```
//**************************************************************************
//Adder Module
//Kamyar Rafati & Naeem Esfahani

module Add (A, B, Y);
    input[31:0]     A, B;
    output[31:0]    Y;
    reg[31:0]       Y;

    always @(A or B)
    begin
        Y = A + B;
    end
endmodule
//**************************************************************************
//Mips ALU Control
//Kamyar Rafati & Naeem Esfahani

module ALU_control (ALUop, Funct, Opcode);
    input[5:0]  Funct;
    input[1:0]  ALUop;
    output[2:0] Opcode;
    reg[2:0]    Opcode;

    always @(ALUop or Funct)
    begin
        case(ALUop)
        0:  Opcode = 2;
        1:  Opcode = 6;
        2:  case(Funct)
            32: Opcode = 2;
            34: Opcode = 6;
            36: Opcode = 0;
            37: Opcode = 1;
            42: Opcode = 7;
            endcase
        endcase
    end
endmodule
//**************************************************************************
//The ALU of the MIPS
//Kamyar Rafati & Naeem Esfahani

module ALU_mips (A, B, Opcode, y, zero);
    input[31:0]     A, B;
    input[2:0]      Opcode;
    output[31:0]    y;
    output          zero;
    reg[31:0]       y;
    reg             zero;

    always @(A or B or Opcode)
    begin
        case(Opcode)
            0: y = A & B;  //bitwise and
            1: y = A|B;  //bitweise or
            2: y = A+B;  //add
            6: y = A-B;  //subtract
            7: y = A<B;  //SLT
        endcase
    end

    always @(y or Opcode)
    begin
        if (Opcode == 5)
        begin
            if (y==0)
                zero = 0;
            else
                zero = 1;
        end
        else
```

```
        begin
            if (y==0)
                zero = 1;
            else
                zero = 0;
        end
    end
endmodule
//***************************************************************************
//Control Unit
//Kamyar Rafati & Naeem Esfahani

module Control (Op, Branch, BranchInvert, ControlData, PCJump, reset, clk);
    input[5:0]  Op;
    input       clk, reset;
    output      Branch, BranchInvert, PCJump;
    output[7:0] ControlData;
    reg         Branch, BranchInvert, PCJump;
    reg[7:0]    ControlData;


    always @(posedge clk or Op)
        if(reset == 0)
        case(Op)
            0:  //R-Type
            begin
                Branch = 0;
                BranchInvert = 0;
                // Control Data:  RegWrite MemToReg MemRead MemWrite ALUSrc
                ALUOp(2) RegDst
                ControlData = 8'b10000101;
                PCJump = 0;
            end

            2: //Jump
            begin
                Branch = 0;
                BranchInvert = 0;
                // Control Data:  RegWrite MemToReg MemRead MemWrite ALUSrc
                ALUOp(2) RegDst
                ControlData = 8'b00000000;
                PCJump = 1;
            end

            4: //BEQ
            begin
                Branch = 1;
                BranchInvert = 0;
                // Control Data:  RegWrite MemToReg MemRead MemWrite ALUSrc
                ALUOp(2) RegDst
                PCJump = 0;
                ControlData = 8'b00000011;
            end

            5: //BNE
            begin
                Branch = 1;
                BranchInvert = 1;
                // Control Data:  RegWrite MemToReg MemRead MemWrite ALUSrc
