|  |  |  |
| --- | --- | --- |
| **0917439 Computer Organization Lab** | **Experiment 10** | **Fall 2023** |
| **Student#1 Name: Tasneem Qat** | **Reg ID: 0201502** | **Section: 2** |
| **Student#2 Name: Sarah Shaheen** | **Reg ID: 0207658** | **Section: 2** |

In this experiment, students have to modify the design of their pipelined processor implementation in experiment 9 to implement branch-if-greater-than-or-equal (bge) and load-upper-immediate (lui) instructions. Afterwards, you need to test your updated design using the program given in Table 3.

For the implementation of the **bge** instruction, the instruction assembly is: *bge rs1, rs2, immediate* and the machine code is as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 |
| imm[12], imm[10:5] | rs2 | rs1 | Funct3 = 101 | imm[4:1], imm[11] | Opcode = 1100011 |

The bge instruction is executed as follows:

if ((rs1) >= (rs2))

PC = target address = PC of bge + sign-extend(imm) \* 2

else

PC = PC + 4

For the implementation of the **lui** instruction, the instruction assembly is: *lui rd, immediate* and the machine code is as follows:

|  |  |  |
| --- | --- | --- |
| 31 12 | 11 7 | 6 0 |
| imm[31:12] | rd | Opcode = 0110111 |

The lui instruction is executed as follows:

(rd) = {imm[31:12], 12’b0}

Take into consideration the following hints:

* Similar to the beq instruction, the bge instruction should be resolved in the ID stage. Hence, if the bge instruction is taken then the instruction in the IF stage must be flushed.
* Similar to the R-type and I-type instructions, the result of lui instruction can be forwarded from the MEM or the WB stage. The tables below show examples of these cases:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Clock Cycle** | **1** | **2** | **3** | **4** | **5** | **6** |
| **LUI X5, 100** | **F** | **D** | **E** | **M** | **W** |  |
| **ADD X7, X5, X6** |  | **F** | **D** | **E** | **M** | **W** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Clock Cycle** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **LUI X5, 100** | **F** | **D** | **E** | **M** | **W** |  |  |
| **ADD X7, X3, X2** |  | **F** | **D** | **E** | **M** | **W** |  |
| **ADDI X9, X5, 129** |  |  | **F** | **D** | **E** | **M** | **W** |

Unlike previous experiments, this is not a guided experiment, i.e., students are free to make decisions on how to modify all modules in their processor. However, you are not allowed to remove any of the old instructions or start a new implementation from scratch. Students are expected to extend their processor implementation in the lab. In addition, you are free to choose between structural or behavioral modeling for your modifications.

**Step1:** In Table 1, describe the changes you did for each file (if any). Note that there are some files where modifications are not needed.

**Table 1:** Verilog files

|  |  |  |
| --- | --- | --- |
| **File** | **Modified (yes/no)?** | **Changes Description** |
| Library439.v | no |  |
| Exp 2 FA and MUX8\_1.v | no |  |
| Exp 2 ALUs.v | no |  |
| Exp 3 REG32 and MUX32\_1.v | no |  |
| Exp 3 Decoders and RegFile.v | no |  |
| Instruction\_memory.v | yes | Loaded the program in Table 3 |
| DataMem.v | no |  |
| ControlUnit.v | yes | * New 1-bit outputs ‘BGE’ and ‘LUI’. * Branch and alusrc output signal logic changed. * Logic for LUI and BGE added. * Aluop[1] logic addition for LUI logic. * Rformat logic condition addition. |
| Exp6 modules.v | yes | * Comparator32bit extra output for greater than functionality. * SignExtend unit extra input ‘LUI’, and else statement to implement rd = {imm[31:12], 12’b0}. |
| Exp7 modules.v | yes | * Extended the size of IDEX register by 1 bit to pass the LUI control unit output to be the select line for a new 2-to-1 mux controlling ALU’s first input (A). |
| Exp8 modules.v | no |  |
| Exp9 modules.v | no |  |
| Piplining\_Procssor.v | yes | * 1-bit wires for ‘greater’ signal, BGE signal, LUI signal, LUIX (LUI from control unit but in the execute stage). * Comparator32bit module gives extra 1-bit output ‘greater’. * IDEX register has extra input and ouput bit LUI and LUIX. * ALU input A now comes from a newly added 2-to-1 mux choosing between the original input from previous experiments and 12’b0 depending on the signal LUIX. * Control unit instance has 2 extra outputs ‘BGE’ and ‘LUI’. * New AND gate for BGE flush logic. * New OR gate for BGE flush logic. * Modification to OR gate responsible for Flush1. * SignExtend unit instance has extra input ‘LUI’. |

