**Final Project and Extra Credit**

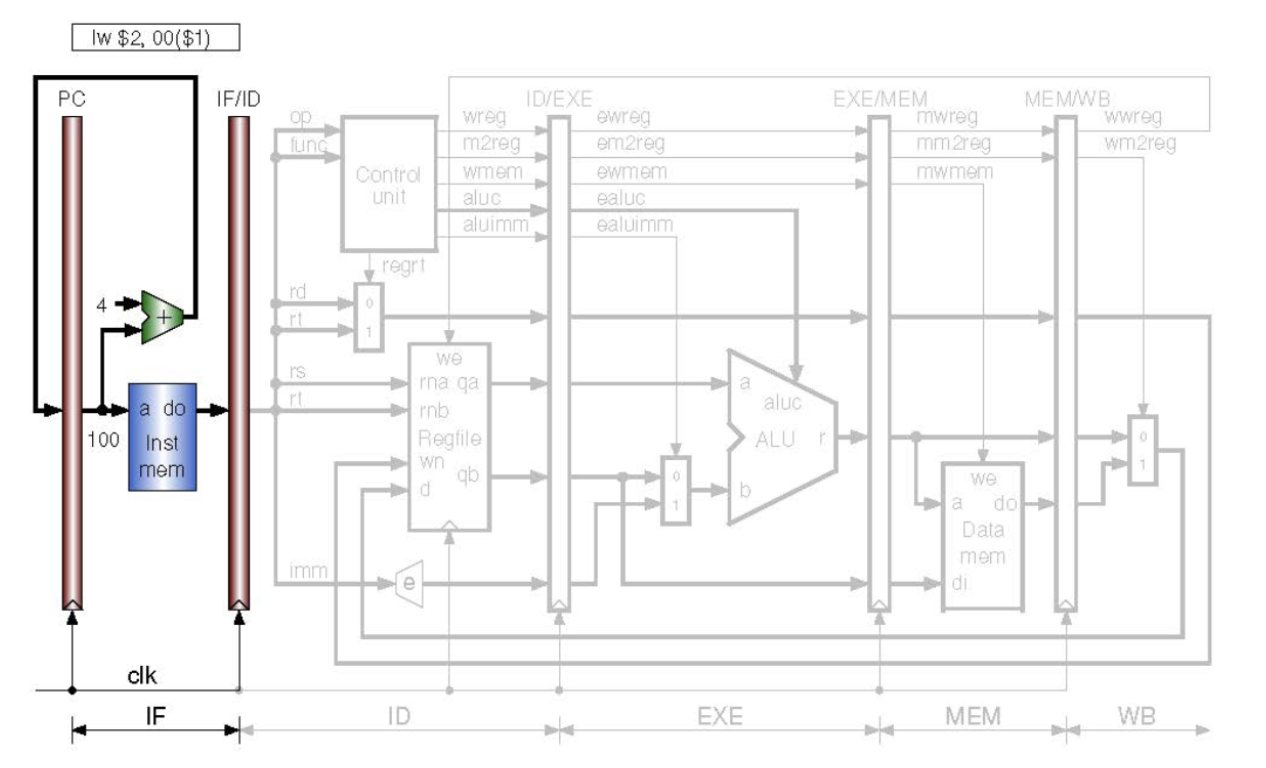
**CMPEN 331**

**Sethu Senthil**

**Abstract**

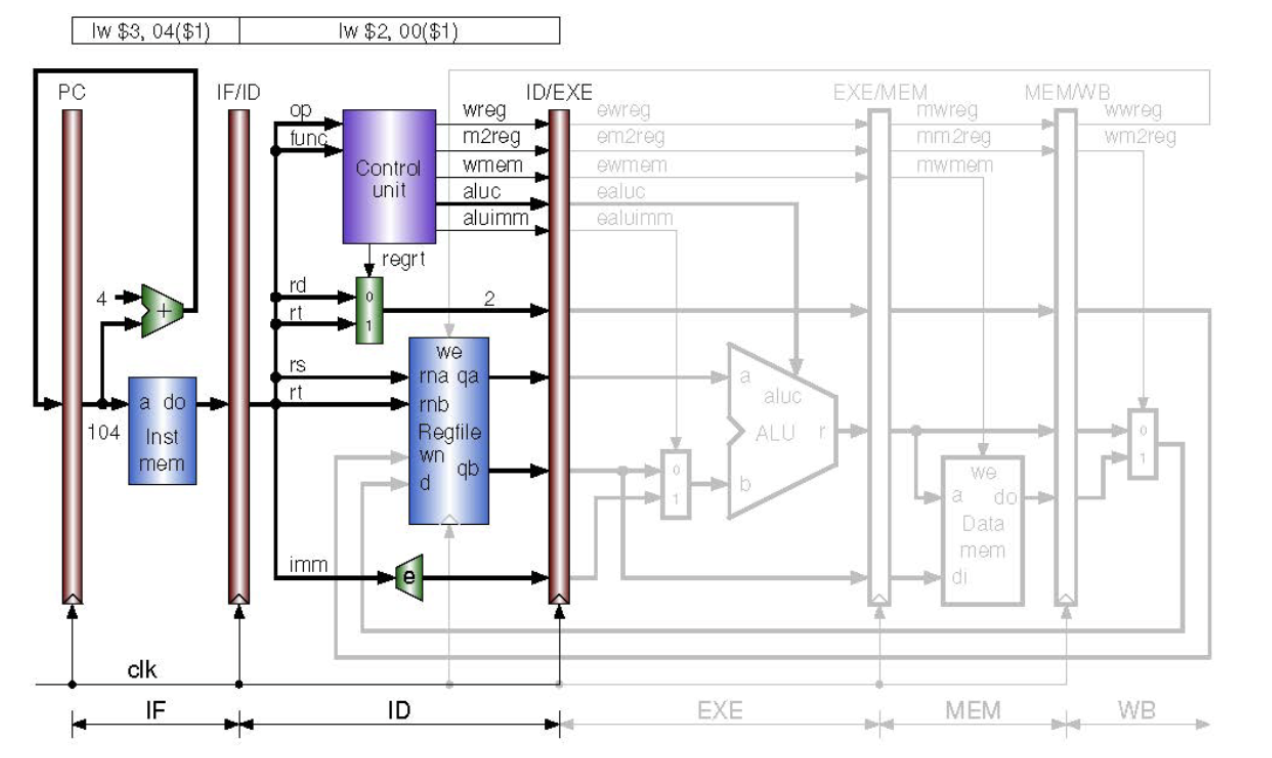
In this project we look at how an MIPS CPU works, focusing on its five main parts: Instruction Fetch (IF), Instruction Decode (ID), Execution (EXE), Memory (MEM), and Writeback (WB). It explains what each part does and how they work together. Additionally, it discusses improvements for handling different kinds of instructions and avoiding slowdowns. Overall, this paper provides a clear understanding of how a MIPS CPU operates, including its main parts, handling different instructions, and avoiding slowdowns.