                ALUOp(2) RegDst
                ControlData = 8'b00000011;
                PCJump = 0;
            end

            8:  //addi
            begin
                Branch = 0;
                BranchInvert = 0;
                // Control Data:  RegWrite MemToReg MemRead MemWrite ALUSrc
                ALUOp(2) RegDst
                ControlData = 8'b10001000;
                PCJump = 0;
```

```
            end

            35: //LW
            begin
                Branch = 0;
                BranchInvert = 0;
                // Control Data:  RegWrite MemToReg MemRead MemWrite ALUSrc
                ALUOp(2) RegDst
                ControlData = 8'b11101000;
                PCJump = 0;
            end

            43: //SW
            begin
                Branch = 0;
                BranchInvert = 0;
                // Control Data:  RegWrite MemToReg MemRead MemWrite ALUSrc
                ALUOp(2) RegDst
                ControlData = 8'b00011000;
                PCJump = 0;
            end
        endcase
        else
        begin
            Branch = 0;
            BranchInvert = 0;
            ControlData = 8'b00000000;
        end
endmodule
//***************************************************************************
//Forwarding Unit
//Kamyar Rafati & Naeem Esfahani

module Forward_Unit (EX_Rt, EX_Rs, EX_Rd,  ID_Rt, ID_Rs, MEM_Rd, WB_Rd,
                     MEM_RegWrite, WB_RegWrite, EX_RegWrite,
                     EX_ForwardMuxA,
                     EX_ForwardMuxB, ID_ForwardEqualA, ID_ForwardEqualB,
                     EX_MemRead);

    input[4:0]  EX_Rt, EX_Rs, EX_Rd,  ID_Rt, ID_Rs, MEM_Rd, WB_Rd;
    input       MEM_RegWrite, WB_RegWrite, EX_RegWrite, EX_MemRead;
    output[1:0] EX_ForwardMuxA, EX_ForwardMuxB, ID_ForwardEqualA,
    ID_ForwardEqualB;
    reg[1:0]    EX_ForwardMuxA, EX_ForwardMuxB, ID_ForwardEqualA,
    ID_ForwardEqualB;

    //if we want to have branch after lw we must add extra logic to both
    this part and data-path
    always @(EX_Rt or EX_Rs or EX_Rd or  ID_Rt or ID_Rs or MEM_Rd or WB_Rd
    or MEM_RegWrite
         or WB_RegWrite or EX_RegWrite)
    begin
    // Branch Check
        //EX_Hazard
        if(MEM_RegWrite && MEM_Rd != 0 && EX_Rd != ID_Rs && MEM_Rd ==
        ID_Rs)
            ID_ForwardEqualA = 1;
        //ID Hazard
        else if(EX_RegWrite && EX_Rd != 0 && EX_Rd == ID_Rs)
            ID_ForwardEqualA = 2;
        else ID_ForwardEqualA = 0;

        //EX Hazard
        if(MEM_RegWrite && MEM_Rd != 0 && EX_Rd != ID_Rt && MEM_Rd ==
        ID_Rt)
            ID_ForwardEqualB = 1;
        //ID Hazard
        else if(EX_RegWrite && EX_Rd != 0 && EX_Rd == ID_Rt)
            ID_ForwardEqualB = 2;
        else ID_ForwardEqualB = 0;

    // ALU Check
```

```
        // Mem hazard
        if(WB_RegWrite && WB_Rd != 0 && MEM_Rd !=EX_Rs && WB_Rd == EX_Rs)
            EX_ForwardMuxA = 1;
        //EX hazard
        else if(MEM_RegWrite && MEM_Rd != 0 && MEM_Rd == EX_Rs)
            EX_ForwardMuxA = 2;
        else EX_ForwardMuxA = 0;

        // Mem HAzard
        if(WB_RegWrite && WB_Rd != 0 && MEM_Rd !=EX_Rt && WB_Rd == EX_Rt)
        //if(WB_RegWrite && WB_Rd != 0 && (MEM_Rd !=EX_Rt && EX_MemRead !=
        0) && WB_Rd == EX_Rt)
        //if(WB_RegWrite && WB_Rd != 0 && WB_Rd == EX_Rt)
