Lab 5

Sam Sahli and Shaun Gordon

February 19, 2017

1 Introduction

The purpose of this lab was to build the first two modules in the decode stage of the MIPS computer. The first was a sign extender and the second was a control unit. The sign extender is responsible for extending a 16-bit number to a 32-bit number while maintaining its sign (positive of negative). The control unit is responsible for interpreting the opcode and setting the control bits in the computer. Verilog code to construct both modules was written and a test benches for each were also created to test their function. Both of these modules, after being tested for correct operation, will later be included in the final decode stage code.

2 Interface

The sign extender consists of one input and one output. The input is 16 bits and the output is 32. The 16th bit of the input is replicated 16 times and concatenated onto the original 16 bits. So the output is the original 16 bits in bit positions 15-0, and the bit from position 15 is placed in bit positions 32-16.

The control unit has one 6 bit input and eight outputs. These eight outputs are the computer control bits. The 6 input bits, or the opcode, is interpreted and used to set the control bits. So the control unit module must take the opcode and use it to correctly assign values to each of the outputs.

3 Design

The design of the sign extender is fairly basic as only one function needs to be executed. Its only purpose is to turn a 16 bit value into a 32 bit value while maintaining its sign and value. It is required to do this for both positive and negative values. So a module must be built to make a new 16 bit number and concatenate that new number onto the original 16 bit number, and output that value.

The design of the control unit is slightly more complicated. The control needs to be built to recognize the value of the opcode input and then set correct

output values. Code will be written to read the input and set different outputs based on that input. The control unit must also be built with a default setting that will generate an output without meaningful opcode input. This default output must be designed to not make any changes to the computers control bits.

4 Implementation

As mentioned in the interface section, the sign extender was designed with a single input and output. For the MIPS computer the input must be 16 bits and the output 32, but these values are not universal. Because of this, the code was designed to be easily modified. This was done by referencing the value WORD in the Verilog header file. WORD is currently set to 32, but can be changed with one edit in the header file. The input is set to be half of WORD, 16 bits, and the output is set to be WORD, 32 bits. The following assign statement takes the original 16 bit input and concatenates 16 more bits into the higher bit positions. These new bits are determined by the 16th bit of the input. By taking the 16th bit, copying it 16 times, and placing it on the front of the input, the sign and value of the input is maintained. So if the input is negative a 1 will be placed in bit positions 32-15 and if the input is positive a 0 will be placed in bit positions 32-15. This agrees with the practice of a negative being represented with a twos-complement. This code for the sign extender can be seen in Listing 1.

The control unit is created with one input, named Opcode, and 8 outputs that represent the different control bits of the computer. The input is used to determine the value each output is set to. This is implemented with a case statement. The case statement takes the value assigned to input Opcode and if that value matches with a MIPS command's opcode the outputs are assigned accordingly. Four commands are included in the code shown in Listing 2. Each case has its own outputs, and these outputs will be used to execute the command indicated by the opcode. This is why the code was implemented with a case statement.

Listing 1: Verilog code for implementing a sign extender.

```
`include "definitions.vh"

module sign_extender(
    input [(`WORD/2)-1:0] in,
    output [`WORD-1:0] out
    );
    assign out = {{(`WORD/2){in[(`WORD/2)-1]}} , in };
endmodule
```

Listing 2: Verilog code for implementing a control unit.

```
`include "definitions.vh"

module control(
    input [5:0] Opcode,
    output reg  RegDst,
    output reg  Branch,
    output reg  MemRead,
    output reg  MemtoReg,
    output reg  [1:0] ALUOp,
    output reg  MemWrite,
    output reg  ALUSrc,
    output reg  RegWrite
    );
    always@(*) begin
        case (Opcode)
        `RTYPE: begin
                RegDst<=1;
                Branch<=0;
                MemRead<=0;
                MemtoReg<=0;
                ALUOp<=`ALUOp_R;
                MemWrite<=0;
                ALUSrc<=0;
                RegWrite<=1;
            end
        `LW: begin
                RegDst<=0;
                Branch<=0;
                MemRead<=1;
                MemtoReg<=1;
                ALUOp<=`ALUOp_ADD;
                MemWrite<=0;
                ALUSrc<=1;
                RegWrite<=1;
            end
        `SW: begin
                RegDst<=0;
                Branch<=0;
                MemRead<=0;
                MemtoReg<=0;
                ALUOp<=`ALUOp_ADD;
                MemWrite<=1;
                ALUSrc<=1;
                RegWrite<=0;
            end
        `BEQ: begin
```

```
                    RegDst<=0;
                    Branch<=1;
                    MemRead<=0;
                    MemtoReg<=0;
                    ALUOp<=`ALUOp_SUB;
                    MemWrite<=0;
                    ALUSrc<=0;
                    RegWrite<=0;
                end
            default: begin
                    RegDst<=0;
                    Branch<=0;
                    MemRead<=0;
                    MemtoReg<=0;
                    ALUOp<=0;
                    MemWrite<=0;
                    ALUSrc<=0;
                    RegWrite<=0;
                end
            endcase
    end
endmodule
```