**Stage 1: Instruction Fetch (IF)**

****

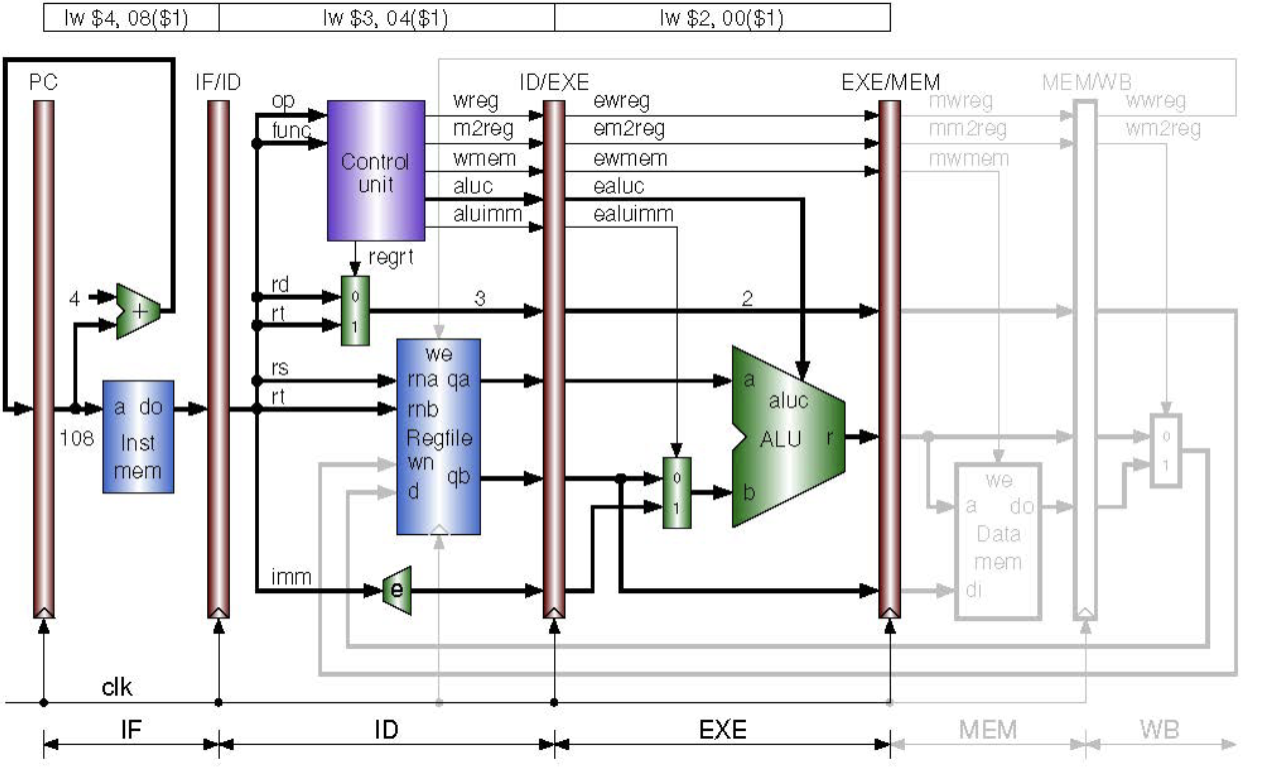
* In the initial stage of the pipeline, the program counter (PC) is generated from the clock signal produced in the upper module (testbench). Four modules are active in this stage, while the remaining are for later discussion.
* The first module, the program counter, controls the update of the program counter to the next value only on the rising edge of the clock signal. It begins at an initial value of 100.
* Next is the program counter adder, responsible for incrementing the program counter by 4 with every signal, as dictated by the program counter module.
* Following that is the Instruction Memory (Inst mem), storing instructions for CPU processing, starting from memory slot 25 to 29.
* The final component of this stage is the IFID Pipeline, acting as the separator between the instruction fetch and instruction decode stage. It passes the instruction memory output on the positive clock edge, as do all five pipelines.

