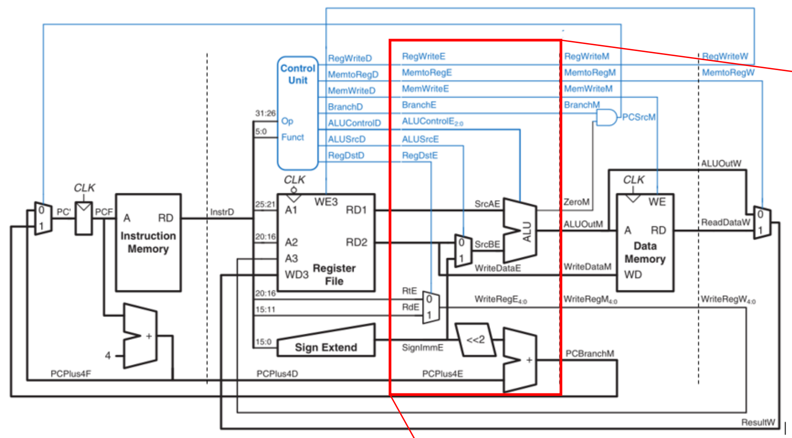
LAB 7
MIPS Execute Stage

In the single cycle MIPS processor, there are 5 stages of execution. The Execute stage primarily involves the ALU which is used to calculate the result of R-Type instructions, performs subtraction to check equality of operands of BEQ, and calculates address for load/store operations:



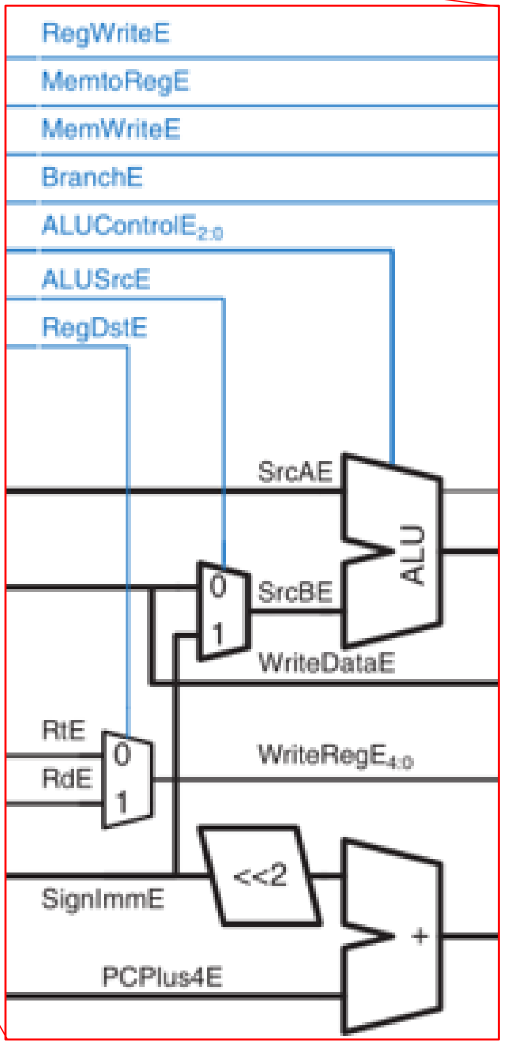


Figure 7.1: a) MIPS data path; b) Close-up of the EXE stage

RegWriteE, MemtoRegE, MemWriteE, and BrancheE signals are passed along but do not control any hardware in the EX stage.

ALUControlE: makes the ALU perform a specific operation

ALUSrcE: when equal to 0 the ALU input B receives output of register File, when equal to 1 the ALU input B receives immediate data

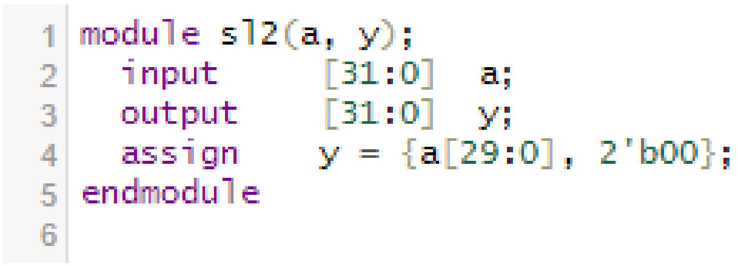
RegDstE: When equal to 0 then RT will choose the register to be written for the current isntruction’s writeback stage, When equal to 1 then RD will choose the destination register (for R-Type instructions)

The adder is sued to calculate the branch target address (only used in the beq instruction)

Get the mux2.v file from a previous lab.

