

**32-Bit MIPS CPU Verification through Questa**

**EGC 447 Functional Verification of Hardware Systems**

George Dagis (CE), Lukas Mazzetti (CE)

March 31st, 2019

Instructor: Yi-Chung Chen

**Abstract**

This project outlines the simulation and functional verification of a pipelined MIPS processor as well as the design of its respective testbench. The MIPS processor discussed within the project has been designed with system verilog and has been simulated using Questa through a Linux virtual machine. The testbench has been designed in order to test 9 instructions which are made possible by the simulated processor. Each instruction is read from its own unique .txt file. The result is then stored in its an appropriate .txt file.

**Table of Contents**

Introduction ………………………………………………….…............................……......... 4

Testbench Design ...........….................………………….…………….….......................….... 4

Verification …….....................……………………………….………….…..............….......... 4

Results .............…….....................………………………..………………….….............…... 6

Discussion ........…….....................……………………….………………….….............…... 10

5.1 Problems Encountered ………………………………………….……………....…..... 10

5.2 Lessons Learned …………........……………………………….…………………..... 10

Appendices ……………………………………........………………………….……..…….... 12

Appendix A - MIPSpipeline.sv .............................…...............……………………….…… 12

Appendix B - MIPSpipeline\_tb.sv ...........….......……..............……………………….…… 37

Bibliography ……………………………………………….…………………………............ 38

**1. Introduction**

The purpose of this project was to simulate a pipelined MIPS processor and verify its various instructions using Questa software. The testbench has been designed in order to test 9 instructions which are made possible by the simulated processor. Each instruction is read from its own unique .txt file. The result is then stored in its an appropriate .txt file. The MIPS processor tested was retrieved from FPGA 4Student [1, 2, 3].

**2. Testbench Design**

Several different verilog commands were used to write the MIPS outputs to a text file. One of the first commands used in the test bench is "$fopen." This is done in an initial begin block. This command opens an output file and gives it a title so it can be referenced and accessed later in the code. "$fclose" must be used after this. This command closes the output file and allows it to be accessed by other programs. "$fwrite" is the main command used in the testbench. This writes the desired behavior to the output file and saves it.

**3. Verification**

The vast majority of the verification process came from the testbench file and tcl files. Once our files were open in Questa the first step was to run the following command:

*vopt +acc work.test -o test\_opt*

This command does several different things. *Vopt* runs the vopt compiler before simulation. *+acc* adds accessibility to the test file in the work folder. *-o* disables optimizations.

If the commands ran successfully, next the .do file is executed. Two .do files have been created for the purposes of this project; one for each instruction to test as well as an all-in-one file that tests every instruction. Each tcl file contains roughly the following code:

*mv Instructions/testADD.txt instr.txt*

*vsim -c test\_opt*

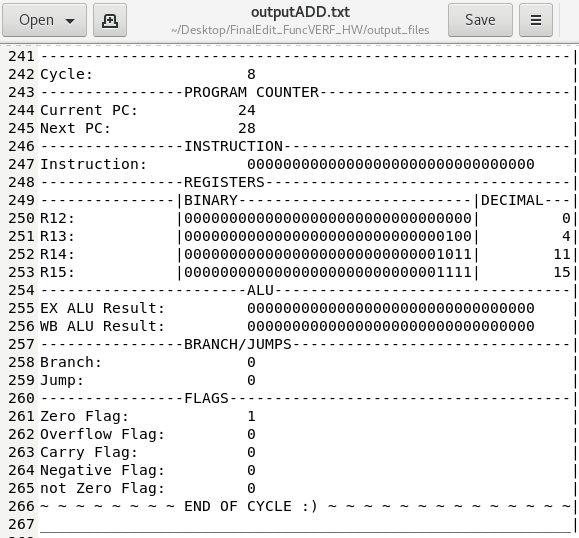
*run 500ns*

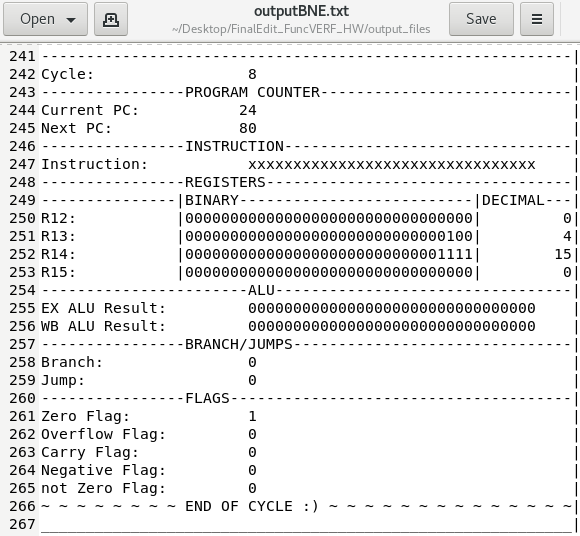
*mv output.txt output\_files/outputADD.txt*

*mv instr.txt Instructions/testADD.txt*

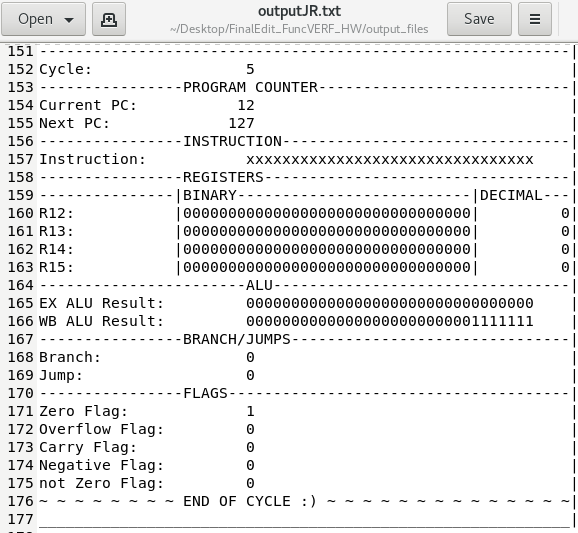
These commands moves each instruction file, which have been written in binary, into a file called instr.txt. This file is then read in the main code with the command *$readmemh("instr.txt", instrmem);*. Various signals were used to determine if the instruction has functioned correctly and are recorded to an output .txt file every cycle for 500ns. The file output.txt is then moved to out output\_files folder and saved there.

**4. Results**

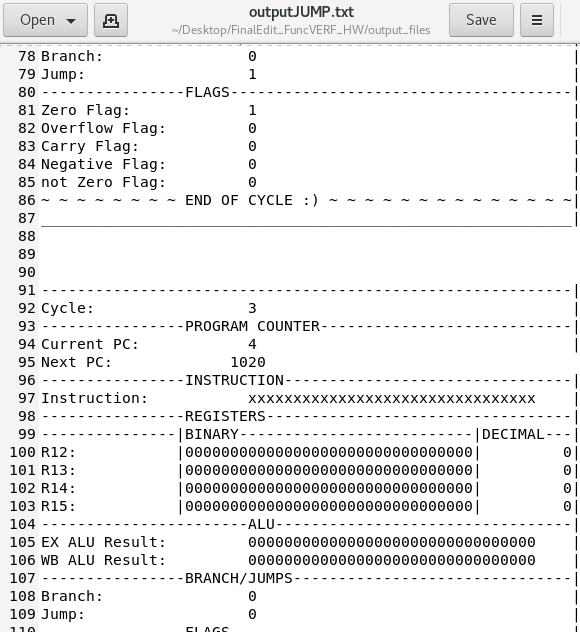




As you can see, R13 and R14 are not equal, so the pointer jumps from PC24 to PC80.



For the JR instruction, we give register the value 1111111 in binary or 127 in decimal, and the Program Counter jumps from PC12 to PC127.



As you can see, in cycle 2 the Jump flag is high, and in the next cycle the Program Counter prepares to jump from PC4 to PC1020.

**5. Discussion**

While the design works as desired, some things can be learned from the design process Some reflections can be made about our design and many things were reinforced throughout designing our project.

**5.1 Problems Encountered**

When testing our MIPS processor, the group had trouble using Questa, as neither group member had much experience using the software. Through the use of online research, the group was able to locate the necessary console commands in order to get the design to run and perform as desired.

A second issue encountered was reading and writing to and from.txt files. The group initially read instructions which were hard coded into the testbench file. Similarly, the outputs were observed through waveforms rather than an output file. This issue was solved similarly to the issue above, meaning the group did necessary research in order to overcome the obstacle. Before this project, the group was only familiar with testing instructions through hard coding in the testbench file itself.

**5.2 Lessons Learned**

This project improved the groups' proficiency in Verilog coding. The group has also become fairly familiar with using Questa, as this software has not been used by either group member prior to this project. Similarly, the group gained valuable knowledge through using the Linux virtual machine. In projects prior, the group had simply created verilog code and hard code instructions via the testbench. This project required the group to read and write to and from .txt files for easier adaptation.

**Appendices**

In the two appendices below, the two system verilog files used in this design can be found. The corresponding MIPS CPU code can be found in Appendix A, while the testbench for the program can be found in Appendix B..

**Appendix A - MIPSpipeline.sv**

`timescale 1 ps / 100 fs

//

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Top level Verilog code for 32-bit 5-stage Pipelined MIPS Processor

module MIPSpipeline(clk, reset);

input clk, reset;

wire [31:0] PC, PCin;

wire [31:0] PC4,ID\_PC4,EX\_PC4;

wire [31:0] PCbne,PC4bne,PCj,PC4bnej,PCjr; // PC signals in MUX

wire [31:0] Instruction,ID\_Instruction,EX\_Instruction; // Output of Instruction Memory

wire [5:0] Opcode,Function; // Opcode, Function

// Extend

wire [15:0] imm16; // immediate in I type instruction

wire [31:0] Im16\_Ext,EX\_Im16\_Ext;

wire [31:0] sign\_ext\_out,zero\_ext\_out;

// regfile

wire [4:0] rs,rt,rd,EX\_rs,EX\_rt,EX\_rd,EX\_WriteRegister,MEM\_WriteRegister,WB\_WriteRegister;

wire [31:0] WB\_WriteData, ReadData1, ReadData2,ReadData1Out,ReadData2Out, EX\_ReadData1, EX\_ReadData2;

// ALU

wire [31:0] Bus\_A\_ALU,Bus\_B\_ALU,Bus\_B\_forwarded;

wire [31:0] EX\_ALUResult,MEM\_ALUResult,WB\_ALUResult;

wire ZeroFlag, OverflowFlag, CarryFlag, NegativeFlag,notZeroFlag;

wire [31:0] WriteDataOfMem,MEM\_ReadDataOfMem,WB\_ReadDataOfMem;

//Control signals

wire RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,Jump,SignZero,JRControl;

wire ID\_RegDst,ID\_ALUSrc,ID\_MemtoReg,ID\_RegWrite,ID\_MemRead,ID\_MemWrite,ID\_Branch,ID\_JRControl;

wire EX\_RegDst,EX\_ALUSrc,EX\_MemtoReg,EX\_RegWrite,EX\_MemRead,EX\_MemWrite,EX\_Branch,EX\_JRControl;

wire MEM\_MemtoReg,MEM\_RegWrite,MEM\_MemRead,MEM\_MemWrite;

wire WB\_MemtoReg,WB\_RegWrite;

wire [1:0] ALUOp,ID\_ALUOp,EX\_ALUOp;

wire [1:0] ALUControl;

wire bneControl,notbneControl;

wire JumpControl,JumpFlush;

wire [1:0] ForwardA,ForwardB;

//flush

wire IF\_flush,IFID\_flush,notIFID\_flush,Stall\_flush,flush;

//shift left

wire [31:0] shiftleft2\_bne\_out,shiftleft2\_jump\_out; // shift left output

// PC Write Enable, IF/ID Write Enable

wire PC\_WriteEn,IFID\_WriteEn;

//====== PC register======

register PC\_Reg(PC,PCin,PC\_WriteEn,reset,clk);

Add Add1(PC4,PC,{29'b0,3'b100}); // PC4 = PC + 4

InstructionMem InstructionMem1(Instruction, PC);

// register IF/ID

register IFID\_PC4(ID\_PC4,PC4,IFID\_WriteEn,reset,clk);

register IFID\_Instruction(ID\_Instruction,Instruction,IFID\_WriteEn,reset,clk);

