Lab 3 - The Datapath and ALU Control Units

Prelab

You will need to submit your testbench on ilearn (check online for due dates) prior to coming to lab. Your testbench should demonstrate that you have read through the lab specifications and understand the goal of this lab. You will need to consider the boundary cases. You do not need to begin designing yet, but this testbench will be helpful during the lab while you are designing.

Introduction

For this lab you will be building the control units for a MIPS processor. See the next figure.

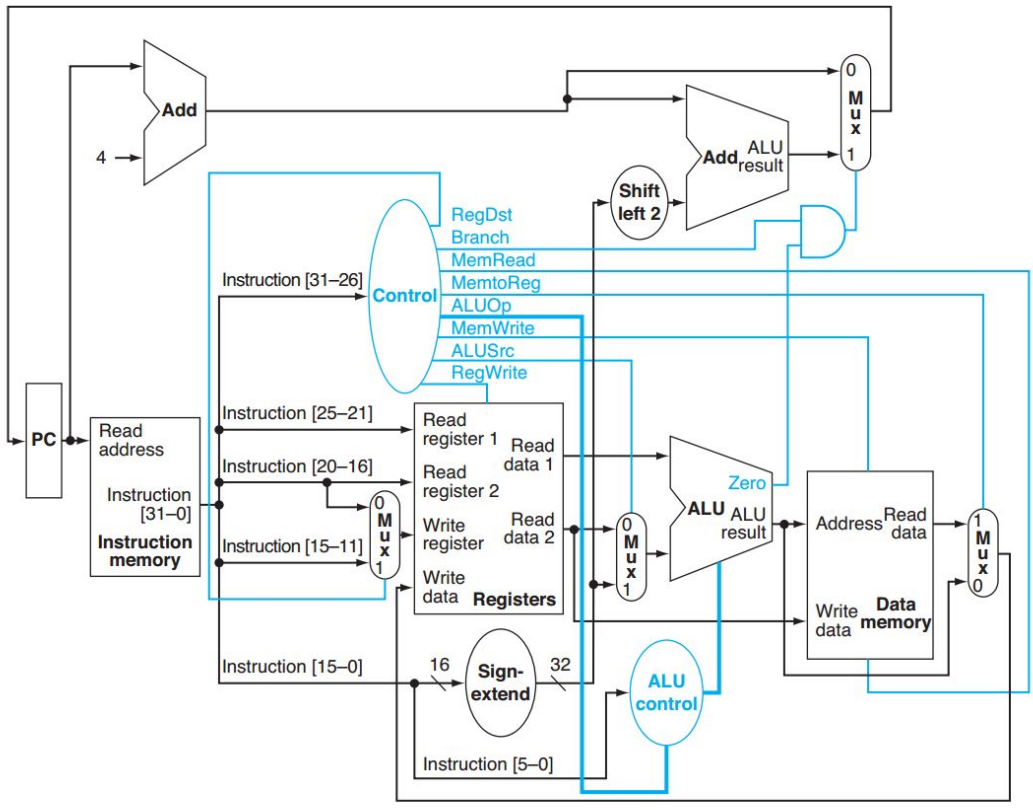

FIGURE 4.17 The simple datapath with the control unit. The input to the control unit is the 6-bit opcode field from the instruction.

There are two control units. The *main control unit* manages the datapath. It receives an opcode input from the currently executing instructions and based on this opcode it configures the datapath accordingly. A truth table for the unit functionality (shown below) can be found in the slides of CS161. The The table (read vertically) shows the output for `R-format, lw, sw` and `beq` instructions, additionally, you will need to implement the immediate type (`addi,subi`) instructions. To do this, you will trace through the datapath (shown above) to determine which control lines will need to be set.