5 Test Bench Design

The test bench for the sign extender was designed to confirm the sign of a number is extended from 16 to 32 bits for different input values. Low and high values are tested to determine whether the 16th bit of the input is copied 16 times and placed on the front of the output. The tests confirm this operation for both negative and positive input values. The sign extender test bench code is displayed in Listing 3.

The test bench for the control unit needs to confirm that for a specific value of the opcode, the control bits are set to their correct values. This is done by testing known opcode values, and confirming that the outputs are correct for that known input. The test bench also includes random values for the input to make sure the default case works. This test bench code is displayed in Listing 4.

Listing 3: Verilog code for testing a sign extender.

```
`include "definitions.vh"

module sign_extender_test;

    reg [(`WORD/2)-1:0] in;
    wire [`WORD-1:0] out;
```

```
    sign_extender UUT (
                .in(in),
                .out(out)
        );

        initial begin
            in = 'b1000000000000000;
            #CYCLE;

            in = 'b0111111111111111;
            #CYCLE;

            in = 'b1111111111111111;
            #CYCLE;

            in = 'b0000000000000000;
            #CYCLE;
        end

endmodule
```

Listing 4: Verilog code for testing a control unit.

```
`include "definitions.vh"

module control_test;
    reg [5:0] Opcode;
    wire  RegDst;
    wire  Branch;
    wire  MemRead;
    wire  MemtoReg;
    wire  [1:0] ALUOp;
    wire  MemWrite;
    wire  ALUSrc;
    wire  RegWrite;

    control UUT (
        .Opcode(Opcode),
        .RegDst(RegDst),
        .Branch(Branch),
        .MemRead(MemRead),
        .MemtoReg(MemtoReg),
        .ALUOp(ALUOp),
        .MemWrite(MemWrite),
        .ALUSrc(ALUSrc),
```

Figure 1: Timing diagram for the sign extender test.

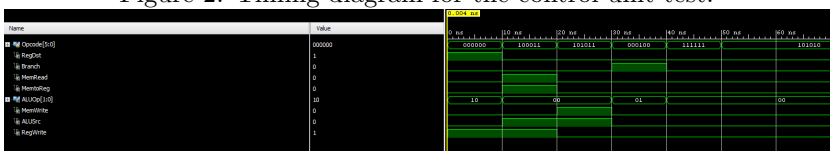
```
        .RegWrite(RegWrite)
    );


    initial begin
            Opcode = `RTYPE;
            #`CYCLE;

            Opcode = `LW;
            #`CYCLE;

            Opcode = `SW;
            #`CYCLE;

            Opcode = `BEQ;
            #`CYCLE;

            Opcode = 'b111111;
            #`CYCLE;

            Opcode = 'b101010;
            #`CYCLE;
    end

endmodule
```

6 Simulation

The timing diagram for the sign extender simulation is shown in Figure 1. Four inputs are shown, and for each value the 16th bit is copied 16 times, and placed into bit locations 32-16. This waveform confirms the operation of the sign extender.

The timing diagram for the control unit is displayed in Figure 2. The opcode for the four desired MIPS commands were each tested, and this waveform shows that the desired control bits are set on the output. This waveform also sets all of the control bits to zero in the default case. This proves that for random values of the input, no changes are made to the control bits. The output for known and random values of the opcode confirms the operation of the control unit.

Figure 2: Timing diagram for the control unit test.



7 Conclusions

The purpose of this lab was to generate, test, and confirm the code for the first two modules of the decode stage of the MIPS computer. The sign extender is required to sign extend inputs without altering their value. This extended value is then output from the decode stage and used in the execute stage. The control unit interprets the opcode and sets the computers control bits. These bits are used in the execute stage as well. So this lab generated two modules that are integral to the decode stage, as they produce values that are used in other stages of the computer.