RegBit IF\_flush\_bit(IFID\_flush,IF\_flush, IFID\_WriteEn,reset, clk);

//========= ID STAGE===========

assign Opcode = ID\_Instruction[31:26];

assign Function = ID\_Instruction[5:0];

assign rs = ID\_Instruction[25:21];

assign rt = ID\_Instruction[20:16];

assign rd = ID\_Instruction[15:11];

assign imm16= ID\_Instruction[15:0];

// Main Control

Control MainControl(

RegDst,

ALUSrc,

MemtoReg,

RegWrite,

MemRead,

MemWrite,

Branch,

ALUOp,

Jump,

SignZero,

Opcode

);

// Regfile

regfile Register\_File(

ReadData1,

ReadData2,

WB\_WriteData,

rs,

rt,

WB\_WriteRegister,

WB\_RegWrite,

reset,

clk);

// forward Read Data if Write and Read at the same time

WB\_forward WB\_forward\_block(ReadData1Out,ReadData2Out,ReadData1,ReadData2,rs,rt,WB\_WriteRegister,WB\_WriteData,WB\_RegWrite);

// Sign-extend

sign\_extend sign\_extend1(sign\_ext\_out,imm16);

// Zero-extend

zero\_extend zero\_extend1(zero\_ext\_out,imm16);

// immediate extend: sign or zero

mux2x32to32 muxSignZero( Im16\_Ext,sign\_ext\_out,zero\_ext\_out, SignZero);

JRControl\_Block JRControl\_Block1( JRControl, ALUOp, Function);

Discard\_Instr Discard\_Instr\_Block(ID\_flush,IF\_flush,JumpControl,bneControl,EX\_JRControl);

or #(50) OR\_flush(flush,ID\_flush,IFID\_flush,Stall\_flush);

flush\_block flush\_block1(ID\_RegDst,ID\_ALUSrc,ID\_MemtoReg,ID\_RegWrite,ID\_MemRead,ID\_MemWrite,ID\_Branch,ID\_ALUOp,

ID\_JRControl,flush,RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,ALUOp,JRControl);

//==========EX STAGE=========================

// thanh ghi ID/EX

register IDEX\_PC4(EX\_PC4,ID\_PC4,1'b1,reset,clk);

register IDEX\_ReadData1(EX\_ReadData1,ReadData1Out,1'b1,reset,clk);

register IDEX\_ReadData2(EX\_ReadData2,ReadData2Out,1'b1,reset,clk);

register IDEX\_Im16\_Ext(EX\_Im16\_Ext,Im16\_Ext,1'b1,reset,clk);

register IDEX\_rs\_rt\_rd(EX\_Instruction[31:0],ID\_Instruction,1'b1,reset,clk);

assign EX\_rs = EX\_Instruction[25:21];

assign EX\_rt = EX\_Instruction[20:16];

assign EX\_rd = EX\_Instruction[15:11];

// 9 control signals via ID/EX

RegBit IDEX\_RegDst(EX\_RegDst, ID\_RegDst, 1'b1,reset, clk);

RegBit IDEX\_ALUSrc(EX\_ALUSrc, ID\_ALUSrc, 1'b1,reset, clk);

RegBit IDEX\_MemtoReg(EX\_MemtoReg, ID\_MemtoReg, 1'b1,reset, clk);

RegBit IDEX\_RegWrite(EX\_RegWrite, ID\_RegWrite, 1'b1,reset, clk);

RegBit IDEX\_MemRead(EX\_MemRead, ID\_MemRead, 1'b1,reset, clk);

RegBit IDEX\_MemWrite(EX\_MemWrite, ID\_MemWrite, 1'b1,reset, clk);

RegBit IDEX\_Branch(EX\_Branch, ID\_Branch, 1'b1,reset, clk);

RegBit IDEX\_JRControl(EX\_JRControl, ID\_JRControl, 1'b1,reset, clk);

RegBit IDEX\_ALUOp1(EX\_ALUOp[1], ID\_ALUOp[1], 1'b1,reset, clk);

RegBit IDEX\_ALUOp0(EX\_ALUOp[0], ID\_ALUOp[0], 1'b1,reset, clk);

// Forwarding unit

ForwardingUnit Forwarding\_Block(ForwardA,ForwardB,MEM\_RegWrite,WB\_RegWrite,MEM\_WriteRegister,WB\_WriteRegister,EX\_rs,EX\_rt);

// mux 3 x32 to 32 to choose source of ALU (forwarding)

mux3x32to32 mux3A(Bus\_A\_ALU,EX\_ReadData1,MEM\_ALUResult,WB\_WriteData,ForwardA);

mux3x32to32 mux3B(Bus\_B\_forwarded,EX\_ReadData2,MEM\_ALUResult,WB\_WriteData,ForwardB);

// mux 2x32 to 32 to select source Bus B of ALU

mux2x32to32 muxALUSrc( Bus\_B\_ALU,Bus\_B\_forwarded,EX\_Im16\_Ext, EX\_ALUSrc);

// ALU Control

ALUControl\_Block ALUControl\_Block1( ALUControl, EX\_ALUOp, EX\_Im16\_Ext[5:0]);

// EX\_Im16\_Ext[5:0] is function

// ALU

alu alu\_block(EX\_ALUResult, CarryFlag, ZeroFlag, OverflowFlag, NegativeFlag, Bus\_A\_ALU, Bus\_B\_ALU, ALUControl);

// mux 2x5 to 5 choose shift register is Rd or Rt

mux2x5to5 muxRegDst( EX\_WriteRegister,EX\_rt,EX\_rd, EX\_RegDst);

//==============MEM STAGE=================

// register EX/MEM

register EXMEM\_ALUResult(MEM\_ALUResult,EX\_ALUResult,1'b1,reset,clk);

register EXMEM\_WriteDataOfMem(WriteDataOfMem, Bus\_B\_forwarded,1'b1,reset,clk);

RegBit EXMEM\_MemtoReg(MEM\_MemtoReg, EX\_MemtoReg, 1'b1,reset, clk);

RegBit EXMEM\_RegWrite(MEM\_RegWrite, EX\_RegWrite, 1'b1,reset, clk);

RegBit EXMEM\_MemRead(MEM\_MemRead, EX\_MemRead, 1'b1,reset, clk);

RegBit EXMEM\_MemWrite(MEM\_MemWrite, EX\_MemWrite, 1'b1,reset, clk);

RegBit EXMEM\_WriteRegister4(MEM\_WriteRegister[4], EX\_WriteRegister[4], 1'b1,reset, clk);

RegBit EXMEM\_WriteRegister3(MEM\_WriteRegister[3], EX\_WriteRegister[3], 1'b1,reset, clk);

RegBit EXMEM\_WriteRegister2(MEM\_WriteRegister[2], EX\_WriteRegister[2], 1'b1,reset, clk);

RegBit EXMEM\_WriteRegister1(MEM\_WriteRegister[1], EX\_WriteRegister[1], 1'b1,reset, clk);

RegBit EXMEM\_WriteRegister0(MEM\_WriteRegister[0], EX\_WriteRegister[0], 1'b1,reset, clk);

// Data Memory

dataMem dataMem1(MEM\_ReadDataOfMem, //data

MEM\_ALUResult, //address

WriteDataOfMem, //writedata

MEM\_MemWrite, //writeenable

MEM\_MemRead,

clk);

//==========WB STAGE====================

// register MEM/WB

register MEMWB\_ReadDataOfMem(WB\_ReadDataOfMem,MEM\_ReadDataOfMem,1'b1,reset,clk);

register MEMWB\_ALUResult(WB\_ALUResult,MEM\_ALUResult,1'b1,reset,clk);

RegBit MEMWB\_WriteRegister4(WB\_WriteRegister[4], MEM\_WriteRegister[4], 1'b1,reset, clk);

RegBit MEMWB\_WriteRegister3(WB\_WriteRegister[3], MEM\_WriteRegister[3], 1'b1,reset, clk);

RegBit MEMWB\_WriteRegister2(WB\_WriteRegister[2], MEM\_WriteRegister[2], 1'b1,reset, clk);

RegBit MEMWB\_WriteRegister1(WB\_WriteRegister[1], MEM\_WriteRegister[1], 1'b1,reset, clk);

RegBit MEMWB\_WriteRegister0(WB\_WriteRegister[0], MEM\_WriteRegister[0], 1'b1,reset, clk);

RegBit MEMWB\_MemtoReg(WB\_MemtoReg, MEM\_MemtoReg, 1'b1,reset, clk);

RegBit MEMWB\_RegWrite(WB\_RegWrite, MEM\_RegWrite, 1'b1,reset, clk);

// Select Data to WriteData for regfile

mux2x32to32 muxMemtoReg( WB\_WriteData, WB\_ALUResult, WB\_ReadDataOfMem,WB\_MemtoReg);

//Stalling

StallControl StallControl\_block(PC\_WriteEn,IFID\_WriteEn,Stall\_flush,EX\_MemRead,EX\_rt,rs,rt,Opcode);

//Jump,bne, JRs

// bne: Branch if not equal

shift\_left\_2 shiftleft2\_bne(shiftleft2\_bne\_out, EX\_Im16\_Ext);

Add Add\_bne(PCbne,EX\_PC4,shiftleft2\_bne\_out);

not #(50) notZero(notZeroFlag,ZeroFlag);

and #(50) andbneControl(bneControl,EX\_Branch,notZeroFlag);

mux2x32to32 muxbneControl( PC4bne,PC4, PCbne, bneControl);

// jump

shift\_left\_2 shiftleft2\_jump(shiftleft2\_jump\_out, {6'b0,ID\_Instruction[25:0]});

assign PCj = {ID\_PC4[31:28],shiftleft2\_jump\_out[27:0]};

not #(50) notIFIDFlush(notIFID\_flush,IFID\_flush);

and #(50) andJumpFlush(JumpFlush,Jump,notIFID\_flush);

not #(50) notbne(notbneControl,bneControl);

and #(50) andJumpBNE(JumpControl,JumpFlush,notbneControl);

mux2x32to32 muxJump( PC4bnej,PC4bne, PCj, JumpControl);

// JR: Jump Register

assign PCjr = Bus\_A\_ALU;

mux2x32to32 muxJR( PCin,PC4bnej, PCjr, EX\_JRControl);

endmodule`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Top level Verilog code for 32-bit 5-stage Pipelined MIPS Processor

module MIPSpipeline(clk, reset);

input clk, reset;

wire [31:0] PC, PCin;

wire [31:0] PC4,ID\_PC4,EX\_PC4;

wire [31:0] PCbne,PC4bne,PCj,PC4bnej,PCjr; // PC signals in MUX

wire [31:0] Instruction,ID\_Instruction,EX\_Instruction; // Output of Instruction Memory

wire [5:0] Opcode,Function; // Opcode, Function

// Extend

wire [15:0] imm16; // immediate in I type instruction

wire [31:0] Im16\_Ext,EX\_Im16\_Ext;

wire [31:0] sign\_ext\_out,zero\_ext\_out;

// regfile

wire [4:0] rs,rt,rd,EX\_rs,EX\_rt,EX\_rd,EX\_WriteRegister,MEM\_WriteRegister,WB\_WriteRegister;

wire [31:0] WB\_WriteData, ReadData1, ReadData2,ReadData1Out,ReadData2Out, EX\_ReadData1, EX\_ReadData2;

// ALU

wire [31:0] Bus\_A\_ALU,Bus\_B\_ALU,Bus\_B\_forwarded;

wire [31:0] EX\_ALUResult,MEM\_ALUResult,WB\_ALUResult;

wire ZeroFlag, OverflowFlag, CarryFlag, NegativeFlag,notZeroFlag;

wire [31:0] WriteDataOfMem,MEM\_ReadDataOfMem,WB\_ReadDataOfMem;

//Control signals

wire RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,Jump,SignZero,JRControl;

wire ID\_RegDst,ID\_ALUSrc,ID\_MemtoReg,ID\_RegWrite,ID\_MemRead,ID\_MemWrite,ID\_Branch,ID\_JRControl;

wire EX\_RegDst,EX\_ALUSrc,EX\_MemtoReg,EX\_RegWrite,EX\_MemRead,EX\_MemWrite,EX\_Branch,EX\_JRControl;

wire MEM\_MemtoReg,MEM\_RegWrite,MEM\_MemRead,MEM\_MemWrite;

wire WB\_MemtoReg,WB\_RegWrite;

wire [1:0] ALUOp,ID\_ALUOp,EX\_ALUOp;

wire [1:0] ALUControl;

wire bneControl,notbneControl;

wire JumpControl,JumpFlush;

wire [1:0] ForwardA,ForwardB;

