Introduction to Digital Logic Design Lab
EECS 31L

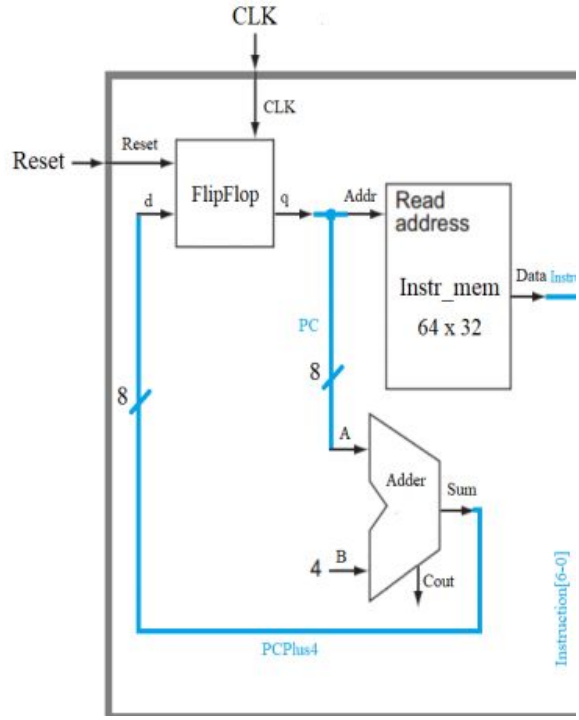
Lab 3

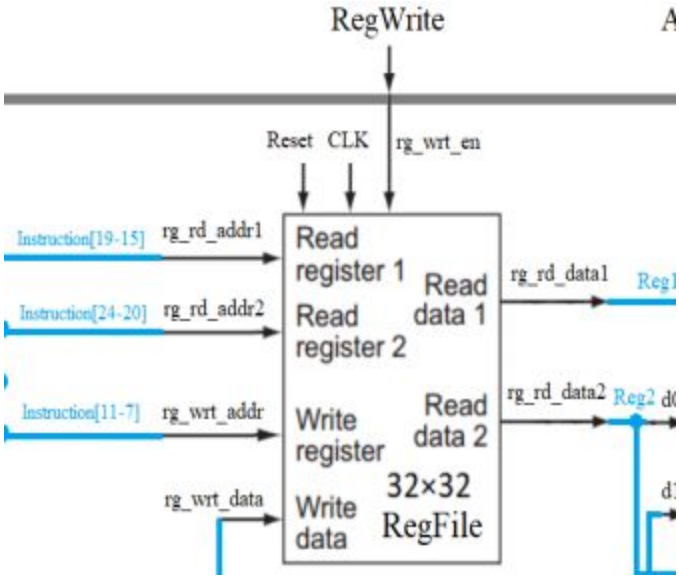
Kyle Zyler Cayanan
80576955

02/14/2021

1 - Objective

The objective of this lab was to design three different modules for a single cycle processor; Flip Flop, Instruction Memory, and Register File. The first module is designed to simply store an input, that input being a register that holds the address of the current instruction. Instruction Memory is designed to store all the instructions specified for this processor in an array made for 64 instructions consisting of 32 bits each. The Register File is designed to read and write to and from registers based on several inputs and outputs. Below are the block diagrams for each respective module.


CLK
CLK
Reset
Reset
FlipFlop
d
q
Addr
Read address
Instr_mem
64 x 32
Data Instr
PC
8
8
A
Adder
Sum
4
B
Cout
PCPlus4
Instruction[6-0]


RegWrite
Reset
CLK
rg_wrt_en
Instruction[19-15]
rg_rd_addr1
Read register 1
Read data 1
rg_rd_data1
Instruction[24-20]
rg_rd_addr2
Read register 2
rg_rd_data2
Instruction[11-7]
rg_wrt_addr
Write register
Read data 2
32×32 RegFile
rg_wrt_data
Write data

2 - Procedure

2.1 - Flip Flop

The Flip Flop runs on three different inputs and one output. The inputs are d (The address of the current instruction), Reset (Keeps the output at 0), and Clk (which runs on the positive edge and controls the timing of input and output). Simply put, the d input will be stored in the Flip Flop so long as reset is NOT 1 and the clock is on a positive edge. The code is shown below.

