Lab4: Beginning to Decode

Jon Johnston and Justin Roessler

February 11, 2019

1 Executive Summary

The purpose of this lab is to create an Instruction Parse module that will break up the 32-bit instruction data into the R and D format cases. This module prepares an instruction to be executed and saves each part of an instruction according to its purpose (Rn,Rm,Rd, Opcode, etc.). Also, a Register File module was created in order to read and write information to registers. The regfile is the core memory of our processor. The Register File will read data from a read_register location to send it out to the ALU. If RegWrite is set, it will write data from the ALU or memory to a write_register location. After testing our files and analyzing the results, our lab was sucessful.

2 Test Report

To verify operation of these modules, this lab requires 2 test benches.

1. Instr_Parse Test Bench
2. RegFile Test Bench

Figure 1: Expected Results of the Instr_Parse Test.

Time (ns)	0-10	10-20	20-30
instruction	4165927241	2332623530	4161732937
rm_num	15	9	15
rn_num	10	21	10
rd_num	9	10	9
address	240	144	240
opcode	1986	1112	1984

Figure 2: Timing diagram for the Instr_Parse test.

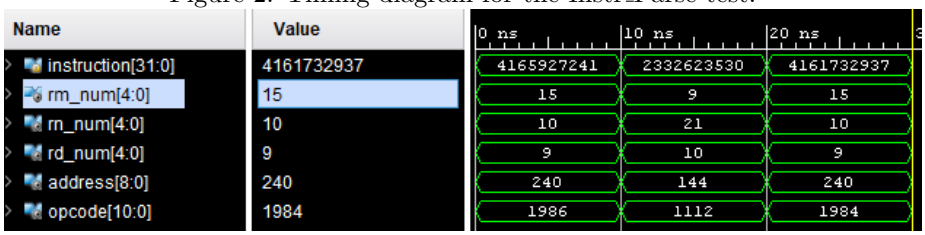
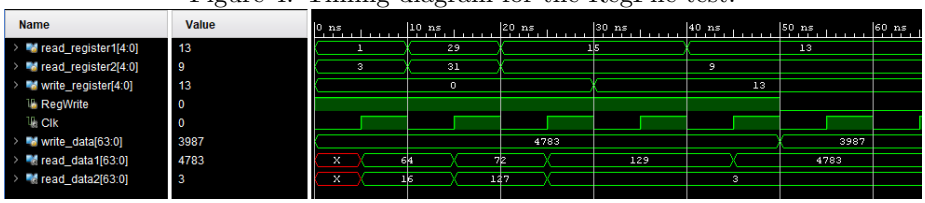


Figure 3: Expected Results for the RegFile test.

Time(ns)	0-5	5-10	10-15	15-20	20-25	25-30	30-35	35-40	40-45	45-50	50-55
read_register1	1	1	29	29	15	15	15	15	13	13	13
read_register2	3	3	31	31	9	9	9	9	9	9	9
write_register	0	0	0	0	0	0	13	13	13	13	13
RegWrite	1	1	1	1	1	1	1	1	1	1	20
Clk	0	1	0	1	0	1	0	1	0	1	0
wrtie_data	4783	4783	4783	4783	4783	4783	4783	4783	4783	3987	3987
read_data1	X	64	64	72	72	129	129	4783	4783	4783	4783
read_data2	X	16	16	127	127	3	3	3	3	3	3

Figure 4: Timing diagram for the RegFile test.



3 Code Appendix

Listing 1: Verilog code for testing the instr_parse_test.

```
`include "definitions.vh"

module instr_parse_test;

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

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

initial
begin
// LDUR X9, [X10, #240]
instruction = `INSTR_LEN'b11111000010011110000000101001001;
#(`CYCLE);

// ADD X10, X21, X9
instruction = `INSTR_LEN'b10001011000010010000001010101010;
#(`CYCLE);

// STUR X9, [X10, #240]
instruction = `INSTR_LEN'b11111000000011110000000101001001;
#(`CYCLE);

$finish;
end
endmodule
```

Listing 2: Verilog code for implementing the instr_parse.

```
`include "definitions.vh"

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

assign rd_num = instruction[4:0];
assign rn_num = instruction[9:5];
assign rm_num = instruction[20:16];
assign address = instruction[20:12];
assign opcode = instruction[31:21];

endmodule
```

Listing 3: Verilog code for testing the regfile_test.

```
`include "definitions.vh"

module regfile_test;

    reg [4:0] read_register1, read_register2, write_register;
    reg RegWrite;
    wire Clk;
    reg [`WORD-1:0] write_data;
    wire [`WORD-1:0] read_data1, read_data2;

    regfile RegFile(
        .read_register1(read_register1),
        .read_register2(read_register2),
        .write_register(write_register),
        .RegWrite(RegWrite),
        .Clk(Clk),
        .write_data(write_data),
        .read_data1(read_data1),
        .read_data2(read_data2)
        );

    oscillator ReadClock(
        .clk(Clk)
        );

    initial
    begin
```

```
        RegWrite = 1;
        write_register = 0;
        write_data = 4783;
        read_register1 = 1;
        read_register2 = 3; #10;
        read_register1 = 29;
        read_register2 = 31; #10;
        read_register1 = 15;
        read_register2 = 9; #10;

        write_register = 13; #10;
        read_register1 = 13; #10;

        RegWrite = 0;
        write_data = 3987; #10;

        write_register = 13; #10;
        read_register1 = 13; #10;

    end
endmodule
```

Listing 4: Verilog code for implementing the regfile.

```
`include "definitions.vh"

module regfile(
    input [4:0] read_register1, read_register2, write_register,
    input RegWrite, Clk,
    input [`WORD-1:0] write_data,
    output reg [`WORD-1:0] read_data1, read_data2);

    reg [`WORD-1:0] rmem [31:0];
    wire WriteClk;

    always @(posedge Clk)
    begin
        read_data1 <= rmem[read_register1];
        read_data2 <= rmem[read_register2];
    end

    delay WClk(.a(Clk), .a_delayed(WriteClk));

    always @(posedge WriteClk)
    begin
        if(RegWrite)
```

```
            rmem[write_register] <= write_data;
    end

    initial
        $readmemb('RMEMFILE, rmem);
endmodule
```