**Step2:** In Table 2, complete the new rows added for the **bge** and **lui** instructions. **In case** new control signals are needed, use the extra three columns given to show your changes to the control unit. **Highlight your changes in yellow, bold color**. You can also add new columns if needed.

**Table2:** Truth Table for the Control Unit

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **instruction** | **opcode** | **func3** | **func7** | **aluop[2]** | **aluop[1]** | **aluop[0]** | **alusrc** | **pcsrc[1]** | **pcsrc[0]** | **memtoreg[1]** | **memtoreg[0]** | **regwrite** | **memread** | **memwrite** | **branch** | **LUI** |  |  |
| **OR** | 0110011 | 110 | 0000000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |  |  |
| **AND** | 0110011 | 111 | 0000000 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |  |  |
| **XOR** | 0110011 | 100 | 0000000 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |  |  |
| **ADD** | 0110011 | 000 | 0000000 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |  |  |
| **SLT** | 0110011 | 010 | 0000000 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |  |  |
| **SUB** | 0110011 | 000 | 0100000 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |  |  |
| **ORI** | 0010011 | 110 | **-** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |  |  |
| **ANDI** | 0010011 | 111 | **-** | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |  |  |
| **XORI** | 0010011 | 100 | **-** | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |  |  |
| **ADDI** | 0010011 | 000 | **-** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |  |  |
| **SLTI** | 0010011 | 010 | **-** | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |  |  |
| **LW** | 0000011 | 010 | **-** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |  |  |
| **SW** | 0100011 | 010 | **-** | 0 | 1 | 1 | 1 | 0 | 0 | x | x | 0 | 0 | 1 | 0 | 0 |  |  |
| **BEQ** | 1100011 | 000 | **-** | x | x | x | x | 0 | 0 | x | x | 0 | 0 | 0 | 1 | 0 |  |  |
| **BGE** | **1100011** | **101** | **-** | x | x | x | x | 0 | 0 | x | x | 0 | 0 | 0 | 1 | 0 |  |  |
| **JAL** | 1101111 | **-** | **-** | x | x | x | x | 0 | 1 | 1 | 0 | 1 | 0 | 0 | x | 0 |  |  |
| **JALR** | 1100111 | 000 | **-** | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | x | 0 |  |  |
| **LUI** | **0110111** | **-** | **-** | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |  |  |

**Step3:** Table 3 shows the Verilog program students are required to use for test. Fill-in the machine codes for this program and also load the instruction memory with this program.

**Table 3:** The content of the instruction memory

|  |  |  |
| --- | --- | --- |
| **Address** | **Instruction** | **Machine Code** |
| **00** | **LW X1, 12(X0)** | 00C02083 h |
| **01** | **LW X2, 4(X0)** | 00402103 h |
| **02** | **LW X3, 100(X0)** | 06402183 h |
| **03** | **LUI X5, 5** | 000052B7 h |
| **04** | **ADD X6, X3, X5** | 00518333 h |
| **05** | **LUI X5, -1** | FFFFF2B7 h |
| **06** | **ADDI X9, X0, 3** | 00300493 h |
| **07** | **ADDI X7, X5, 2048** | 80028393 h |
| **08** | **BEQ X3, X6, 4** | 00618463 h |
| **09** | **BGE X6, X3, 8** | 00335863 h |
| **10** | **ADDI X7, X0, 25** | 01900393 h |
| **11** | **SW X6, 23(X7)** | 0063ABA3 h |
| **12** | **JALR X1, 60(X0)** | 03C000E7 h |
| **13** | **XOR X0, X1, X2** | 0020C033 h |
| **14** | **BGE X1, X9, -8** | FE90D8E3 h |
| **15** | **LW X5, 48(X0)** | 03002283 h |

**Step4:** Generate the timing diagram for the program in Table 3, with the below signals (in-order). Note that the test module is already given to you.

