Name: Alexander Wise **Lab #5 – Instruction Decoding**

Instructor Name: Farzad Niknia **March 19, 2024**

Background

The instruction decoder is another fundamental part of the ALU. It automatically changes inputs and enable values based on the opcode from the user or application. This simplifies operations and increases overall computing speed of various operations. Similarly, it makes direct user interaction with the CPU much easier.

Pre-Lab

2.1.1 - Complete the Instruction Decoder Table:

	Inputs				Outputs								
operation	**opcode**	**immediate**	**offset**	**nzimm**	**RegWrite**	**RegDst**	**instr_i**	**ALUSrc1**	**ALUSrc2**	**ALUOp**	**MemWrite**	**MemToReg**	**Regsrc**
Load Word (lw)	4'b0000	value from vio	x	x	1	1	vio	0	1	0	0	1	0
Store Word (sw)	4'b0001	value from vio	x	x	0	0	vio	0	1	0	1	0	0
add	4'b0010	x	x	x	1	1	x	0	0	0	0	0	1
add immediate (addi)	4'b0011	x	x	value from vio	1	1	vio	0	1	0	0	0	1
and	4'b0100	x	x	x	1	1	x	0	0	2	0	0	1
andi	4'b0101	value from vio	x	x	1	1	vio	0	1	2	0	0	1
or	4'b0110	x	x	x	1	1	x	0	0	3	0	0	1
xor	4'b0111	x	x	x	1	1	x	0	0	8	0	0	1
sra immedate (srai)	4'b1000	x	x	value from vio	1	1	vio	0	1	4	0	0	1
slli	4'b1001	x	x	value from vio	1	1	vio	0	1	5	0	0	1
beqz	4'b1010	x	value from vio	x	0	x	vio	0	x	6	0	0	0
bneqz	4'b1011	x	value from vio	x	0	x	vio	0	x	7	0	0	0

Figure 1: Table of Instruction Decoder Values

Design Implementation

2.2.1 - Instruction Decoder:

The instruction decoder automatically applies specific operations based on opcode input. The purpose of the instruction_decoder module is to automatically apply these operations based on the input opcode. To achieve this, a case statement was used within an always @ statement that depended on the opcode input. Each opcode from values 0 to 11 is represented in this case statement with specific instructions for all regfile and multiplexer values. A screenshot of the code for this can be found in appendix A labeled as figures 3 through 5.

2.2.2 – Simulate the Instruction Decoder

To simulate the instruction decoder, a test bench was created to test each instruction using random values for the immediate values such as immediate, offset, and nzimm. In total, there are 12 opcodes to be tested in the test bench. The waveform from this simulation as well as the test bench code can be found in appendix A labeled as figures 6 and 7 respectively.

2.2.3 – Designing the Top Level

Next, the top-level file, debounce, and display adapter are imported to the project. Some changes had to be made to the top-level module to ensure that the processor worked properly such as multiplexer renaming and the renaming of several module components.

2.3 – Testing in Hardware

After uploading the full top level module to the board, test cases were run from the table provided in the lab instructions. Using the module, empty values were filled in based on hardware outputs. The following table will show the outputs from each of these test cases:

	Inputs							Outputs							
operation	**opcode**	**immediate**	**offset**	**nzimm**	**rd**	**rs1**	**rs2**	**WriteData**	**ReadData1**	**ReadData2**	**input1**	**input2**	**alu_out**	**alu_ovf**	**take_branch**
addi value of 5 with register 2	3	x	x	5	2	0	0	5	0	0	0	5	5	0	0
Let 1 clk cycle pass using BTN1	3	x	x	5	2	0	0	A	5	0	5	5	A	0	0
addi value of 3 with register 1	3	x	x	3	1	0	0	3	0	0	0	3	3	0	0
Let 1 clk cycle pass using BTN1	3	x	x	3	1	0	0	6	3	0	3	3	6	0	0
store value at register 1 to memory address 6	1	6	x	x	1	0	0	6	0	0	0	6	6	0	0
Let 1 clk cycle pass using BTN1	1	6	x	x	1	0	0	6	0	0	0	6	6	0	0
load value at memory address 6 to register 4	0	6	x	x	4	0	0	0	0	0	0	6	6	0	0
Let 1 clk cycle pass using BTN1	0	6	x	x	4	0	0	0	0	0	0	6	6	0	0
srai value at regiter 4 with immediate value of 1	8	x	x	1	4	0	0	3	6	0	6	1	3	0	0
Let 1 clk cycle pass using BTN1	8	x	x	1	4	0	0	1	3	0	3	1	1	0	0
xor value at register 2 with value at register 4	7	x	x	x	2	0	4	6	5	3	5	3	6	0	0
addi value of 5 with 2 register	3	x	x	5	2	0	0	A	5	0	5	5	A	0	0
beqz value at register address 3 with offset 0	A	x	0	x	0	3	0	0	0	0	0	0	0	0	1
bneqz value at register address 3 with offset 0	B	x	0	x	0	3	0	0	0	0	0	0	0	0	0

