>

ALUctrl<2> = ALUop<2> & ALUop<1> + 
ALUop<2> & func<2> & func<1> & func<0> + 
ALUop<2> & func<3> & func<2> & func<1> & func<0> + 
ALUop<2> & func<3> & func<2> & func<1> & func<0>
The ALU Control Block

The effect of load in a real MIPS Processor is delayed:

- lw $1, 100 ($2) // Load Register R1
- add $3, $1, $0 // Move “old” R1 into R3
- add $4, $1, $0 // Move “new” R1 into R4

The effect of load in our single cycle processor is NOT delayed:

- lw $1, 100 ($2) // Load Register R1
- add $3, $1, $0 // Move “new” R1 into R3

* The effect of branch and jump in a real MIPS Processor is NOT delayed:
- Instruction Address: 0x00 $1000
- Instruction Address: 0x1000 sub $1, $2, $3
- Branch and jump in our single cycle processor is NOT delayed
- Instruction Address: 0x00 $1000
- Instruction Address: 0x1000 sub $1, $2, $3
Worst Case Timing

<table>
<thead>
<tr>
<th>Stage</th>
<th>Old Value</th>
<th>New Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>Rs, Rt, Rd, Op, Func</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>ALUct</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>Instruction Memory Access Time</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>Old Value</td>
<td>New Value</td>
<td></td>
</tr>
<tr>
<td>Delay through Control Logic</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>Old Value</td>
<td>New Value</td>
<td></td>
</tr>
<tr>
<td>MEMct</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>Old Value</td>
<td>New Value</td>
<td></td>
</tr>
<tr>
<td>Register File Access Time</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>Old Value</td>
<td>New Value</td>
<td></td>
</tr>
<tr>
<td>ExtOp</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>Old Value</td>
<td>New Value</td>
<td></td>
</tr>
<tr>
<td>ALUSrc</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>Old Value</td>
<td>New Value</td>
<td></td>
</tr>
<tr>
<td>MemtoReg</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>Old Value</td>
<td>New Value</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>Old Value</td>
<td>New Value</td>
<td></td>
</tr>
<tr>
<td>Data Memory Access Time</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>Old Value</td>
<td>New Value</td>
<td></td>
</tr>
<tr>
<td>AluA</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>Old Value</td>
<td>New Value</td>
<td></td>
</tr>
<tr>
<td>Data Memory Access Time</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>Old Value</td>
<td>New Value</td>
<td></td>
</tr>
<tr>
<td>AluB</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>Old Value</td>
<td>New Value</td>
<td></td>
</tr>
<tr>
<td>Data Memory Access Time</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>Old Value</td>
<td>New Value</td>
<td></td>
</tr>
<tr>
<td>MemWrite</td>
<td>Old Value</td>
<td>New Value</td>
</tr>
<tr>
<td>Old Value</td>
<td>New Value</td>
<td></td>
</tr>
</tbody>
</table>

Drawback of this Single Cycle Processor

- Long cycle time:
  - Cycle time must be long enough for the load instruction:
    - PC’s Clock -to-Q
    - Instruction Memory Access Time
    - Register File Access Time
    - ALU Delay (address calculation)
    - Data Memory Access Time
    - Register File Setup Time
    - Clock Skew

- Cycle time is much longer than needed for all other instructions

Where to get more information?

- Chapter 5.1 to 5.3 of your textbook:

- For a reference on the MIPS architecture:
  - Gerry Kane, “MIPS RISC Architecture,” Prentice Hall.