CECS 530 - Lab 9

"Making Hazard Detection and Forwarding units"

Due date: December 17, 2020

Student Name: Julie Kim

[redacted]

I certify that this submission is my original work

Julie Kim—December 17, 2020

<u>Your signature or electronic signature</u>

Lab Report: Lab Assignment 9- "Data Hazard"

1. **Goal:**
   To build a pipeline process with the ability to detect data hazard caused by source operand of the proceeding instruction need the data from the destination register of the preceding instruction. The instruction as ADD X2, X31, X1 produce the result of X2, which is used by three proceeding instructions as SUB X5, X4, X2 ; AND X8,X6,X2 and OR X12,X9,X2. The first two instruction are data hazard as X2 is a source operand available at the WB stage 5. The content of X2 need to be forwarded from EX/MEM pipeline3 as ALU input, and from MEM/WB pipeline4 as ALU input respectively. The instruction need X2 at stage5 WB, which can be forwarded directly from the register file where the first half write back X2 into register and the second half is to consume the X2 by instruction OR X12,X9,X2. This is called half cycle technique. We are going to build Hazard Detector in stage2, ID-stage, to develop three HA signals as HARn2Rd4, HARn2Rt4 and HARn2Rd3. HARn2Rt4 is hazard signal when produced by instruction ADD X2,X31,X1 and SUB X5,X4,X2. The last two signal produced by instruction ADD X2,X31,X1 and AND X8,X6,X2. The choice between the two signals is whether the X2 is the content of memory read or the result from ALU calculation.

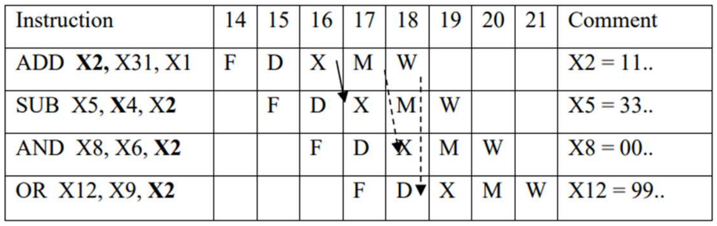
2. **Steps:**
   - Build pipeline registers, IF/ID-pipe, ID/EX-pipe, EX/MEM-pipe and MEM/WB-pipe
   - Hazard Detection units. The unit generate for signal 100, 001, 010 and 011. The first signal produced by last instruction OR X12,X9,X2, which can be resolved by half cycle technique. 011 signal is produced by instruction SUB X5,X4,X2 which can be resolved by forwarding X2 result from EX/MEM pipeline in stage-4, MEM-stage to ALU input. 001 and 010 is produced by AND X8,X6,X2. In this case the signal

hazard will be 001 because X2 is forwarded from ALU result instead of memory read.

- Build another unit, the Forwarding unit, which take in Hazard produced by previous Hazard Detector stored in ID/EX pipeline. Forwarding unit will take signals as 100, 001, 010 and 011 to decide the inputs to the ALU mux whether inputs are from EX/MEM pipeline for 011-signal, from MEM/WB pipeline for 001 (ALU result)-signal, from MEM/WB pipeline for 010 (memory read), and 100-signal register files half-cycle technique.

3. **Results:**

There are two parts of the results signals from the Hazard detection unit and from the Forwarding unit:



Instruction	14	15	16	17	18	19	20	21	Comment
ADD **X2,** X31, X1	F	D	X	M	W				X2 = 11..
SUB X5, **X**4, X**2**		F	D	X	M	W			X5 = 33..
AND X8, X6, **X2**			F	D	X	M	W		X8 = 00..
OR X12, X9, **X2**				F	D	X	M	W	X12 = 99..

a. The Hazard Detection unit produce 3 signals to indication three type of hazards of the instruction SUB, AND and OR proceeding instruction ADD X2,X31,X1 whose X2 is the destination register producing result 64'h111….at stage5-WB-stage. At time 300ns, the three instruction with the two operands register as: SUB(X4,X2), AND(X6,X2) and OR(X9,X2) respectively. The destination register X2 indicate as blue color as the same address. SUB(sub x5,x4,x2) instruction produce signal 3'b011, AND(and x8,x6,x2) instruction produce signal hazard 3'b010, and OR(or x12,x9,x2) instruction produces signal hazard 3'b100.

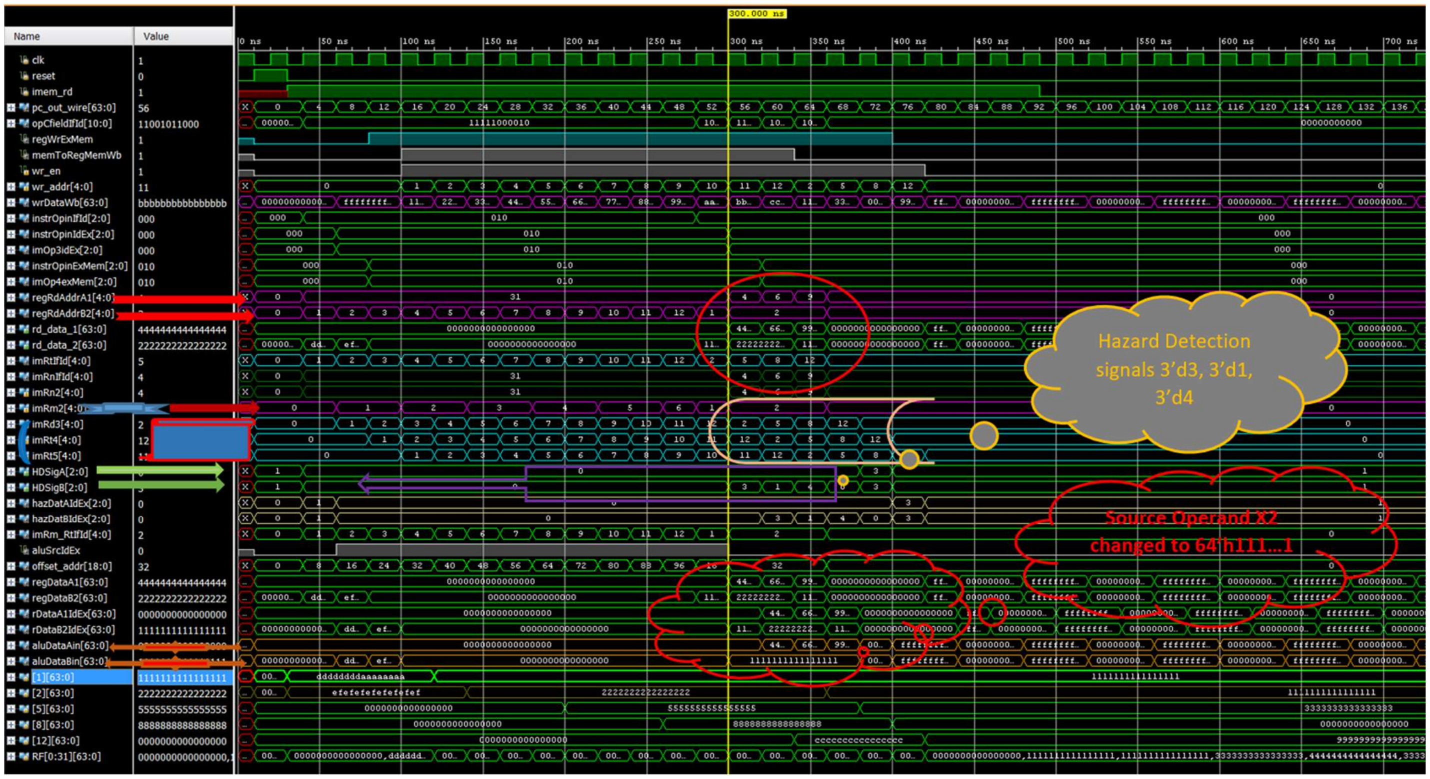
b. The Forwarding unit takes the those three signals and decide upon the inputs to the ALU unit. 3'd3 for alu result from EX/MEM, 3'd1 for alu result from MEM/WB and 3'd4 from register file as half cycle technique. By comparing the register read results (enclosing by the read cloud circle) regDataA1 and regDataB2 as raw data read from register file for X4, X6, X9, and X2 as 64'h44…, 64'h66…, 64'h99… and 64'h22… and 64'h11… respectively. The alu inputs, aluDataAin and aluDataBin as source operand of X2 are all 64'h11…. This is because of the Hazard Detection unit changes the ALU input options.

4. **Conclusion:**

From this lab I learn how to construct Hazard Detection unit and Forwarding unit. The signals are passed from one pipeline to another are in timely manner at the edge of the clock edge. The data and signals are passed from one pipeline to another is at the edge of clock edge. The challenge in this lab is make sure to get the right data as input to the ALU.