* Clock, reset, enable
* PC (the output of the program counter)
* Instruction (the output of the instruction memory)
* The output of the IF/ID register
* The output of the registers X1, X2, X3, X5, X6, X7, and X9
* **Memtoreg, regwrite, and all new control signals** in the decode stage

**Paste a screenshot of your timing diagram below. Make sure it is clear and readable.**

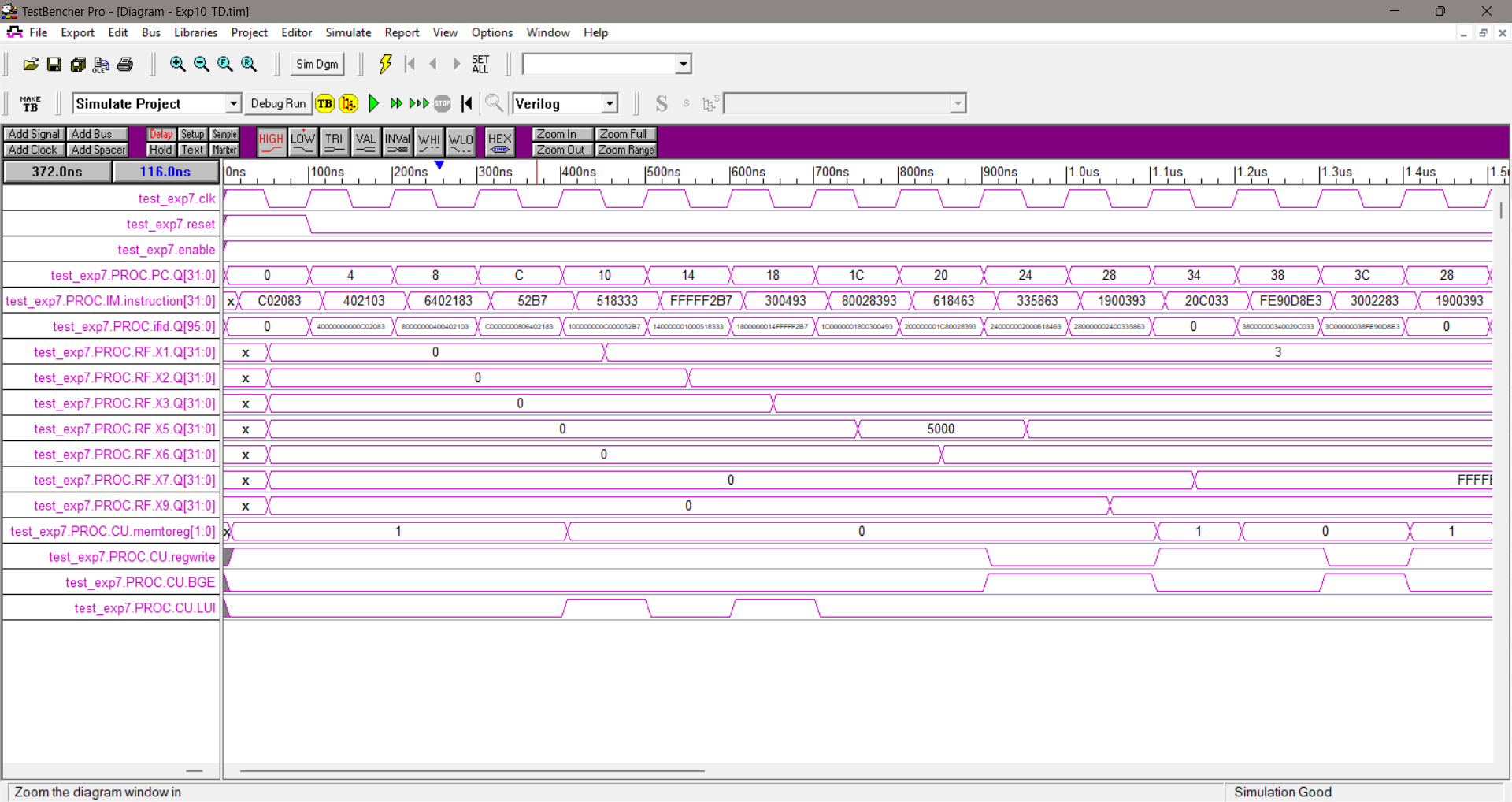
****

Figure 1: Timing diagram screenshot.

