EECE 2323 Digital Logic Design Lab Report

Lab 6 Adding Instruction Decoding to the Datapath

Student Name:
Ousmane Toure & Kaitlyn O'Flaherty

Section #: 1

Instructor Name: Dr. Aboelela

Date:08/09/2022

Lab TA Name: Keshav Bharadwaj Vaidyanathan, Srinidi Somasundaram Subbulakshmi, Yuchao Su

1. Background & Purpose:

In this experiment, we implemented a decoder to control the datapath we created in the previous Labs. The objective was to translate the inputted bits into different data signals and instruction words based on the operation that was selected. From our previous labs, our ALU performs all necessary operations to generate new values and needs a register file to remember and store these values in memory. The results were then displayed utilizing the VIO Dashboard on Vivado and the LEDs on the add on board of the PYNQ. Decoders are essential combinational logic elements because they give central processing units the ability to read instructions from the computer ensuring the correct operation is done, saving time and space in your circuit. The goal of this lab was to create an instruction set for our decoder then utilize verilog code in vivado to create, simulate, and physically implement the new datapath. Furthermore enhancing confidence and proficiency in vivado and computational architecture.

This lab is important to the scientific community because each processor needs to be able to receive instructions, process the instruction correctly and then output the desired resul. This allows the processor to quickly access information, and without memory a computer woud not be able to function properly.

2. Pre-Lab Response:

Opcode	RegDst	RegWrite	AluSrc1	Alusrc2	AluOp[2:0]	Memwrite	MemToReg
0000 lw	0	1	0	1	000	0	1
0001 sw	0	0	0	1	000	1	x
0010 add	1	1	0	0	000	0	0
0011 addi	0	1	0	1	000	0	0
0100 inv	1	1	0	0	001	0	0
0101 and	1	1	0	0	010	0	0
0110 andi	0	1	0	1	010	0	0
0111 or	1	1	0	0	011	0	0
1000 ori	0	1	1	0	011	0	0

1001 sra	0	1	0	0	100	0	0
1010 sll	0	1	0	0	101	0	0
1011 beq	0	1	0	0	110	0	x
1100 bne	0	1	0	0	111	0	x
1101 clr	x	1	1	0	010	0	0

Type	**Instruction**	**Machine Code Binary**	**Machine Code Hex**
R	**inv $1, $1**	**010000010100000**	**0x4140**
I	**sll $1, $1, 0x03**	**1010 01 01 0000011**	**0x5283**
I	**sw $1, 0xFF($3)**	**0001101111111111**	**0x1BFF**
I	**lw $2, 0xFF($3)**	**0000111011111111**	**0x0EFF**
I	**ori $2, $2, 0xF0**	**1000101011110000**	**0x8AF0**

3. Summary of Design Implementation

3.1.Results and Analysis:

When conducting this experiment, the single-cycle datapath from Lab 5 was utilized and updated to include a decoder element.. The simulation was run by creating a test bench from the prelab tables seen above. The Test Bench Code resulted in the correct operation being selected and being displayed onto the addon board of the PYNQ. which were then used to complete the following operations : loading to a register, storing to a register, arithmetic add of two register elements, adding an constant value to a registers value, inverting a reg value, and then completing various other ALU operations such as ADDI (Appendix B), OR, ORI, BEQ, BNE, SRA, SLL, and CLR. All files can be found in Appendix A. The result of the simulation being run is the Test Bench WaveForm Simulation in Appendix B, which clearly shows the instruction being inputted and the resultant operation being completed.. The results of the test bench verified that our code in fact does work and gives us the green light to program straight into the PYNQ-Z2 board. After programming our board and connecting the add on board, we opened the virtual input output (VIO) dashboard which allowed us to test our values as our PYNQ board did not have enough physical inputs. Storing values in the register and all of the necessary ALU outputs were tested using the VIO dashboard and the resulting screenshots were placed in Appendix B. All tests resulted in values that were consistent with our pre lab test bench truth table highlighting that our circuit was in fact correct. You could face many errors when conducting this lab. For example, when creating your test bench, if you incorrectly have the wrong hex or binary value for your machine code, you could cause various inconsistencies in your data.