Figure 2: Table of Operations and Their Respective Outputs

Similarly, images can be found in Appendix B that show these values represented on the hardware itself. These images are labeled as figures 8 through 11

Conclusion

The instruction decoder is another fundamental component of a processor, allowing for easy operations based on various operations as presented by their respective opcode. With the introduction of the instruction decoder, we can now create basic programs using RISC-V assembly as many ALU operations have now been incorporated into the processor. Similarly, we are continuing to advance our knowledge of System Verilog as well as the Xilinx Vivado IDE though the continued application of System Verilog within our processor.

Appendix A – Xilinx Vivado Screen captures

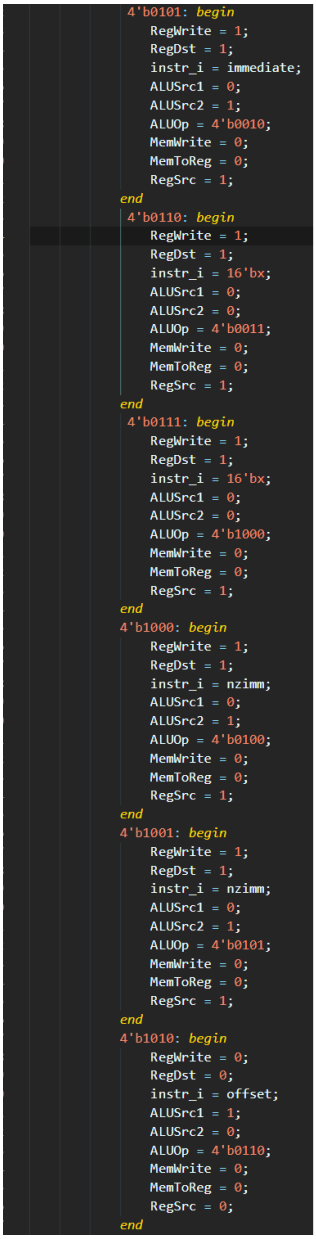
```
module instruction_decoder(
    input logic [6:0] immediate,
    input logic [5:0] nzimm,
    input logic [8:0] offset,
    input logic [3:0] opcode,
    output logic RegWrite, RegDst, ALUSrc1, ALUSrc2, MemWrite, MemTo
    output logic [15:0] instr_i,
    output logic [3:0] ALUOp
    );

    always @(opcode) begin
        case(opcode)
            4'b0000: begin
                RegWrite = 1;
                RegDst = 1;
                instr_i = immediate;
                ALUSrc1 = 0;
                ALUSrc2 = 1;
                ALUOp = 0;
                MemWrite = 0;
                MemToReg = 1;
                RegSrc = 0;
            end

            4'b0001: begin
                RegWrite = 0;
                RegDst = 0;
                instr_i = immediate;
                ALUSrc1 = 0;
                ALUSrc2 = 1;
                ALUOp = 0;
                MemWrite = 1;
                MemToReg = 0;
                RegSrc = 0;
            end

            4'b0010: begin
                RegWrite = 1;
                RegDst = 1;
                instr_i = 16'bx;
                ALUSrc1 = 0;
                ALUSrc2 = 0;
                ALUOp = 0;
                MemWrite = 0;
                MemToReg = 0;
                RegSrc = 1;
            end
             4'b0011: begin
                RegWrite = 1;
                RegDst = 1;
                instr_i = nzimm;
                ALUSrc1 = 0;
                ALUSrc2 = 1;
                ALUOp = 0;
                MemWrite = 0;
                MemToReg = 0;
                RegSrc = 1;
            end
             4'b0100: begin
                RegWrite = 1;
                RegDst = 1;
                instr_i = 16'bx;
                ALUSrc1 = 0;
                ALUSrc2 = 0;
                ALUOp = 4'b0010;
                MemWrite = 0;
                MemToReg = 0;
                RegSrc = 1;
            end
```

