CECS 341 – LAB 6

MIPS Instruction Fetch Stage

15 April 2020

Rifa Safeer Shah

017138353

I certify that this submission is my original work

[illegible]

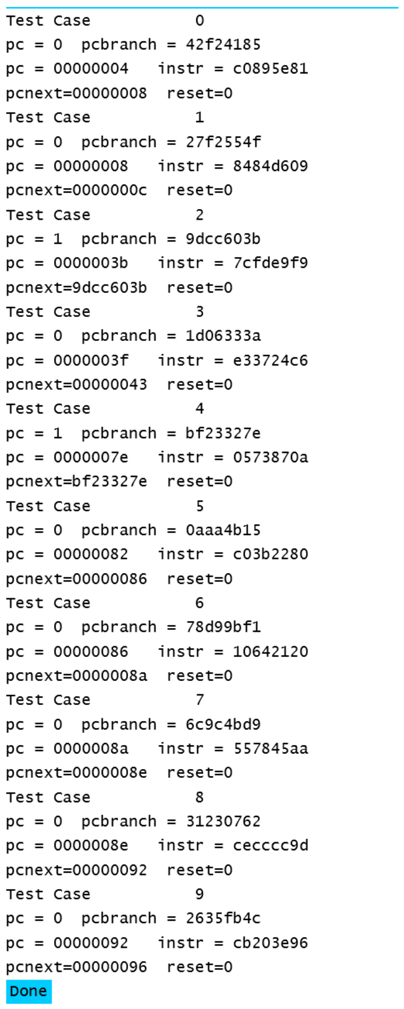
Lab Report: Lab Assignment 6 – MIPS Instruction Fetch Stage

1. Goal: The goal of the MIPS Instruction Fetch Stage Lab Assignment is to implement one of the five stages of execution in a single cycle MIPS processor.

2. Steps:

   a. Go to edaplayground.com and log in to your account.
   b. Rewrite the outline provided with this lab.
   c. Identify what is to be done to produce specified output.
   d. Use mux2.v from the previous lab.
   e. Create instruction memory in file imem.v.
   f. Create 32-bit adder in file adder32.v.
   g. Create register in the file flopr.v from lab instructions.
   h. Check for errors
   i. Run the code for test cases.

3. Results: In this lab assignment we tested for one of the five execution stages in the single cycle MIPS processor. The output displayed 9 different cases which was the same as expected. The cases highlighted the different inputs that dealt with the program counter and the instructions.

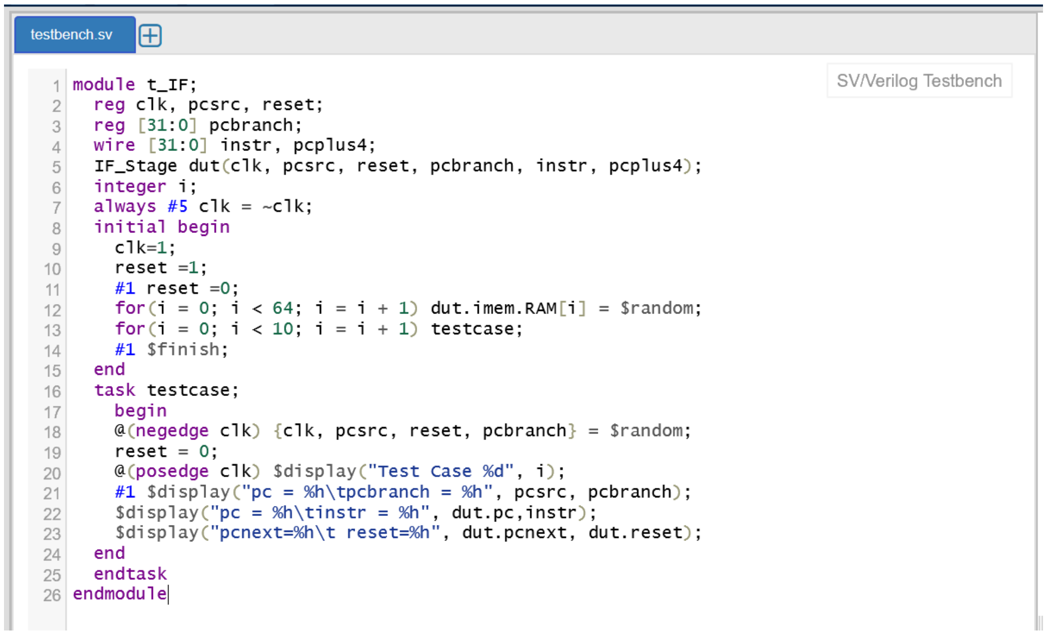


```
Test Case           0
pc = 0  pcbranch = 42f24185
pc = 00000004   instr = c0895e81
pcnext=00000008  reset=0
Test Case           1
pc = 0  pcbranch = 27f2554f
pc = 00000008   instr = 8484d609
pcnext=0000000c  reset=0
Test Case           2
pc = 1  pcbranch = 9dcc603b
pc = 0000003b   instr = 7cfde9f9
pcnext=9dcc603b  reset=0
Test Case           3
pc = 0  pcbranch = 1d06333a
pc = 0000003f   instr = e33724c6
pcnext=00000043  reset=0
Test Case           4
pc = 1  pcbranch = bf23327e
pc = 0000007e   instr = 0573870a
pcnext=bf23327e  reset=0
Test Case           5
pc = 0  pcbranch = 0aaa4b15
pc = 00000082   instr = c03b2280
pcnext=00000086  reset=0
Test Case           6
pc = 0  pcbranch = 78d99bf1
pc = 00000086   instr = 10642120
pcnext=0000008a  reset=0
Test Case           7
pc = 0  pcbranch = 6c9c4bd9
pc = 0000008a   instr = 557845aa
pcnext=0000008e  reset=0
Test Case           8
pc = 0  pcbranch = 31230762
pc = 0000008e   instr = ceccc9d
pcnext=00000092  reset=0
Test Case           9
pc = 0  pcbranch = 2635fb4c
pc = 00000092   instr = cb203e96
pcnext=00000096  reset=0
Done
```