Get the 32-bit adder from a previous lab.

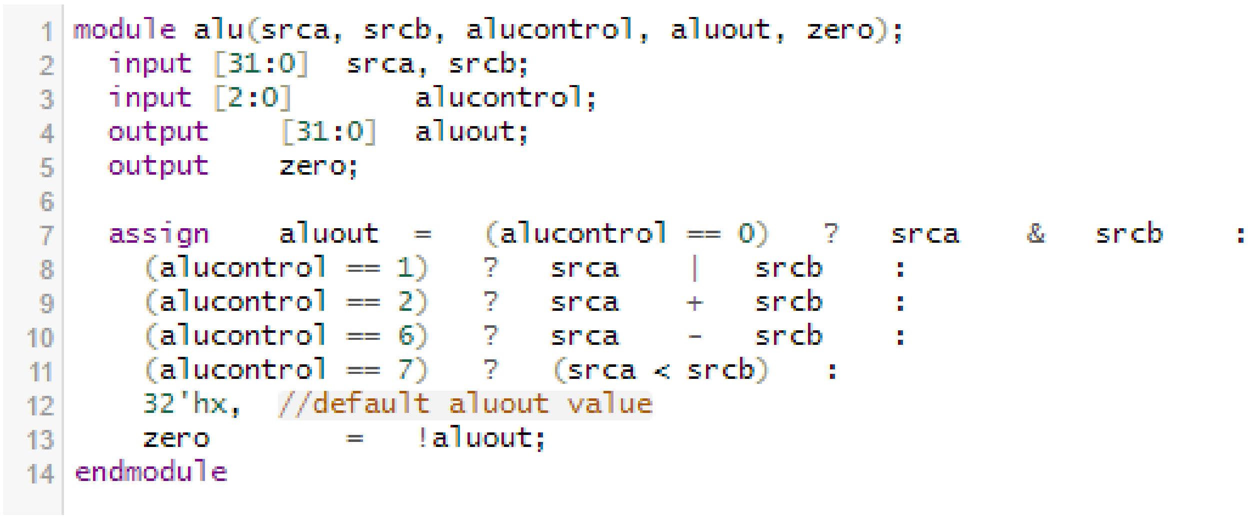
A left shifter is needed. Use **sl2.v** with Verilog module definition given below:



```
module sl2(a, y);
  input      [31:0]   a;
  output     [31:0]   y;
  assign     y = {a[29:0], 2'b00};
endmodule