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
21
22
23 module FlipFlop(clk, reset, d, q);
24
25 input clk;
26 input reset;
27 input [7:0]d;
28 output reg [7:0]q;
29
30 always @(posedge clk)
31     begin
32         if (reset == 1)
33             q <= 0;
34         else
35             q <= d;
36     end
37 endmodule
```

2.2 - Instruction Memory

The Instruction Memory is made up of a 64x32 2d array. Each index in the array holds a hard coded instruction provided by the lab manual. The output is the instruction specified by the input from the flip flop. For now we have only been provided with 18 instructions. The code is shown below. The input addr will specify which instruction is needed for the current cycle.

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////...
10
11
12 module Instmem(
13     input  [7:0] addr,
14     output [31:0] instruction
15     );
16
17 reg [31:0] Instruction_memory[63:0];
18 initial
19  begin
20     Instruction_memory[0] = 32'h00007033;  // and r0, r0, r0 32'h00000000
21     Instruction_memory[1] = 32'h00100093;  // addi r1, r0, 1 32'h00000001
22     Instruction_memory[2] = 32'h00200113;  // addi r2, r0, 2 32'h00000002
23     Instruction_memory[3] = 32'h00308193;  // addi r3, r1, 3 32'h00000004
24     Instruction_memory[4] = 32'h00408213;  // addi r4, r1, 4 32'h00000005
25     Instruction_memory[5] = 32'h00510293;  // addi r5, r2, 5 32'h00000007
26     Instruction_memory[6] = 32'h00610313;  // addi r6, r2, 6 32'h00000008
27     Instruction_memory[7] = 32'h00718393;  // addi r7, r3, 7 32'h0000000B
28     Instruction_memory[8] = 32'h00208433;  // add r8, r1, r2 32'h00000003
29     Instruction_memory[9] = 32'h404404b3;  // sub r9, r8, r4 32'hfffffffe
30     Instruction_memory[10] = 32'h00317533; // and r10, r2, r3 32'h00000000
31     Instruction_memory[11] = 32'h0041e5b3; // or r11, r3, r4 32'h00000005
32     Instruction_memory[12] = 32'h0041a633; // if r3 is less than r4 then r12 = 1 32'h00000001
33     Instruction_memory[13] = 32'h007346b3; // nor r13, r6, r7 32'hfffffff4
34     Instruction_memory[14] = 32'h4d34f713; // andi r14, r9, "4D3" 32'h000004D2
35     Instruction_memory[15] = 32'h8d35e793; // ori r15, r11, "8d3" 32'hfffff8d7
36     Instruction_memory[16] = 32'h4d26a813; // if r13 is less than 32'h000004D2 then r16 = 1 32'h00000000
37     Instruction_memory[17] = 32'h4d244893; // nori r17, r8, "4D2" 32'hfffffb2C
38  end
39
```

2.3 - Register File

The Register File is a much more complicated module. This module has 7 inputs and 2 outputs. The inputs are reset, clk, register write enable (allows input to be written to specified register address), register address 1, register address 2, register write address (specifies the address of the register we are writing to), register write data (data to be written into register write address). The output of the register file is the data from register address 1 and address 2 which will be sent to an ALU to handle the operation. The code is shown below

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
21
22
23 module RegFile(
24     clk,
25     reset,
26     rg_wrt_en,
27     rg_wrt_addr,
28     rg_rd_addr1,
29     rg_rd_addr2,
30     rg_wrt_data,
31     rg_rd_data1,
32     rg_rd_data2
33     );
34
35 //I/O ports
36 parameter N = 4;
37 input clk, reset, rg_wrt_en;
38 input  [N:0] rg_rd_addr1;
39 input  [N:0] rg_rd_addr2;
40 input  [N:0] rg_wrt_addr;
41 input  [31:0] rg_wrt_data;
42 output [31:0] rg_rd_data1, rg_rd_data2;
43 reg    [31:0] register_file [31:0];
44
45
46 //Defining behavior
47 integer i;
48 //async reset
49 always @(reset)
50  begin
```

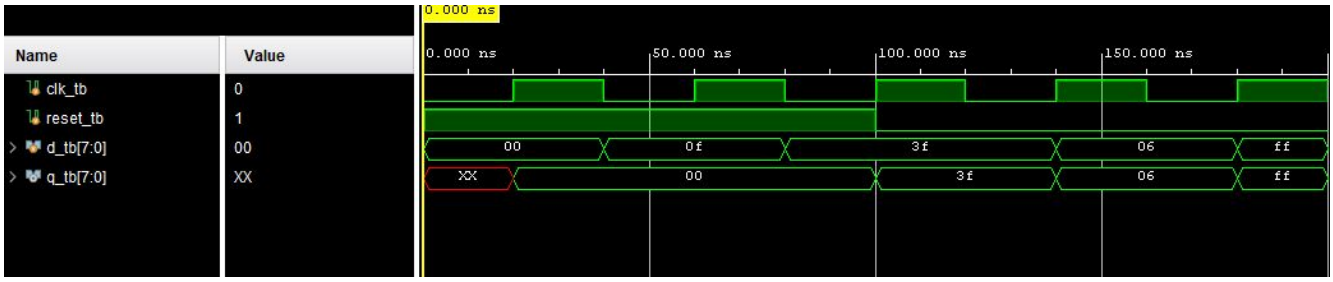
```
input  [31:0] rg_wrt_data;
output [31:0] rg_rd_data1, rg_rd_data2;
reg    [31:0] register_file [31:0];