**Stage 2: Instruction Decode (ID)**



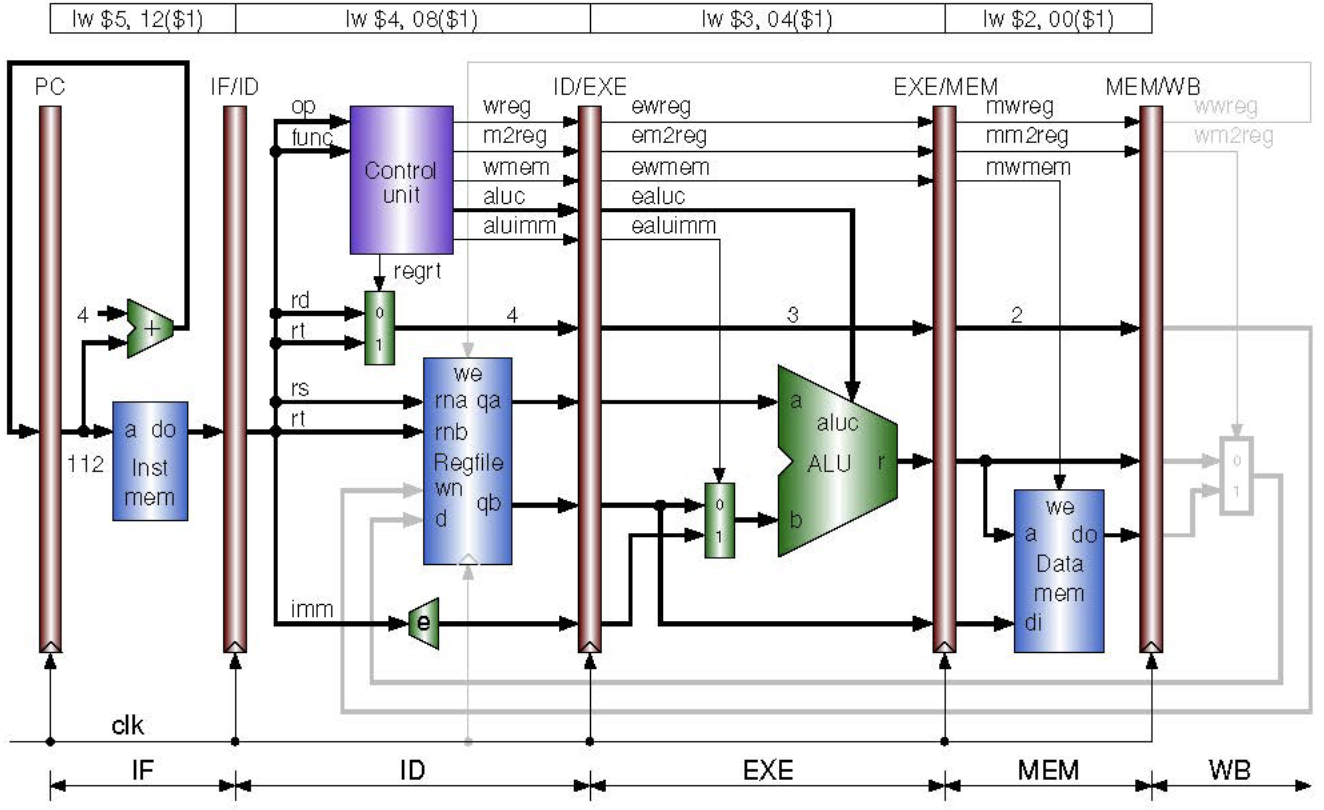
* In the second stage of the pipeline, the primary task is to decode the instructions from the previous stage. As the first instruction enters this stage (assuming no stalls), the next instruction is fetched in the instruction fetch stage. In a higher-level module, the instruction is dissected into its components (rs, rt, rd, op, func, imm, etc.), with wire sizes matching the requirements of each component.
* Firstly, the immediate extender processes the imm portion of the instruction, the last 16 bits, extending them to a 32-bit number. The most significant 16 bits are determined by the 16th bit of the least significant bits: if it's 1, the most significant bits are set to 1; if it's 0, they are set to 0.
* Secondly, the register file stores rs and rt in their designated register addresses dictated by the op code. The values are output as 32-bit addresses, qa for rs and qb for rt.
* Thirdly, the regrtMux selects rd or rt based on the value of regrt (if 1, rt is passed through; if 0, rs is passed through).
* Fourthly, the control unit, the "brain" of the CPU, determines the output signals for each instruction type. While only R-type instructions are covered here, load word was used in previous assignments. For the final project, instructions like add, or, and, xor, sub were covered. Based on the op code and function, the outputs are set accordingly. This is implemented using case statements for clarity and simplicity.
* Finally, this stage concludes with the IDEXE pipeline, which updates its outputs based on the inputs only at the positive edge of the clock.