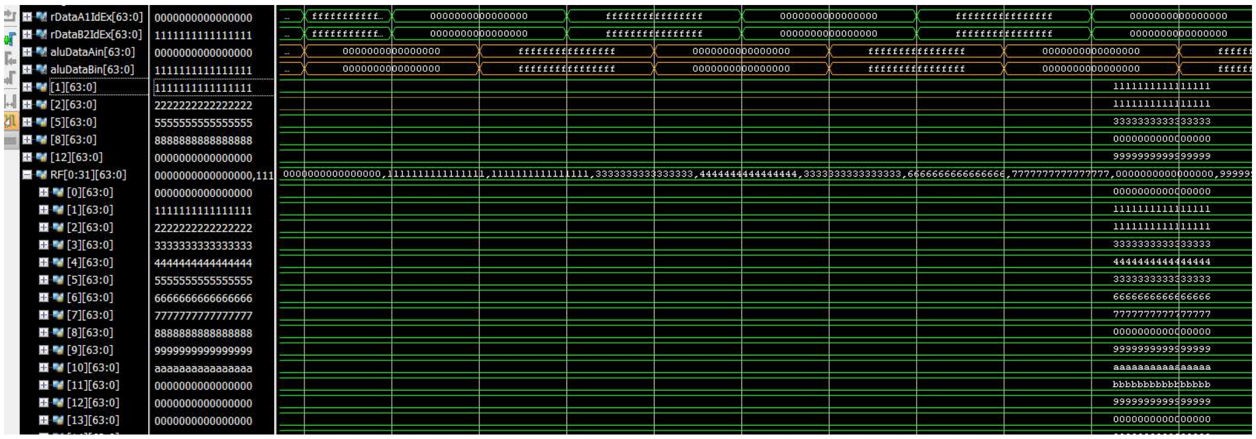
If there is a gap between the source and destination register, the data and signal must be preserved in the pipeline and passed to the ALU at the right clock time. Either before the timing or after, the ALU would be getting the wrong result. I also learn how to write half cycle Forwarding unit, which is as simple as assigning the input write result as the output. It took a quit amount of work to debug how to forward write result to the source operand.

Simulation Results:


300.000 ns
Hazard Detection signals 3'd3, 3'd1, 3'd4
Source Operand X2 changed to 64'h111…1

```
Name                      Value
clk                       1
reset                     0
imem_rd                   1
pc_out_wire[63:0]         56
opCfieldIfId[10:0]        11001011000
regWrExMem                1
memToRegMemWb             1
wr_en                     1
wr_addr[4:0]              11
wrDataWb[63:0]            bbbbbbbbbbbbbbbb
instrOpinIfId[2:0]        000
instrOpinIdEx[2:0]        000
imOp3idEx[2:0]            000
instrOpinExMem[2:0]       010
imOp4exMem[2:0]           010
regRdAddrA1[4:0]
regRdAddrB2[4:0]
rd_data_1[63:0]           4444444444444444
rd_data_2[63:0]           2222222222222222
imRtIfId[4:0]             5
imRnIfId[4:0]             4
imRn2[4:0]                4
imRm2[4:0]
imRd3[4:0]                2
imRt4[4:0]                12
imRt5[4:0]
HDSigA[2:0]
HDSigB[2:0]
hazDatAIdEx[2:0]          0
hazDatBIdEx[2:0]          0
imRm_RtIfId[4:0]          2
aluSrcIdEx                0
offset_addr[18:0]         32
regDataA1[63:0]           4444444444444444
regDataB2[63:0]           2222222222222222
rDataA1IdEx[63:0]         0000000000000000
rDataB2IdEx[63:0]         1111111111111111
aluDataAin[63:0]
aluDataBin[63:0]
[1][63:0]                 1111111111111111
[2][63:0]                 2222222222222222
[5][63:0]                 5555555555555555
[8][63:0]                 8888888888888888
[12][63:0]                0000000000000000
RF[0:31][63:0]            0000000000000000,1
```

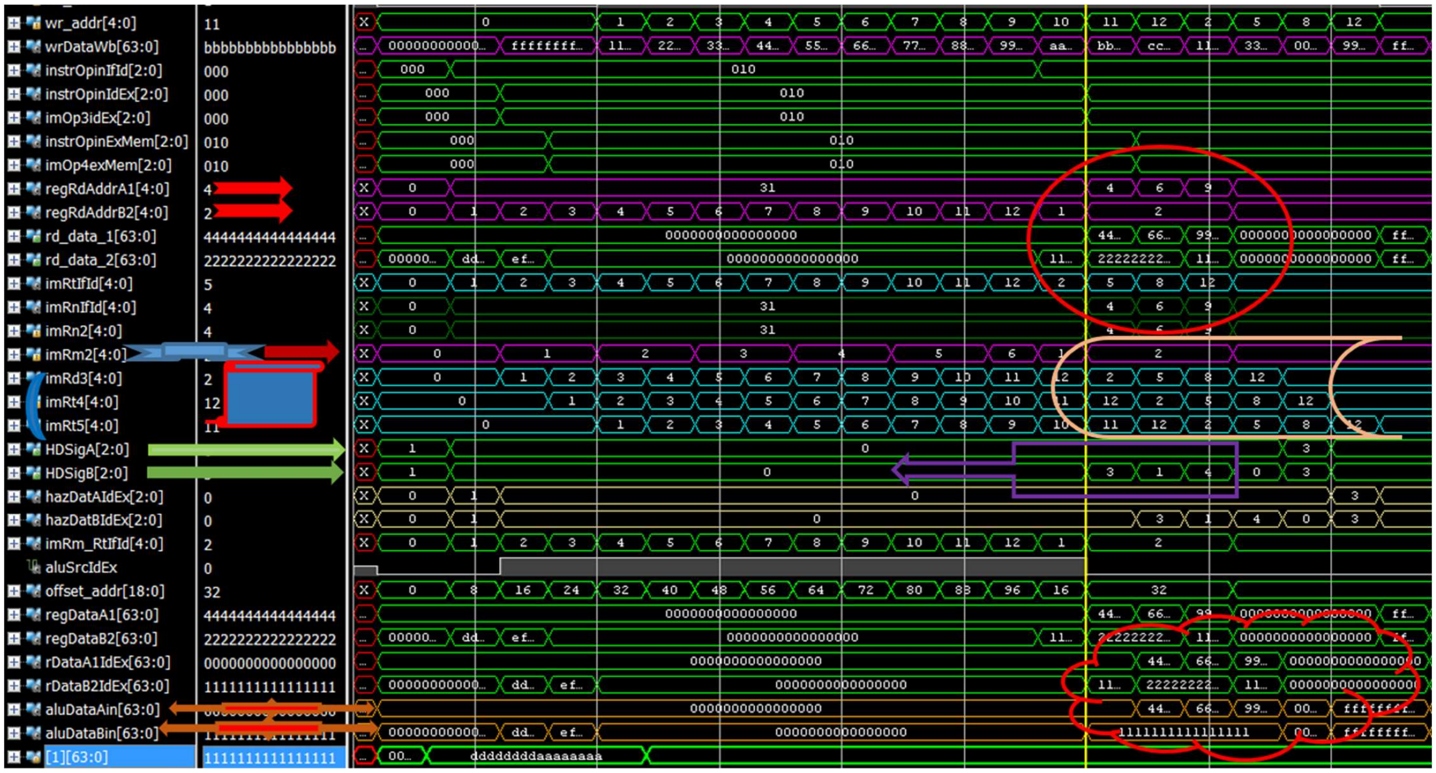
The result inside the register files shows the result after the operation of ADD X2,X31,X1, SUB X5,X4,X2, AND X8,X6,X2 and OR X12,X9,X2