//flush

wire IF\_flush,IFID\_flush,notIFID\_flush,Stall\_flush,flush;

//shift left

wire [31:0] shiftleft2\_bne\_out,shiftleft2\_jump\_out; // shift left output

// PC Write Enable, IF/ID Write Enable

wire PC\_WriteEn,IFID\_WriteEn;

//====== PC register======

register PC\_Reg(PC,PCin,PC\_WriteEn,reset,clk);

Add Add1(PC4,PC,{29'b0,3'b100}); // PC4 = PC + 4

InstructionMem InstructionMem1(Instruction, PC);

// register IF/ID

register IFID\_PC4(ID\_PC4,PC4,IFID\_WriteEn,reset,clk);

register IFID\_Instruction(ID\_Instruction,Instruction,IFID\_WriteEn,reset,clk);

RegBit IF\_flush\_bit(IFID\_flush,IF\_flush, IFID\_WriteEn,reset, clk);

//========= ID STAGE===========

assign Opcode = ID\_Instruction[31:26];

assign Function = ID\_Instruction[5:0];

assign rs = ID\_Instruction[25:21];

assign rt = ID\_Instruction[20:16];

assign rd = ID\_Instruction[15:11];

assign imm16= ID\_Instruction[15:0];

// Main Control

Control MainControl(

RegDst,

ALUSrc,

MemtoReg,

RegWrite,

MemRead,

MemWrite,

Branch,

ALUOp,

Jump,

SignZero,

Opcode

);

// Regfile

regfile Register\_File(

ReadData1,

ReadData2,

WB\_WriteData,

rs,

rt,

WB\_WriteRegister,

WB\_RegWrite,

reset,

clk);

// forward Read Data if Write and Read at the same time

WB\_forward WB\_forward\_block(ReadData1Out,ReadData2Out,ReadData1,ReadData2,rs,rt,WB\_WriteRegister,WB\_WriteData,WB\_RegWrite);

// Sign-extend

sign\_extend sign\_extend1(sign\_ext\_out,imm16);

// Zero-extend

zero\_extend zero\_extend1(zero\_ext\_out,imm16);

// immediate extend: sign or zero

mux2x32to32 muxSignZero( Im16\_Ext,sign\_ext\_out,zero\_ext\_out, SignZero);

JRControl\_Block JRControl\_Block1( JRControl, ALUOp, Function);

Discard\_Instr Discard\_Instr\_Block(ID\_flush,IF\_flush,JumpControl,bneControl,EX\_JRControl);

or #(50) OR\_flush(flush,ID\_flush,IFID\_flush,Stall\_flush);

flush\_block flush\_block1(ID\_RegDst,ID\_ALUSrc,ID\_MemtoReg,ID\_RegWrite,ID\_MemRead,ID\_MemWrite,ID\_Branch,ID\_ALUOp,

ID\_JRControl,flush,RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,ALUOp,JRControl);

//==========EX STAGE=========================

// thanh ghi ID/EX

register IDEX\_PC4(EX\_PC4,ID\_PC4,1'b1,reset,clk);

register IDEX\_ReadData1(EX\_ReadData1,ReadData1Out,1'b1,reset,clk);

register IDEX\_ReadData2(EX\_ReadData2,ReadData2Out,1'b1,reset,clk);

register IDEX\_Im16\_Ext(EX\_Im16\_Ext,Im16\_Ext,1'b1,reset,clk);

register IDEX\_rs\_rt\_rd(EX\_Instruction[31:0],ID\_Instruction,1'b1,reset,clk);

assign EX\_rs = EX\_Instruction[25:21];

assign EX\_rt = EX\_Instruction[20:16];

assign EX\_rd = EX\_Instruction[15:11];

// 9 control signals via ID/EX

RegBit IDEX\_RegDst(EX\_RegDst, ID\_RegDst, 1'b1,reset, clk);

RegBit IDEX\_ALUSrc(EX\_ALUSrc, ID\_ALUSrc, 1'b1,reset, clk);

RegBit IDEX\_MemtoReg(EX\_MemtoReg, ID\_MemtoReg, 1'b1,reset, clk);

RegBit IDEX\_RegWrite(EX\_RegWrite, ID\_RegWrite, 1'b1,reset, clk);

RegBit IDEX\_MemRead(EX\_MemRead, ID\_MemRead, 1'b1,reset, clk);

RegBit IDEX\_MemWrite(EX\_MemWrite, ID\_MemWrite, 1'b1,reset, clk);

RegBit IDEX\_Branch(EX\_Branch, ID\_Branch, 1'b1,reset, clk);

RegBit IDEX\_JRControl(EX\_JRControl, ID\_JRControl, 1'b1,reset, clk);

RegBit IDEX\_ALUOp1(EX\_ALUOp[1], ID\_ALUOp[1], 1'b1,reset, clk);

RegBit IDEX\_ALUOp0(EX\_ALUOp[0], ID\_ALUOp[0], 1'b1,reset, clk);

// Forwarding unit

ForwardingUnit Forwarding\_Block(ForwardA,ForwardB,MEM\_RegWrite,WB\_RegWrite,MEM\_WriteRegister,WB\_WriteRegister,EX\_rs,EX\_rt);

// mux 3 x32 to 32 to choose source of ALU (forwarding)

mux3x32to32 mux3A(Bus\_A\_ALU,EX\_ReadData1,MEM\_ALUResult,WB\_WriteData,ForwardA);

mux3x32to32 mux3B(Bus\_B\_forwarded,EX\_ReadData2,MEM\_ALUResult,WB\_WriteData,ForwardB);

// mux 2x32 to 32 to select source Bus B of ALU

mux2x32to32 muxALUSrc( Bus\_B\_ALU,Bus\_B\_forwarded,EX\_Im16\_Ext, EX\_ALUSrc);

// ALU Control

ALUControl\_Block ALUControl\_Block1( ALUControl, EX\_ALUOp, EX\_Im16\_Ext[5:0]);

// EX\_Im16\_Ext[5:0] is function

// ALU

alu alu\_block(EX\_ALUResult, CarryFlag, ZeroFlag, OverflowFlag, NegativeFlag, Bus\_A\_ALU, Bus\_B\_ALU, ALUControl);

// mux 2x5 to 5 choose shift register is Rd or Rt

mux2x5to5 muxRegDst( EX\_WriteRegister,EX\_rt,EX\_rd, EX\_RegDst);

//==============MEM STAGE=================

// register EX/MEM

register EXMEM\_ALUResult(MEM\_ALUResult,EX\_ALUResult,1'b1,reset,clk);

register EXMEM\_WriteDataOfMem(WriteDataOfMem, Bus\_B\_forwarded,1'b1,reset,clk);

RegBit EXMEM\_MemtoReg(MEM\_MemtoReg, EX\_MemtoReg, 1'b1,reset, clk);

RegBit EXMEM\_RegWrite(MEM\_RegWrite, EX\_RegWrite, 1'b1,reset, clk);

RegBit EXMEM\_MemRead(MEM\_MemRead, EX\_MemRead, 1'b1,reset, clk);

RegBit EXMEM\_MemWrite(MEM\_MemWrite, EX\_MemWrite, 1'b1,reset, clk);

RegBit EXMEM\_WriteRegister4(MEM\_WriteRegister[4], EX\_WriteRegister[4], 1'b1,reset, clk);

RegBit EXMEM\_WriteRegister3(MEM\_WriteRegister[3], EX\_WriteRegister[3], 1'b1,reset, clk);

RegBit EXMEM\_WriteRegister2(MEM\_WriteRegister[2], EX\_WriteRegister[2], 1'b1,reset, clk);

RegBit EXMEM\_WriteRegister1(MEM\_WriteRegister[1], EX\_WriteRegister[1], 1'b1,reset, clk);

RegBit EXMEM\_WriteRegister0(MEM\_WriteRegister[0], EX\_WriteRegister[0], 1'b1,reset, clk);

// Data Memory

dataMem dataMem1(MEM\_ReadDataOfMem, //data

MEM\_ALUResult, //address

WriteDataOfMem, //writedata

MEM\_MemWrite, //writeenable

MEM\_MemRead,

clk);

//==========WB STAGE====================

// register MEM/WB

register MEMWB\_ReadDataOfMem(WB\_ReadDataOfMem,MEM\_ReadDataOfMem,1'b1,reset,clk);

register MEMWB\_ALUResult(WB\_ALUResult,MEM\_ALUResult,1'b1,reset,clk);

RegBit MEMWB\_WriteRegister4(WB\_WriteRegister[4], MEM\_WriteRegister[4], 1'b1,reset, clk);

RegBit MEMWB\_WriteRegister3(WB\_WriteRegister[3], MEM\_WriteRegister[3], 1'b1,reset, clk);

RegBit MEMWB\_WriteRegister2(WB\_WriteRegister[2], MEM\_WriteRegister[2], 1'b1,reset, clk);

RegBit MEMWB\_WriteRegister1(WB\_WriteRegister[1], MEM\_WriteRegister[1], 1'b1,reset, clk);

RegBit MEMWB\_WriteRegister0(WB\_WriteRegister[0], MEM\_WriteRegister[0], 1'b1,reset, clk);

RegBit MEMWB\_MemtoReg(WB\_MemtoReg, MEM\_MemtoReg, 1'b1,reset, clk);

RegBit MEMWB\_RegWrite(WB\_RegWrite, MEM\_RegWrite, 1'b1,reset, clk);

// Select Data to WriteData for regfile

mux2x32to32 muxMemtoReg( WB\_WriteData, WB\_ALUResult, WB\_ReadDataOfMem,WB\_MemtoReg);

//Stalling

StallControl StallControl\_block(PC\_WriteEn,IFID\_WriteEn,Stall\_flush,EX\_MemRead,EX\_rt,rs,rt,Opcode);

//Jump,bne, JRs

// bne: Branch if not equal

shift\_left\_2 shiftleft2\_bne(shiftleft2\_bne\_out, EX\_Im16\_Ext);

Add Add\_bne(PCbne,EX\_PC4,shiftleft2\_bne\_out);

not #(50) notZero(notZeroFlag,ZeroFlag);

and #(50) andbneControl(bneControl,EX\_Branch,notZeroFlag);

mux2x32to32 muxbneControl( PC4bne,PC4, PCbne, bneControl);

// jump

shift\_left\_2 shiftleft2\_jump(shiftleft2\_jump\_out, {6'b0,ID\_Instruction[25:0]});

assign PCj = {ID\_PC4[31:28],shiftleft2\_jump\_out[27:0]};

not #(50) notIFIDFlush(notIFID\_flush,IFID\_flush);

and #(50) andJumpFlush(JumpFlush,Jump,notIFID\_flush);

not #(50) notbne(notbneControl,bneControl);

and #(50) andJumpBNE(JumpControl,JumpFlush,notbneControl);

mux2x32to32 muxJump( PC4bnej,PC4bne, PCj, JumpControl);

// JR: Jump Register

assign PCjr = Bus\_A\_ALU;

mux2x32to32 muxJR( PCin,PC4bnej, PCjr, EX\_JRControl);

endmodule

/\* Instruction memory module. Change the $readmemb line to have the name of the program you want to load \*/

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Instruction memory module

`timescale 1 ps / 100 fs

module InstructionMem(instruction, address);

input [31:0] address;

output [31:0] instruction;

reg [31:0]instrmem[1023:0];

reg [31:0] temp;

buf #1000 buf0(instruction[0],temp[0]),

buf1(instruction[1],temp[1]),

buf2(instruction[2],temp[2]),

buf3(instruction[3],temp[3]),

buf4(instruction[4],temp[4]),

buf5(instruction[5],temp[5]),

buf6(instruction[6],temp[6]),

buf7(instruction[7],temp[7]),

buf8(instruction[8],temp[8]),

buf9(instruction[9],temp[9]),

buf10(instruction[10],temp[10]),

buf11(instruction[11],temp[11]),

buf12(instruction[12],temp[12]),

buf13(instruction[13],temp[13]),

buf14(instruction[14],temp[14]),

buf15(instruction[15],temp[15]),

buf16(instruction[16],temp[16]),

buf17(instruction[17],temp[17]),

buf18(instruction[18],temp[18]),

buf19(instruction[19],temp[19]),

buf20(instruction[20],temp[20]),

buf21(instruction[21],temp[21]),

buf22(instruction[22],temp[22]),

buf23(instruction[23],temp[23]),

buf24(instruction[24],temp[24]),

buf25(instruction[25],temp[25]),

buf26(instruction[26],temp[26]),

buf27(instruction[27],temp[27]),

buf28(instruction[28],temp[28]),

buf29(instruction[29],temp[29]),

buf30(instruction[30],temp[30]),

buf31(instruction[31],temp[31]);

always @(address)

begin

temp=instrmem[address/4];

end

initial

begin

$readmemh("instr.txt", instrmem);

end

endmodule

module instrmemstimulous();

reg [31:0] addr;

wire [31:0] instr;