**Stage 3: Execution (EXE)**

****

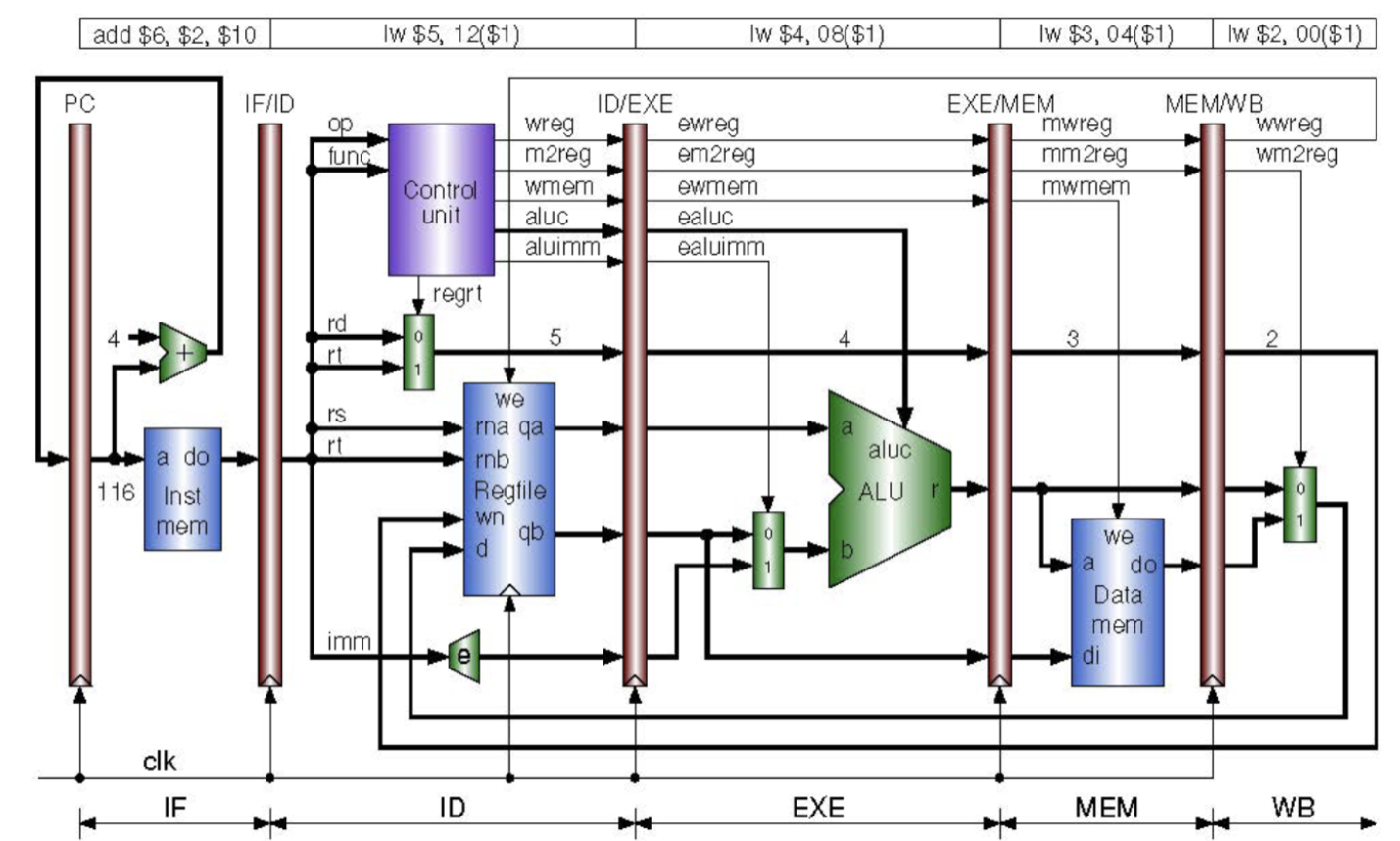
In this stage, we begin with the IDEXE pipeline from the previous stage. Following that, we encounter a mux right before the ALU, which selects between qb and the immediate extended value based on the ealuimm signal. The ALU then performs arithmetic operations on the selected values based on the aluc signal from the control unit. The resulting value (r) is passed out of the ALU. Finally, the EXEMEM pipeline updates the values on the positive edge of the clock, as in previous pipelines.

