Lab 5: Control Unit and Sign Extender

Matthew Carrano and Breana Leal

February 12, 2018

1 Simplified Report

The control module uses a portion of the instruction data, the opcode field, as inputs to define which control signals to set. The set signals will determine which signals the processor will use. The Sign Extender module uses the address from the instruction code and outputs a 64 bit sign extended version. Using the expected results table, the module is verified.

2 Code

Listing 1: Verilog code for implementing the control module.

```
`include "definitions.vh"

module control(
    input [10:0] opcode_bits,
    output reg reg2_loc,
    output reg uncondbranch,
    output reg branch,
    output reg mem_read,
    output reg mem_to_reg,
    output reg [1:0] alu_op,
    output reg mem_write,
    output reg alu_src,
    output reg reg_write
    );

    always @(*) begin
        casex (opcode_bits)
        // ADD, SUB, AND, ORR, LDUR, STUR, CBZ, and B should be
        // defined in definitions.vh.  CBZ and B instructions should
        // use X to fill in the extra bits in the 11 bit opcode
        // casex treats these bits as 'don't cares'
        `ADD:   begin
```

```
            reg2_loc <=0;
            uncondbranch <=0;
            branch <=0;
            mem_read <=0;
            mem_to_reg <=0;
            alu_op <='ALUOp_RTYPE;
            mem_write <=0;
            alu_src <=0;
            reg_write <=1;
          end
    'SUB: begin
            reg2_loc <=0;
            uncondbranch <=0;
            branch <=0;
            mem_read <=0;
            mem_to_reg <=0;
            alu_op <='ALUOp_RTYPE;
            mem_write <=0;
            alu_src <=0;
            reg_write <=1;
          end
    'AND: begin
            reg2_loc <=0;
            uncondbranch <=0;
            branch <=0;
            mem_read <=0;
            mem_to_reg <=0;
            alu_op <='ALUOp_RTYPE;
            mem_write <=0;
            alu_src <=0;
            reg_write <=1;
          end
    'ORR: begin
            reg2_loc <=0;
            uncondbranch <=0;
            branch <=0;
            mem_read <=0;
            mem_to_reg <=0;
            alu_op <='ALUOp_RTYPE;
            mem_write <=0;
            alu_src <=0;
            reg_write <=1;
          end
    'LDUR: begin
            reg2_loc <=0;
            uncondbranch <=0;
```

```
                branch<=0;
                mem_read<=1;
                mem_to_reg<=1;
                alu_op<=`ALUOp_DTYPE;
                mem_write<=0;
                alu_src <=1;
                reg_write <=1;
              end
        `LDUR: begin
                reg2_loc <=0;
                uncondbranch<=0;
                branch<=0;
                mem_read<=1;
                mem_to_reg<=1;
                alu_op<=`ALUOp_DTYPE;
                mem_write<=0;
                alu_src <=1;
                reg_write <=1;
              end
        `STUR: begin
                reg2_loc <=1;
                uncondbranch<=0;
                branch<=0;
                mem_read<=0;
                mem_to_reg<=0;
                alu_op<=`ALUOp_DTYPE;
                mem_write<=1;
                alu_src <=1;
                reg_write <=0;
              end
        `CBZ: begin
                reg2_loc <=1;
                uncondbranch<=0;
                branch<=1;
                mem_read<=0;
                mem_to_reg<=0;
                alu_op<=`ALUOp_BRANCH;
                mem_write<=0;
                alu_src <=0;
                reg_write <=0;
              end
        `B: begin
                reg2_loc <=0;
                uncondbranch<=1;
                branch<=0;
                mem_read<=0;
```

```
                    mem_to_reg<=0;
                    alu_op<=`ALUOp_BRANCH;
                    mem_write<=0;
                    alu_src<=0;
                    reg_write<=0;
                end
        endcase
    end
endmodule
```