InstructionMem instructionmemory(instr, addr);

initial

begin

$monitor("Mem Address=%h instruction=%b",addr,instr);

addr=32'd0;

#10000 addr=32'd4;

#10000 addr=32'd8;

#10000 addr=32'd12;

#10000 addr=32'd16;

#10000 addr=32'd20;

#10000 addr=32'd24;

#10000 addr=32'd28;

#10000;

$finish;

end

endmodule

`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Verilog code for 32-bit adder

module Add(S,A,B);

output [31:0] S;

input [31:0] A,B;

wire [31:0] C;

adder1bit adder1bit0(S[0],C[0],A[0],B[0],1'b0);

adder1bit adder1bit1(S[1],C[1],A[1],B[1],C[0]);

adder1bit adder1bit2(S[2],C[2],A[2],B[2],C[1]);

adder1bit adder1bit3(S[3],C[3],A[3],B[3],C[2]);

adder1bit adder1bit4(S[4],C[4],A[4],B[4],C[3]);

adder1bit adder1bit5(S[5],C[5],A[5],B[5],C[4]);

adder1bit adder1bit6(S[6],C[6],A[6],B[6],C[5]);

adder1bit adder1bit7(S[7],C[7],A[7],B[7],C[6]);

adder1bit adder1bit8(S[8],C[8],A[8],B[8],C[7]);

adder1bit adder1bit9(S[9],C[9],A[9],B[9],C[8]);

adder1bit adder1bit10(S[10],C[10],A[10],B[10],C[9]);

adder1bit adder1bit11(S[11],C[11],A[11],B[11],C[10]);

adder1bit adder1bit12(S[12],C[12],A[12],B[12],C[11]);

adder1bit adder1bit13(S[13],C[13],A[13],B[13],C[12]);

adder1bit adder1bit14(S[14],C[14],A[14],B[14],C[13]);

adder1bit adder1bit15(S[15],C[15],A[15],B[15],C[14]);

adder1bit adder1bit16(S[16],C[16],A[16],B[16],C[15]);

adder1bit adder1bit17(S[17],C[17],A[17],B[17],C[16]);

adder1bit adder1bit18(S[18],C[18],A[18],B[18],C[17]);

adder1bit adder1bit19(S[19],C[19],A[19],B[19],C[18]);

adder1bit adder1bit20(S[20],C[20],A[20],B[20],C[19]);

adder1bit adder1bit21(S[21],C[21],A[21],B[21],C[20]);

adder1bit adder1bit22(S[22],C[22],A[22],B[22],C[21]);

adder1bit adder1bit23(S[23],C[23],A[23],B[23],C[22]);

adder1bit adder1bit24(S[24],C[24],A[24],B[24],C[23]);

adder1bit adder1bit25(S[25],C[25],A[25],B[25],C[24]);

adder1bit adder1bit26(S[26],C[26],A[26],B[26],C[25]);

adder1bit adder1bit27(S[27],C[27],A[27],B[27],C[26]);

adder1bit adder1bit28(S[28],C[28],A[28],B[28],C[27]);

adder1bit adder1bit29(S[29],C[29],A[29],B[29],C[28]);

adder1bit adder1bit30(S[30],C[30],A[30],B[30],C[29]);

adder1bit adder1bit31(S[31],C[31],A[31],B[31],C[30]);

endmodule

//---------------------------------------------------------------------------------------------------

`timescale 1 ps / 100 fs

module adder1bit(sum,cout,a,b,cin);

input a,b,cin;

output cout,sum;

// sum = a xor b xor cin

xor #(50) (sum,a,b,cin);

// carry out = a.b + cin.(a+b)

and #(50) and1(c1,a,b);

or #(50) or1(c2,a,b);

and #(50) and2(c3,c2,cin);

or #(50) or2(cout,c1,c3);

endmodule

`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Register file

module regfile(

ReadData1,

ReadData2,

WriteData,

ReadRegister1,

ReadRegister2,

WriteRegister,

RegWrite,

reset,

clk);

input [4:0]ReadRegister1,ReadRegister2,WriteRegister;

input [31:0] WriteData;

input RegWrite,reset, clk;

output [31:0] ReadData1, ReadData2;

wire [31:0] WriteEn;

wire [31:0] RegArray [0:31];

integer i;

//----Decoder Block

decoder Decoder1( WriteEn,RegWrite,WriteRegister);

register reg0 (RegArray[0],32'b0,1'b1,1'b0, clk);

register reg1 (RegArray[1],WriteData,WriteEn[1],reset,clk);

register reg2 (RegArray[2],WriteData,WriteEn[2],reset,clk);

register reg3 (RegArray[3],WriteData,WriteEn[3],reset,clk);

register reg4 (RegArray[4],WriteData,WriteEn[4],reset,clk);

register reg5 (RegArray[5],WriteData,WriteEn[5],reset,clk);

register reg6 (RegArray[6],WriteData,WriteEn[6],reset,clk);

register reg7 (RegArray[7],WriteData,WriteEn[7],reset,clk);

register reg8 (RegArray[8],WriteData,WriteEn[8],reset,clk);

register reg9 (RegArray[9],WriteData,WriteEn[9],reset,clk);

register reg10 (RegArray[10],WriteData,WriteEn[10],reset,clk);

register reg11 (RegArray[11],WriteData,WriteEn[11],reset,clk);

register reg12 (RegArray[12],WriteData,WriteEn[12],reset,clk);

register reg13 (RegArray[13],WriteData,WriteEn[13],reset,clk);

register reg14 (RegArray[14],WriteData,WriteEn[14],reset,clk);

register reg15 (RegArray[15],WriteData,WriteEn[15],reset,clk);

register reg16 (RegArray[16],WriteData,WriteEn[16],reset,clk);

register reg17 (RegArray[17],WriteData,WriteEn[17],reset,clk);

register reg18 (RegArray[18],WriteData,WriteEn[18],reset,clk);

register reg19 (RegArray[19],WriteData,WriteEn[19],reset,clk);

register reg20 (RegArray[20],WriteData,WriteEn[20],reset,clk);

register reg21 (RegArray[21],WriteData,WriteEn[21],reset,clk);

register reg22 (RegArray[22],WriteData,WriteEn[22],reset,clk);

register reg23 (RegArray[23],WriteData,WriteEn[23],reset,clk);

register reg24 (RegArray[24],WriteData,WriteEn[24],reset,clk);

register reg25 (RegArray[25],WriteData,WriteEn[25],reset,clk);

register reg26 (RegArray[26],WriteData,WriteEn[26],reset,clk);

register reg27 (RegArray[27],WriteData,WriteEn[27],reset,clk);

register reg28 (RegArray[28],WriteData,WriteEn[28],reset,clk);

register reg29 (RegArray[29],WriteData,WriteEn[29],reset,clk);

register reg30 (RegArray[30],WriteData,WriteEn[30],reset,clk);

register reg31 (RegArray[31],WriteData,WriteEn[31],reset,clk);

//----32x32to32 Multiplexor1 Block----

mux32x32to32 Mux1(ReadData1,RegArray[0], RegArray[1],RegArray[2], RegArray[3],RegArray[4],RegArray[5],RegArray[6],RegArray[7],

RegArray[8],RegArray[9],RegArray[10],RegArray[11],RegArray[12],RegArray[13],RegArray[14],RegArray[15],RegArray[16],RegArray[17],

RegArray[18], RegArray[19],RegArray[20],RegArray[21],RegArray[22],RegArray[23],RegArray[24],RegArray[25],RegArray[26],

RegArray[27], RegArray[28], RegArray[29],RegArray[30],RegArray[31], ReadRegister1

);

//----32x32to32 Multiplexor2 Block----

mux32x32to32 Mux2(ReadData2,RegArray[0], RegArray[1],RegArray[2], RegArray[3],RegArray[4],RegArray[5],RegArray[6],RegArray[7],

RegArray[8],RegArray[9],RegArray[10],RegArray[11],RegArray[12],RegArray[13],RegArray[14],RegArray[15],RegArray[16],RegArray[17],

RegArray[18], RegArray[19],RegArray[20],RegArray[21],RegArray[22],RegArray[23],RegArray[24],RegArray[25],RegArray[26],

RegArray[27], RegArray[28], RegArray[29],RegArray[30],RegArray[31], ReadRegister2

);

endmodule

//------------DFF-------------------

module D\_FF (q, d, reset, clk);

output q;

input d, reset, clk;

reg q; // Indicate that q is stateholding

always @(posedge clk or posedge reset)

if (reset)

q = 0; // On reset, set to 0

else

q = d; // Otherwise out = d

endmodule

// 1 bit register

module RegBit(BitOut, BitData, WriteEn,reset, clk);

output BitOut; // 1 bit of register

input BitData, WriteEn;

input reset,clk;

wire d,f1, f2; // input of D Flip-Flop

wire reset;

//assign reset=0;

and #(50) U1(f1, BitOut, (~WriteEn));

and #(50) U2(f2, BitData, WriteEn);

or #(50) U3(d, f1, f2);

D\_FF DFF0(BitOut, d, reset, clk);

endmodule

//32 bit register

module register(RegOut,RegIn,WriteEn,reset,clk);

output [31:0] RegOut;

input [31:0] RegIn;

input WriteEn,reset, clk;

RegBit bit31(RegOut[31],RegIn[31],WriteEn,reset,clk);

RegBit bit30(RegOut[30],RegIn[30],WriteEn,reset,clk);

RegBit bit29(RegOut[29],RegIn[29],WriteEn,reset,clk);

RegBit bit28(RegOut[28],RegIn[28],WriteEn,reset,clk);

RegBit bit27(RegOut[27],RegIn[27],WriteEn,reset,clk);

RegBit bit26(RegOut[26],RegIn[26],WriteEn,reset,clk);

RegBit bit25(RegOut[25],RegIn[25],WriteEn,reset,clk);

RegBit bit24(RegOut[24],RegIn[24],WriteEn,reset,clk);

RegBit bit23(RegOut[23],RegIn[23],WriteEn,reset,clk);

RegBit bit22(RegOut[22],RegIn[22],WriteEn,reset,clk);

RegBit bit21(RegOut[21],RegIn[21],WriteEn,reset,clk);

RegBit bit20(RegOut[20],RegIn[20],WriteEn,reset,clk);

RegBit bit19(RegOut[19],RegIn[19],WriteEn,reset,clk);

RegBit bit18(RegOut[18],RegIn[18],WriteEn,reset,clk);

RegBit bit17(RegOut[17],RegIn[17],WriteEn,reset,clk);

RegBit bit16(RegOut[16],RegIn[16],WriteEn,reset,clk);

RegBit bit15(RegOut[15],RegIn[15],WriteEn,reset,clk);

RegBit bit14(RegOut[14],RegIn[14],WriteEn,reset,clk);

RegBit bit13(RegOut[13],RegIn[13],WriteEn,reset,clk);

RegBit bit12(RegOut[12],RegIn[12],WriteEn,reset,clk);

RegBit bit11(RegOut[11],RegIn[11],WriteEn,reset,clk);

RegBit bit10(RegOut[10],RegIn[10],WriteEn,reset,clk);

RegBit bit9 (RegOut[9], RegIn[9], WriteEn,reset,clk);

RegBit bit8 (RegOut[8], RegIn[8], WriteEn,reset,clk);

RegBit bit7 (RegOut[7], RegIn[7], WriteEn,reset,clk);

RegBit bit6 (RegOut[6], RegIn[6], WriteEn,reset,clk);

RegBit bit5 (RegOut[5], RegIn[5], WriteEn,reset,clk);