**A screenshot of a computer

Description automatically generated**

Figure 2: Timing diagram screenshot continued.

**Step5:** To make it easier for us to grade your work, in below, copy/paste all modified or new modules in your code. Also, highlight in bold font, the modified lines in the module. For example, if you made changes on the processor module, then copy/paste the entire processor module and highlight only the lines that have been modified. It is okay to use several pages for this step. However, please try your best to show your code in an easy-to-read fashion and avoid large fonts.

1. **Control Unit**

module ControlUnit(aluop, alusrc, pcsrc, memtoreg, regwrite ,memread, memwrite, branch, Iformat, LW, SW, BEQ, **BGE**, **LUI**, JAL, JALR, opcode, func3, func7);

input [6:0] opcode, func7;

input [2:0] func3;

output [2:0] aluop;

output [1:0] memtoreg, pcsrc;

output alusrc, regwrite, memread, memwrite, branch, Iformat, LW, SW, BEQ, **BGE**, **LUI**, JAL, JALR;

wire Rformat;

assign #2 LW = ~opcode[6] & ~opcode[5] & ~opcode[4];

assign #2 SW = ~opcode[6] & opcode[5] & ~opcode[4];

assign #2 Iformat = ~opcode[6] & ~opcode[5] & opcode[4];

assign #2 Rformat = ~opcode[6] & opcode[5] & opcode[4] & ~opcode[2];

assign #2 BEQ = opcode[6] & ~opcode[2];

assign #2 JAL = opcode[6] & opcode[3];

assign #2 JALR = opcode[6] & ~opcode[3] & opcode[2];

**assign #2 BGE = opcode[6] & ~opcode[2];**

**assign #2 LUI = opcode[2] & ~opcode[6];**

assign #4 memread = LW;

assign #4 memwrite = SW;

assign #4 branch = BEQ **|** **BGE**;

assign #4 alusrc = Iformat | LW | SW | JALR **|** **LUI**;

assign #4 regwrite = Rformat | Iformat | LW | JAL | JALR **|** **LUI**;

assign #4 memtoreg[1] = JAL | JALR;

assign #4 memtoreg[0] = LW;

assign #4 pcsrc[1] = JALR;

assign #4 pcsrc[0] = JAL;

assign #4 aluop[2] = (Rformat & (func7[5]|(~func3[2]&func3[1]))) | (Iformat & (~func3[2]&func3[1]));

//change in aluop[1] below, LUI added to logic condition

assign #4 aluop[1] = (Rformat & (func7[5]|(~func3[2]&func3[1])|(~func3[2]&~func3[1]&~func3[0])|(func3[2]&~func3[1]))) | (Iformat & ((~func3[2]&func3[1])|(~func3[2]&~func3[1]&~func3[0])|(func3[2]&~func3[1]))) | LW | SW | JALR **|** **LUI**;

assign #4 aluop[0] = (Rformat & (func7[5]|(~func3[2]&~func3[1])|(func3[1]&func3[0]))) | (Iformat & ((~func3[2]&~func3[1])|(func3[1]&func3[0]))) | LW | SW | JALR;

endmodule

1. **Exp6 modules**
2. **Sign extension unit:**

module SignExtend (SEout, in, Iformat, LW, SW, BEQ, **LUI**, JAL, JALR);

input [31:0] in;

input Iformat, LW, SW, BEQ, JAL, JALR, **LUI**;

output [31:0] SEout;

reg [31:0] SEout;

always @(in)

begin

#3;

if (Iformat || LW || JALR)

SEout = {{20{in[31]}},in[31:20]};

else if (SW)

SEout = {{20{in[31]}},in[31:25],in[11:7]};

else if (BEQ)

SEout = {{21{in[31]}},in[7],in[30:25],in[11:8]};

else if (JAL)