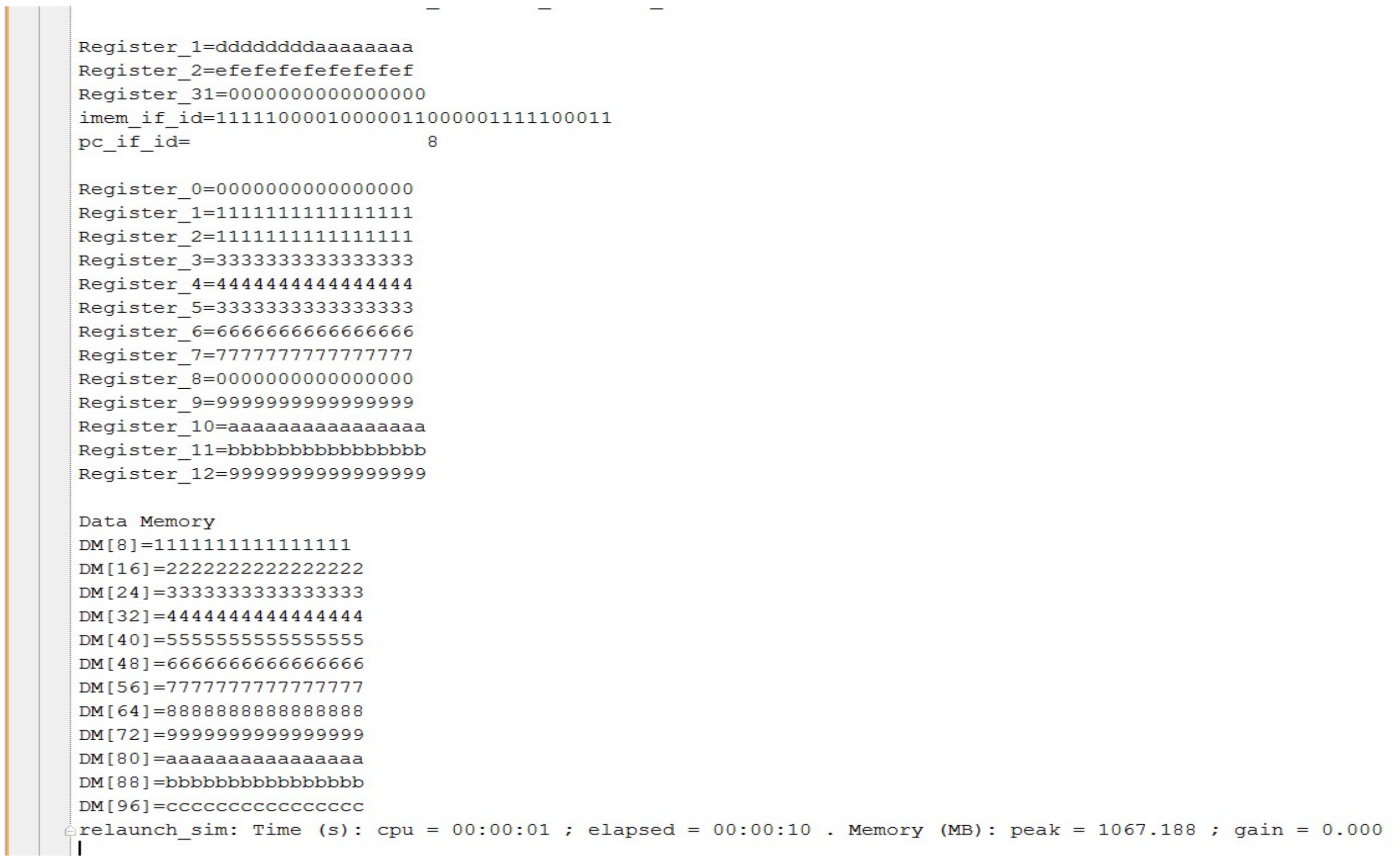
```
rDataA1IdEx[63:0]         0000000000000000
rDataB2IdEx[63:0]         1111111111111111
aluDataAin[63:0]          0000000000000000
aluDataBin[63:0]          1111111111111111
[1][63:0]                 1111111111111111
[2][63:0]                 2222222222222222
[5][63:0]                 5555555555555555
[8][63:0]                 8888888888888888
[12][63:0]                0000000000000000
RF[0:31][63:0]            0000000000000000,111
  [0][63:0]               0000000000000000
  [1][63:0]               1111111111111111
  [2][63:0]               2222222222222222
  [3][63:0]               3333333333333333
  [4][63:0]               4444444444444444
  [5][63:0]               5555555555555555
  [6][63:0]               6666666666666666
  [7][63:0]               7777777777777777
  [8][63:0]               8888888888888888
  [9][63:0]               9999999999999999
  [10][63:0]              aaaaaaaaaaaaaaaa
  [11][63:0]              0000000000000000
  [12][63:0]              0000000000000000
  [13][63:0]              0000000000000000
```

```
0000000000000000,1111111111111111,1111111111111111,3333333333333333,4444444444444444,3333333333333333,6666666666666666,7777777777777777,0000000000000000,99999
0000000000000000
1111111111111111
1111111111111111
3333333333333333
4444444444444444
3333333333333333
6666666666666666
7777777777777777
0000000000000000
9999999999999999
aaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbb
9999999999999999
0000000000000000
```

Zoom out:

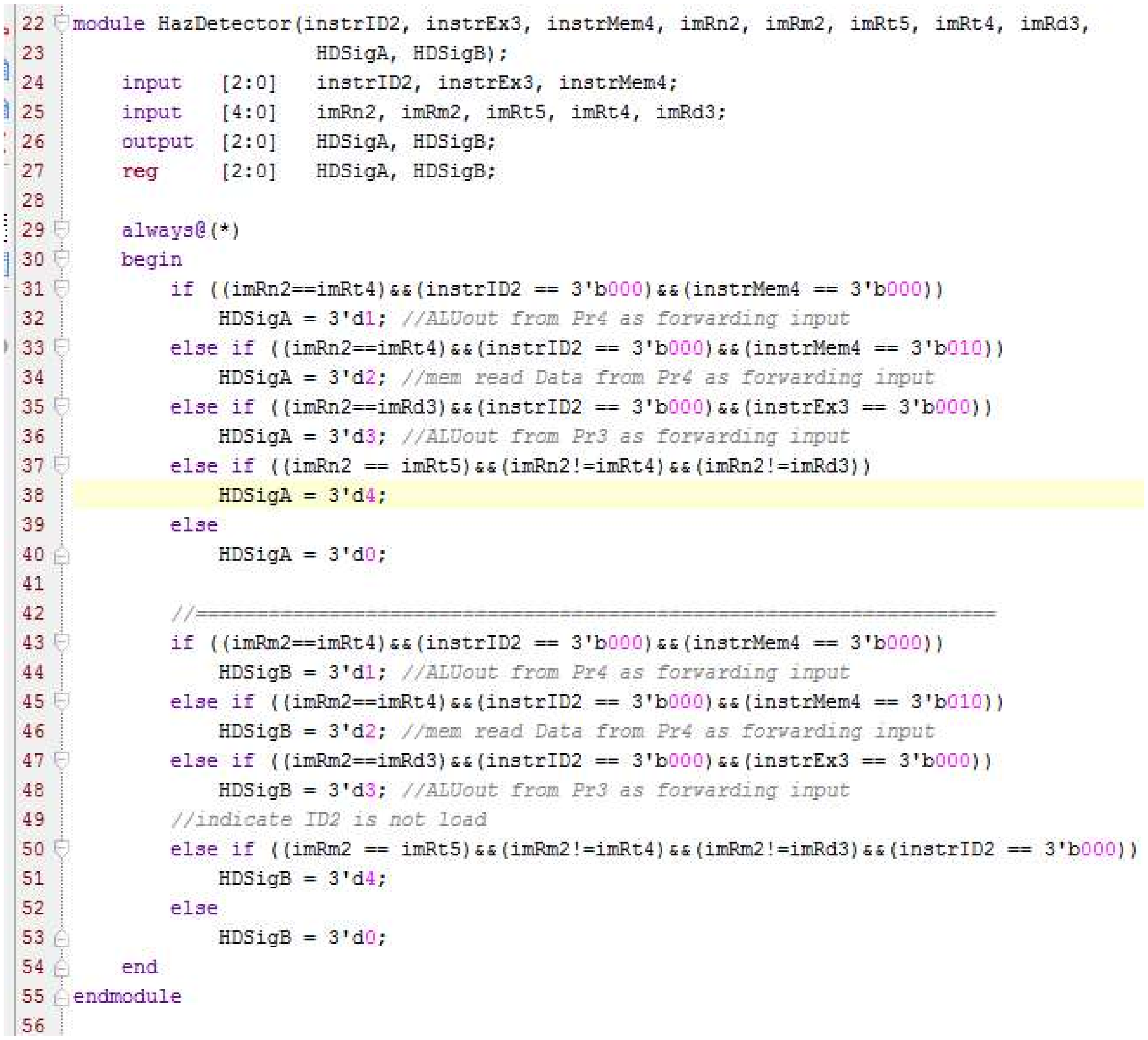


```
wr_addr[4:0]          11
wrDataWb[63:0]        bbbbbbbbbbbbbbbb
instrOpinIfId[2:0]    000
instrOpinIdEx[2:0]    000
imOp3idEx[2:0]        000
instrOpinExMem[2:0]   010
imOp4exMem[2:0]       010
regRdAddrA1[4:0]      4
regRdAddrB2[4:0]      2
rd_data_1[63:0]       4444444444444444
rd_data_2[63:0]       2222222222222222
imRtIfId[4:0]         5
imRnIfId[4:0]         4
imRn2[4:0]            4
imRm2[4:0]
imRd3[4:0]            2
imRt4[4:0]            12
imRt5[4:0]            11
HDSigA[2:0]
HDSigB[2:0]
hazDatAIdEx[2:0]      0
hazDatBIdEx[2:0]      0
imRm_RtIfId[4:0]      2
aluSrcIdEx            0
offset_addr[18:0]     32
regDataA1[63:0]       4444444444444444
regDataB2[63:0]       2222222222222222
rDataA1IdEx[63:0]     0000000000000000
rDataB2IdEx[63:0]     1111111111111111
aluDataAin[63:0]
aluDataBin[63:0]      1111111111111111
[1][63:0]             1111111111111111
```



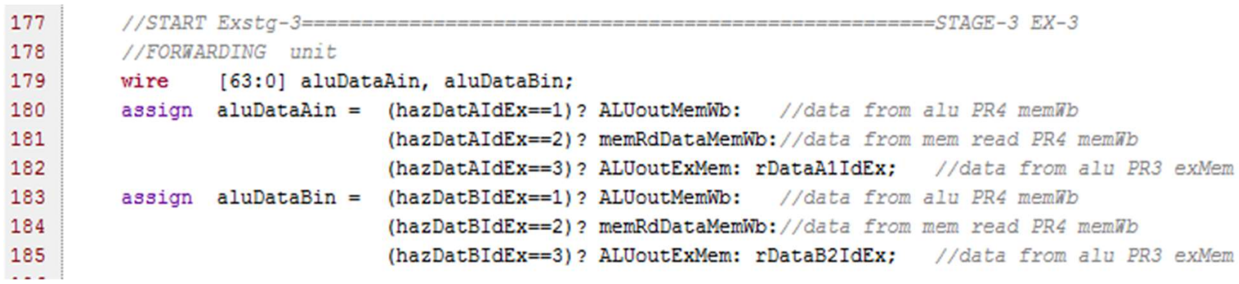
```
Register_1=ddddddddaaaaaaaa
Register_2=efefefefefefefef
Register_31=0000000000000000
imem_if_id=11111000010000011000001111100011
pc_if_id=                   8

Register_0=0000000000000000
Register_1=1111111111111111
Register_2=1111111111111111
Register_3=3333333333333333
Register_4=4444444444444444
Register_5=3333333333333333
Register_6=6666666666666666
Register_7=7777777777777777
Register_8=0000000000000000
Register_9=9999999999999999
Register_10=aaaaaaaaaaaaaaaa
Register_11=bbbbbbbbbbbbbbbb
Register_12=9999999999999999

Data Memory
DM[8]=1111111111111111
DM[16]=2222222222222222
DM[24]=3333333333333333
DM[32]=4444444444444444
DM[40]=5555555555555555
DM[48]=6666666666666666
DM[56]=7777777777777777
DM[64]=8888888888888888
DM[72]=9999999999999999
DM[80]=aaaaaaaaaaaaaaaa
DM[88]=bbbbbbbbbbbbbbbb
DM[96]=cccccccccccccccc
relaunch_sim: Time (s): cpu = 00:00:01 ; elapsed = 00:00:10 . Memory (MB): peak = 1067.188 ; gain = 0.000
```