RegBit bit4 (RegOut[4], RegIn[4], WriteEn,reset,clk);

RegBit bit3 (RegOut[3], RegIn[3], WriteEn,reset,clk);

RegBit bit2 (RegOut[2], RegIn[2], WriteEn,reset,clk);

RegBit bit1 (RegOut[1], RegIn[1], WriteEn,reset,clk);

RegBit bit0 (RegOut[0], RegIn[0], WriteEn,reset,clk);

endmodule

// Decoder

module decoder(WriteEn,RegWrite, WriteRegister);

input RegWrite;

input [4:0] WriteRegister;

output [31:0] WriteEn;

wire [31:0] OE; // Output Enable

dec5to32 dec(OE,WriteRegister);

assign WriteEn[0]=0;

and #(50) gate1(WriteEn[1],OE[1],RegWrite);

and #(50) gate2(WriteEn[2],OE[2],RegWrite);

and #(50) gate3(WriteEn[3],OE[3],RegWrite);

and #(50) gate4(WriteEn[4],OE[4],RegWrite);

and #(50) gate5(WriteEn[5],OE[5],RegWrite);

and #(50) gate6(WriteEn[6],OE[6],RegWrite);

and #(50) gate7(WriteEn[7],OE[7],RegWrite);

and #(50) gate8(WriteEn[8],OE[8],RegWrite);

and #(50) gate9(WriteEn[9],OE[9],RegWrite);

and #(50) gate10(WriteEn[10],OE[10],RegWrite);

and #(50) gate11(WriteEn[11],OE[11],RegWrite);

and #(50) gate12(WriteEn[12],OE[12],RegWrite);

and #(50) gate13(WriteEn[13],OE[13],RegWrite);

and #(50) gate14(WriteEn[14],OE[14],RegWrite);

and #(50) gate15(WriteEn[15],OE[15],RegWrite);

and #(50) gate16(WriteEn[16],OE[16],RegWrite);

and #(50) gate17(WriteEn[17],OE[17],RegWrite);

and #(50) gate18(WriteEn[18],OE[18],RegWrite);

and #(50) gate19(WriteEn[19],OE[19],RegWrite);

and #(50) gate20(WriteEn[20],OE[20],RegWrite);

and #(50) gate21(WriteEn[21],OE[21],RegWrite);

and #(50) gate22(WriteEn[22],OE[22],RegWrite);

and #(50) gate23(WriteEn[23],OE[23],RegWrite);

and #(50) gate24(WriteEn[24],OE[24],RegWrite);

and #(50) gate25(WriteEn[25],OE[25],RegWrite);

and #(50) gate26(WriteEn[26],OE[26],RegWrite);

and #(50) gate27(WriteEn[27],OE[27],RegWrite);

and #(50) gate28(WriteEn[28],OE[28],RegWrite);

and #(50) gate29(WriteEn[29],OE[29],RegWrite);

and #(50) gate30(WriteEn[30],OE[30],RegWrite);

and #(50) gate31(WriteEn[31],OE[31],RegWrite);

endmodule

module andmore(g,a,b,c,d,e);

output g;

input a,b,c,d,e;

and #(50) and1(f1,a,b,c,d),

and2(g,f1,e);

endmodule

module dec5to32(Out,Adr);

input [4:0] Adr; // Adr=Address of register

output [31:0] Out;

not #(50) Inv4(Nota, Adr[4]);

not #(50) Inv3(Notb, Adr[3]);

not #(50) Inv2(Notc, Adr[2]);

not #(50) Inv1(Notd, Adr[1]);

not #(50) Inv0(Note, Adr[0]);

andmore a0(Out[0], Nota,Notb,Notc,Notd,Note); // 00000

andmore a1(Out[1], Nota,Notb,Notc,Notd,Adr[0]); // 00001

andmore a2(Out[2], Nota,Notb,Notc,Adr[1],Note); //00010

andmore a3(Out[3], Nota,Notb,Notc,Adr[1],Adr[0]);

andmore a4(Out[4], Nota,Notb,Adr[2],Notd,Note);

andmore a5(Out[5], Nota,Notb,Adr[2],Notd,Adr[0]);

andmore a6(Out[6], Nota,Notb,Adr[2],Adr[1],Note);

andmore a7(Out[7], Nota,Notb,Adr[2],Adr[1],Adr[0]);

andmore a8(Out[8], Nota,Adr[3],Notc,Notd,Note);

andmore a9(Out[9], Nota,Adr[3],Notc,Notd,Adr[0]);

andmore a10(Out[10], Nota,Adr[3],Notc,Adr[1],Note);

andmore a11(Out[11], Nota,Adr[3],Notc,Adr[1],Adr[0]);

andmore a12(Out[12], Nota,Adr[3],Adr[2],Notd,Note);

andmore a13(Out[13], Nota,Adr[3],Adr[2],Notd,Adr[0]);

andmore a14(Out[14], Nota,Adr[3],Adr[2],Adr[1],Note);

andmore a15(Out[15], Nota,Adr[3],Adr[2],Adr[1],Adr[0]);

andmore a16(Out[16], Adr[4],Notb,Notc,Notd,Note);

andmore a17(Out[17], Adr[4],Notb,Notc,Notd,Adr[0]);

andmore a18(Out[18], Adr[4],Notb,Notc,Adr[1],Note);

andmore a19(Out[19], Adr[4],Notb,Notc,Adr[1],Adr[0]);

andmore a20(Out[20], Adr[4],Notb,Adr[2],Notd,Note);

andmore a21(Out[21], Adr[4],Notb,Adr[2],Notd,Adr[0]);

andmore a22(Out[22], Adr[4],Notb,Adr[2],Adr[1],Note);

andmore a23(Out[23], Adr[4],Notb,Adr[2],Adr[1],Adr[0]);

andmore a24(Out[24], Adr[4],Adr[3],Notc,Notd,Note);

andmore a25(Out[25], Adr[4],Adr[3],Notc,Notd,Adr[0]);

andmore a26(Out[26], Adr[4],Adr[3],Notc,Adr[1],Note);

andmore a27(Out[27], Adr[4],Adr[3],Notc,Adr[1],Adr[0]);

andmore a28(Out[28], Adr[4],Adr[3],Adr[2],Notd,Note);

andmore a29(Out[29], Adr[4],Adr[3],Adr[2],Notd,Adr[0]);

andmore a30(Out[30], Adr[4],Adr[3],Adr[2],Adr[1],Note);

andmore a31(Out[31], Adr[4],Adr[3],Adr[2],Adr[1],Adr[0]); // 11111

endmodule

//------------module multiplexor 32 to 1----------------

module mux32to1(Out, In , Select);

output Out;

input [31:0] In;

input [4:0] Select;

wire [31:0] OE,f; // OE = Output Enable

dec5to32 dec1(OE,Select);

and #(50) g\_0(f[0],OE[0],In[0]);

and #(50) g\_1(f[1],OE[1],In[1]);

and #(50) g\_2(f[2],OE[2],In[2]);

and #(50) g\_3(f[3],OE[3],In[3]);

and #(50) g\_4(f[4],OE[4],In[4]);

and #(50) g\_5(f[5],OE[5],In[5]);

and #(50) g\_6(f[6],OE[6],In[6]);

and #(50) g\_7(f[7],OE[7],In[7]);

and #(50) g\_8(f[8],OE[8],In[8]);

and #(50) g\_9(f[9],OE[9],In[9]);

and #(50) g\_10(f[10],OE[10],In[10]);

and #(50) g\_11(f[11],OE[11],In[11]);

and #(50) g\_12(f[12],OE[12],In[12]);

and #(50) g\_13(f[13],OE[13],In[13]);

and #(50) g\_14(f[14],OE[14],In[14]);

and #(50) g\_15(f[15],OE[15],In[15]);

and #(50) g\_16(f[16],OE[16],In[16]);

and #(50) g\_17(f[17],OE[17],In[17]);

and #(50) g\_18(f[18],OE[18],In[18]);

and #(50) g\_19(f[19],OE[19],In[19]);

and #(50) g\_20(f[20],OE[20],In[20]);

and #(50) g\_21(f[21],OE[21],In[21]);

and #(50) g\_22(f[22],OE[22],In[22]);

and #(50) g\_23(f[23],OE[23],In[23]);

and #(50) g\_24(f[24],OE[24],In[24]);

and #(50) g\_25(f[25],OE[25],In[25]);

and #(50) g\_26(f[26],OE[26],In[26]);

and #(50) g\_27(f[27],OE[27],In[27]);

and #(50) g\_28(f[28],OE[28],In[28]);

and #(50) g\_29(f[29],OE[29],In[29]);

and #(50) g\_30(f[30],OE[30],In[30]);

and #(50) g\_31(f[31],OE[31],In[31]);

or #(50) gate3(g3,f[0],f[1],f[2],f[3]);

or #(50) gate4(g4,f[4],f[5],f[6],f[7]);

or #(50) gate5(g5,f[8],f[9],f[10],f[11]);

or #(50) gate6(g6,f[12],f[13],f[14],f[15]);

or #(50) gate7(g7,f[16],f[17],f[18],f[19]);

or #(50) gate8(g8,f[20],f[21],f[22],f[23]);

or #(50) gate9(g9,f[24],f[25],f[26],f[27]);

or #(50) gate10(g10,f[28],f[29],f[30],f[31]);

or #(50) gate11(g11,g3,g4,g5,g6);

or #(50) gate12(g12,g7,g8,g9,10);

or #(50) gate(Out,g11,g12);

endmodule

module mux32x32to32(ReadData,In0, In1,In2,In3,In4,In5,In6,In7,In8,In9,In10,In11,In12,In13,In14,In15,In16,In17,In18,In19,In20,In21,In22,In23, In24,In25,In26,In27,In28,In29,In30,In31,ReadRegister);

input [31:0] In0, In1,In2,In3,In4,In5,In6,In7,In8,In9,In10,In11,In12,In13,In14,In15,In16,In17,In18,In19,In20,In21,In22,In23,In24,In25,In26,In27,In28,In29,In30,In31;

input [4:0] ReadRegister;

output [31:0] ReadData;

reg [31:0] ArrayReg [0:31];

integer j;

always @(\*)

begin

for (j=0;j<=31;j=j+1)

ArrayReg[j] = {In31[j], In30[j],In29[j],In28[j],In27[j],In26[j],In25[j],In24[j],In23[j],In22[j],In21[j],

In20[j],In19[j],In18[j],In17[j],In16[j],In15[j],In14[j],In13[j],In12[j],In11[j],

In10[j],In9[j],In8[j],In7[j],In6[j],In5[j],In4[j],In3[j],In2[j],In1[j],In0[j]};

end

mux32to1 mux0(ReadData[0],ArrayReg[0],ReadRegister);

mux32to1 mux1(ReadData[1],ArrayReg[1],ReadRegister);

mux32to1 mux2(ReadData[2],ArrayReg[2],ReadRegister);

mux32to1 mux3(ReadData[3],ArrayReg[3],ReadRegister);

mux32to1 mux4(ReadData[4],ArrayReg[4],ReadRegister);

mux32to1 mux5(ReadData[5],ArrayReg[5],ReadRegister);

mux32to1 mux6(ReadData[6],ArrayReg[6],ReadRegister);

mux32to1 mux7(ReadData[7],ArrayReg[7],ReadRegister);

mux32to1 mux8(ReadData[8],ArrayReg[8],ReadRegister);

mux32to1 mux9(ReadData[9],ArrayReg[9],ReadRegister);

mux32to1 mux10(ReadData[10],ArrayReg[10],ReadRegister);

mux32to1 mux11(ReadData[11],ArrayReg[11],ReadRegister);

mux32to1 mux12(ReadData[12],ArrayReg[12],ReadRegister);

mux32to1 mux13(ReadData[13],ArrayReg[13],ReadRegister);

mux32to1 mux14(ReadData[14],ArrayReg[14],ReadRegister);

mux32to1 mux15(ReadData[15],ArrayReg[15],ReadRegister);

mux32to1 mux16(ReadData[16],ArrayReg[16],ReadRegister);

mux32to1 mux17(ReadData[17],ArrayReg[17],ReadRegister);

mux32to1 mux18(ReadData[18],ArrayReg[18],ReadRegister);

mux32to1 mux19(ReadData[19],ArrayReg[19],ReadRegister);

mux32to1 mux20(ReadData[20],ArrayReg[20],ReadRegister);