            EX_ForwardMuxB = 1;

        //EX Hazard
        else if(MEM_RegWrite && MEM_Rd != 0 && MEM_Rd == EX_Rt)
            EX_ForwardMuxB = 2;
        else EX_ForwardMuxB = 0;
    end
endmodule
//*************************************************************************
//Hazard Detection Unit
//Kamyar Rafati & Naeem Esfahani

module Hazard_Detect (Last_IDEX_NOP, ID_Rt, ID_Rs, EX_Rt, EX_MemRead,
IFID_Write, PCWrite, IDEX_NOP, reset, clk);
    input[4:0]  ID_Rt, ID_Rs, EX_Rt;
    input       EX_MemRead, reset, clk;
    input       Last_IDEX_NOP;
    output      IFID_Write, PCWrite, IDEX_NOP;
    reg         IFID_Write, PCWrite, IDEX_NOP;

    //always @(ID_Rt or ID_Rs or EX_Rt)
    //We must stall pipe for two clocks when we have branch after lw but we
    assume that we'll have no branch fter lw
    always @(negedge clk)
    begin
        if(reset == 0)
        begin
            if(EX_MemRead && (EX_Rt == ID_Rs || EX_Rt  == ID_Rt) &&
            Last_IDEX_NOP != 0)
                //Stall the pipeline
                begin
                    PCWrite = 0;
                    IDEX_NOP = 1;
                    IFID_Write = 0;
                end
            else
                //Don't Stall the Pipeline
                begin
                    PCWrite = 1;
                    IDEX_NOP = 0;
                    IFID_Write = 1;
                end
        end
        else
        begin
          PCWrite = 1;
          IDEX_NOP = 1;
          IFID_Write = 1;
        end
    end
endmodule
//*************************************************************************
//JumpBox: Shifts and combines bits for jump address calculation
//Kamyar Rafati & Naeem Esfahani

module JumpBox (PC, Address, y);
    input[25:0]     Address;
    input[3:0]      PC;
    output[31:0]    y;
```

```
    reg[31:0]       y;

    always @(PC or Address)
    begin
        y[27:0] = Address * 4;
        y[31:27] = PC;
    end
endmodule
//***************************************************************************
//Memmory unit
//Kamyar Rafati & Naeem Esfahani

module Memory(address,ind,outd,we, re,clk);
// this is a 4K * 32 memory module, so the address line must be 12 bits
// make sure that test program will not use an address greater than
00000fff
    input[31:0]     address,ind;
    input           re, we,clk;
    output[31:0]    outd;
    reg[31:0]       memory[4095:0];
    reg[31:0]       outd;
    wire            we_clk;
    wire            re_clk;


    assign we_clk = we & clk;
    assign re_clk = re & clk;

    initial
        $readmemh("mem.dat", memory);

// loads a hex code from file “mem.dat” to the memory, the format for the
file
//is as follows: you specify a 3-bit hex address at each line then 8 bit
instruction
//is followed(start at address 000)

    always @ (posedge re_clk or address)
    begin
        if (re) outd = memory[address];
    end

    always @(negedge we_clk)
    begin
        memory[address] = ind;
    end
endmodule
//***************************************************************************
//32 bit MUX 2 to 1
//Kamyar Rafati & Naeem Esfahani

module MUX2x1_32bit (s, a0, a1, y);
    input           s;
    input[31:0]     a0, a1;
    output[31:0]    y;
    reg[31:0]       y;

    always @(s or a0 or a1)
        case (s)
            0:  y=a0;