**Stage 4: Memory (MEM)**

****

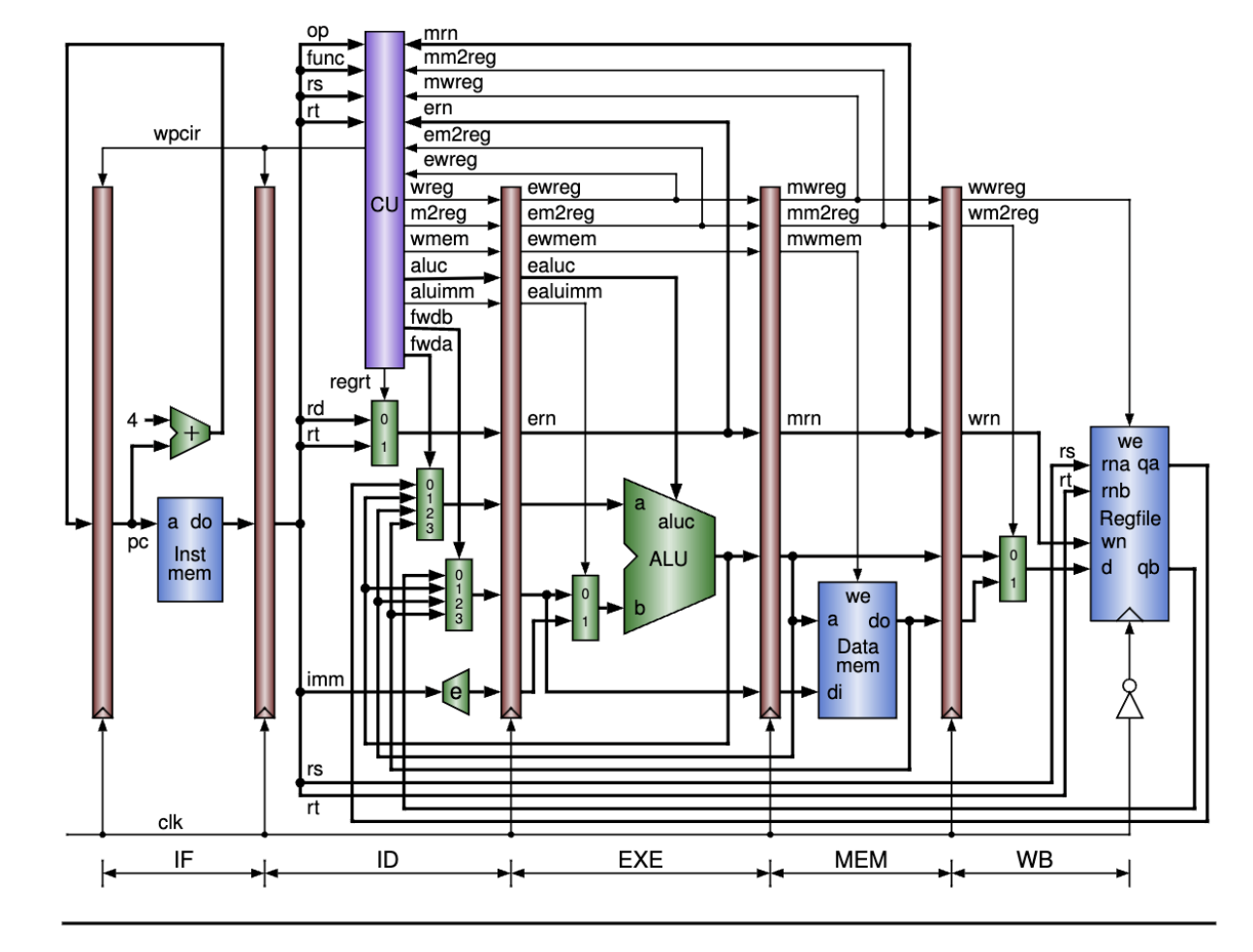
* In the fourth stage, the memory module stores the results of the ALU into memory if required. The EXEMEM pipeline, discussed previously, passes values only on the positive clock edge. Following that, the data memory module handles memory storage and register writing as necessary. Finally, the MEMWB pipeline updates the outputs of this stage on the positive clock edge.