//Defining behavior
integer i;
//async reset
always @(reset)
 begin
    if (reset == 1)
     begin
        for (i = 0; i < 32; i = i + 1)
        begin
            register_file[i] = 32'h00000000;
        end
     end
 end

 //read data
assign rg_rd_data1 = register_file[rg_rd_addr1];
assign rg_rd_data2 = register_file[rg_rd_addr2];

always @(posedge clk)
 begin
    if (reset == 0 && rg_wrt_en == 1)
     begin
        register_file[rg_wrt_addr] = rg_wrt_data;
     end
 end
endmodule

```

3 - Simulation Results

3.1 - Flip Flop Results

The expected results was that the q output would reflect the d input one clock cycle later as shown by the waveform below. The values used are shown by the test bench below as well.

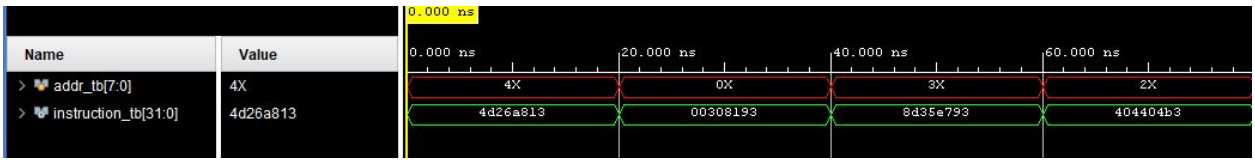
```
reg reset_tb;
reg [7:0]d_tb;
wire [7:0]q_tb;

FlipFlop instant
(
.clk(clk_tb),
.reset(reset_tb),
.d(d_tb),
.q(q_tb)
);

always #20 clk_tb = ~clk_tb;
initial
    begin
        clk_tb = 1'b0;
        reset_tb = 1'b1;

        #0   d_tb = 8'b00000000;
        #40  d_tb = 8'b00001111;
        #40  d_tb = 8'b00111111;

        #20 reset_tb = 1'b0;


        #40 d_tb = 8'b00000110;
        #40 d_tb = 8'b11111111;
        #20
        $finish;
    end
endmodule

```



3.2 - Instruction Memory Results

The expected results were to have the output display an instruction stored in the 2d array. Each index (aka the address bits [7:2]) would store an instruction. In our test bench we checked addresses 16, 3, 15, and 9 in decimal. Each test case reflected their rightful instruction. The test bench and waveform are shown below.

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////
21
22
23 module tb_Instmem();
24 reg [7:0]addr_tb;
25 wire [31:0]instruction_tb;
26
27 Instmem instant(
28     .addr(addr_tb),
29     .instruction(instruction_tb)
30     );
31
32 initial
33  begin
34     addr_tb[7:2] = 'd16;
35     #20
36     addr_tb[7:2] = 'd3;
37     #20
38     addr_tb[7:2] = 'd15;
39     #20
40     addr_tb[7:2] = 'd9;
41     #20
42     $finish;
43   end
44 endmodule
45
46
47
48
49
50
```



3.3 - Register File Results

The expected results of the register file was to have an address in the register file array to contain some data written from the register write data input. For this to happen, Reset had to 0, register write enable had to be set to 1, and clock had to appear on a positive edge. A test bench was not written to test this module, instead the features “Force Constant” and “Force Clock” were used. Here in our waveform below we had written the value to ff20 to register 18 in the processor.