```

Figure 7.2: Shift-left module

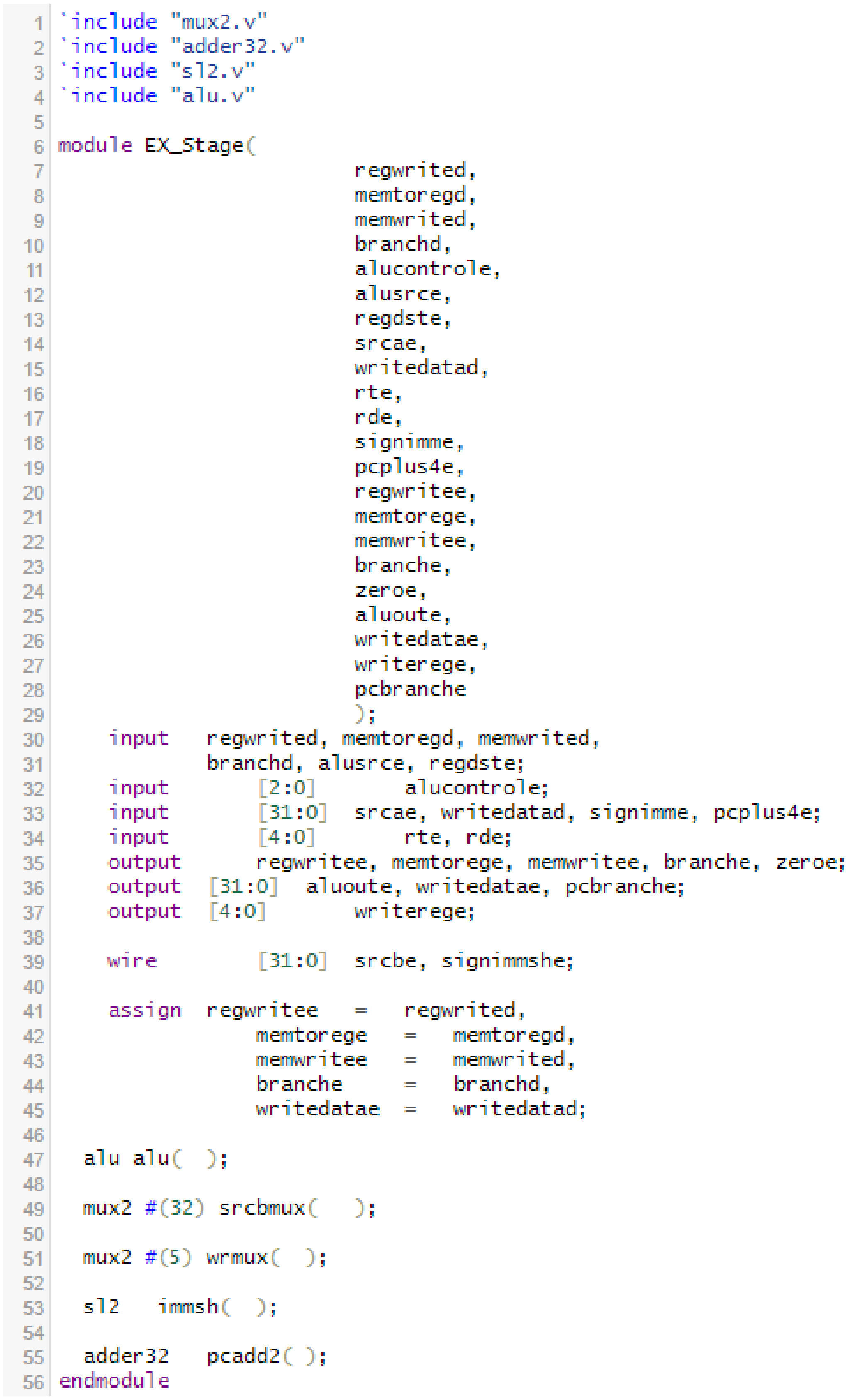
Create Arithmetic Logic Unit in a file named **alu.v** and name the module **alu**:



```
module alu(srca, srcb, alucontrol, aluout, zero);
  input [31:0]  srca, srcb;
  input [2:0]       alucontrol;
  output     [31:0]  aluout;
  output     zero;

  assign     aluout  =   (alucontrol == 0)   ?   srca    &   srcb    :
    (alucontrol == 1)   ?   srca    |   srcb    :
    (alucontrol == 2)   ?   srca    +   srcb    :
    (alucontrol == 6)   ?   srca    -   srcb    :
    (alucontrol == 7)   ?   (srca < srcb)   :
    32'hx,  //default aluout value
    zero         =    !aluout;
endmodule
```