Figure 3: Instruction Decoder Code Up to Opcode 0100



```
4'b0101: begin
    RegWrite = 1;
    RegDst = 1;
    instr_i = immediate;
    ALUSrc1 = 0;
    ALUSrc2 = 1;
    ALUOp = 4'b0010;
    MemWrite = 0;
    MemToReg = 0;
    RegSrc = 1;
end
4'b0110: begin
    RegWrite = 1;
    RegDst = 1;
    instr_i = 16'bx;
    ALUSrc1 = 0;
    ALUSrc2 = 0;
    ALUOp = 4'b0011;
    MemWrite = 0;
    MemToReg = 0;
    RegSrc = 1;
end
4'b0111: begin
    RegWrite = 1;
    RegDst = 1;
    instr_i = 16'bx;
    ALUSrc1 = 0;
    ALUSrc2 = 0;
    ALUOp = 4'b1000;
    MemWrite = 0;
    MemToReg = 0;
    RegSrc = 1;
end
4'b1000: begin
    RegWrite = 1;
    RegDst = 1;
    instr_i = nzimm;
    ALUSrc1 = 0;
    ALUSrc2 = 1;
    ALUOp = 4'b0100;
    MemWrite = 0;
    MemToReg = 0;
    RegSrc = 1;
end
4'b1001: begin
    RegWrite = 1;
    RegDst = 1;
    instr_i = nzimm;
    ALUSrc1 = 0;
    ALUSrc2 = 1;
    ALUOp = 4'b0101;
    MemWrite = 0;
    MemToReg = 0;
    RegSrc = 1;
end
4'b1010: begin
    RegWrite = 0;
    RegDst = 0;
    instr_i = offset;
    ALUSrc1 = 1;
    ALUSrc2 = 0;
    ALUOp = 4'b0110;
    MemWrite = 0;
    MemToReg = 0;
    RegSrc = 0;
end
```

Figure 4: Instruction Decoder Code up to Opcode 1010

```
            4'b1010: begin
                RegWrite = 0;
                RegDst = 0;
                instr_i = offset;
                ALUSrc1 = 1;
                ALUSrc2 = 0;
                ALUOp = 4'b0110;
                MemWrite = 0;
                MemToReg = 0;
                RegSrc = 0;
            end
            4'b1011: begin
                RegWrite = 0;
                RegDst = 0;
                instr_i = offset;
                ALUSrc1 = 1;
                ALUSrc2 = 0;
                ALUOp = 4'b0111;
                MemWrite = 0;
                MemToReg = 0;
                RegSrc = 0;
            end
        endcase
    end
endmodule
```