SEout = {{13{in[31]}},in[19:12],in[20],in[30:21]};

**else if(LUI)**

**SEout = {in[31:12], 12'b0};**

end

endmodule

1. **Comparator module:**

module Comparator32bit (equal, **greater**, a, b);

input [31:0] a, b;

output equal, **greater**;

reg equal, **greater**;

always @(a or b)

#10 begin

if(a==b)

equal = 1'b1;

else

equal = 1'b0;

**if(a>b)**

**greater = 1'b1;**

**else**

**greater = 1'b0;**

end

endmodule

1. **Exp7 modules**

module IDEX (Q, D, clk, reset, enable);

input clk, reset, enable;

input [**153**:0] D;

output [**153**:0] Q;

IFID r0(Q[95:0], D[95:0], clk, reset, enable);

ProgramCounter r1(Q[127:96], D[127:96], clk, reset, enable);

REG8 r2(Q[135:128], D[135:128], clk, reset, enable);

REG8 r3(Q[143:136], D[143:136], clk, reset, enable);

REG8 r4(Q[151:144], D[151:144], clk, reset, enable);

DFF r5(Q[152], D[152], clk, reset,enable);

**DFF r6(Q[153], D[153], clk, reset,enable);**

endmodule

1. **Pipelining Processor module**

module PipelinedProcessor(clk, reset, enable);

input clk, reset, enable;

wire [31:0] PCout, SEout, inst, PC4, PCin, writedata, readdata, readdata1, readdata2, ALUout, ALUinA, ALUin, ShiftOut, BTA, muxOut, readdata1X, readdata2X, SEoutX, mux1out;

wire [31:0] instF, PCoutF, PC4F, ALUoutM, readdata2M, readdataW, ALUoutW;

wire [31:0] PC4X, PC4M, PC4W, muxAout, muxBout;

wire [2:0] aluop, aluopX;

wire [1:0] memtoreg, pcsrc, memtoregX, memtoregM, memtoregW;

wire [1:0] pcsrcX, ForwardA, ForwardB;

wire [4:0] writereg, readreg1X, readreg2X, writeregX, writeregM;

wire Flush1, Flush2, memtoregAND, alusrc, regwrite, memread, memwrite, branch, Iformat, LW, SW, BEQ, BGE, LUI, LUIX, JAL, JALR, equal, greater, branchsel, branchsel2, greaterORequal, memwriteX, memreadX, alusrcX, mux1sel, regwriteX, regwriteM, memwriteM, memreadM, regwriteW;

//ProgramCounter (Q, D, clk, reset, enable);

ProgramCounter PC(PCoutF, PCin, clk, reset, enable);

//Instruction\_Memory(PC, instruction);

Instruction\_Memory IM (PCoutF, instF);

//Adder32bit (out, a, b); for PC + 4

Adder32bit PCadd4 (PC4F, PCoutF, 32'b0100);

//IFID (Q, D, clk, reset, enable);

IFID ifid({PC4, PCout, inst}, {PC4F, PCoutF, instF}, clk, Flush1, enable);

//OR3 (out, in1, in2, in3);

OR3 flush\_1(Flush1, reset, mux1sel, pcsrcX[1]);

//ControlUnit(aluop, …, JALR, opcode, func3, func7);

**ControlUnit CU** (aluop, alusrc, pcsrc, memtoreg, regwrite ,memread, memwrite, branch, Iformat, LW, SW, BEQ, **BGE**, **LUI**, JAL, JALR, inst[6:0], inst[14:12],inst[31:25]);

//SignExtend SE (SEout, inst, Iformat, LW, SW, BEQ, JAL, JALR);

SignExtend SE(SEout, inst, Iformat, LW, SW, BEQ, **LUI**, JAL, JALR);

//RegFile(readdata1 ,readdata2, ………, clk, reset);

RegFile RF (readdata1 ,readdata2, inst[19:15], inst[24:20], writedata,writereg , regwriteW, clk, reset);

//IDEX (Q, D, clk, reset, enable);