Figure 7.3: ALU module

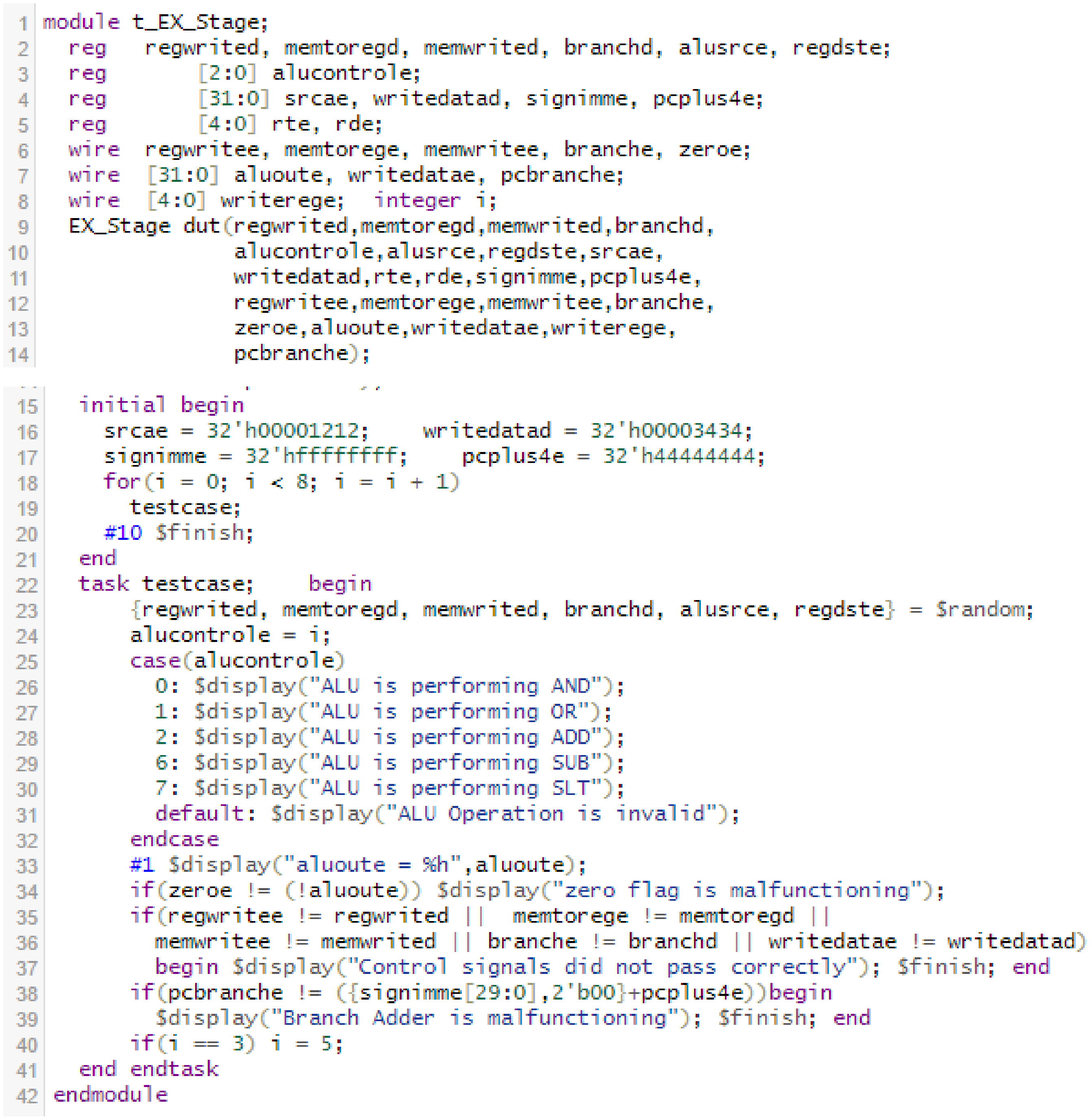
Use the block diagram on page 1 and complete the module skeleton in Figure 7.4 to create the IF stage:



```verilog
`include "mux2.v"
`include "adder32.v"
`include "sl2.v"
`include "alu.v"

module EX_Stage(
                        regwrited,
                        memtoregd,
                        memwrited,
                        branchd,
                        alucontrole,
                        alusrce,
                        regdste,
                        srcae,
                        writedatad,
                        rte,
                        rde,
                        signimme,
                        pcplus4e,
                        regwritee,
                        memtorege,
                        memwritee,
                        branche,
                        zeroe,
                        aluoute,
                        writedatae,
                        writerege,
                        pcbranche
                        );
    input   regwrited, memtoregd, memwrited,
            branchd, alusrce, regdste;
    input       [2:0]       alucontrole;
    input       [31:0]  srcae, writedatad, signimme, pcplus4e;
    input       [4:0]       rte, rde;
    output      regwritee, memtorege, memwritee, branche, zeroe;
    output  [31:0]  aluoute, writedatae, pcbranche;
    output  [4:0]       writerege;

    wire        [31:0]  srcbe, signimmshe;

    assign  regwritee   =   regwrited,
                memtorege   =   memtoregd,
                memwritee   =   memwrited,
                branche     =   branchd,
                writedatae  =   writedatad;

  alu alu(  );

  mux2 #(32) srcbmux(   );

  mux2 #(5) wrmux(  );

  sl2   immsh(  );

  adder32   pcadd2( );
endmodule
```