3.2.Conclusion & Recommendations:

Based on our results, we can conclude that Lab 6 consists of creating and implementing a decoder into a single-cycle datapath. Testing its implementation virtually with a test bench confirmed that our verilog code was correct which gave us the green light to program the PYNQ Board using Virtual Input Output (VIO) ports. . The lab resulted in successfully being

able to give instructions to our central processing unit, allowing better functionality to complete different arithmetic and logical functions like addition and bit shifting, BNE, as well as checking if values were equal. These tools are crucial for a Central Processing Unit, and gives it the ability to perform varying tasks of different degrees. Completing this lab showed highlighted the importance of decoders in not only single-cycle datapaths but for all cpus. Furthermore, the clock button was very resourceful as it made incrememnting step by step much easier when testing the functionality of our datapth.

Recommendations going forward would be to give better clarification on the prelab instruction set, many of us struggled to figure out what was actually needed of us from the start which caused a delay in completing the lab as we had to wait for office hours or for a TA to assist.

Appendix A: Design Program Files (Verilog modules, testbenches, etc)

TEST BENCH TABLE:

Type	**Machine Code Binary**	**executing**
0000 lw	**0000_11_10_00_110011**	**load reg 1**
0001 sw	**0001_01_10_00_010001**	**store reg 1**
0010 add	**0010_00_01_10_00000**	**reg 0 + reg 1 into reg 2**
0011 addi	**0011__00_11_00_001010**	**reg 0 + 10 into reg 3**
0100 inv	**0100_00_01_10_000000**	**not reg 1 = reg 2**
0101 and	**0101_00_01_10_000000**	**reg 0 & reg 1**
0110 andi	**0110_01_11_00_001111**	**reg 1 & 0xFFFFFFF**
0111 or	**0111_00_01_10_000000**	**reg0 or reg 1**
1000 ori	**1000_01_11_00_010100**	**reg 1 or 0x14**
1001 sra	**1001_01_11_00_000011**	**reg1 << 3**
1010 sll	**1010_01_11_00_000001**	**reg 1 >>> 1**

1011 beq	**1011_00_10_01_000000**	**reg 0 == reg 2?**
1100 bne	**1100_00_10_01_000000**	**reg 0 != reg 2?**
1101 clr	**1101_00_11_00_000000**	**reg**