            1:  y=a1;
        endcase
endmodule
//***************************************************************************
//Data-path
//Kamyar Rafati & Naeem Esfahani

module Pipelined_MIPS(reset, clk);
    input       reset, clk;
    wire[31:0]  PCcurrent, PCwriteBack, NextInstPC, IF_Inst, PCcalculated,
    SignExtended, ShiftedLeft,
                SelectedRegDataA, SelectedRegDataB, EX_SignExtended,
```

```
            EX_DataB,
            EX_DataA, EX_Inst, MEM_ALUResult, EX_ALUResult,
            EX_MUX_DataA, EX_MUX_DataB,
            MEM_Data, MEM_ReadData, WB_ReadData, EX_SrcMUX_DataB,
            ID_JumpAddr, ID_BranchAddr,
            WB_Address, ID_Inst, ID_PC, RegFileA, RegFileB,
            WB_WriteData;
wire[7:0]   ID_Control, ID_CorrectedControl;
wire[4:0]   WB_WriteAddr, EX_Rd, MEM_Rd;
wire[2:0]   EX_Opcode;
wire[1:0]   ForwardEqualA, ForwardEqualB, EX_ALUOp, EX_ForwardMuxA,
EX_ForwardMuxB;
wire        PCenable, IFID_WriteEnable, IF_Flush, RegDataBranch,
RegDataEqual, BranchInvert,
            IDEX_FlushSelect, WB_RegWrite, Branch, EX_MemToReg,
            EX_RegWrite, EX_MemRead, EX_ALUSrc, EX_RegDst,
            PCSelect, EX_MemWrite, EX_AluSrc, PCJump;

// Instruction Fetch
Memory          IRMemory (PCcurrent, 32'b0, IF_Inst, 1'b0, 1'b1, clk);
Reg_32bit       PC (PCwriteBack, PCcurrent, PCenable, reset, clk);
Add             PCadder (PCcurrent, 32'b100, NextInstPC);
MUX2x1_32bit    IF_PCMux (PCSelect, NextInstPC, ID_BranchAddr,
PCcalculated);
and             BranchAnd (PCSelect, RegDataBranch, Branch);
or              FlushOr (IF_Flush, reset, PCSelect, PCJump);
MUX2x1_32bit    IF_JumpMux (PCJump, PCcalculated, ID_JumpAddr,
PCwriteBack);

// IF/ID reg
Reg_32bit   IFIDPC(NextInstPC, ID_PC, IFID_WriteEnable, IF_Flush, clk);
Reg_32bit   IFIDInst(IF_Inst, ID_Inst, IFID_WriteEnable, IF_Flush,
clk);

//Instruction Decode
Hazard_Detect       HazardDetectUnit(PCenable, ID_Inst[20:16], ID_Inst[
25:21], EX_Inst[20:16], EX_MemRead,
                                IFID_WriteEnable, PCenable,
                                IDEX_FlushSelect, reset, clk);
Control             ControlUnit(ID_Inst[31:26], Branch, BranchInvert,
ID_Control, PCJump, reset, clk);
Add                 PCcalc(ShiftedLeft, ID_PC, ID_BranchAddr);

xor                 BranchInverter (RegDataBranch, BranchInvert,
RegDataEqual);
MUX2x1_8bit         IDEX_Flush(IDEX_FlushSelect, ID_Control, 8'b0,
ID_CorrectedControl);
ShiftLeft2_32bit    SiftLeft(SignExtended, ShiftedLeft);
RegFile_32bit       RegFile(ID_Inst[25:21], RegFileA, ID_Inst[20:16],
RegFileB, WB_WriteAddr,
                    WB_WriteData, WB_RegWrite, clk);
SignExtend_32bit    SignExtend (ID_Inst[15:0], SignExtended);
MUX3x1_32bit        CompareMuxA(ForwardEqualA, RegFileA, MEM_ALUResult,
EX_ALUResult, SelectedRegDataA);
MUX3x1_32bit        CompareMuxB(ForwardEqualB, RegFileB, MEM_ALUResult,
EX_ALUResult, SelectedRegDataB);
RegFile_Compare     RegCompare(SelectedRegDataA, SelectedRegDataB,
RegDataEqual, nil, clk);
JumpBox             MyJumpBox(ID_PC[31:28], ID_Inst[25:0],
ID_JumpAddr);

// ID/EX reg
Reg_32bit   IDEXInst(ID_Inst, EX_Inst, 1'b1, reset, clk);
Reg_32bit   DataA(RegFileA, EX_DataA, 1'b1, reset, clk);
Reg_32bit   DataB(RegFileB, EX_DataB, 1'b1, reset, clk);
Reg_32bit   IDEX_SignExtended(SignExtended, EX_SignExtended, 1'b1,