**Stage 5: Writeback (WB)**

****

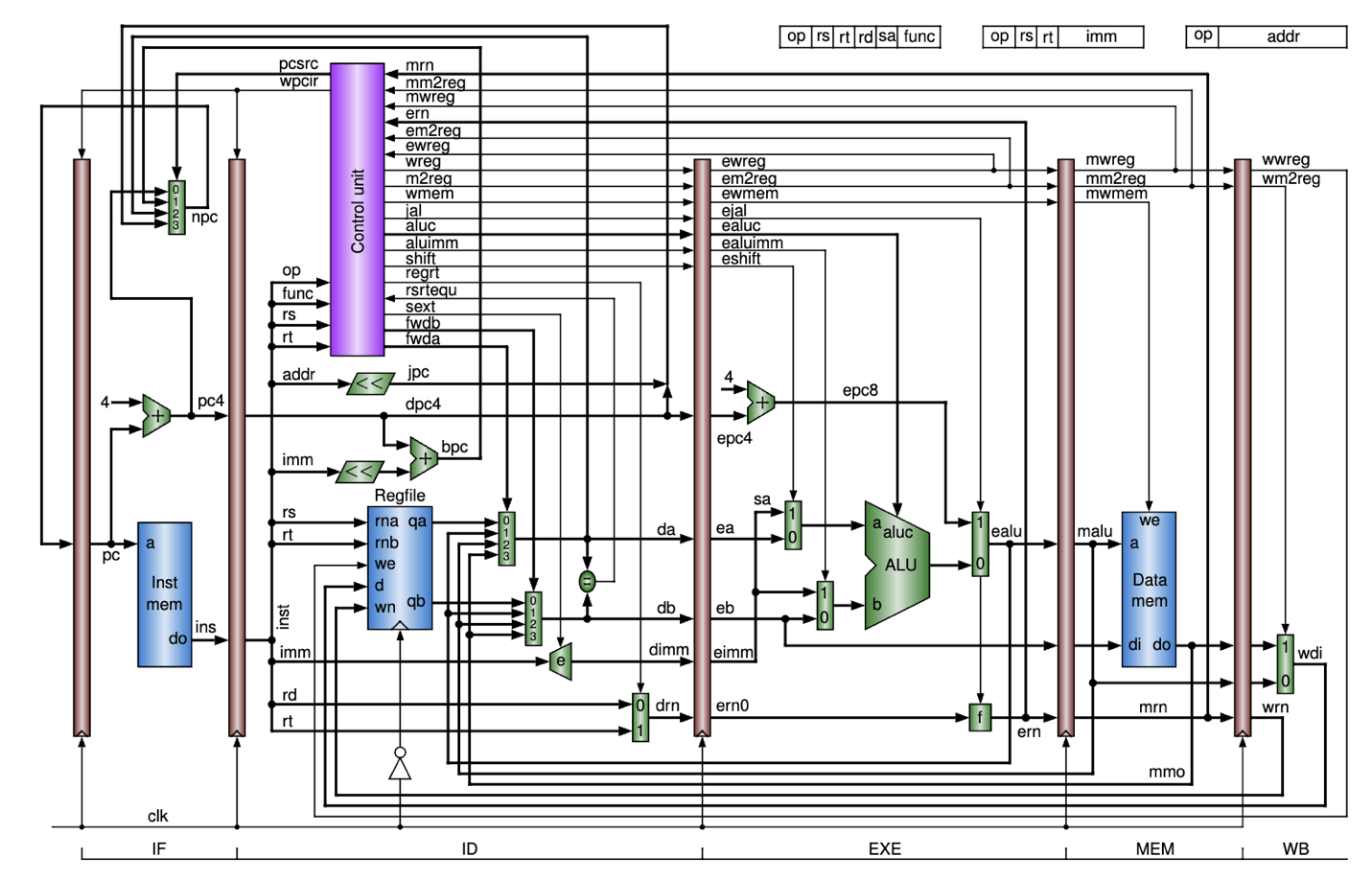
* In the final stage, the writeback stage comprises two modules: the MEMWB pipeline and the writeback mux. The MEMWB pipeline passes values out of the fourth stage on the positive edge of the clock, while the writeback mux determines which value will be passed back into the register file module in stage two.
* If wm2reg is 0, the delayed result wr is passed through the mux to the register file. If wm2reg is 1, wdo, the delayed output of the data memory module from stage four, is passed through.
* The signal wwreg determines whether writing to a register is necessary, and this operation occurs on the negative edge of the clock.

**Forwarding:**

****

* To complete the project, additional modules and modifications are introduced to each pipeline stage. The aim is to enhance our CPU to include proper hazard detection and utilize forwarding as needed. Stall necessity was implemented, which isn't part of the final project but for the extra credit.
* Two modules, ForwardMuxA and ForwardMuxB, are integrated into our CPU's functionality. These muxes determine whether ALU forwarding, memory forwarding, data memory forwarding, or no forwarding is required, with one mux handling qa and the other qb.
* To ascertain if forwarding is necessary, a hazard detection segment is incorporated into the Control Unit, requiring eight new inputs. This set of conditional statements checks for any potential hazards and determines the values of fwda and fwdb accordingly. These values are then passed to the forwarding muxes, which select the appropriate value to be fed back into the CPU, if required. This completes all essential aspects of the final project as outlined in the rubric.