Topfile:

```
`timescale 1ns / 1ps

module pdatapath_top(
        input wire clk,// General clock input
        input wire top_pb_clk,// PBN1 clock input
      input wire rst_general,// PBN0 clock reset for memory blocks
        output [7:0] led,// add-on board led[5:0], + LD0, LD1
        output wire ovf_ctrl,    // LD3 for overflow
        output [3:0] disp_en,// 7-Segment display enable
        output [6:0] seg7_output// 7-segment display output
   );

   wire [7:0] alu_input1, alu_input2;
   wire [7:0] alu_output;
   wire [2:0] ALUOp;
   wire       alu_ovf;
   wire       take_branch;

   wire [15:0] instruction;
   //insturction fields
   wire [3:0] opcode;
   wire [1:0] rs_addr;
   wire [1:0] rt_addr;
   wire [1:0] rd_addr;
   wire [7:0] immediate;
   //control signals
   wire RegDst;
   wire RegWrite;
   wire ALUSrc1;
   wire ALUSrc2;
   wire MemWrite;
   wire MemToReg;

   wire [1:0] regfile_WriteAddress; //destination register address
   wire [8:0] regfile_WriteData;//result data
   wire [8:0] regfile_ReadData1;//source register1 data
   wire [8:0] regfile_ReadData2;//source register2 data

   wire [8:0] alu_result;
   wire [8:0] Data_Mem_Out;
        reg [7:0] zero_register = 0;//ZERO constant
```

```
    wire pb_clk_debounced;

        assign alu_result = {alu_ovf, alu_output};

        // Assign LEDs
    assign led = alu_output;
        assign ovf_ctrl = alu_ovf;

        // Push button debounce
    debounce debounce_clk(
       .clk_in(clk),
       .rst_in(rst_general),
       .sig_in(top_pb_clk),
       .sig_debounced_out(pb_clk_debounced)
    );

        // 7-Segment display module
        Adaptor_display display(
        .clk(clk), // system clock
        .input_value(alu_output),// 8-bit input [7:0] value to display
        .disp_en(disp_en),// output [3:0] 7 segment display enable
        .seg7_output(seg7_output)// output [6:0] 7 segment signals
        );

    //Instantiate Your instruction decoder here
inst_decoder (.instruction(instruction), .opcode(opcode), .rs_addr(rs_addr), .rt_addr(rt_addr),
.rd_addr(rd_addr), .immediate(immediate), .RegDst(RegDst), .RegWrite(RegWrite),
.ALUSrc1(ALUSrc1),.ALUSrc2(ALUSrc2), .ALUOp(ALUOp), .MemWrite(MemWrite),
.MemToReg(MemToReg));

        //Instantiate Your alu-regfile here
regfile
rf(.rd0_data(regfile_ReadData1),.rd1_data(regfile_ReadData2),.wr_data(regfile_WriteData),.rd0_ad
dr(rs_addr),.rd1_addr(rt_addr),.wr_addr(regfile_WriteAddress),.wr_en(RegWrite),.clk(pb_clk_debo
unced),.rst(rst_general));
    Mux m1(.in1(regfile_ReadData1),.sel(ALUSrc1),.in2(zero_register),.out(alu_input1));
//instantiate template
    Mux m2(.in1(regfile_ReadData2),.sel(ALUSrc2),.in2(immediate),.out(alu_input2)); //instantiate
template
    alu
a1(.a(alu_input1),.b(alu_input2),.sel(ALUOp),.f(alu_output),.ovf(alu_ovf),.take_branch(take_branc
h));

 //Instantiate Your data memory here
data_memory data (
  .a(alu_output),      // input wire [7 : 0] a
  .d(regfile_ReadData2),      // input wire [8 : 0] d
  .clk(pb_clk_debounced),  // input wire clk
  .we(MemWrite),    // input wire we
  .spo(Data_Mem_Out));  // output wire [8 : 0] spo

        //Mux for regfile_writedata
Mux m3(.in1(alu_result),.sel(MemtoReg),.in2(Data_Mem_Out),.out(regfile_WriteData));
```