Figure 5: Instruction Decoder Code up to Opcode 1011

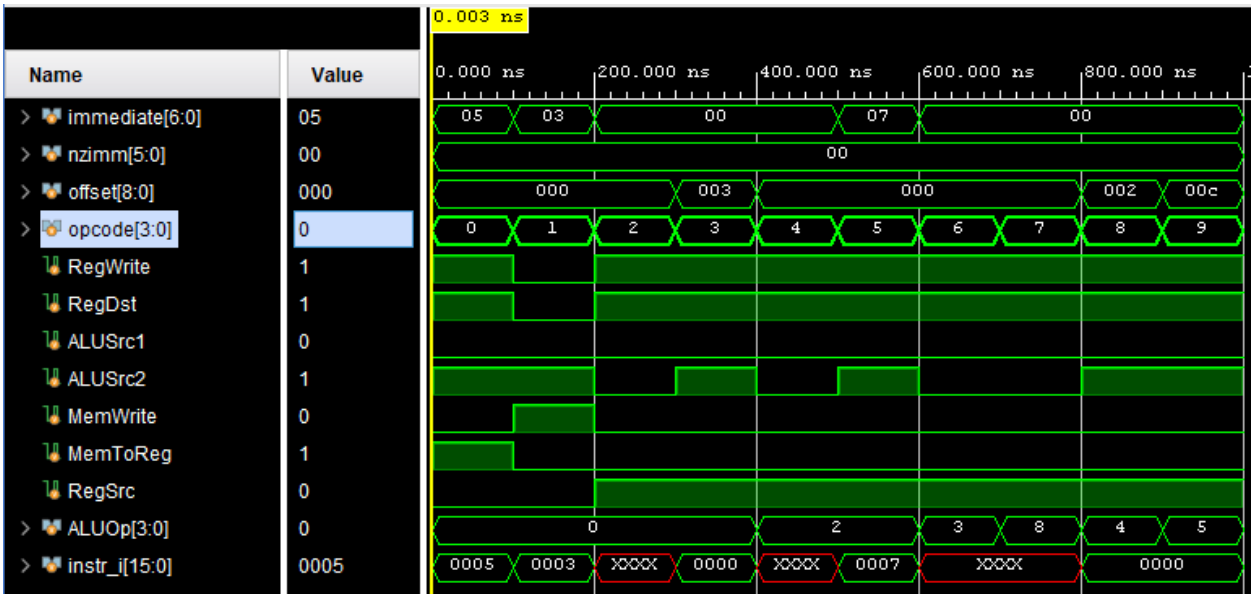


Figure 6: Waveform Generated from Test Bench Run

```
module inst_decoder_tb();

logic RegWrite, RegDst, ALUSrc1, ALUSrc2, MemWrite, MemToReg, RegSrc;
logic [6:0]immediate;
logic [5:0]nzimm;
logic [8:0]offset;
logic [3:0]opcode, ALUOp;
logic [15:0]instr_i;

instruction_decoder id(
    .RegWrite(RegWrite),
    .RegDst(RegDst),
    .ALUSrc1(ALUSrc1),
    .ALUSrc2(ALUSrc2),
    .MemWrite(MemWrite),
    .MemToReg(MemToReg),
    .RegSrc(RegSrc),
    .immediate(immediate),
    .nzimm(nzimm),
    .offset(offset),
    .opcode(opcode),
    .ALUOp(ALUOp),
    .instr_i(instr_i)
);

initial begin
    opcode = 4'b0000; immediate = 7'b0000101; nzimm = 6'b000000; offset = 9'b000000000;
    #100 opcode = 4'b0001; immediate = 7'b0000011; nzimm = 6'b000000; offset = 9'b000000000;
    #100 opcode = 4'b0010; immediate = 7'b0000000; nzimm = 6'b000000; offset = 9'b000000000;
    #100 opcode = 4'b0011; immediate = 7'b0000000; nzimm = 6'b000000; offset = 9'b000000011;
    #100 opcode = 4'b0100; immediate = 7'b0000000; nzimm = 6'b000000; offset = 9'b000000000;
    #100 opcode = 4'b0101; immediate = 7'b0000111; nzimm = 6'b000000; offset = 9'b000000000;
    #100 opcode = 4'b0110; immediate = 7'b0000000; nzimm = 6'b000000; offset = 9'b000000000;
    #100 opcode = 4'b0111; immediate = 7'b0000000; nzimm = 6'b000000; offset = 9'b000000000;
    #100 opcode = 4'b1000; immediate = 7'b0000000; nzimm = 6'b000000; offset = 9'b000000010;
    #100 opcode = 4'b1001; immediate = 7'b0000000; nzimm = 6'b000000; offset = 9'b000001100;
    #100 opcode = 4'b1010; immediate = 7'b0000000; nzimm = 6'b000010; offset = 9'b000000000;
    #100 opcode = 4'b1011; immediate = 7'b0000000; nzimm = 6'b000110; offset = 9'b000000000;
end
endmodule
```