reset, clk);
Reg_1bit    IDEX_MemToReg(ID_CorrectedControl[6], EX_MemToReg, 1'b1,
reset, clk);
Reg_1bit    IDEX_RegWrite(ID_CorrectedControl[7], EX_RegWrite, 1'b1,
reset, clk);
Reg_1bit    IDEX_MemWrite(ID_CorrectedControl[4], EX_MemWrite, 1'b1,
```

```
    reset, clk);
    Reg_1bit    IDEX_MemRead(ID_CorrectedControl[5], EX_MemRead, 1'b1,
    reset, clk);
    Reg_1bit    IDEX_ALUSrc(ID_CorrectedControl[3], EX_AluSrc, 1'b1, reset,
    clk);
    Reg_1bit    IDEX_RegDst(ID_CorrectedControl[0], EX_RegDst, 1'b1, reset,
    clk);
    Reg_2bit    IDEX_ALUOp(ID_CorrectedControl[2:1], EX_ALUOp, 1'b1, reset,
    clk);

    // Execute
    Forward_Unit    Forward (EX_Inst[20:16], EX_Inst[25:21], EX_Rd,
    ID_Inst[20:16], ID_Inst[25:21],
                            MEM_Rd, WB_WriteAddr,
                            MEM_RegWrite, WB_RegWrite, EX_RegWrite,
                            EX_ForwardMuxA,
                            EX_ForwardMuxB, ForwardEqualA, ForwardEqualB,
                            EX_MemRead);
    MUX3x1_32bit    ALUMUXForwardA(EX_ForwardMuxA, EX_DataA, WB_WriteData,
    MEM_ALUResult, EX_MUX_DataA);
    MUX3x1_32bit    ALUMUXForwardB(EX_ForwardMuxB, EX_DataB, WB_WriteData,
    MEM_ALUResult, EX_MUX_DataB);
    MUX2x1_32bit    ALUMUXSourceB(EX_AluSrc, EX_MUX_DataB, EX_SignExtended,
    EX_SrcMUX_DataB);
    ALU_mips        ALU(EX_MUX_DataA, EX_SrcMUX_DataB, EX_Opcode,
    EX_ALUResult, nil);
    MUX2x1_5bit     EX_RdMUX(EX_RegDst, EX_Inst[20:16], EX_Inst[15:11],
    EX_Rd);
    ALU_control     EX_ALUControl(EX_ALUOp, EX_SignExtended[5:0],
    EX_Opcode);

    // EX/MEM reg
    Reg_32bit   EXMEM_ALUResult(EX_ALUResult, MEM_ALUResult, 1'b1, reset,
    clk);
    Reg_32bit   EXMEM_Data(EX_MUX_DataB, MEM_Data, 1'b1, reset, clk);
    Reg_1bit    EXMEM_MemToReg(EX_MemToReg, MEM_MemToReg, 1'b1, reset,
    clk);
    Reg_1bit    EXMEM_RegWrite(EX_RegWrite, MEM_RegWrite, 1'b1, reset,
    clk);
    Reg_1bit    EXMEM_MemWrite(EX_MemWrite, MEM_MemWrite, 1'b1, reset,
    clk);
    Reg_1bit    EXMEM_MemRead(EX_MemRead, MEM_MemRead, 1'b1, reset, clk);
    Reg_5bit    EXMEM_Rd(EX_Rd, MEM_Rd, 1'b1, reset, clk);

    //MEMory
    Memory  DataMemory(MEM_ALUResult, MEM_Data, MEM_ReadData, MEM_MemWrite,
    MEM_MemRead, clk);

    // MEM/WB reg
    Reg_32bit   MEMWB_ReadData(MEM_ReadData,  WB_ReadData, 1'b1, reset,
    clk);
    Reg_32bit   MEMWB_Address(MEM_ALUResult, WB_Address, 1'b1, reset, clk);
    Reg_5bit    MEMWB_Rd(MEM_Rd,        WB_WriteAddr, 1'b1, reset, clk);
    Reg_1bit    MEMWB_MemToReg(MEM_MemToReg,  WB_MemToReg, 1'b1, reset,
    clk);
    Reg_1bit    MEMWB_RegWrite(MEM_RegWrite,  WB_RegWrite, 1'b1, reset,
    clk);

    //WB
    MUX2x1_32bit    WB_MUX(WB_MemToReg, WB_Address, WB_ReadData,
    WB_WriteData);
endmodule
//**********************************************************************
//1 bit Register
//Kamyar Rafati & Naeem Esfahani

module Reg_1bit (MData, Y, enable, reset, CLK);
    input   MData;
    input   CLK, reset, enable;
    output  Y;
    reg     Y;
    reg     Data;
```

```
    always @(negedge CLK or MData)
    begin
        if (!reset && enable && !CLK) Data=MData;
    end

    always @(posedge CLK)
    begin
        Y = Data;
    end

    always @(posedge reset)
    begin
        Data = 0;
    end
endmodule
//***************************************************************************