Hazard Detector:

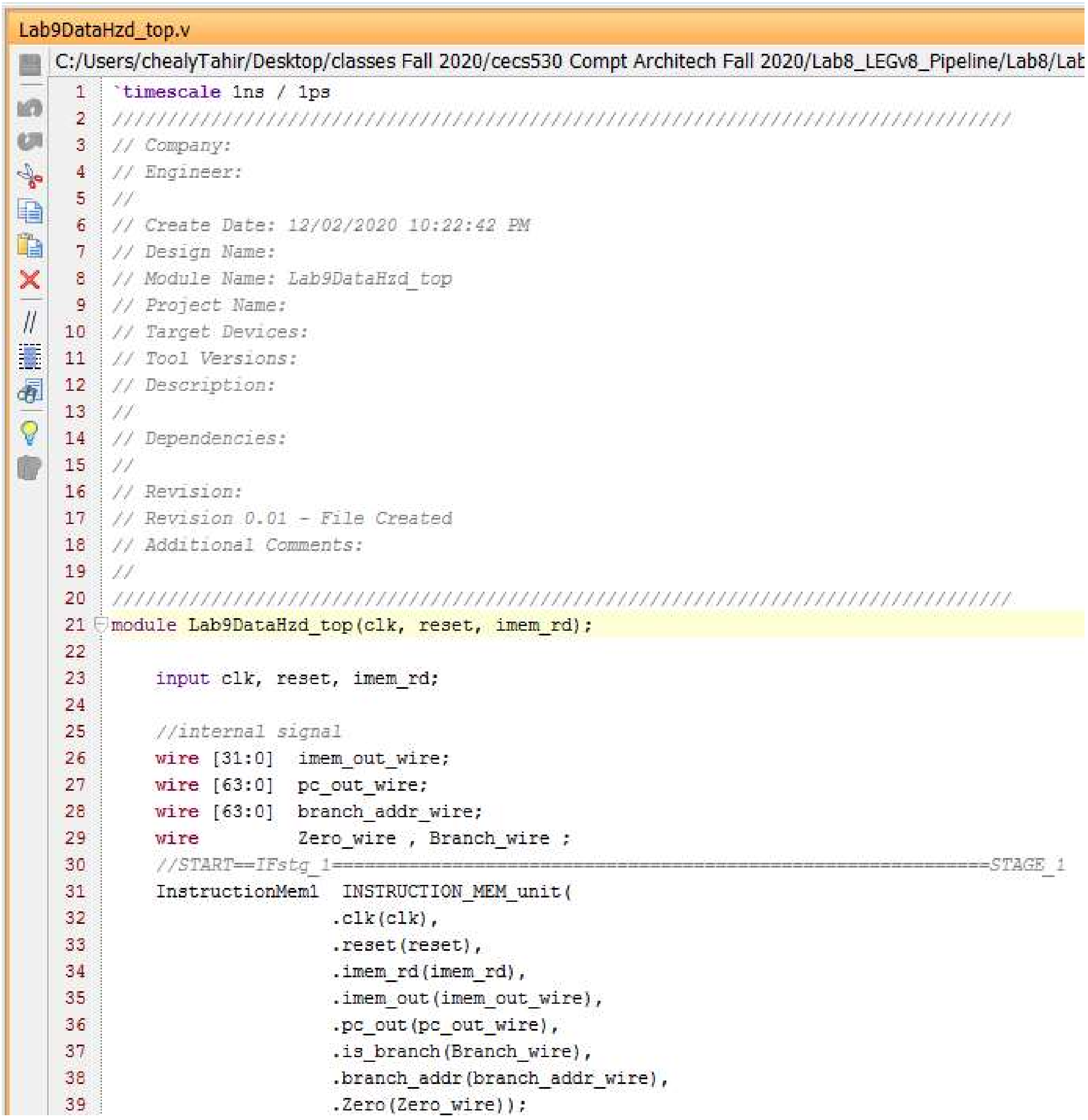


```
module HazDetector(instrID2, instrEx3, instrMem4, imRn2, imRm2, imRt5, imRt4, imRd3,
                    HDSigA, HDSigB);
    input   [2:0]   instrID2, instrEx3, instrMem4;
    input   [4:0]   imRn2, imRm2, imRt5, imRt4, imRd3;
    output  [2:0]   HDSigA, HDSigB;
    reg     [2:0]   HDSigA, HDSigB;

    always@(*)
    begin
        if ((imRn2==imRt4)&&(instrID2 == 3'b000)&&(instrMem4 == 3'b000))
            HDSigA = 3'd1; //ALUout from Pr4 as forwarding input
        else if ((imRn2==imRt4)&&(instrID2 == 3'b000)&&(instrMem4 == 3'b010))
            HDSigA = 3'd2; //mem read Data from Pr4 as forwarding input
        else if ((imRn2==imRd3)&&(instrID2 == 3'b000)&&(instrEx3 == 3'b000))
            HDSigA = 3'd3; //ALUout from Pr3 as forwarding input
        else if ((imRn2 == imRt5)&&(imRn2!=imRt4)&&(imRn2!=imRd3))
            HDSigA = 3'd4;
        else
            HDSigA = 3'd0;

        //================================================================
        if ((imRm2==imRt4)&&(instrID2 == 3'b000)&&(instrMem4 == 3'b000))
            HDSigB = 3'd1; //ALUout from Pr4 as forwarding input
        else if ((imRm2==imRt4)&&(instrID2 == 3'b000)&&(instrMem4 == 3'b010))
            HDSigB = 3'd2; //mem read Data from Pr4 as forwarding input
        else if ((imRm2==imRd3)&&(instrID2 == 3'b000)&&(instrEx3 == 3'b000))
            HDSigB = 3'd3; //ALUout from Pr3 as forwarding input
        //indicate ID2 is not load
        else if ((imRm2 == imRt5)&&(imRm2!=imRt4)&&(imRm2!=imRd3)&&(instrID2 == 3'b000))
            HDSigB = 3'd4;
        else
            HDSigB = 3'd0;
    end
endmodule
```