**IDEX idex**({PC4X[31:0], pcsrcX[1], **LUIX**, regwriteX, memtoregX[1:0], memwriteX, memreadX, aluopX[2:0], alusrcX, readdata1X[31:0], readdata2X[31:0], SEoutX[31:0], readreg2X[4:0], readreg1X[4:0], writeregX[4:0]}, {PC4[31:0], pcsrc[1], **LUI** ,regwrite, memtoreg[1:0], memwrite, memread, aluop[2:0], alusrc, readdata1[31:0], readdata2[31:0], SEout[31:0], inst[24:20], inst[19:15], inst[11:7]}, clk, Flush2, enable);

//OR (out, in1, in2);

OR flush\_2(Flush2, reset, pcsrcX[1]);

//ForwardingUnit(ForwardA, ForwardB, EXMEM\_Rd, MEMWB\_Rd, IDEX\_Rs1, IDEX\_Rs2, EXMEM\_RegWrite, MEMWB\_RegWrite);

ForwardingUnit FU(ForwardA, ForwardB, writeregM, writereg, readreg1X, readreg2X, regwriteM, regwriteW);

//Mux\_3\_to\_1\_32bit(out, s, i2, i1, i0);

Mux\_3\_to\_1\_32bit muxforwardA(muxAout, ForwardA, writedata, ALUoutM, readdata1X);

Mux\_3\_to\_1\_32bit muxforwardB(muxBout, ForwardB, writedata, ALUoutM, readdata2X);

**//Mux\_2\_to\_1\_32bit(out, s, i1, i0);**

**Mux\_2\_to\_1\_32bit ALU\_A (ALUinA, LUIX, 32'b0, muxAout);**

//Mux\_2\_to\_1\_32bit(out, s, i1, i0); for the input b of the ALU

Mux\_2\_to\_1\_32bit ALU\_B (ALUin, alusrcX, SEoutX, muxBout);

//ALU\_32(result, a, b, m);

**ALU\_32 AL**U (ALUout , **ALUinA**, ALUin , aluopX);

//ShiftLeft32\_by1(out, in);

ShiftLeft32\_by1 shifter (ShiftOut, SEout);

//Adder32bit (out, a, b); branch/jal target Address

Adder32bit BTAddress (BTA, PCout, ShiftOut);

//Comparator32bit (equal, **greater**, a, b);

**Comparator32bit comp** (equal, **greater**, readdata1, readdata2);

//AND (out, in1, in2);

AND andbranch (branchsel, branch, equal);

**//AND (out, in1, in2);**

**AND andbranch2(branchsel2, BGE, greater);**

**//OR (out, in1, in2);**

**OR branchselOR(greaterORequal, branchsel, branchsel2);**

//OR (out, in1, in2);

**OR mux1select**(mux1sel, **greaterORequal**, pcsrc[0]);

//Mux\_2\_to\_1\_32bit(out, s, i1, i0);

Mux\_2\_to\_1\_32bit mux1(mux1out, mux1sel, BTA, PC4F);

//Mux\_2\_to\_1\_32bit(out, s, i1, i0);

Mux\_2\_to\_1\_32bit mux2 (PCin, pcsrcX[1], ALUout, mux1out);

//EXMEM (Q, D, clk, reset, enable);

EXMEM exmem({PC4M[31:0], regwriteM, memtoregM[1:0], memwriteM, memreadM, ALUoutM[31:0], readdata2M[31:0], writeregM[4:0]}, {PC4X[31:0], regwriteX, memtoregX[1:0], memwriteX, memreadX, ALUout[31:0], muxBout, writeregX[4:0]}, clk, reset, enable);

//Data\_Memory(readdata , address, ……., clk );

Data\_Memory DM(readdata , ALUoutM, readdata2M , memwriteM , memreadM , clk);

//MEMWB (Q, D, clk, reset, enable)

MEMWB memwb({PC4W[31:0], regwriteW, memtoregW[1:0],readdataW[31:0], ALUoutW[31:0], writereg[4:0]}, {PC4M[31:0], regwriteM, memtoregM[1:0],readdata, ALUoutM[31:0], writeregM[4:0]}, clk, reset, enable);

//Mux\_3\_to\_1\_32bit(out, s, i2, i1, i0);

Mux\_3\_to\_1\_32bit muxFinal (writedata, memtoregW, PC4W, readdataW, ALUoutW);

endmodule