mux32to1 mux21(ReadData[21],ArrayReg[21],ReadRegister);

mux32to1 mux22(ReadData[22],ArrayReg[22],ReadRegister);

mux32to1 mux23(ReadData[23],ArrayReg[23],ReadRegister);

mux32to1 mux24(ReadData[24],ArrayReg[24],ReadRegister);

mux32to1 mux25(ReadData[25],ArrayReg[25],ReadRegister);

mux32to1 mux26(ReadData[26],ArrayReg[26],ReadRegister);

mux32to1 mux27(ReadData[27],ArrayReg[27],ReadRegister);

mux32to1 mux28(ReadData[28],ArrayReg[28],ReadRegister);

mux32to1 mux29(ReadData[29],ArrayReg[29],ReadRegister);

mux32to1 mux30(ReadData[30],ArrayReg[30],ReadRegister);

mux32to1 mux31(ReadData[31],ArrayReg[31],ReadRegister);

endmodule

`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Verilog code for ALU

module alu(Output, CarryOut, zero, overflow, negative, BussA, BussB, ALUControl);

output CarryOut,overflow,negative,zero;

output [31:0] Output;

input [31:0] BussA,BussB;

input [1:0] ALUControl;

wire lessthan;

wire [31:0] crrout;

alu1bit alu0(Output[0],crrout[0],BussA[0],BussB[0],ALUControl[1],lessthan,ALUControl);

alu1bit alu1(Output[1],crrout[1],BussA[1],BussB[1],crrout[0],1'b0,ALUControl);

alu1bit alu2(Output[2],crrout[2],BussA[2],BussB[2],crrout[1],1'b0,ALUControl);

alu1bit alu3(Output[3],crrout[3],BussA[3],BussB[3],crrout[2],1'b0,ALUControl);

alu1bit alu4(Output[4],crrout[4],BussA[4],BussB[4],crrout[3],1'b0,ALUControl);

alu1bit alu5(Output[5],crrout[5],BussA[5],BussB[5],crrout[4],1'b0,ALUControl);

alu1bit alu6(Output[6],crrout[6],BussA[6],BussB[6],crrout[5],1'b0,ALUControl);

alu1bit alu7(Output[7],crrout[7],BussA[7],BussB[7],crrout[6],1'b0,ALUControl);

alu1bit alu8(Output[8],crrout[8],BussA[8],BussB[8],crrout[7],1'b0,ALUControl);

alu1bit alu9(Output[9],crrout[9],BussA[9],BussB[9],crrout[8],1'b0,ALUControl);

alu1bit alu10(Output[10],crrout[10],BussA[10],BussB[10],crrout[9],1'b0,ALUControl);

alu1bit alu11(Output[11],crrout[11],BussA[11],BussB[11],crrout[10],1'b0,ALUControl);

alu1bit alu12(Output[12],crrout[12],BussA[12],BussB[12],crrout[11],1'b0,ALUControl);

alu1bit alu13(Output[13],crrout[13],BussA[13],BussB[13],crrout[12],1'b0,ALUControl);

alu1bit alu14(Output[14],crrout[14],BussA[14],BussB[14],crrout[13],1'b0,ALUControl);

alu1bit alu15(Output[15],crrout[15],BussA[15],BussB[15],crrout[14],1'b0,ALUControl);

alu1bit alu16(Output[16],crrout[16],BussA[16],BussB[16],crrout[15],1'b0,ALUControl);

alu1bit alu17(Output[17],crrout[17],BussA[17],BussB[17],crrout[16],1'b0,ALUControl);

alu1bit alu18(Output[18],crrout[18],BussA[18],BussB[18],crrout[17],1'b0,ALUControl);

alu1bit alu19(Output[19],crrout[19],BussA[19],BussB[19],crrout[18],1'b0,ALUControl);

alu1bit alu20(Output[20],crrout[20],BussA[20],BussB[20],crrout[19],1'b0,ALUControl);

alu1bit alu21(Output[21],crrout[21],BussA[21],BussB[21],crrout[20],1'b0,ALUControl);

alu1bit alu22(Output[22],crrout[22],BussA[22],BussB[22],crrout[21],1'b0,ALUControl);

alu1bit alu23(Output[23],crrout[23],BussA[23],BussB[23],crrout[22],1'b0,ALUControl);

alu1bit alu24(Output[24],crrout[24],BussA[24],BussB[24],crrout[23],1'b0,ALUControl);

alu1bit alu25(Output[25],crrout[25],BussA[25],BussB[25],crrout[24],1'b0,ALUControl);

alu1bit alu26(Output[26],crrout[26],BussA[26],BussB[26],crrout[25],1'b0,ALUControl);

alu1bit alu27(Output[27],crrout[27],BussA[27],BussB[26],crrout[26],1'b0,ALUControl);

alu1bit alu28(Output[28],crrout[28],BussA[28],BussB[28],crrout[27],1'b0,ALUControl);

alu1bit alu29(Output[29],crrout[29],BussA[29],BussB[29],crrout[28],1'b0,ALUControl);

alu1bit alu30(Output[30],crrout[30],BussA[30],BussB[30],crrout[29],1'b0,ALUControl);

alu1bit alu31(Output[31],crrout[31],BussA[31],BussB[31],crrout[30],1'b0,ALUControl);

not #(50) notcarry(notcr31,crrout[31]);

// Carryout = Not carry out 31 if it is subtraction

mux21 muxcarry31(CarryOut,crrout[31],notcr31,ALUControl[1]);

xor #(50) xor5(overflow,crrout[30],crrout[31]);

// SLT

addsub add2(addsub31Out,crrout31,BussA[31],BussB[31],crrout[30],ALUControl[1]);

xor #(50) xor6(lessthan,overflow,addsub31Out);

assign negative = Output[31];

or #(50) or1(o1,Output[0],Output[1],Output[2],Output[3]);

or #(50) or2(o2,Output[4],Output[5],Output[6],Output[7]);

or #(50) or3(o3,Output[8],Output[9],Output[10],Output[11]);

or #(50) or4(o4,Output[12],Output[13],Output[14],Output[15]);

or #(50) or5(o5,Output[16],Output[17],Output[18],Output[19]);

or #(50) or6(o6,Output[20],Output[21],Output[22],Output[23]);

or #(50) or7(o7,Output[24],Output[25],Output[26],Output[27]);

or #(50) or8(o8,Output[28],Output[29],Output[30],Output[31]);

or #(50) or9(o9,o1,o2,o3,o4);

or #(50) or10(o10,o5,o6,o7,o8);

nor #(50) nor1(zero,o9,o10);

endmodule

`timescale 1 ps / 100 fs

module alu1bit(result,crrout,a,b,carryin,less,ALUControl);

output result,crrout;

input a,b,carryin,less;

input [1:0] ALUControl;

addsub add1(addsubOut,crrout,a,b,carryin,ALUControl[1]);

xor #(50) xor1(xorOut,a,b);

mux21 mux2(xorlessOut,xorOut,less,ALUControl[1]);

mux21 mux3(result,addsubOut,xorlessOut,ALUControl[0]);

endmodule

`timescale 1 ps / 100 fs

module addsub(Out,cout,a,b,cin,select);

input a,b,cin,select;

output Out,cout; // the result and carry out

not #(50) not1(notb,b);

mux21 mux1(b1,b,notb,select);

adder adder1(Out,cout,a,b1,cin);

endmodule

`timescale 1 ps / 100 fs

module adder(sum,cout,a,b,cin);

input a,b,cin;

output cout,sum;

// sum = a xor b xor cin

xor #(50) (sum,a,b,cin);

// carry out = a.b + cin.(a+b)

and #(50) and1(c1,a,b);

or #(50) or1(c2,a,b);

and #(50) and2(c3,c2,cin);

or #(50) or2(cout,c1,c3);

endmodule

`timescale 1 ps / 100 fs

module mux21(O,A,B,sel);

// sel = 0 thi O = A

// sel = 1 thi O =B

output O;

input A,B,sel;

not #(50) not1(nsel,sel);

and #(50) and1(O1,A,nsel);

and #(50) and2(O2,B,sel);

or #(50) or2(O,O1,O2);

endmodule

`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Data Memory

module dataMem(data, address, writedata, writeenable,MemRead,clk);

input [31:0] address, writedata;

input writeenable,MemRead, clk;

output [31:0] data;

reg [7:0] datamem[1023:0];

reg [31:0] temp;

buf #1000 buf0(data[0],temp[0]),

buf1(data[1],temp[1]),

buf2(data[2],temp[2]),

buf3(data[3],temp[3]),

buf4(data[4],temp[4]),

buf5(data[5],temp[5]),

buf6(data[6],temp[6]),

buf7(data[7],temp[7]),

buf8(data[8],temp[8]),

buf9(data[9],temp[9]),

buf10(data[10],temp[10]),

buf11(data[11],temp[11]),

buf12(data[12],temp[12]),

buf13(data[13],temp[13]),

buf14(data[14],temp[14]),

buf15(data[15],temp[15]),

buf16(data[16],temp[16]),

buf17(data[17],temp[17]),

buf18(data[18],temp[18]),

buf19(data[19],temp[19]),

buf20(data[20],temp[20]),

buf21(data[21],temp[21]),

buf22(data[22],temp[22]),

buf23(data[23],temp[23]),

buf24(data[24],temp[24]),

buf25(data[25],temp[25]),

buf26(data[26],temp[26]),

buf27(data[27],temp[27]),

buf28(data[28],temp[28]),

buf29(data[29],temp[29]),

buf30(data[30],temp[30]),

buf31(data[31],temp[31]);

always @(posedge clk)

if (writeenable)

begin

datamem[address]=writedata[31:24];

datamem[address+1]=writedata[23:16];

datamem[address+2]=writedata[15:8];

datamem[address+3]=writedata[7:0];

end

always @(address or datamem[address] or datamem[address+1] or datamem[address+2] or datamem[address+3])

begin

temp={datamem[address],datamem[address+1],datamem[address+2],datamem[address+3]};

end

// initial

// begin

// $readmemh("data.dat", datamem);

// end

endmodule

`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Zero-Extension

module zero\_extend(zOut32,zIn16);

output [31:0] zOut32;

input [15:0] zIn16;

assign zOut32 = {{16{1'b0}},zIn16};

endmodule

`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// mux2x32to32

module mux2x32to32( DataOut,Data0, Data1, Select);

output [31:0] DataOut; // Data Out

input [31:0] Data0, Data1; // Data In 1 and 2

input Select;

// neu Select = 0 thi DataOut = Data0

// nguoc lai thi DataOut = Data1

mux2\_1 mux0(DataOut[0],Data0[0],Data1[0],Select);

mux2\_1 mux1(DataOut[1],Data0[1],Data1[1],Select);

mux2\_1 mux2(DataOut[2],Data0[2],Data1[2],Select);

mux2\_1 mux3(DataOut[3],Data0[3],Data1[3],Select);

mux2\_1 mux4(DataOut[4],Data0[4],Data1[4],Select);

mux2\_1 mux5(DataOut[5],Data0[5],Data1[5],Select);

mux2\_1 mux6(DataOut[6],Data0[6],Data1[6],Select);

mux2\_1 mux7(DataOut[7],Data0[7],Data1[7],Select);

mux2\_1 mux8(DataOut[8],Data0[8],Data1[8],Select);

mux2\_1 mux9(DataOut[9],Data0[9],Data1[9],Select);

mux2\_1 mux10(DataOut[10],Data0[10],Data1[10],Select);

mux2\_1 mux11(DataOut[11],Data0[11],Data1[11],Select);

mux2\_1 mux12(DataOut[12],Data0[12],Data1[12],Select);

mux2\_1 mux13(DataOut[13],Data0[13],Data1[13],Select);

mux2\_1 mux14(DataOut[14],Data0[14],Data1[14],Select);

mux2\_1 mux15(DataOut[15],Data0[15],Data1[15],Select);

mux2\_1 mux16(DataOut[16],Data0[16],Data1[16],Select);

mux2\_1 mux17(DataOut[17],Data0[17],Data1[17],Select);

mux2\_1 mux18(DataOut[18],Data0[18],Data1[18],Select);

mux2\_1 mux19(DataOut[19],Data0[19],Data1[19],Select);

mux2\_1 mux20(DataOut[20],Data0[20],Data1[20],Select);