Forwarding unit: built in Top level in EX-stage, stage3

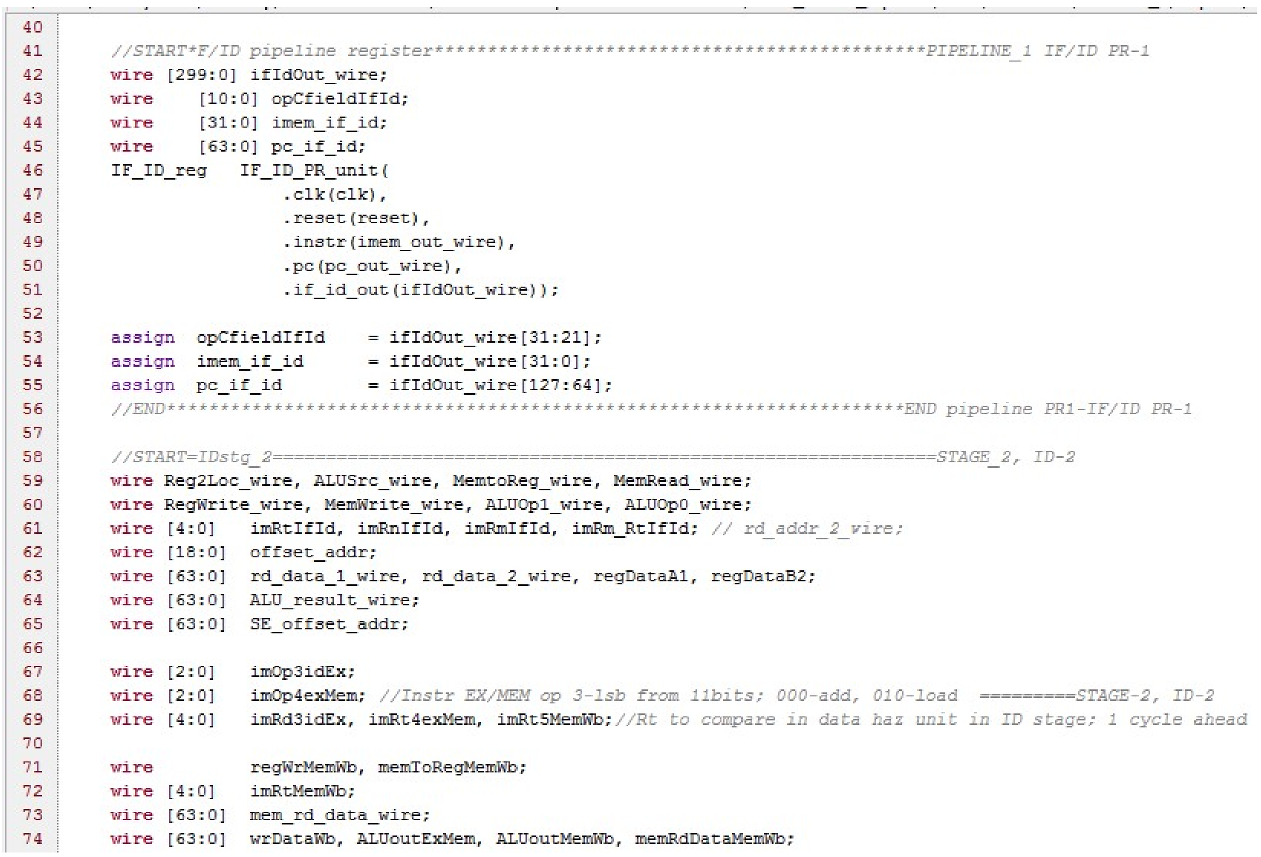


```
    //START Exstg-3=====================================================STAGE-3 EX-3
    //FORWARDING  unit
    wire    [63:0] aluDataAin, aluDataBin;
    assign  aluDataAin =  (hazDatAIdEx==1)? ALUoutMemWb:   //data from alu PR4 memWb
                          (hazDatAIdEx==2)? memRdDataMemWb://data from mem read PR4 memWb
                          (hazDatAIdEx==3)? ALUoutExMem: rDataA1IdEx;   //data from alu PR3 exMem
    assign  aluDataBin =  (hazDatBIdEx==1)? ALUoutMemWb:   //data from alu PR4 memWb
                          (hazDatBIdEx==2)? memRdDataMemWb://data from mem read PR4 memWb
                          (hazDatBIdEx==3)? ALUoutExMem: rDataB2IdEx;   //data from alu PR3 exMem
```

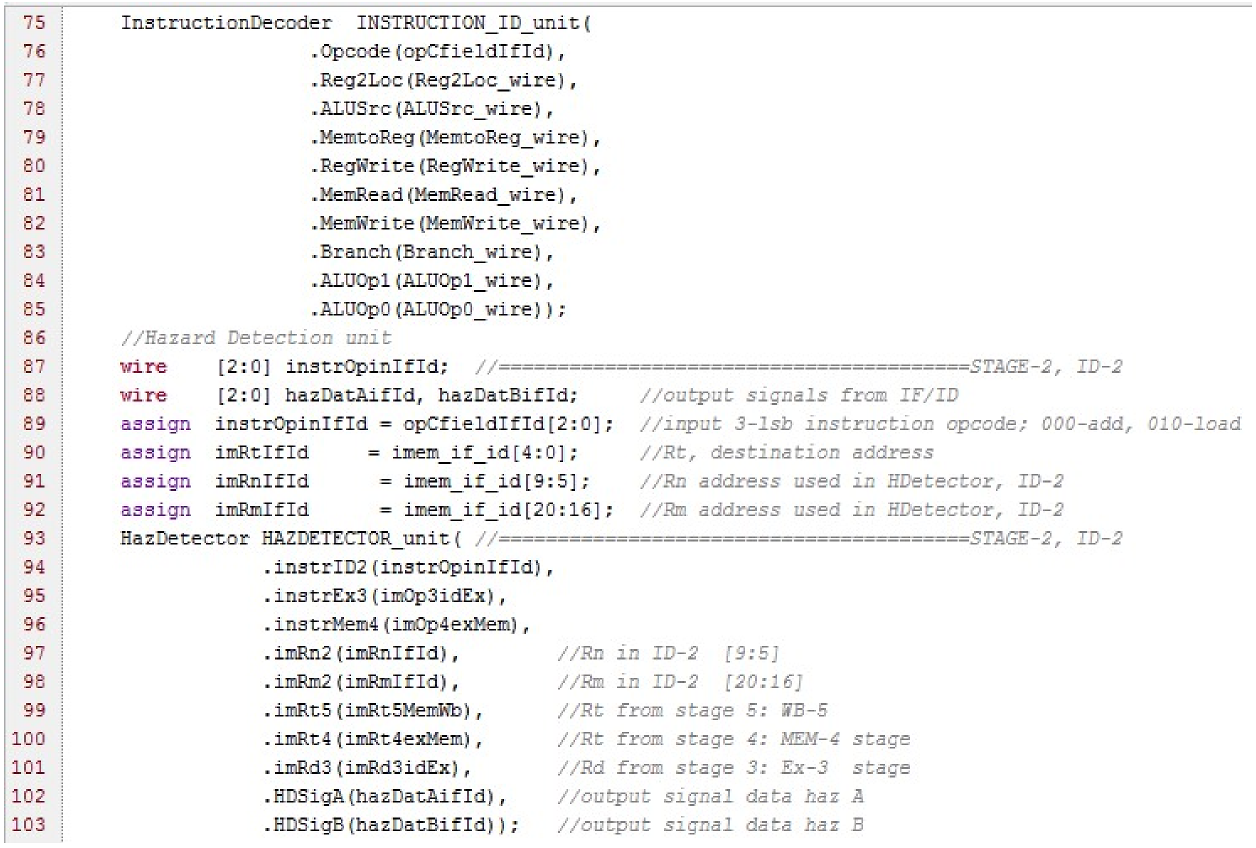

Lab9DataHzd_top.v

C:/Users/chealyTahir/Desktop/classes Fall 2020/cecs530 Compt Architech Fall 2020/Lab8_LEGv8_Pipeline/Lab8/Lab

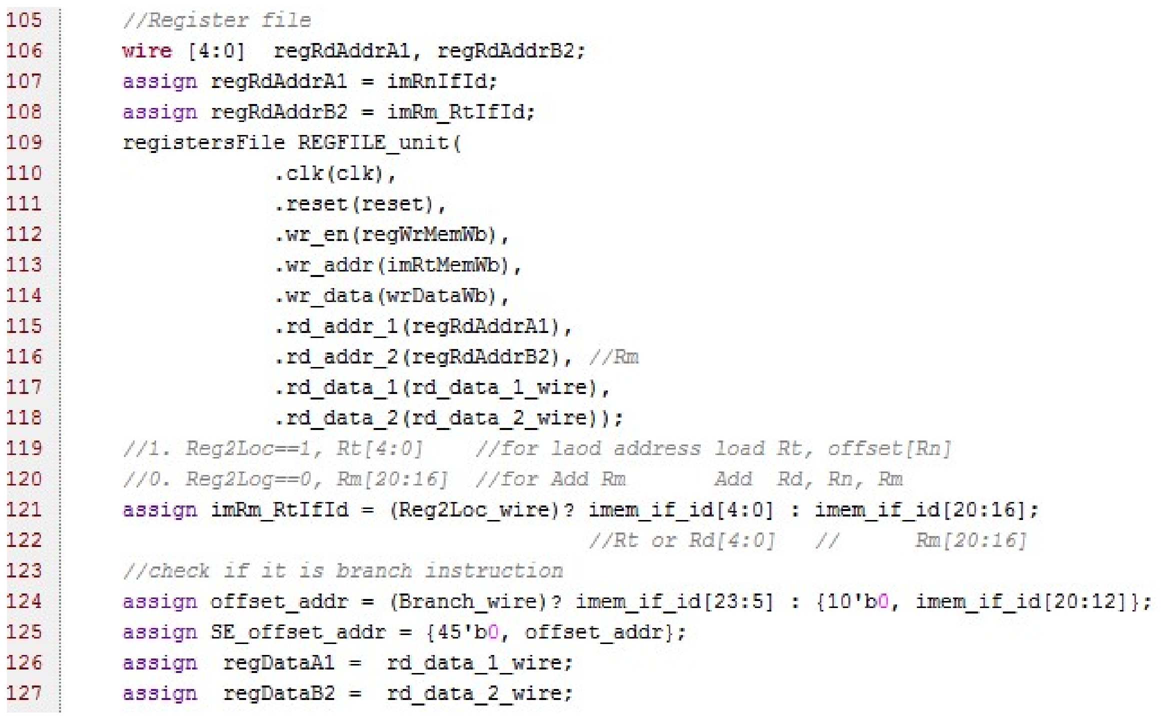
```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/02/2020 10:22:42 PM
// Design Name:
// Module Name: Lab9DataHzd_top
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module Lab9DataHzd_top(clk, reset, imem_rd);

    input clk, reset, imem_rd;

    //internal signal
    wire [31:0]  imem_out_wire;
    wire [63:0]  pc_out_wire;
    wire [63:0]  branch_addr_wire;
    wire         Zero_wire , Branch_wire ;
    //START==IFstg_1=========================================================STAGE_1
    InstructionMem1  INSTRUCTION_MEM_unit(
                     .clk(clk),
                     .reset(reset),
                     .imem_rd(imem_rd),
                     .imem_out(imem_out_wire),
                     .pc_out(pc_out_wire),
                     .is_branch(Branch_wire),
                     .branch_addr(branch_addr_wire),
                     .Zero(Zero_wire));
```