//Register file
//Kamyar Rafati & Naeem Esfahani

module RegFile_32bit (RAdr1, Out1, RAdr2, Out2, WAdr, WData, RegWrite,
CLK);

    input[4:0]      RAdr1, RAdr2, WAdr;
    input[31:0]     WData;
    input           RegWrite, CLK;
    output[31:0]    Out1, Out2;
    reg[31:0]       Out1, Out2;
//We split them so we'll have easier test. It means that we can see them in
the waveform
    reg[31:0]       Reg1, Reg2, Reg3, Reg4, Reg5, Reg6, Reg7, Reg8, Reg9,
    Reg10,
                    Reg11, Reg12, Reg13, Reg14, Reg15, Reg16, Reg17, Reg18,
                    Reg19,
                    Reg20, Reg21, Reg22, Reg23, Reg24, Reg25, Reg26, Reg27,
                    Reg28, Reg29,
                    Reg30, Reg31;

// Write
    always @(WAdr or WData or RegWrite or posedge CLK)
    begin
        if (RegWrite &&  CLK)
            case (WAdr)
                1: Reg1=WData;
                2: Reg2=WData;
                3: Reg3=WData;
                4: Reg4=WData;
                5: Reg5=WData;
                6: Reg6=WData;
                7: Reg7=WData;
                8: Reg8=WData;
                9: Reg9=WData;
                10: Reg10=WData;
                11: Reg11=WData;
                12: Reg12=WData;
                13: Reg13=WData;
                14: Reg14=WData;
                15: Reg15=WData;
                16: Reg16=WData;
                17: Reg17=WData;
                18: Reg18=WData;
                19: Reg19=WData;
                20: Reg20=WData;
                21: Reg21=WData;
                22: Reg22=WData;
                23: Reg23=WData;
                24: Reg24=WData;
                25: Reg25=WData;
                26: Reg26=WData;
                27: Reg27=WData;
                28: Reg28=WData;
                29: Reg29=WData;
```

```
                30: Reg30=WData;
                31: Reg31=WData;
            endcase
        end

//Read
    always @(negedge CLK)
    begin
        case(RAdr1)
            0: Out1 = 0;
            1: Out1 = Reg1;
            2: Out1 = Reg2;
            3: Out1 = Reg3;
            4: Out1 = Reg4;
            5: Out1 = Reg5;
            6: Out1 = Reg6;
            7: Out1 = Reg7;
            8: Out1 = Reg8;
            9: Out1 = Reg9;
            10: Out1 = Reg10;
            11: Out1 = Reg11;
            12: Out1 = Reg12;
            13: Out1 = Reg13;
            14: Out1 = Reg14;
            15: Out1 = Reg15;
            16: Out1 = Reg16;
            17: Out1 = Reg17;
            18: Out1 = Reg18;
            19: Out1 = Reg19;
            20: Out1 = Reg20;
            21: Out1 = Reg21;
            22: Out1 = Reg22;
            23: Out1 = Reg23;
            24: Out1 = Reg24;
            25: Out1 = Reg25;
            26: Out1 = Reg26;
            27: Out1 = Reg27;
            28: Out1 = Reg28;
            29: Out1 = Reg29;
            30: Out1 = Reg30;
            31: Out1 = Reg31;
        endcase
        case(RAdr2)
            0: Out2 = 0;
            1: Out2 = Reg1;
            2: Out2 = Reg2;
            3: Out2 = Reg3;
            4: Out2 = Reg4;
            5: Out2 = Reg5;
            6: Out2 = Reg6;
            7: Out2 = Reg7;
            8: Out2 = Reg8;
            9: Out2 = Reg9;
            10: Out2 = Reg10;
            11: Out2 = Reg11;
            12: Out2 = Reg12;