Figure 7: Instruction Decoder Test Bench Code

Appendix B – Images of Instruction Decoder Working on Hardware

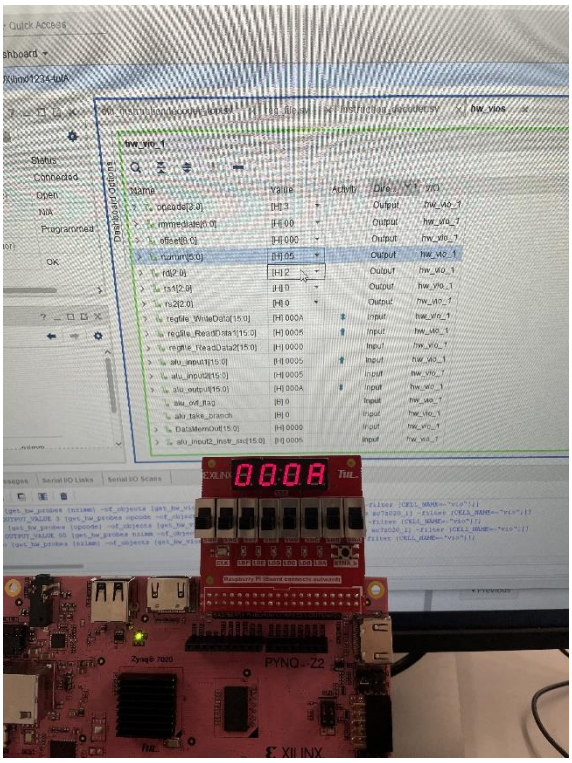


Figure 8: Table Operation 1 on Hardware

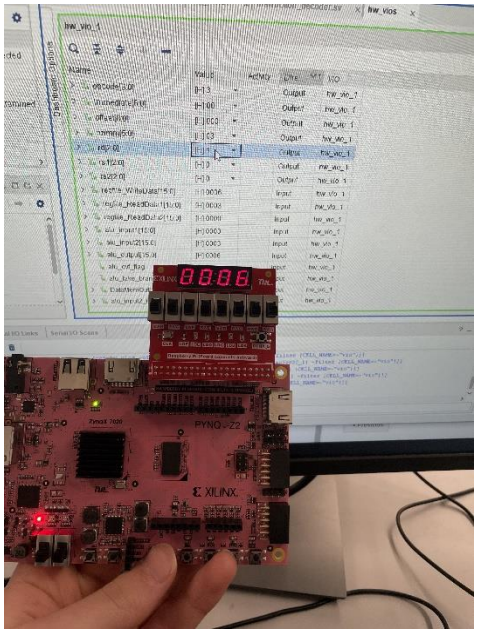


Figure 9: Operation 2 on Hardware

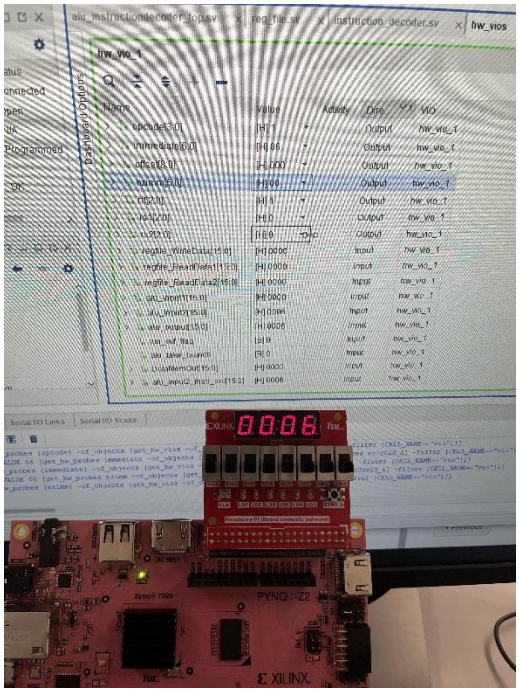


Figure 10: Operation 3 on Hardware

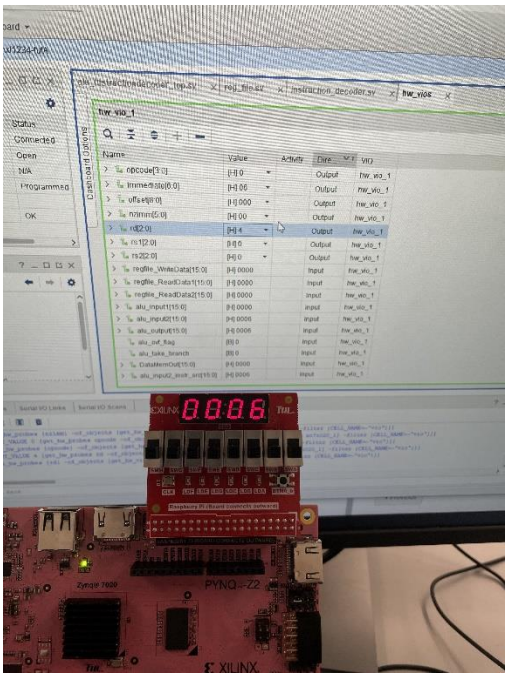


Figure 11: Operation 4 on Hardware