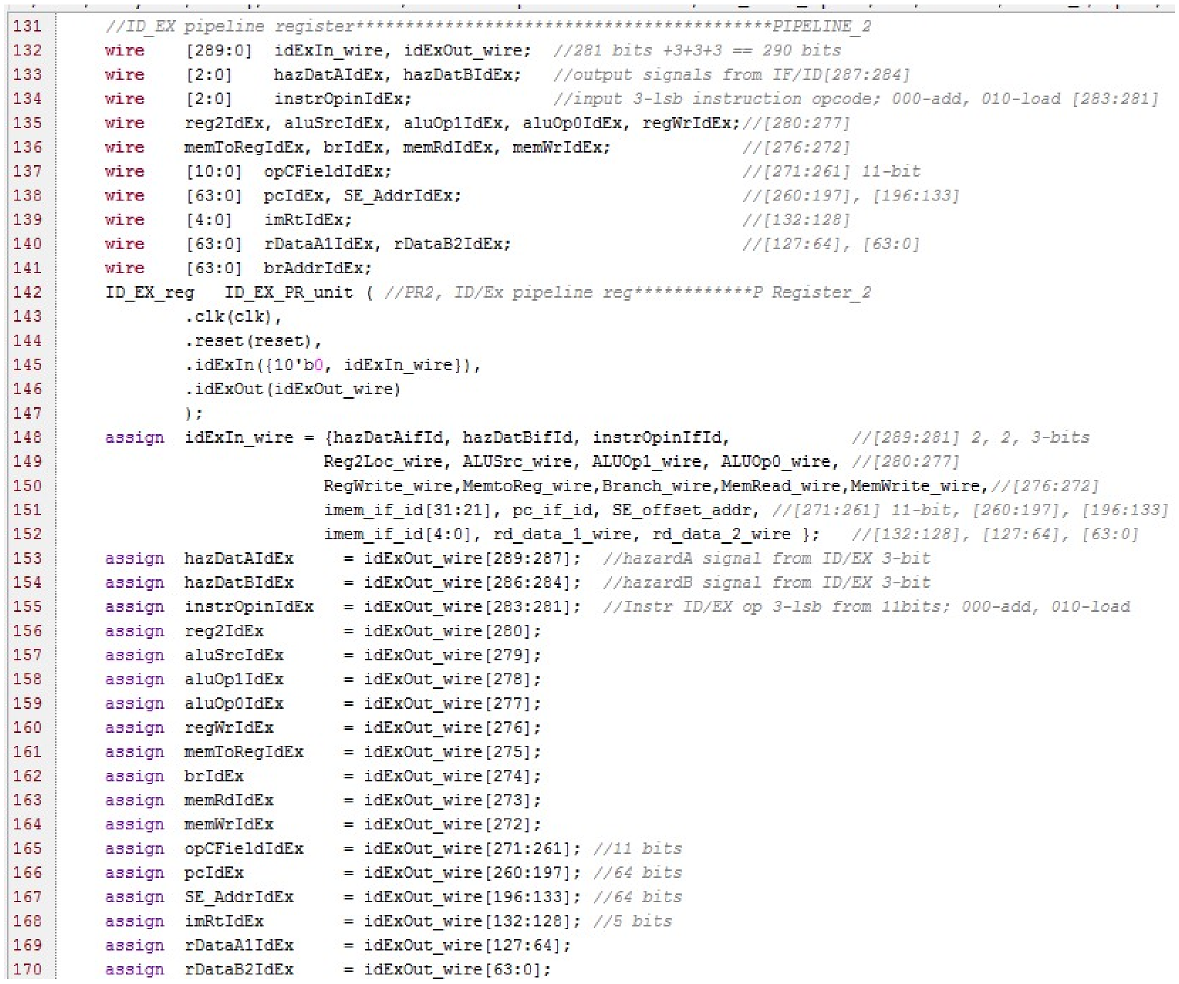
```verilog

    //START*F/ID pipeline register***********************************************PIPELINE_1 IF/ID PR-1
    wire [299:0] ifIdOut_wire;
    wire    [10:0] opCfieldIfId;
    wire    [31:0] imem_if_id;
    wire    [63:0] pc_if_id;
    IF_ID_reg   IF_ID_PR_unit(
                    .clk(clk),
                    .reset(reset),
                    .instr(imem_out_wire),
                    .pc(pc_out_wire),
                    .if_id_out(ifIdOut_wire));

    assign  opCfieldIfId    = ifIdOut_wire[31:21];
    assign  imem_if_id      = ifIdOut_wire[31:0];
    assign  pc_if_id        = ifIdOut_wire[127:64];
    //END**************************************************************END pipeline PR1-IF/ID PR-1

    //START=IDstg_2=====================================================STAGE_2, ID-2
    wire Reg2Loc_wire, ALUSrc_wire, MemtoReg_wire, MemRead_wire;
    wire RegWrite_wire, MemWrite_wire, ALUOp1_wire, ALUOp0_wire;
    wire [4:0]   imRtIfId, imRnIfId, imRmIfId, imRm_RtIfId; // rd_addr_2_wire;
    wire [18:0]  offset_addr;
    wire [63:0]  rd_data_1_wire, rd_data_2_wire, regDataA1, regDataB2;
    wire [63:0]  ALU_result_wire;
    wire [63:0]  SE_offset_addr;

    wire [2:0]   imOp3idEx;
    wire [2:0]   imOp4exMem; //Instr EX/MEM op 3-lsb from 11bits; 000-add, 010-load  =========STAGE-2, ID-2
    wire [4:0]   imRd3idEx, imRt4exMem, imRt5MemWb;//Rt to compare in data haz unit in ID stage; 1 cycle ahead

    wire         regWrMemWb, memToRegMemWb;
    wire [4:0]   imRtMemWb;
    wire [63:0]  mem_rd_data_wire;
    wire [63:0]  wrDataWb, ALUoutExMem, ALUoutMemWb, memRdDataMemWb;
```



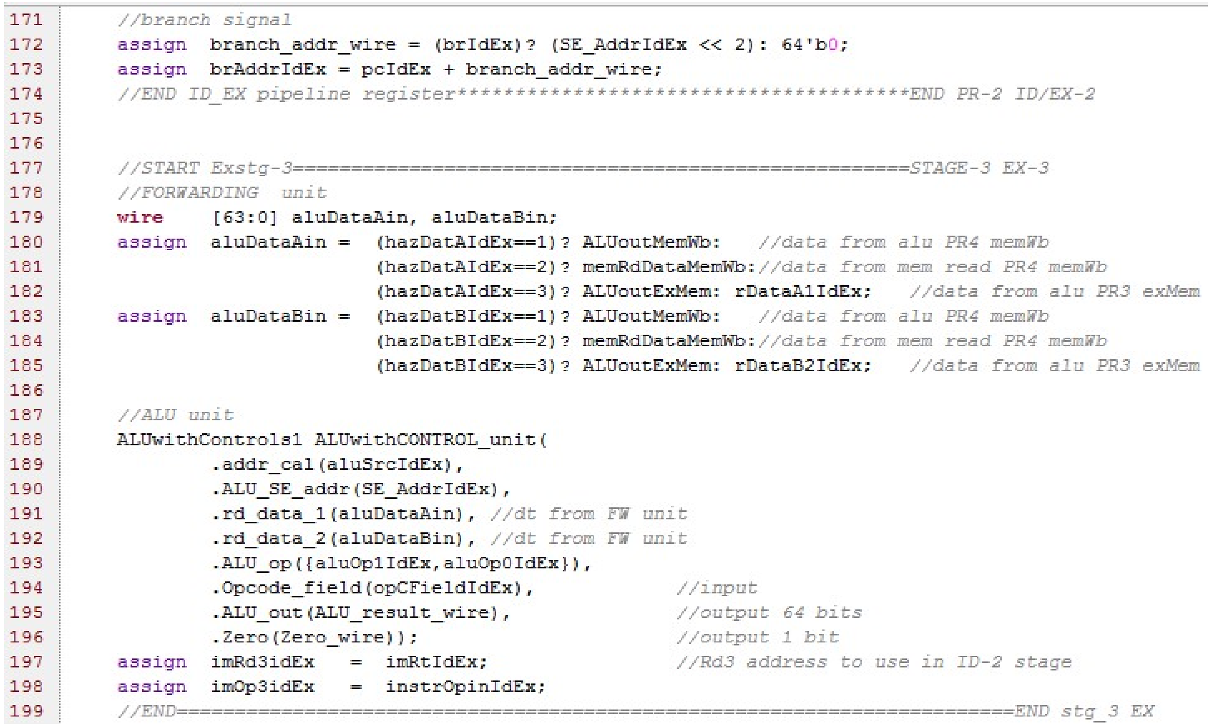
```verilog
    InstructionDecoder  INSTRUCTION_ID_unit(
                    .Opcode(opCfieldIfId),
                    .Reg2Loc(Reg2Loc_wire),
                    .ALUSrc(ALUSrc_wire),
                    .MemtoReg(MemtoReg_wire),
                    .RegWrite(RegWrite_wire),
                    .MemRead(MemRead_wire),
                    .MemWrite(MemWrite_wire),
                    .Branch(Branch_wire),
                    .ALUOp1(ALUOp1_wire),
                    .ALUOp0(ALUOp0_wire));
    //Hazard Detection unit
    wire     [2:0] instrOpinIfId;  //========================================STAGE-2, ID-2
    wire     [2:0] hazDatAifId, hazDatBifId;     //output signals from IF/ID
    assign  instrOpinIfId = opCfieldIfId[2:0];  //input 3-lsb instruction opcode; 000-add, 010-load
    assign  imRtIfId     = imem_if_id[4:0];     //Rt, destination address
    assign  imRnIfId      = imem_if_id[9:5];    //Rn address used in HDetector, ID-2
    assign  imRmIfId      = imem_if_id[20:16];  //Rm address used in HDetector, ID-2
    HazDetector HAZDETECTOR_unit( //=======================================STAGE-2, ID-2
                .instrID2(instrOpinIfId),
                .instrEx3(imOp3idEx),
                .instrMem4(imOp4exMem),
                .imRn2(imRnIfId),         //Rn in ID-2  [9:5]
                .imRm2(imRmIfId),         //Rm in ID-2  [20:16]
                .imRt5(imRt5MemWb),       //Rt from stage 5: WB-5
                .imRt4(imRt4exMem),       //Rt from stage 4: MEM-4 stage
                .imRd3(imRd3idEx),        //Rd from stage 3: Ex-3  stage
                .HDSigA(hazDatAifId),     //output signal data haz A
                .HDSigB(hazDatBifId));    //output signal data haz B
```