Control	Signal name	R-format	lw	sw	beq	imm
Inputs	Op5	0	1	1	0	0
	Op4	0	0	0	0	0
	Op3	0	0	1	0	1
	Op2	0	0	0	1	0
	Op1	0	1	1	0	0
	Op0	0	1	1	0	0
Outputs	RegDst	1	0	X	X	
	ALUSrc	0	1	1	0	
	MemtoReg	0	1	X	X	
	RegWrite	1	1	0	0	
	MemRead	0	1	0	0	
	MemWrite	0	0	1	0	
	Branch	0	0	0	1	
	ALUOp1	1	0	0	0	
	ALUOp0	0	0	0	1	

Table 1. The control function for the simple one-clock implementation. Read each column vertically. I.e. if the `Op[5:0]` is `000000`, the `RegDst` would be '`1`', `ALUSrc` would be '`0`', etc. You will need to fill in the `imm` column. Based on Figure D.2.4 from the book.

The second control unit manages the *ALU*. It receives an ALU opcode from the datapath controller, and the 'Funct Field' from the current instruction. With these, the ALU controller decides what operation the ALU is to perform. The following figures from the CS161 slides give an idea of the inputs and outputs of the ALU controller.

Input				Output	
Instruction Opcode	**ALUOp**	**Instruction operation**	**Funct field**	**Desired ALU action**	**ALU select input**
lw	00	Load word	XXXXXX	add	0010
sw	00	Store word	XXXXXX	add	0010
beq	01	Branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	and	0000
R-type	10	OR	100101	or	0001
R-type	10	NOR	100111	nor	1100
R-type	10	Set on less than	101010	Set on less than	0111

ALU select bits based on ALUop, and Funct field

ALUOp		Funct field						Operation
ALUOp1	**ALUOp0**	**F5**	**F4**	**F3**	**F2**	**F1**	**F0**	
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111
1	X	X	X	0	1	1	1	1100

Truth Table for ALU Control

Deliverables

For this lab you are expected to build and test both the datapath and ALU control units. The target processor architecture will only support a subset of the MIPS instructions. These instructions are listed below. You only have to offer control for these instructions. Signal values can be found within your textbook (and in the images above).

- `add, addu, addi`
- `sub, subi`
- `slt`
- `not*, nor`
- `or`
- `and`
- `lw, sw`
- `beq`

Notice that for the `addu` it is sufficient to generate the same control signals as the `add` operation.
* `not` is not an instruction, it is a *pseudo-op*, which means it can be implemented using other operations. Think about how you would implement it using the other operations.
To complete this lab you are provided with the following skeleton verilog files.
control_unit.v

```
module control_unit  (
    input wire [5:0]  instr_op ,
    output wire reg_dst      ,
    output wire  branch    ,
    output wire  mem_read ,
    output wire  mem_to_reg  ,
    output wire [1:0]  alu_op  ,
    output wire  mem_write  ,
    output wire  alu_src     ,
    output wire  reg_write  ,
    );
```

alu_control.v

```
module alu_control  (
    input wire [1:0] alu_op ,
    input wire [5:0] instruction_5_0 ,
    output wire [3:0] alu_out
    );
```

Architecture Case Study

For lab this week you are also expected to perform a simple case study. It is meant to show how important understanding a computer's architecture, and compiler is when developing efficient code. For this study you are to compare and analyze the execution time of the two programs given here. You should run a number of experiments varying the input size from 100 to 30,000. Based on the results you are to write a report of your findings. The report should contain a graph of your data, and useful analysis of it. You should draw conclusions based on your findings. Reports that simply restate what is in the graph will not get credit. To make it clear, make sure you used the concepts you have learned so far in 161 and 161L when explaining the differences in performance. If a confusing or fuzzy explanation is given you will get low or no marks. The report should be in PDF format.

Turn-In:

Each student should turn in one tar file to iLearn. Your Verilog code can, and should, be identical to the other members of your group, your PDF report **must** be done individually. The contents of which should be:

- A README file with the group members names, and any incomplete or incorrect functionality as described on iLearn.
- A completed version of the `control_unit.v`, and `alu_control.v` files
- A Verilog file(s) for the Testbenches
- A one page or two pages PDF report for your case study.
  Grading Verilog portion 75% Report 25%