            13: Out2 = Reg13;
            14: Out2 = Reg14;
            15: Out2 = Reg15;
            16: Out2 = Reg16;
            17: Out2 = Reg17;
            18: Out2 = Reg18;
            19: Out2 = Reg19;
            20: Out2 = Reg20;
            21: Out2 = Reg21;
            22: Out2 = Reg22;
            23: Out2 = Reg23;
            24: Out2 = Reg24;
            25: Out2 = Reg25;
            26: Out2 = Reg26;
            27: Out2 = Reg27;
            28: Out2 = Reg28;
```

```
            29: Out2 = Reg29;
            30: Out2 = Reg30;
            31: Out2 = Reg31;
        endcase
    end
endmodule
//***************************************************************************
// Regfile Compare
//Kamyar Rafati & Naeem Esfahani

module RegFile_Compare (A, B, equal, flush, clk);
    input[31:0] A, B;
    input       clk;
    output      equal, flush;
    reg         equal, flush;

    always @(A or B)
    begin
        if (~clk)
            if (A == B)
            begin
                equal = 1;
                flush = 1;
            end
            else
            begin
                equal = 0;
                flush = 0;
            end
    end
endmodule
//***************************************************************************
//32 bit 2 bit shifter
//Kamyar Rafati & Naeem Esfahani

module ShiftLeft2_32bit (a, y);
    input[31:0]     a;
    output[31:0]    y;
    reg[31:0]       y;

    always @(a)
    begin
        y = a * 4;
    end
endmodule
//***************************************************************************
//16 bit to 32 bit sign extender
//Kamyar Rafati & Naeem Esfahani

module SignExtend_32bit (a, y);
    input[15:0]     a;
    output[31:0]    y;
    reg[31:0]       y;

    always @(a)
    begin
        if (a[15] == 1)
            y[31:16] = 16'b1111111111111111;
        else
            y[31:16] = 16'b0000000000000000;

        y[15:0] = a;
    end
endmodule
//***************************************************************************
//Test Bench
//Kamyar Rafati & Naeem Esfahani

module Startup(reset, clk);
    output  reset, clk;
    reg     reset, clk;
```

```
    Pipelined_MIPS  MIPS(reset, clk);

    initial
    begin
        clk= 1;
        reset = 1;

        #20 reset = 0;
        #4750 $finish;
    end

    always
        #10  clk=~clk;
endmodule
//**************************************************************************
//mem.dat:
//@000 20030200 ->  addi    $3,$0,512
//@004 00003020 ->  add     $6,$0,$0
//@008 00001020 ->  add     $2,$0,$0
//@00c 00002020 ->  add     $4,$0,$0
//@010 2005000a ->  addi    $5,$0,10
//@014 8c660000 ->  lw      $6,0($3)
//@018 00461020 ->  add     $2,$2,$6
//@01c 20630004 ->  addi    $3,$3,4
//@020 20840001 ->  addi    $4,$4,1
//@024 1485fffb ->  bne     $4,$5,-5    -> @014
//@028 ac020c00 ->  sw      $2,3027($0)
//@02c 08000100 ->  j       1024        -> @400
//@200 00000001 ->  DATA SEGMENT
//@204 00000002 ->  DATA SEGMENT
//@208 00000003 ->  DATA SEGMENT
//@20c 00000004 ->  DATA SEGMENT
//@210 00000005 ->  DATA SEGMENT
//@214 00000006 ->  DATA SEGMENT
//@218 00000007 ->  DATA SEGMENT
//@21c 00000008 ->  DATA SEGMENT
//@220 00000009 ->  DATA SEGMENT
//@224 0000000a ->  DATA SEGMENT
//@400 0082382a ->  slt     $7,$4,$2
//**************************************************************************
```