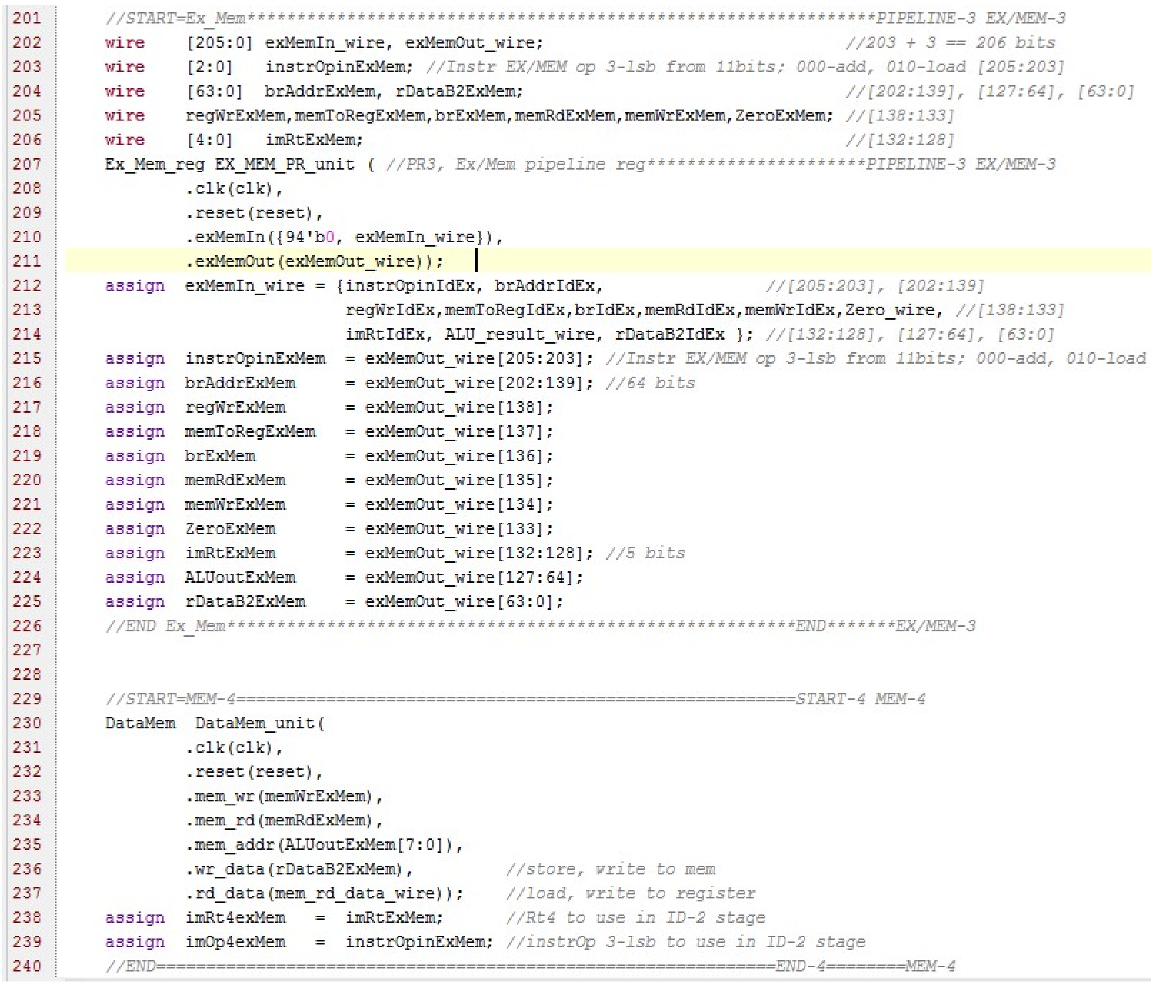
```
//Register file
wire [4:0]  regRdAddrA1, regRdAddrB2;
assign regRdAddrA1 = imRnIfId;
assign regRdAddrB2 = imRm_RtIfId;
registersFile REGFILE_unit(
            .clk(clk),
            .reset(reset),
            .wr_en(regWrMemWb),
            .wr_addr(imRtMemWb),
            .wr_data(wrDataWb),
            .rd_addr_1(regRdAddrA1),
            .rd_addr_2(regRdAddrB2), //Rm
            .rd_data_1(rd_data_1_wire),
            .rd_data_2(rd_data_2_wire));
//1. Reg2Loc==1, Rt[4:0]    //for laod address load Rt, offset[Rn]
//0. Reg2Log==0, Rm[20:16]  //for Add Rm       Add  Rd, Rn, Rm
assign imRm_RtIfId = (Reg2Loc_wire)? imem_if_id[4:0] : imem_if_id[20:16];
                                     //Rt or Rd[4:0]   //      Rm[20:16]
//check if it is branch instruction
assign offset_addr = (Branch_wire)? imem_if_id[23:5] : {10'b0, imem_if_id[20:12]};
assign SE_offset_addr = {45'b0, offset_addr};
assign  regDataA1 =  rd_data_1_wire;
assign  regDataB2 =  rd_data_2_wire;
```



