Lab 4: Beginning to Decode

Matthew Carrano and Breana Leal

February 5, 2018

1 Simplified Report

The instr_parse module uses both R and D format to break down the instruction code. The module parsed the instruction bits successfully and it is verified using four instruction codes provided in class. The regfile module reads the data from the regData file and outputs the bits into an array. The module progresses along the positive clock edge. When write_data is HIGH data will be written to the registers. This action is verified through the simulation.

2 Code

Listing 1: Verilog code for implementing the instr_parse module.

```
`include "definitions.vh"

module instr_parse(
    input clk,
    input [`INSTR_LEN-1:0] instruction,
    output reg [10:0] opcode,
    output reg [8:0] address,
    output reg [4:0] rm_num,
    output reg [4:0] rn_num,
    output reg [4:0] rd_num

    );

    always @(posedge(clk)) begin
            opcode <= instruction[31:21];

            address <= instruction[20:12];

            rm_num <= instruction[20:16];

            rn_num <= instruction[9:5];
```

```
            rd_num <= instruction[4:0];

        end



endmodule
```

Listing 2: Verilog code for implementing the regfile module.

```
`include "definitions.vh"

module regfile(
    input read_clk,
    input write_clk,
    input regWrite,
    input [4:0] read_reg1,
    input [4:0] read_reg2,
    input [4:0] write_reg,
    input [`WORD-1:0] write_data,
    output reg [`WORD-1:0] read_data1,
    output reg [`WORD-1:0] read_data2
    );


    reg [`WORD-1:0] regs [31:0];


    always @(posedge(read_clk)) begin
        read_data1 <= regs[read_reg1];
        read_data2 <= regs[read_reg2];
    end

    always @(posedge(write_clk)) begin
        if (regWrite == 1'b1)
            regs[write_reg] <= write_data;

    end

    initial
            $readmemb(`RMEMFILE, regs);


endmodule
```

3 Testbench

Listing 3: Verilog code for implementing the instr_parse testbench.

```
`include "definitions.vh"

module instr_parse_test;

    wire clk;
    reg [`INSTR_LEN-1:0] instruction;
    wire [10:0] opcode;
    wire [8:0] address;
    wire [4:0] rm_num;
    wire [4:0] rn_num;
    wire [4:0] rd_num;

oscillator clk_gen(clk);

instr_parse UUT
    (
    .clk(clk),
    .instruction(instruction),
    .address(address),
    .opcode(opcode),
    .rm_num(rm_num),
    .rn_num(rn_num),
    .rd_num(rd_num)
    );

initial begin

instruction = 32'b11111000010011110000000101001001;
#(2*`CYCLE);
instruction = 32'b10001011000010010000001010101001;
#(2*`CYCLE);
instruction = 32'b10010001000000000000011010101001;
#(2*`CYCLE);
instruction = 32'b11111000000011110000000101001001;

end

endmodule
```

Listing 4: Verilog code for implementing the regfile testbench.

```
`include "definitions.vh"
```

```
module regfile_test;

    wire read_clk;
    wire write_clk;
    reg regWrite;
    reg [4:0] read_reg1; //rn
    reg [4:0] read_reg2; //rm
    reg [4:0] write_reg; //rd
    reg [`WORD-1:0] write_data;
    wire [`WORD-1:0] read_data_1;
    wire [`WORD-1:0] read_data_2;

    oscillator clk_gen_read(write_clk);

    oscillator clk_gen_write(read_clk);

    regfile UUT(
        .read_clk(read_clk),
        .write_clk(write_clk),
        .regWrite(regWrite),
        .read_reg1(read_reg1),
        .read_reg2(read_reg2),
        .write_reg(write_reg),
        .write_data(write_data),
        .read_data1(read_data_1),
        .read_data2(read_data_2)
    );

initial begin

regWrite <= 1'b0;
read_reg1 <= 10;
read_reg2 <= 15;

#(`CYCLE*6);

regWrite <= 1'b1;
write_reg = 5'b01001;
write_data = 256;

#(`CYCLE*2);

regWrite <= 1'b0;
read_reg1 <= 2;
read_reg2 <= 32;
```

```
end

endmodule
```

Figure 1: Timing diagram for instr_parse module test.

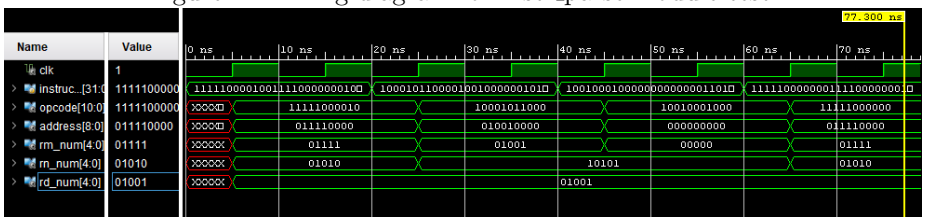
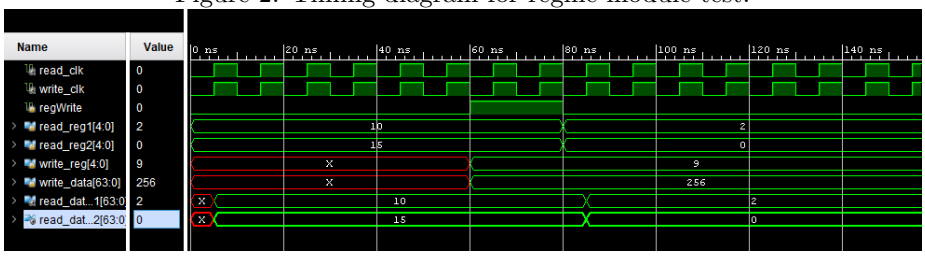


Figure 2: Timing diagram for regfile module test.