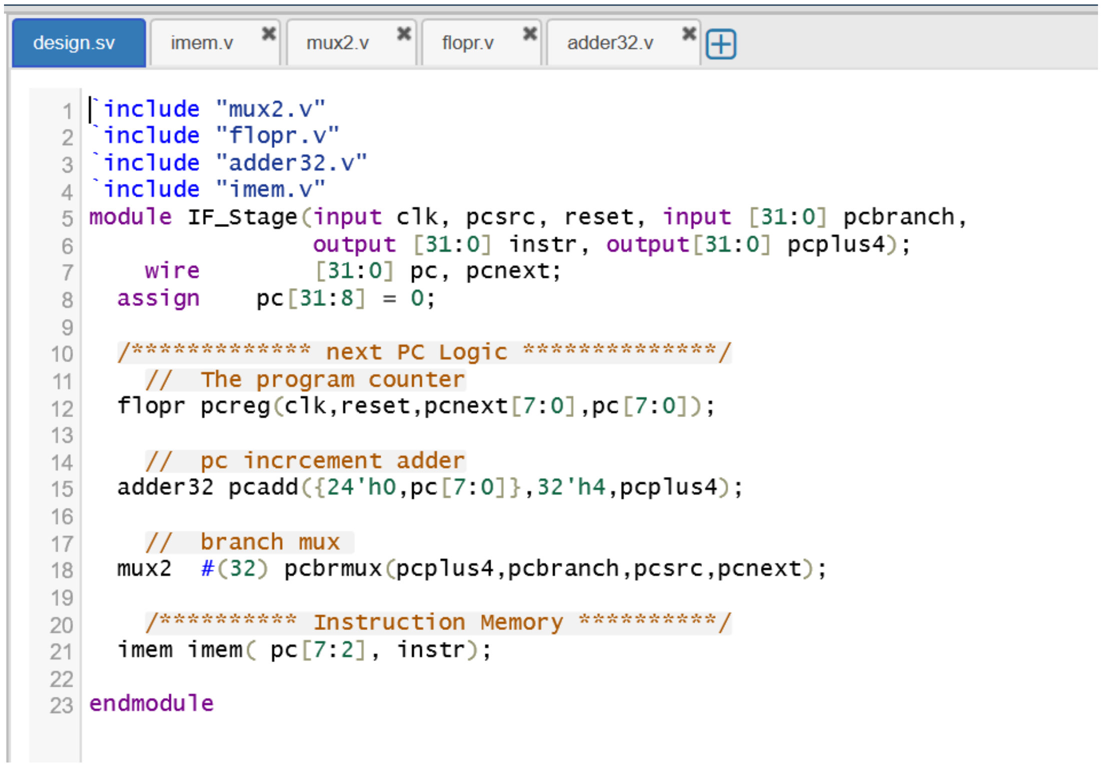
4. Conclusion: In conclusion of this lab, I learned the inner process of the implementation of the Instruction Fetch stage of the single cycle MIPS processor. The challenge in this lab was to understand each item of the output and how we achieved it.

5. Notes: The Instruction Fetch access stage is where an instruction's machine code is fetched from IMEM and the program counter value is incremented. Then program counter is updated according to next PC logic. pcsrc controls how the mux is loaded. If pcsrc == 0 then pc is incremented as usual. If pcsrc == 1 then a branch is taken, and pc is loaded with the branch target address. instr is the 32-bit machine code for a particular instruction that was loaded from IMEM.

6. Lab Screenshots:



```
module t_IF;
  reg clk, pcsrc, reset;
  reg [31:0] pcbranch;
  wire [31:0] instr, pcplus4;
  IF_Stage dut(clk, pcsrc, reset, pcbranch, instr, pcplus4);
  integer i;
  always #5 clk = ~clk;
  initial begin
    clk=1;
    reset =1;
    #1 reset =0;
    for(i = 0; i < 64; i = i + 1) dut.imem.RAM[i] = $random;
    for(i = 0; i < 10; i = i + 1) testcase;
    #1 $finish;
  end
  task testcase;
    begin
    @(negedge clk) {clk, pcsrc, reset, pcbranch} = $random;
    reset = 0;
    @(posedge clk) $display("Test Case %d", i);
    #1 $display("pc = %h\tpcbranch = %h", pcsrc, pcbranch);
    $display("pc = %h\tinstr = %h", dut.pc,instr);
    $display("pcnext=%h\t reset=%h", dut.pcnext, dut.reset);
  end
  endtask
endmodule
```



```
`include "mux2.v"
`include "flopr.v"
`include "adder32.v"
`include "imem.v"
module IF_Stage(input clk, pcsrc, reset, input [31:0] pcbranch,
                output [31:0] instr, output[31:0] pcplus4);
    wire        [31:0] pc, pcnext;
  assign     pc[31:8] = 0;

  /************* next PC Logic **************/
    //  The program counter
  flopr pcreg(clk,reset,pcnext[7:0],pc[7:0]);

    //  pc incrcement adder
  adder32 pcadd({24'h0,pc[7:0]},32'h4,pcplus4);

    //  branch mux
  mux2  #(32) pcbrmux(pcplus4,pcbranch,pcsrc,pcnext);

    /********** Instruction Memory **********/
  imem imem( pc[7:2], instr);

endmodule
```