LAB 6
MIPS Instruction Fetch Stage

In the single cycle MIPS processor, there are 5 stages of execution. The Instruction Fetch access stage is where the 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:

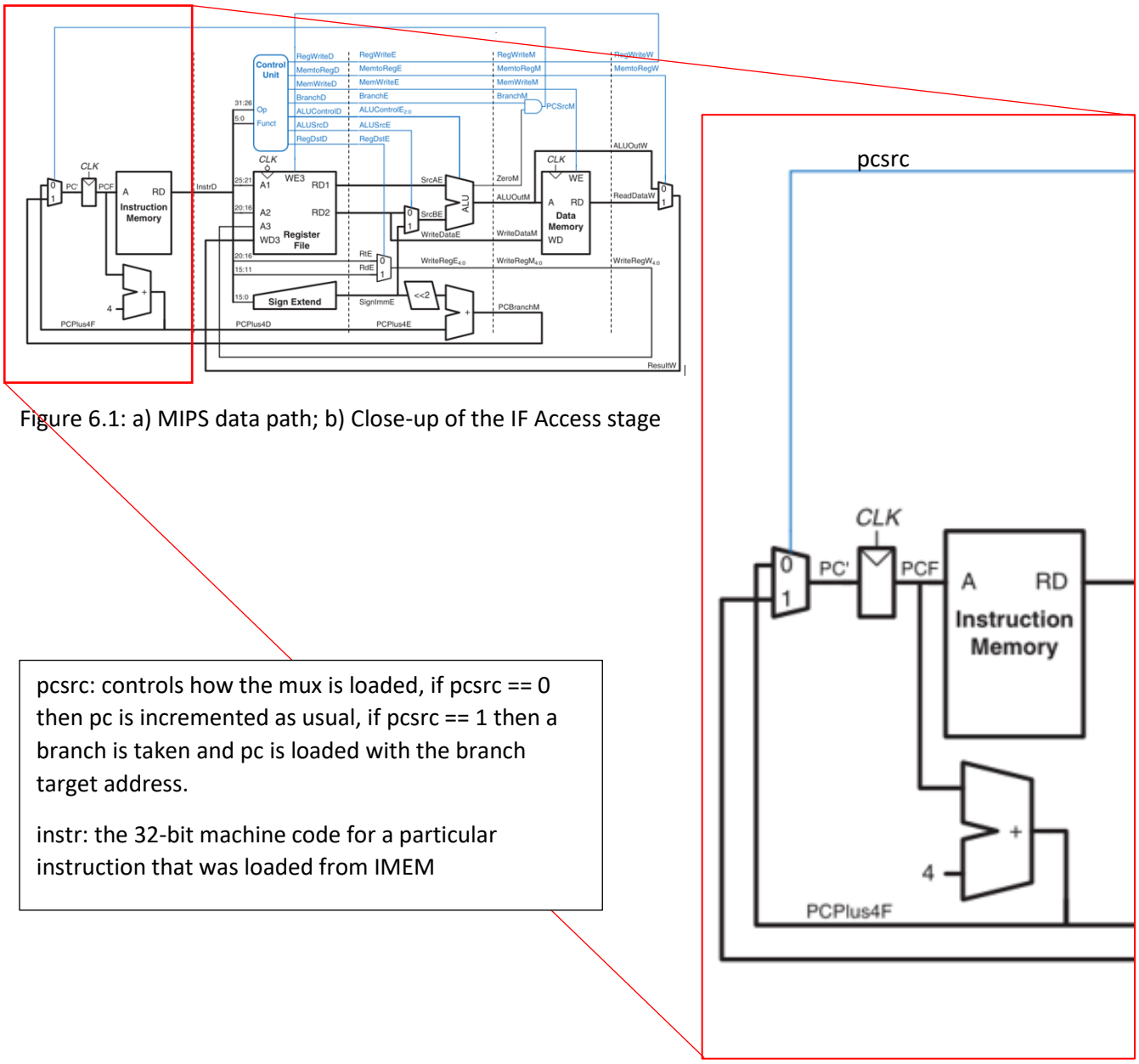

Figure 6.1: a) MIPS data path; b) Close-up of the IF Access stage

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: the 32-bit machine code for a particular instruction that was loaded from IMEM

Get the mux2.v file from a previous lab.