Listing 2: Verilog code for implementing the sign extender module.

```
`include "definitions.vh"

module sign_extender(
    input [`INSTR_LEN-1:0] instr,
    output reg [`WORD-1:0] extended
    );

    always @(*) begin
        casex (instr[`INSTR_LEN-1:21])

        `LDUR: begin
                extended[8:0] <= instr[20:12];
                if(extended[8] == 1'b1)
                    extended[63:9] <= 55'b1111111111111111111111111111111111111111111
                else
                    extended[63:9] <= 55'b0000000000000000000000000000000000000000000
               end

        `STUR: begin
                extended[8:0] <= instr[20:12];
                if(extended[8] == 1'b1)
                     extended[63:9] <= 55'b1111111111111111111111111111111111111111111
                else
                     extended[63:9] <= 55'b0000000000000000000000000000000000000000000
                end

        `CBZ:  begin
               extended[18:0] <= instr[23:5];
               if(extended[18] == 1'b1)
                    extended[63:19] <= 45'b1111111111111111111111111111111111111111111
               else
                    extended[63:19] <= 45'b0000000000000000000000000000000000000000000
               end
```

```
        `B:     begin
                extended[25:0] <= instr[25:0];
                if(extended[25] == 1'b1)
                    extended[63:26] <= 38'b11111111111111111111111111111111111111
                else
                    extended[63:26] <= 38'b00000000000000000000000000000000000000
                end

        endcase
    end

endmodule
```

3 Testbench

Listing 3: Verilog code for implementing the control testbench.

```
`include "definitions.vh"

module control_test;

    reg [10:0] opcode_bits;
    wire reg2_loc;
    wire uncondbranch;
    wire branch;
    wire mem_read;
    wire mem_to_reg;
    wire [1:0] alu_op;
    wire mem_write;
    wire alu_src;
    wire reg_write;

    control UUT
        (
        .opcode_bits(opcode_bits),
        .reg2_loc(reg2_loc),
        .uncondbranch(uncondbranch),
        .branch(branch),
        .mem_read(mem_read),
        .mem_to_reg(mem_to_reg),
        .alu_op(alu_op),
        .mem_write(mem_write),
        .alu_src(alu_src),
        .reg_write(reg_write)
```

```
        );

    initial begin

    opcode_bits <= 11'b11111000010; //1
    #CYCLE;
    opcode_bits <= 11'b10001011000; //2
    #CYCLE;
    opcode_bits <= 11'b11001011000; //3
    #CYCLE;
    opcode_bits <= 11'b11111000000; //4
    #CYCLE;
    opcode_bits <= 8'b10110100; //5
    #CYCLE;
    opcode_bits <= 8'b10110100; //6
    #CYCLE;
    opcode_bits <= 6'b000101; //7
    #CYCLE;
    opcode_bits <= 6'b000101; //8
    #CYCLE;
    opcode_bits <= 11'b10101010000; //9
    #CYCLE;
    opcode_bits <= 11'b10001010000; //10

    end

endmodule
```

Listing 4: Verilog code for implementing the sign extender testbench.

```
`include "definitions.vh"

module sign_ext_test;

    reg [`INSTR_LEN-1:0] instr;
    wire[`WORD-1:0] extended;

    sign_extender UUT
        (
        .instr(instr),
        .extended(extended)
        );

        initial begin

        instr <= 64'hF84402C9; //ldur
```

```
        #CYCLE;
        instr <= 64'hF80602CB; //stur
        #CYCLE;
        instr <= 64'hB4FFFF6B; //cbz
        #CYCLE;
        instr <= 64'h17FFFFC9; //b

        end

endmodule
```

4 Simulation

Figure 1: Timing diagram for control module test.

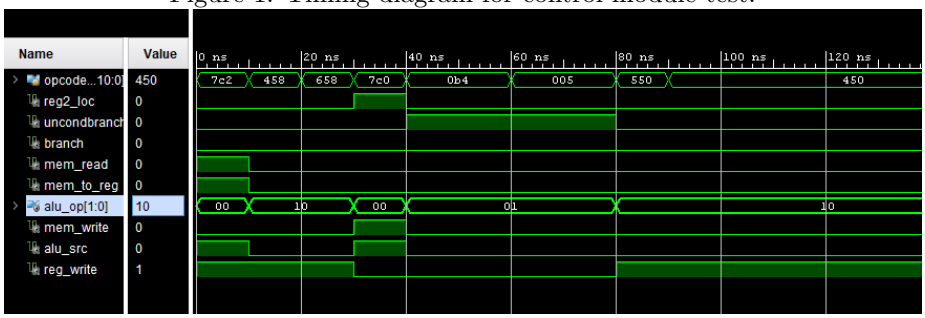

Figure 2: Timing diagram for sign extender module test.