mux2\_1 mux21(DataOut[21],Data0[21],Data1[21],Select);

mux2\_1 mux22(DataOut[22],Data0[22],Data1[22],Select);

mux2\_1 mux23(DataOut[23],Data0[23],Data1[23],Select);

mux2\_1 mux24(DataOut[24],Data0[24],Data1[24],Select);

mux2\_1 mux25(DataOut[25],Data0[25],Data1[25],Select);

mux2\_1 mux26(DataOut[26],Data0[26],Data1[26],Select);

mux2\_1 mux27(DataOut[27],Data0[27],Data1[27],Select);

mux2\_1 mux28(DataOut[28],Data0[28],Data1[28],Select);

mux2\_1 mux29(DataOut[29],Data0[29],Data1[29],Select);

mux2\_1 mux30(DataOut[30],Data0[30],Data1[30],Select);

mux2\_1 mux31(DataOut[31],Data0[31],Data1[31],Select);

endmodule

`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Sign-Extension

module sign\_extend(sOut32,sIn16);

output [31:0] sOut32;

input [15:0] sIn16;

assign sOut32 = {{16{sIn16[15]}},sIn16};

endmodule

// Shift left 2 module

module shift\_left\_2(Out32, In32);

output [31:0] Out32;

input [31:0] In32;

assign Out32 = {In32[29:0],2'b00};

endmodule

`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// mux2x5to5

// Select Write Register

module mux2x5to5( AddrOut,Addr0, Addr1, Select);

output [4:0] AddrOut; // Address Out

input [4:0] Addr0, Addr1; // Address In 1 and 2

input Select;

mux21 mux0(AddrOut[0],Addr0[0],Addr1[0],Select);

mux21 mux1(AddrOut[1],Addr0[1],Addr1[1],Select);

mux21 mux2(AddrOut[2],Addr0[2],Addr1[2],Select);

mux21 mux3(AddrOut[3],Addr0[3],Addr1[3],Select);

mux21 mux4(AddrOut[4],Addr0[4],Addr1[4],Select);

endmodule

module mux2\_1(O,A,B,sel);

// sel = 0 : O = A

// sel = 1 : O =B

output O;

input A,B,sel;

not #(50) not1(nsel,sel);

and #(50) and1(O1,A,nsel);

and #(50) and2(O2,B,sel);

or #(50) or2(O,O1,O2);

endmodule

`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Control unit

module Control(

RegDst,

ALUSrc,

MemtoReg,

RegWrite,

MemRead,

MemWrite,

Branch,

ALUOp,

Jump,

SignZero,

Opcode

);

output RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,Jump,SignZero;

output [1:0] ALUOp;

input [5:0] Opcode;

reg RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,Jump,SignZero;

reg [1:0] ALUOp;

always @(\*)

casex (Opcode)

6'b000000 : begin // R - type

RegDst = 1'b1;

ALUSrc = 1'b0;

MemtoReg= 1'b0;

RegWrite= 1'b1;

MemRead = 1'b0;

MemWrite= 1'b0;

Branch = 1'b0;

ALUOp = 2'b10;

Jump = 1'b0;

SignZero= 1'b0;

end

6'b100011 : begin // lw - load word

RegDst = 1'b0;

ALUSrc = 1'b1;

MemtoReg= 1'b1;

RegWrite= 1'b1;

MemRead = 1'b1;

MemWrite= 1'b0;

Branch = 1'b0;

ALUOp = 2'b00;

Jump = 1'b0;

SignZero= 1'b0; // sign extend

end

6'b101011 : begin // sw - store word

RegDst = 1'bx;

ALUSrc = 1'b1;

MemtoReg= 1'bx;

RegWrite= 1'b0;

MemRead = 1'b0;

MemWrite= 1'b1;

Branch = 1'b0;

ALUOp = 2'b00;

Jump = 1'b0;

SignZero= 1'b0;

end

6'b000101 : begin // bne - branch if not equal

RegDst = 1'b0;

ALUSrc = 1'b0;

MemtoReg= 1'b0;

RegWrite= 1'b0;

MemRead = 1'b0;

MemWrite= 1'b0;

Branch = 1'b1;

ALUOp = 2'b01;

Jump = 1'b0;

SignZero= 1'b0; // sign extend

end

6'b001110 : begin // XORI - XOR immidiate

RegDst = 1'b0;

ALUSrc = 1'b1;

MemtoReg= 1'b0;

RegWrite= 1'b1;

MemRead = 1'b0;

MemWrite= 1'b0;

Branch = 1'b0;

ALUOp = 2'b11;

Jump = 1'b0;

SignZero= 1'b1; // zero extend

end

6'b000010 : begin // j - Jump

RegDst = 1'b0;

ALUSrc = 1'b0;

MemtoReg= 1'b0;

RegWrite= 1'b0;

MemRead = 1'b0;

MemWrite= 1'b0;

Branch = 1'b0;

ALUOp = 2'b00;

Jump = 1'b1;

SignZero= 1'b0;

end

default : begin

RegDst = 1'b0;

ALUSrc = 1'b0;

MemtoReg= 1'b0;

RegWrite= 1'b0;

MemRead = 1'b0;

MemWrite= 1'b0;

Branch = 1'b0;

ALUOp = 2'b10;

Jump = 1'b0;

SignZero= 1'b0;

end

endcase

endmodule

`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// ALU Control unit

module ALUControl\_Block( ALUControl, ALUOp, Function);

output [1:0] ALUControl;

reg [1:0] ALUControl;

input [1:0] ALUOp;

input [5:0] Function;

wire [7:0] ALUControlIn;

assign ALUControlIn = {ALUOp,Function};

always @(ALUControlIn)

casex (ALUControlIn)

8'b11xxxxxx: ALUControl=2'b01;

8'b00xxxxxx: ALUControl=2'b00;

8'b01xxxxxx: ALUControl=2'b10;

8'b10100000: ALUControl=2'b00;

8'b10100010: ALUControl=2'b10;

8'b10101010: ALUControl=2'b11;

default: ALUControl=2'b00;

endcase

endmodule

`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Control singals for JR instruction

module JRControl\_Block( JRControl, ALUOp, Function);

output JRControl;

reg JRControl;

input [1:0] ALUOp;

input [5:0] Function;

wire [7:0] test;

assign test = {ALUOp,Function};

always @(test)

case (test)

8'b10001000 : JRControl=1'b1;

default: JRControl=1'b0;

endcase

endmodule

`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Forwarding Unit

module ForwardingUnit(ForwardA,ForwardB,MEM\_RegWrite,WB\_RegWrite,MEM\_WriteRegister,WB\_WriteRegister,EX\_rs,EX\_rt);

output [1:0] ForwardA,ForwardB;

wire [1:0] ForwardA,ForwardB;

input MEM\_RegWrite,WB\_RegWrite;

input [4:0] MEM\_WriteRegister,WB\_WriteRegister,EX\_rs,EX\_rt;

// a= 1 if ( MEM\_WriteRegister != 0 )

or #(50) orMEM\_WriteReg(a,MEM\_WriteRegister[4],MEM\_WriteRegister[3],MEM\_WriteRegister[2],MEM\_WriteRegister[1],MEM\_WriteRegister[0]);

CompareAddress CompMEM\_WriteReg\_EXrs(b,MEM\_WriteRegister,EX\_rs);

and #(50) andx(x,MEM\_RegWrite,a,b);

// x=1 if ((MEM\_RegWrite==1)&&(MEM\_WriteRegister != 0)&&(MEM\_WriteRegister==EX\_rs))

// c= 1 if ( WB\_WriteRegister != 0 )

or #(50) orWB\_WriteReg(c,WB\_WriteRegister[4],WB\_WriteRegister[3],WB\_WriteRegister[2],WB\_WriteRegister[1],WB\_WriteRegister[0]);

CompareAddress CompWB\_WriteReg\_EXrs(d,WB\_WriteRegister,EX\_rs);

and #(50) andy(y,WB\_RegWrite,c,d);

// y=1 if ((WB\_RegWrite==1)&&(WB\_WriteRegister != 0)&&(WB\_WriteRegister==EX\_rs))

// ForwardA[1] = x; va ForwardA[0] = (NOT x). y ;

assign ForwardA[1] = x;

not #(50) notxgate(notx,x);

and #(50) NOTxANDy(ForwardA[0],notx,y);

// ForwardB

CompareAddress CompMEM\_WriteReg\_EXrt(b1,MEM\_WriteRegister,EX\_rt);

CompareAddress CompWB\_WriteReg\_EXrt(d1,WB\_WriteRegister,EX\_rt);

and #(50) andx1(x1,MEM\_RegWrite,a,b1);

and #(50) andy1(y1,WB\_RegWrite,c,d1);

assign ForwardB[1] = x1;

not #(50) notx1gate(notx1,x1);

and #(50) NOTx1ANDy1(ForwardB[0],notx1,y1);

endmodule

`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// mux3x32to32

module mux3x32to32(DataOut,A,B,C,Select);

output [31:0] DataOut;

input [1:0] Select;

input [31:0] A,B,C;

wire [31:0] DataOut1,DataOut2;

mux2x32to32 muxAB(DataOut1,A,B, Select[1]);

mux2x32to32 muxCA(DataOut2,C,A, Select[1]);

mux2x32to32 muxABC(DataOut,DataOut1,DataOut2, Select[0]);

endmodule

`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Stall Control

module StallControl(PC\_WriteEn,IFID\_WriteEn,Stall\_flush,EX\_MemRead,EX\_rt,ID\_rs,ID\_rt,ID\_Op);

output PC\_WriteEn,IFID\_WriteEn,Stall\_flush;

wire PC\_WriteEn,IFID\_WriteEn,Stall\_flush;

input EX\_MemRead,EX\_rt,ID\_rs,ID\_rt;

input [5:0] ID\_Op;

wire [4:0] EX\_rt,ID\_rs,ID\_rt,xorRsRt,xorRtRt;

wire [5:0] xoropcodelw,xoropcodexori;

wire EX\_MemRead;

//wire xoropcode1,xoroprt;

// write in behavior model

/\*always @(EX\_MemRead or EX\_rt or ID\_rs or ID\_rt)

begin

if ((EX\_MemRead==1)&&((EX\_rt==ID\_rs)||((EX\_rt==ID\_rt)&&(Opcode!= 6'b001110)&&(Opcode!= 6'b100011)))

begin

PC\_WriteEn=1'b0;

IFID\_WriteEn=1'b0;

Stall\_flush =1'b1;

end

else

begin

PC\_WriteEn=1'b1;

IFID\_WriteEn=1'b1;

Stall\_flush =1'b0;

end

end

\*/

// write in structural model

xor #(50) xorRsRt4(xorRsRt[4],EX\_rt[4],ID\_rs[4]);

xor #(50) xorRsRt3(xorRsRt[3],EX\_rt[3],ID\_rs[3]);

xor #(50) xorRsRt2(xorRsRt[2],EX\_rt[2],ID\_rs[2]);

xor #(50) xorRsRt1(xorRsRt[1],EX\_rt[1],ID\_rs[1]);

xor #(50) xorRsRt0(xorRsRt[0],EX\_rt[0],ID\_rs[0]);

or #(50) OrRsRt1(OrRsRt,xorRsRt[4],xorRsRt[3],xorRsRt[2],xorRsRt[1],xorRsRt[0]);

not #(50) notgate1(notOrRsRt,OrRsRt);

// neu EX\_rt==ID\_rs thi notOrRsRt = 1

xor #(50) xorRtRt4(xorRtRt[4],EX\_rt[4],ID\_rt[4]);

xor #(50) xorRtRt3(xorRtRt[3],EX\_rt[3],ID\_rt[3]);

xor #(50) xorRtRt2(xorRtRt[2],EX\_rt[2],ID\_rt[2]);

xor #(50) xorRtRt1(xorRtRt[1],EX\_rt[1],ID\_rt[1]);

xor #(50) xorRtRt0(xorRtRt[0],EX\_rt[0],ID\_rt[0]);

or #(50) OrRtRt1(OrRtRt,xorRtRt[4],xorRtRt[3],xorRtRt[2],xorRtRt[1],xorRtRt[0]);

not #(50) notgate2(notOrRtRt,OrRtRt);