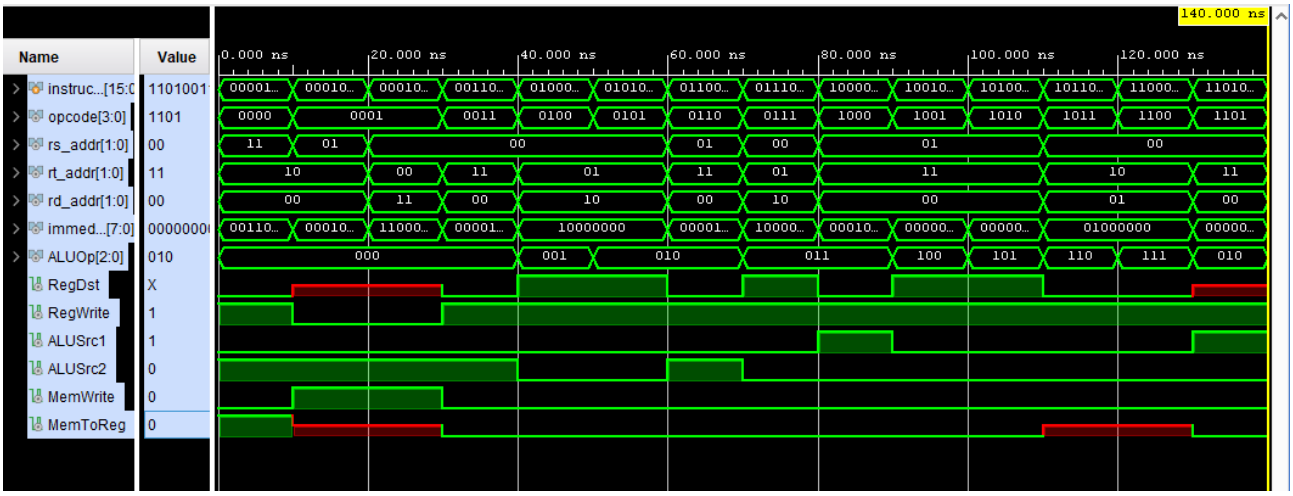
```
        //Mux for RegDST
Mux m4(.in1(rt_addr),.sel(RegDst),.in2(rd_addr),.out(regfile_WriteAddress)); //instantiate template

        //Instantiate Your VIO core here
vio_0 vio (
  .clk(clk),                // input wire clk
  .probe_in0(regfile_WriteData),    // input wire [8 : 0] probe_in0
  .probe_in1(regfile_ReadData1),    // input wire [7 : 0] probe_in1
  .probe_in2(regfile_ReadData2),    // input wire [7 : 0] probe_in2
  .probe_in3(alu_input1),    // input wire [7 : 0] probe_in3
  .probe_in4(alu_input2),    // input wire [7 : 0] probe_in4
  .probe_in5(take_branch),    // input wire [0 : 0] probe_in5
  .probe_in6(alu_ovf),    // input wire [0 : 0] probe_in6
  .probe_in7(opcode),    // input wire [3 : 0] probe_in7
  .probe_in8(alu_output),    // input wire [7 : 0] probe_in8
  .probe_in9(Data_Mem_Out),    // input wire [8 : 0] probe_in9
  .probe_out0(instruction)  // output wire [15 : 0] probe_out0
);


endmodule



`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 08/05/2022 11:35:30 AM
// Design Name:
// Module Name: inst_decoder_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module inst_decoder_tb(
    );

reg [15:0] instruction; //input
 //outputs
 wire [3:0] opcode;
 wire [1:0] rs_addr, rt_addr, rd_addr;
 wire [7:0] immediate;
```