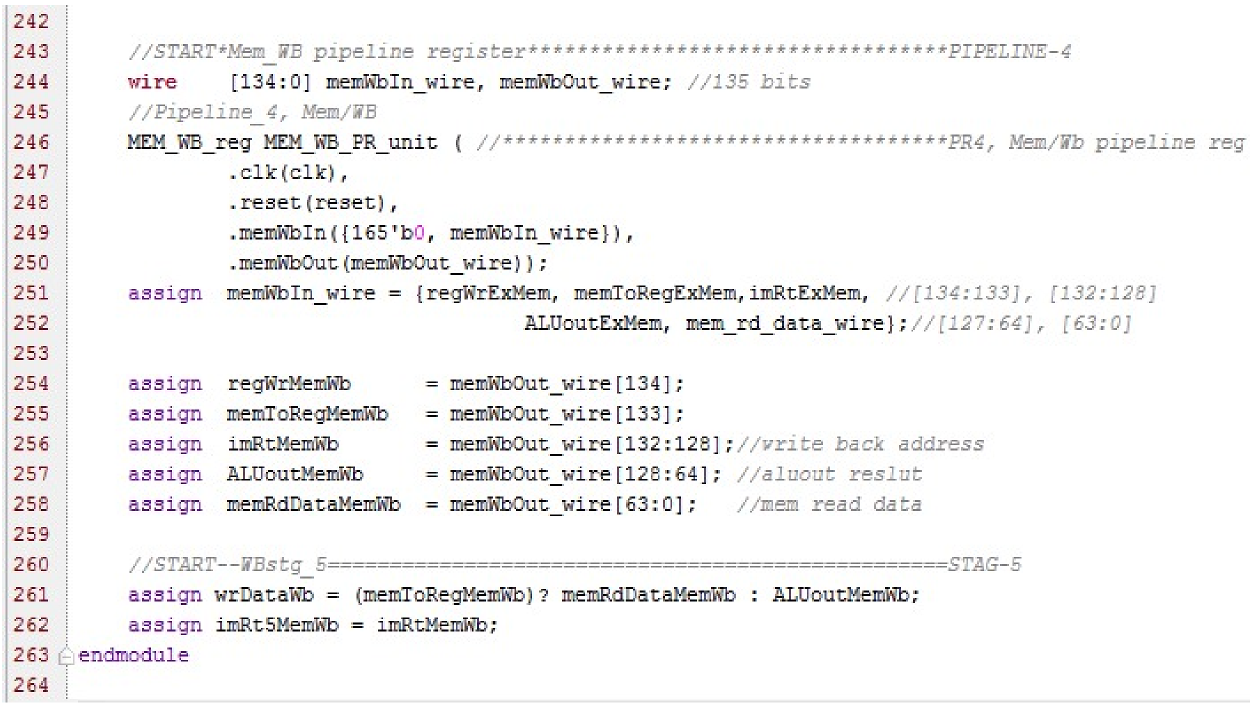
```
//ID_EX pipeline register******************************************PIPELINE_2
wire    [289:0]  idExIn_wire, idExOut_wire;  //281 bits +3+3+3 == 290 bits
wire    [2:0]    hazDatAIdEx, hazDatBIdEx;   //output signals from IF/ID[287:284]
wire    [2:0]    instrOpinIdEx;              //input 3-lsb instruction opcode; 000-add, 010-load [283:281]
wire    reg2IdEx, aluSrcIdEx, aluOp1IdEx, aluOp0IdEx, regWrIdEx;//[280:277]
wire    memToRegIdEx, brIdEx, memRdIdEx, memWrIdEx;            //[276:272]
wire    [10:0]  opCFieldIdEx;                                  //[271:261] 11-bit
wire    [63:0]  pcIdEx, SE_AddrIdEx;                           //[260:197], [196:133]
wire    [4:0]   imRtIdEx;                                      //[132:128]
wire    [63:0]  rDataA1IdEx, rDataB2IdEx;                      //[127:64], [63:0]
wire    [63:0]  brAddrIdEx;
ID_EX_reg   ID_EX_PR_unit ( //PR2, ID/Ex pipeline reg************P Register_2
        .clk(clk),
        .reset(reset),
        .idExIn({10'b0, idExIn_wire}),
        .idExOut(idExOut_wire)
        );
assign  idExIn_wire = {hazDatAifId, hazDatBifId, instrOpinIfId,            //[289:281] 2, 2, 3-bits
                       Reg2Loc_wire, ALUSrc_wire, ALUOp1_wire, ALUOp0_wire, //[280:277]
                       RegWrite_wire,MemtoReg_wire,Branch_wire,MemRead_wire,MemWrite_wire,//[276:272]
                       imem_if_id[31:21], pc_if_id, SE_offset_addr, //[271:261] 11-bit, [260:197], [196:133]
                       imem_if_id[4:0], rd_data_1_wire, rd_data_2_wire };   //[132:128], [127:64], [63:0]
assign  hazDatAIdEx     = idExOut_wire[289:287];  //hazardA signal from ID/EX 3-bit
assign  hazDatBIdEx     = idExOut_wire[286:284];  //hazardB signal from ID/EX 3-bit
assign  instrOpinIdEx   = idExOut_wire[283:281];  //Instr ID/EX op 3-lsb from 11bits; 000-add, 010-load
assign  reg2IdEx        = idExOut_wire[280];
assign  aluSrcIdEx      = idExOut_wire[279];
assign  aluOp1IdEx      = idExOut_wire[278];
assign  aluOp0IdEx      = idExOut_wire[277];
assign  regWrIdEx       = idExOut_wire[276];
assign  memToRegIdEx    = idExOut_wire[275];
assign  brIdEx          = idExOut_wire[274];
assign  memRdIdEx       = idExOut_wire[273];
assign  memWrIdEx       = idExOut_wire[272];
assign  opCFieldIdEx    = idExOut_wire[271:261]; //11 bits
assign  pcIdEx          = idExOut_wire[260:197]; //64 bits
assign  SE_AddrIdEx     = idExOut_wire[196:133]; //64 bits
assign  imRtIdEx        = idExOut_wire[132:128]; //5 bits
assign  rDataA1IdEx     = idExOut_wire[127:64];
assign  rDataB2IdEx     = idExOut_wire[63:0];
```



```
    //branch signal
    assign  branch_addr_wire = (brIdEx)? (SE_AddrIdEx << 2): 64'b0;
    assign  brAddrIdEx = pcIdEx + branch_addr_wire;
    //END ID_EX pipeline register****************************************END PR-2 ID/EX-2


    //START Exstg-3=====================================================STAGE-3 EX-3
    //FORWARDING  unit
    wire    [63:0] aluDataAin, aluDataBin;
    assign  aluDataAin =  (hazDatAIdEx==1)? ALUoutMemWb:   //data from alu PR4 memWb
                          (hazDatAIdEx==2)? memRdDataMemWb://data from mem read PR4 memWb
                          (hazDatAIdEx==3)? ALUoutExMem: rDataA1IdEx;   //data from alu PR3 exMem
    assign  aluDataBin =  (hazDatBIdEx==1)? ALUoutMemWb:   //data from alu PR4 memWb
                          (hazDatBIdEx==2)? memRdDataMemWb://data from mem read PR4 memWb
                          (hazDatBIdEx==3)? ALUoutExMem: rDataB2IdEx;   //data from alu PR3 exMem

    //ALU unit
    ALUwithControls1 ALUwithCONTROL_unit(
            .addr_cal(aluSrcIdEx),
            .ALU_SE_addr(SE_AddrIdEx),
            .rd_data_1(aluDataAin), //dt from FW unit
            .rd_data_2(aluDataBin), //dt from FW unit
            .ALU_op({aluOp1IdEx,aluOp0IdEx}),
            .Opcode_field(opCFieldIdEx),             //input
            .ALU_out(ALU_result_wire),               //output 64 bits
            .Zero(Zero_wire));                       //output 1 bit
    assign  imRd3idEx   =  imRtIdEx;                 //Rd3 address to use in ID-2 stage
    assign  imOp3idEx   =  instrOpinIdEx;
    //END==================================================================END stg_3 EX
```



```
    //START=Ex_Mem**************************************************************PIPELINE-3 EX/MEM-3
    wire    [205:0] exMemIn_wire, exMemOut_wire;                                //203 + 3 == 206 bits
    wire    [2:0]   instrOpinExMem; //Instr EX/MEM op 3-lsb from 11bits; 000-add, 010-load [205:203]
    wire    [63:0]  brAddrExMem, rDataB2ExMem;                                  //[202:139], [127:64], [63:0]
    wire    regWrExMem,memToRegExMem,brExMem,memRdExMem,memWrExMem,ZeroExMem; //[138:133]
    wire    [4:0]   imRtExMem;                                                  //[132:128]
    Ex_Mem_reg EX_MEM_PR_unit ( //PR3, Ex/Mem pipeline reg*********************PIPELINE-3 EX/MEM-3
            .clk(clk),
            .reset(reset),
            .exMemIn({94'b0, exMemIn_wire}),
            .exMemOut(exMemOut_wire));
    assign  exMemIn_wire = {instrOpinIdEx, brAddrIdEx,                  //[205:203], [202:139]
                            regWrIdEx,memToRegIdEx,brIdEx,memRdIdEx,memWrIdEx,Zero_wire, //[138:133]
                            imRtIdEx, ALU_result_wire, rDataB2IdEx }; //[132:128], [127:64], [63:0]
    assign  instrOpinExMem  = exMemOut_wire[205:203]; //Instr EX/MEM op 3-lsb from 11bits; 000-add, 010-load
    assign  brAddrExMem     = exMemOut_wire[202:139]; //64 bits
    assign  regWrExMem      = exMemOut_wire[138];
    assign  memToRegExMem   = exMemOut_wire[137];
    assign  brExMem         = exMemOut_wire[136];
    assign  memRdExMem      = exMemOut_wire[135];
    assign  memWrExMem      = exMemOut_wire[134];
    assign  ZeroExMem       = exMemOut_wire[133];
    assign  imRtExMem       = exMemOut_wire[132:128]; //5 bits
    assign  ALUoutExMem     = exMemOut_wire[127:64];
    assign  rDataB2ExMem    = exMemOut_wire[63:0];
    //END Ex_Mem*********************************************************END*******EX/MEM-3


    //START=MEM-4=========================================================START-4 MEM-4
    DataMem  DataMem_unit(
            .clk(clk),
            .reset(reset),
            .mem_wr(memWrExMem),
            .mem_rd(memRdExMem),
            .mem_addr(ALUoutExMem[7:0]),
            .wr_data(rDataB2ExMem),          //store, write to mem
            .rd_data(mem_rd_data_wire));     //load, write to register
    assign  imRt4exMem   =  imRtExMem;       //Rt4 to use in ID-2 stage
    assign  imOp4exMem   =  instrOpinExMem; //instrOp 3-lsb to use in ID-2 stage
    //END===============================================================END-4=======MEM-4
```



```verilog

    //START*Mem_WB pipeline register**********************************PIPELINE-4
    wire    [134:0] memWbIn_wire, memWbOut_wire; //135 bits
    //Pipeline_4, Mem/WB
    MEM_WB_reg MEM_WB_PR_unit ( //************************************PR4, Mem/Wb pipeline reg
            .clk(clk),
            .reset(reset),
            .memWbIn({165'b0, memWbIn_wire}),
            .memWbOut(memWbOut_wire));
    assign  memWbIn_wire = {regWrExMem, memToRegExMem,imRtExMem, //[134:133], [132:128]
                            ALUoutExMem, mem_rd_data_wire};//[127:64], [63:0]

    assign  regWrMemWb      = memWbOut_wire[134];
    assign  memToRegMemWb   = memWbOut_wire[133];
    assign  imRtMemWb       = memWbOut_wire[132:128];//write back address
    assign  ALUoutMemWb     = memWbOut_wire[128:64]; //aluout reslut
    assign  memRdDataMemWb  = memWbOut_wire[63:0];   //mem read data

    //START--WBstg_5=================================================STAG-5
    assign wrDataWb = (memToRegMemWb)? memRdDataMemWb : ALUoutMemWb;
    assign imRt5MemWb = imRtMemWb;
endmodule

```