Figure 7.4: Skeleton of the EXE_Stage module

Test your design to ensure it works correctly using the Verilog test fixture shown in the Figure 7.5.The expected output is shown in Figure 7.6.



```
module t_EX_Stage;
  reg   regwrited, memtoregd, memwrited, branchd, alusrce, regdste;
  reg       [2:0] alucontrole;
  reg       [31:0] srcae, writedatad, signimme, pcplus4e;
  reg       [4:0] rte, rde;
  wire  regwritee, memtorege, memwritee, branche, zeroe;
  wire  [31:0] aluoute, writedatae, pcbranche;
  wire  [4:0] writerege;  integer i;
  EX_Stage dut(regwrited,memtoregd,memwrited,branchd,
               alucontrole,alusrce,regdste,srcae,
               writedatad,rte,rde,signimme,pcplus4e,
               regwritee,memtorege,memwritee,branche,
               zeroe,aluoute,writedatae,writerege,
               pcbranche);
  initial begin
    srcae = 32'h00001212;    writedatad = 32'h00003434;
    signimme = 32'hffffffff;    pcplus4e = 32'h44444444;
    for(i = 0; i < 8; i = i + 1)
      testcase;
    #10 $finish;
  end
  task testcase;    begin
      {regwrited, memtoregd, memwrited, branchd, alusrce, regdste} = $random;
      alucontrole = i;
      case(alucontrole)
        0: $display("ALU is performing AND");
        1: $display("ALU is performing OR");
        2: $display("ALU is performing ADD");
        6: $display("ALU is performing SUB");
        7: $display("ALU is performing SLT");
        default: $display("ALU Operation is invalid");
      endcase
      #1 $display("aluoute = %h",aluoute);
      if(zeroe != (!aluoute)) $display("zero flag is malfunctioning");
      if(regwritee != regwrited ||  memtorege != memtoregd ||
        memwritee != memwrited || branche != branchd || writedatae != writedatad)
        begin $display("Control signals did not pass correctly"); $finish; end
      if(pcbranche != ({signimme[29:0],2'b00}+pcplus4e))begin
        $display("Branch Adder is malfunctioning"); $finish; end
      if(i == 3) i = 5;
  end endtask
endmodule
```

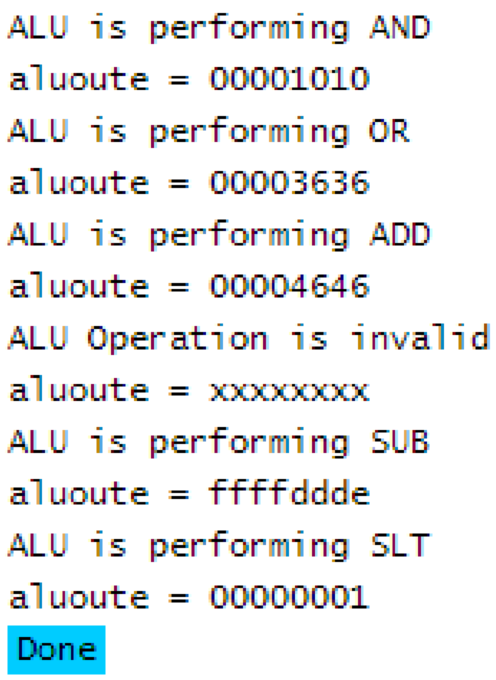
Figure 7.5:Testbench for the EXE_Stage module

WHAT TO SUBMIT

Once you have verified proper functionality of your project, copy the contents of **EXE_Stage module (design.sv)** to a text file named **EXE_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



```
ALU is performing AND
aluoute = 00001010
ALU is performing OR
aluoute = 00003636
ALU is performing ADD
aluoute = 00004646
ALU Operation is invalid
aluoute = xxxxxxxx
ALU is performing SUB
aluoute = ffffddde
ALU is performing SLT
aluoute = 00000001
Done
```

Figure 7.6: Expected output for the Lab 7