// neu EX\_rt==ID\_rt thi notOrRtRt = 1

xor #(50) xoropcode5(xoropcodelw[5],ID\_Op[5],1'b1);

xor #(50) xoropcode4(xoropcodelw[4],ID\_Op[4],1'b0);

xor #(50) xoropcode3(xoropcodelw[3],ID\_Op[3],1'b0);

xor #(50) xoropcode2(xoropcodelw[2],ID\_Op[2],1'b0);

xor #(50) xoropcode1(xoropcodelw[1],ID\_Op[1],1'b1);

xor #(50) xoropcode0(xoropcodelw[0],ID\_Op[0],1'b1);

or #(50) oropcode1(ec1,xoropcodelw[5],xoropcodelw[4],xoropcodelw[3],xoropcodelw[2],xoropcodelw[1],xoropcodelw[0]);

// opcode != opcode[lw] xoropcodelw =1

xor #(50) xoropcod5(xoropcodexori[5],ID\_Op[5],1'b0);

xor #(50) xoropcod4(xoropcodexori[4],ID\_Op[4],1'b0);

xor #(50) xoropcod3(xoropcodexori[3],ID\_Op[3],1'b1);

xor #(50) xoropcod2(xoropcodexori[2],ID\_Op[2],1'b1);

xor #(50) xoropcod1(xoropcodexori[1],ID\_Op[1],1'b1);

xor #(50) xoropcod0(xoropcodexori[0],ID\_Op[0],1'b0);

or #(50) oropcode2(ec2,xoropcodexori[5],xoropcodexori[4],xoropcodexori[3],xoropcodexori[2],xoropcodexori[1],xoropcodexori[0]);

// opcode != opcode[xori] xoropcodexori =1

and #(50) and1(xorop,ec1,ec2);

and #(50) and2(xoroprt,xorop,notOrRtRt);

or #(50) OrEXIDRsRt(OrOut,notOrRsRt,xoroprt);

and #(50) AndCondition(Condition,EX\_MemRead,OrOut);

// Condition =1 when stall is satisfied

not #(50) NotPC\_WriteEn(PC\_WriteEn,Condition);

not #(50) NotIFID\_WriteEn(IFID\_WriteEn,Condition);

buf #(50) bufStallflush(Stall\_flush,Condition);

endmodule

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Flush control signals

`timescale 1 ps / 100 fs

module flush\_block(

ID\_RegDst,ID\_ALUSrc, ID\_MemtoReg,ID\_RegWrite,ID\_MemRead,ID\_MemWrite,

ID\_Branch,ID\_ALUOp,ID\_JRControl,flush,RegDst,ALUSrc,MemtoReg,RegWrite,

MemRead,MemWrite,Branch,ALUOp,JRControl);

output ID\_RegDst,ID\_ALUSrc,ID\_MemtoReg,ID\_RegWrite,ID\_MemRead,ID\_MemWrite,ID\_Branch,ID\_JRControl;

output [1:0] ID\_ALUOp;

input flush,RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,JRControl;

input [1:0] ALUOp;

not #50 (notflush,flush);

and #50 and1(ID\_RegDst,RegDst,notflush);

and #50 and2(ID\_ALUSrc,ALUSrc,notflush);

and #50 and3(ID\_MemtoReg,MemtoReg,notflush);

and #50 and4(ID\_RegWrite,RegWrite,notflush);

and #50 and5(ID\_MemRead,MemRead,notflush);

and #50 and6(ID\_MemWrite,MemWrite,notflush);

and #50 and7(ID\_Branch,Branch,notflush);

and #50 and8(ID\_JRControl,JRControl,notflush);

and #50 and9(ID\_ALUOp[1],ALUOp[1],notflush);

and #50 and10(ID\_ALUOp[0],ALUOp[0],notflush);

endmodule

`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Discard instructions when needed

module Discard\_Instr(ID\_flush,IF\_flush,jump,bne,jr);

output ID\_flush,IF\_flush;

input jump,bne,jr;

or #50 OR1(IF\_flush,jump,bne,jr);

or #50 OR2(ID\_flush,bne,jr);

endmodule

`timescale 1 ps / 100 fs

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Write Back Forwarding

module WB\_forward(ReadData1Out,ReadData2Out,ReadData1,ReadData2,rs,rt,WriteRegister,WriteData,RegWrite);

// WB Hazard: Reading data while writing

// Solve Hazard at the WriteBack Stage

output [31:0] ReadData1Out,ReadData2Out;

input [31:0] ReadData1,ReadData2,WriteData;

input [4:0] rs,rt,WriteRegister;

input RegWrite;

wire ReadSourceRs,ReadSourceRt;

wire compOut1,compOut2;

// behavior model

/\*

always @(rs or rt or WriteRegister or WriteData or RegWrite)

begin

if ((RegWrite==1)&&(WriteRegister != 0)&&(WriteRegister==rs))

ReadSourceRs = 1'b1; //Forwarding WriteData to ReadData1

else

ReadSourceRs = 1'b0;

if ((RegWrite==1)&&(WriteRegister != 0)&&(WriteRegister==rt))

ReadSourceRt = 1'b1; //Forwarding WriteData to ReadData2

else

ReadSourceRt = 1'b0;

end

\*/

// Structural model

or #(50) orWriteReg(orOut1,WriteRegister[4],WriteRegister[3],WriteRegister[2],WriteRegister[1],WriteRegister[0]);

CompareAddress Compare1(compOut1,WriteRegister,rs);

and #(50) andCondition1(ReadSourceRs,RegWrite,orOut1,compOut1);

CompareAddress Compare2(compOut2,WriteRegister,rt);

and #(50) andCondition2(ReadSourceRt,RegWrite,orOut1,compOut2);

mux2x32to32 muxReadData1( ReadData1Out,ReadData1,WriteData, ReadSourceRs);

mux2x32to32 muxReadData2( ReadData2Out,ReadData2,WriteData, ReadSourceRt);

endmodule

`timescale 1 ps / 100 fs

module CompareAddress(equal,Addr1,Addr2);

// fpga4student.com: FPGA projects, Verilog Projects, VHDL projects

// Verilog project: 32-bit 5-stage Pipelined MIPS Processor in Verilog

// Compare Address

output equal;

wire equal;

input [4:0] Addr1,Addr2;

wire [4:0] Addr1,Addr2,xorAddress;

xor #(50) xorAddress4(xorAddress[4],Addr1[4],Addr2[4]);

xor #(50) xorAddress3(xorAddress[3],Addr1[3],Addr2[3]);

xor #(50) xorAddress2(xorAddress[2],Addr1[2],Addr2[2]);

xor #(50) xorAddress1(xorAddress[1],Addr1[1],Addr2[1]);

xor #(50) xorAddress0(xorAddress[0],Addr1[0],Addr2[0]);

or #(50) Orgate1(OrAddr,xorAddress[4],xorAddress[3],xorAddress[2],xorAddress[1],xorAddress[0]);

not #(50) notgate1(equal,OrAddr);

endmodule

**Appendix B - MIPSpipeline\_tb.sv**

`timescale 1ns/1ns

module test;

reg clk, reset;

int note, cycleNum = 0;

MIPSpipeline DUT(.clk(clk), .reset(reset));

initial begin

clk = 0;

reset = 1;

#25

reset = 0;

end

initial begin

note = $fopen("output.txt");

end

always begin

#5

clk = ~clk;

end

always begin

#500

$fclose(note);

end

always @(posedge clk)

begin

$fwrite(note, "-----------------------------------------------------------|\n");

$fwrite(note, "Cycle: %d |\n", cycleNum);

$fwrite(note, "----------------PROGRAM COUNTER----------------------------|\n");

$fwrite(note, "Current PC: %3d |\n", DUT.PC);

$fwrite(note, "Next PC: %3d |\n",DUT.PCin);

#4

$fwrite(note, "----------------INSTRUCTION--------------------------------|\n");

$fwrite(note, "Instruction: %32b |\n", DUT.Instruction);

$fwrite(note, "----------------REGISTERS----------------------------------|\n");

$fwrite(note, "---------------|BINARY--------------------------|DECIMAL---|\n");

$fwrite(note, "R12: |%b|%d|\n", DUT.Register\_File.RegArray[12],DUT.Register\_File.RegArray[12]);

$fwrite(note, "R13: |%b|%d|\n", DUT.Register\_File.RegArray[13],DUT.Register\_File.RegArray[13]);

$fwrite(note, "R14: |%b|%d|\n", DUT.Register\_File.RegArray[14],DUT.Register\_File.RegArray[14]);

$fwrite(note, "R15: |%b|%d|\n", DUT.Register\_File.RegArray[15],DUT.Register\_File.RegArray[15]);

$fwrite(note, "-----------------------ALU---------------------------------|\n");

$fwrite(note, "EX ALU Result: %32b |\n", DUT.EX\_ALUResult);

$fwrite(note, "WB ALU Result: %32b |\n", DUT.WB\_ALUResult);

$fwrite(note, "----------------BRANCH/JUMPS-------------------------------|\n");

$fwrite(note, "Branch: %b |\n", DUT.Branch);

$fwrite(note, "Jump: %b |\n", DUT.Jump);

$fwrite(note, "----------------FLAGS--------------------------------------|\n");

$fwrite(note, "Zero Flag: %b |\n", DUT.ZeroFlag);

$fwrite(note, "Overflow Flag: %b |\n", DUT.OverflowFlag);

$fwrite(note, "Carry Flag: %b |\n", DUT.CarryFlag);

$fwrite(note, "Negative Flag: %b |\n", DUT.NegativeFlag);

$fwrite(note, "not Zero Flag: %b |\n", DUT.notZeroFlag);

$fwrite(note, "~ ~ ~ ~ ~ ~ ~ ~ END OF CYCLE :) ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~|\n");

$fwrite(note, "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

$fwrite(note, "\n \n \n");

cycleNum++;

end

endmodule

**Bibliography**

[1] FPGA 4Student. Pipelined MIPS Processor in Verilog (Part-1). [online] Available at:

<https://www.fpga4student.com/2017/06/32-bit-pipelined-mips-processor-in-verilog-1.html>[Last Accessed, 29 Mar, 2019]

[2] FPGA 4Student. Pipelined MIPS Processor in Verilog (Part-2). [online] Available at:

<https://www.fpga4student.com/2017/06/32-bit-pipelined-mips-processor-in-verilog-2.html>[Last Accessed, 29 Mar, 2019]

[3] FPGA 4Student. Pipelined MIPS Processor in Verilog (Part-3). [online] Available at:

<https://www.fpga4student.com/2017/06/32-bit-pipelined-mips-processor-in-verilog-3.html> [Last Accessed, 29 Mar, 2019]

[4] Cornell University Pipelined Processor. Hakim Weatherspoon [online Powerpoint] Available at:

<https://www.cs.cornell.edu/courses/cs3410/2012sp/lecture/09-pipelined-cpu-i-g.pdf>[Last Accessed, 29 Mar, 2019]

[5] Tala, Deepak Kumar. System Task and Function, 1 Feb. 1970 [online] Available at:

[www.asic-world.com/verilog/sys\_task\_func1.html](http://www.asic-world.com/verilog/sys_task_func1.html). [Last Accessed, 29 Mar, 2019]

[6] Implementation of 32-Bit MIPS CPU on PYNQ-Z1 FPGA (Cover Page). [online] Available at: <https://docs.google.com/document/d/1mvOCYufiIo1EmEzPQfkLDpfEa1zp4e81THK64G7jdtA/edit?usp=sharing>. [Last Accessed, 31 Mar, 2019]

[7] Implementation of 32-Bit MIPS CPU on PYNQ-Z1 FPGA (Table of Contents). [online] Available at: <https://docs.google.com/document/d/1fA4sMpSnxDgmwepLQw9QyC1YCEs0kzP1cP3IaeHtXac/edit?usp=sharing.>[Last Accessed, 31 Mar, 2019]

[8] Implementation of 32-Bit MIPS CPU on PYNQ-Z1 FPGA (Paper). [online] Available at: <https://docs.google.com/document/d/1xrDEOm26zgcvV8N5UDWgOIaoghT4Lz3FTQbEpspRQkc/edit?usp=sharing>[Last Accessed, 31 Mar, 2019]

[9] Implementation of 32-Bit MIPS CPU on PYNQ-Z1 FPGA (Abstract). [online] Available at: <https://docs.google.com/document/d/1QP8msOKYGCNT_ZU8Xi54v1knF6hwBrfD1xOUFMZSdwI/edit?usp=sharing>. [Last Accessed, 31 Mar, 2019]