A configurable size register is needed. Use **flopr.v** with Verilog module definition given below:

```
module flopr #(parameter WIDTH = 8)
  (input clk, reset, input  [WIDTH-1:0] d, output reg   [WIDTH-1:0] q);
    always @(posedge clk, posedge reset)
        if( reset ) q = 0;
        else        q = d;
endmodule
```

Figure 6.2: Register module

Create a 32-bit adder in a file named **adder32.v** which is defined below:

```
module adder32(input [31:0] a,b, output [31:0]  y);
    assign y = a + b;
endmodule
```

Figure 6.3: 32-bit adder module

Create Instruction Memory in a file named **imem.v** and name the module **imem**:

```
module imem(input [5:0] a, output [31:0] rd);
  reg       [31:0] RAM[63:0];
  assign rd=RAM[a]; // word aligned
endmodule
```

Figure 6.4: Instruction memory module

Use the block diagram on page 1 and complete the module skeleton below to create the IF stage:

```
`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(    ,     ,pcnext[7:0],pc[7:0]);

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

    // branch mux
  mux2  #(32) pcbrmux(                          );


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

endmodule
```

Figure 6.5: Skeleton of the code of IF Stage module

Test your design to ensure it works correctly using the Verilog test fixture code below:

testbench.sv

```
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; // this was added

      @(posedge clk) $display("Test Case %d", i);
      #1 $display("pcsrc = %b\tpcbranch = %h", pcsrc, pcbranch);
      $display("pc = %h\tinstr = %h", dut.pc, instr);
      $display("pcnext=%h\t reset=%h", dut.pcnext, dut.reset); // this was added too
    end
  endtask

endmodule
```

Figure 6.6:Testbench for the IF_Stage module – updated on 03/26/2019 @ 11:20 am

WHAT TO SUBMIT

Once you have verified proper functionality of your project, copy the contents of **IF_Stage module (design.sv)** to a text file named **IF_Stage.txt** and upload the BeachBoard Dropbox for Lab 6. Additionally, upload your Lab report (see the LabReportTemplete in the Documents folder on BeachBoard).

NOTE: **keep** these lab files as they will be needed for future labs!

Created by Josh Hayter, updated by Jelena Trajkovic and debugged by one of Jelena's students

Correct test results are shown below:

```
Test Case           0
pcsrc = 0       pcbranch = 42f24185
pc = 00000004   instr = c0895e81
pcnext=00000008  reset=0
Test Case           1
pcsrc = 0       pcbranch = 27f2554f
pc = 00000008   instr = 8484d609
pcnext=0000000c  reset=0
Test Case           2
pcsrc = 1       pcbranch = 9dcc603b
pc = 0000003b   instr = 7cfde9f9
pcnext=9dcc603b  reset=0
Test Case           3
pcsrc = 0       pcbranch = 1d06333a
pc = 0000003f   instr = e33724c6
pcnext=00000043  reset=0
Test Case           4
pcsrc = 1       pcbranch = bf23327e
pc = 0000007e   instr = 0573870a
pcnext=bf23327e  reset=0
Test Case           5
pcsrc = 0       pcbranch = 0aaa4b15
pc = 00000082   instr = c03b2280
pcnext=00000086  reset=0
Test Case           6
pcsrc = 0       pcbranch = 78d99bf1
pc = 00000086   instr = 10642120
pcnext=0000008a  reset=0
Test Case           7
pcsrc = 0       pcbranch = 6c9c4bd9
pc = 0000008a   instr = 557845aa
pcnext=0000008e  reset=0
Test Case           8
pcsrc = 0       pcbranch = 31230762
pc = 0000008e   instr = cecccc9d
pcnext=00000092  reset=0
Test Case           9
pcsrc = 0       pcbranch = 2635fb4c
pc = 00000092   instr = cb203e96
pcnext=00000096  reset=0
```

Figure 6.6: Expected output for the IF_Stage module – updated on 03/26/2019 @ 11:20 am