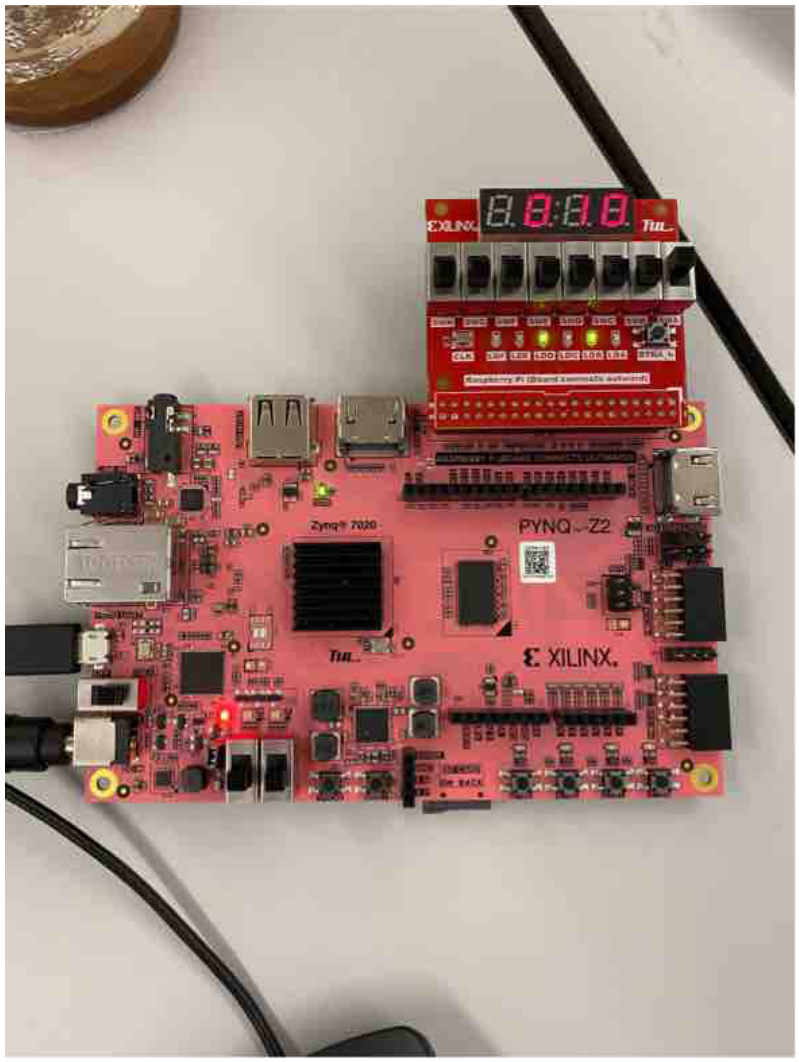
```
 wire [2:0] ALUOp;
 wire RegDst, RegWrite, ALUSrc1, ALUSrc2, MemWrite, MemToReg;

inst_decoder uut(

    .instruction(instruction),
    .opcode(opcode),
    .rs_addr(rs_addr),
    .rt_addr(rt_addr),
    .rd_addr(rd_addr),
    .immediate(immediate),
    .RegWrite(RegWrite),
    .RegDst(RegDst),
    .ALUSrc1(ALUSrc1),
    .ALUSrc2(ALUSrc2),
    .ALUOp(ALUOp),
    .MemWrite(MemWrite),
    .MemToReg(MemToReg)     );

    initial
        begin


        instruction = 16'b0000_11_10_00_110011;  //loading word
        #10
        instruction = 16'b0001_01_10_00_010001; //sw reg 1
         #10
        instruction = 16'b0010_00_01_10_00000; //add
         #10
        instruction = 16'b0011__00_11_00_001010; //addi
          #10
        instruction = 16'b0100_00_01_10_000000; //inv
         #10
        instruction = 16'b0101_00_01_10_000000; //and
          #10
        instruction = 16'b0110_01_11_00_001111; //andi
         #10
        instruction = 16'b0111_00_01_10_000000; //or
          #10
        instruction = 16'b1000_01_11_00_010100; //ori
         #10
        instruction = 16'b1001_01_11_00_000011; //sra
          #10
        instruction = 16'b1010_01_11_00_000001; //sll
         #10
        instruction = 16'b1011_00_10_01_000000; //beq
         #10
        instruction = 16'b1100_00_10_01_000000; //bne
         #10
        instruction = 16'b1101_00_11_00_000000; //clr
        #10  ;
        $finish;
        $monitor("instruction=%b opcode=%b immediate=%b rs_addr=%b rt_addr=%b rd_addr=%b
RegWrite=%b RegDst=%b ALUSrc1=%b ALUSrc2=%b ALUOP=%b MemWrite=%b
```

```
MemToReg=%b",
        instruction,opcode,immediate, rs_addr, rt_addr, rd_addr, RegWrite, RegDst, ALUSrc1,
ALUSrc2,
        ALUOp, MemWrite, MemToReg);

    end


endmodule
```

AppendixB: Design Program Screenshots (Simulations, etc)

TESTBENCH OUTPUT:




PYNQ-Z2
XILINX.