The regfile simulation is working on the positive edge and

4 Simulation

The regfile simulation is working on the positive edge and

Figure 3: Modified Register File Data.

```
1  00000000000000000000000000000000
2  00000000000000000000000000000001
3  00000000000000000000000000000010
4  00000000000000000000000000000011
5  00000000000000000000000000000100
6  00000000000000000000000000000101
7  00000000000000000000000000000110
8  00000000000000000000000000000111
9  00000000000000000000000000001000
10 00000000000000000000000000001001
11 00000000000000000000000000001010
12 00000000000000000000000000001011
13 00000000000000000000000000001100
14 00000000000000000000000000001101
15 00000000000000000000000000001110
16 00000000000000000000000000001111
17 00000000000000000000000000010000
18 00000000000000000000000000010001
19 00000000000000000000000000010010
20 00000000000000000000000000010011
21 00000000000000000000000000010100
22 00000000000000000000000000010101
23 00000000000000000000000000010110
24 00000000000000000000000000010111
25 00000000000000000000000000011000
26 00000000000000000000000000011001
27 00000000000000000000000000011010
28 00000000000000000000000000011011
29 00000000000000000000000000011100
30 00000000000000000000000000011101
31 00000000000000000000000000011110
32 00000000000000000000000000011111
```

Figure 4: Timing diagram for instr_parse module test.

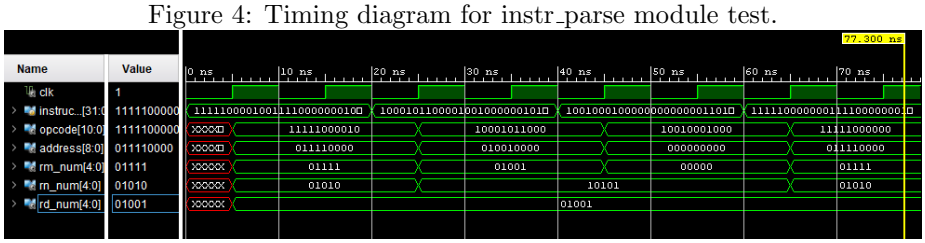


Figure 5: Timing diagram for regfile module test.

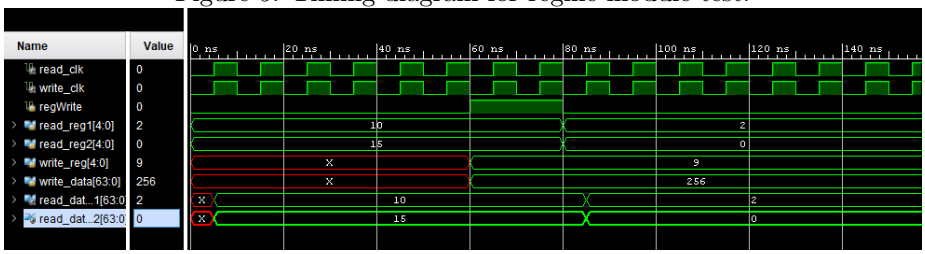


Figure 6: Modified Register File Data.

```
00000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000001
00000000000000000000000000000000000000000000000000000000000000010
00000000000000000000000000000000000000000000000000000000000000011
00000000000000000000000000000000000000000000000000000000000000100
00000000000000000000000000000000000000000000000000000000000000101
00000000000000000000000000000000000000000000000000000000000000110
00000000000000000000000000000000000000000000000000000000000000111
00000000000000000000000000000000000000000000000000000000000001000
00000000000000000000000000000000000000000000000000000000000001001
00000000000000000000000000000000000000000000000000000000000001010
00000000000000000000000000000000000000000000000000000000000001011
00000000000000000000000000000000000000000000000000000000000001100
00000000000000000000000000000000000000000000000000000000000001101
00000000000000000000000000000000000000000000000000000000000001110
00000000000000000000000000000000000000000000000000000000000001111
00000000000000000000000000000000000000000000000000000000000010000
00000000000000000000000000000000000000000000000000000000000010001
00000000000000000000000000000000000000000000000000000000000010010
00000000000000000000000000000000000000000000000000000000000010011
00000000000000000000000000000000000000000000000000000000000010100
00000000000000000000000000000000000000000000000000000000000010101
00000000000000000000000000000000000000000000000000000000000010110
00000000000000000000000000000000000000000000000000000000000010111
00000000000000000000000000000000000000000000000000000000000011000
00000000000000000000000000000000000000000000000000000000000011001
00000000000000000000000000000000000000000000000000000000000011010
00000000000000000000000000000000000000000000000000000000000011011
00000000000000000000000000000000000000000000000000000000000011100
00000000000000000000000000000000000000000000000000000000000011101
00000000000000000000000000000000000000000000000000000000000011110
00000000000000000000000000000000000000000000000000000000000011111
```