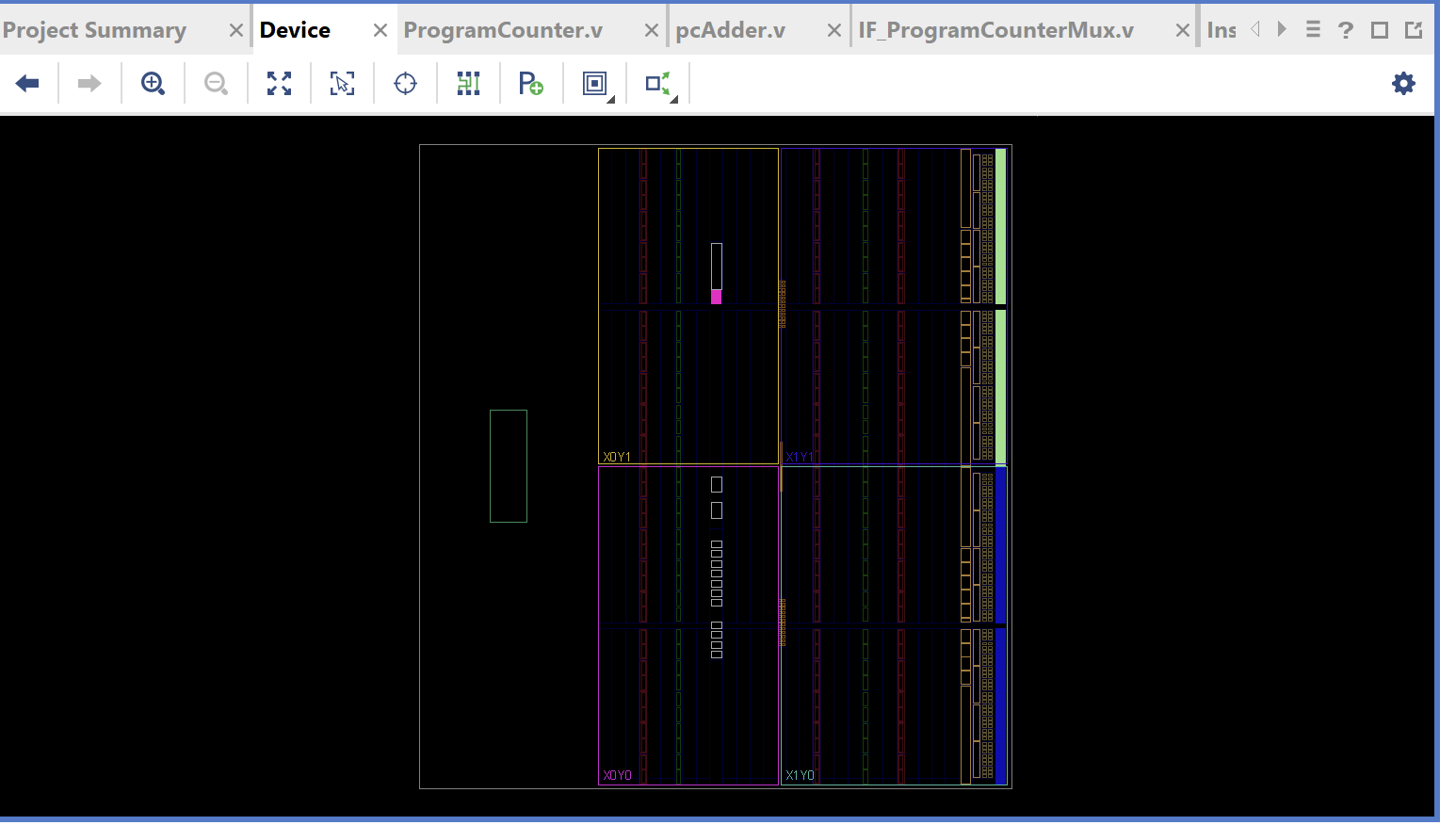
**Implementing Extra Credit:**

****

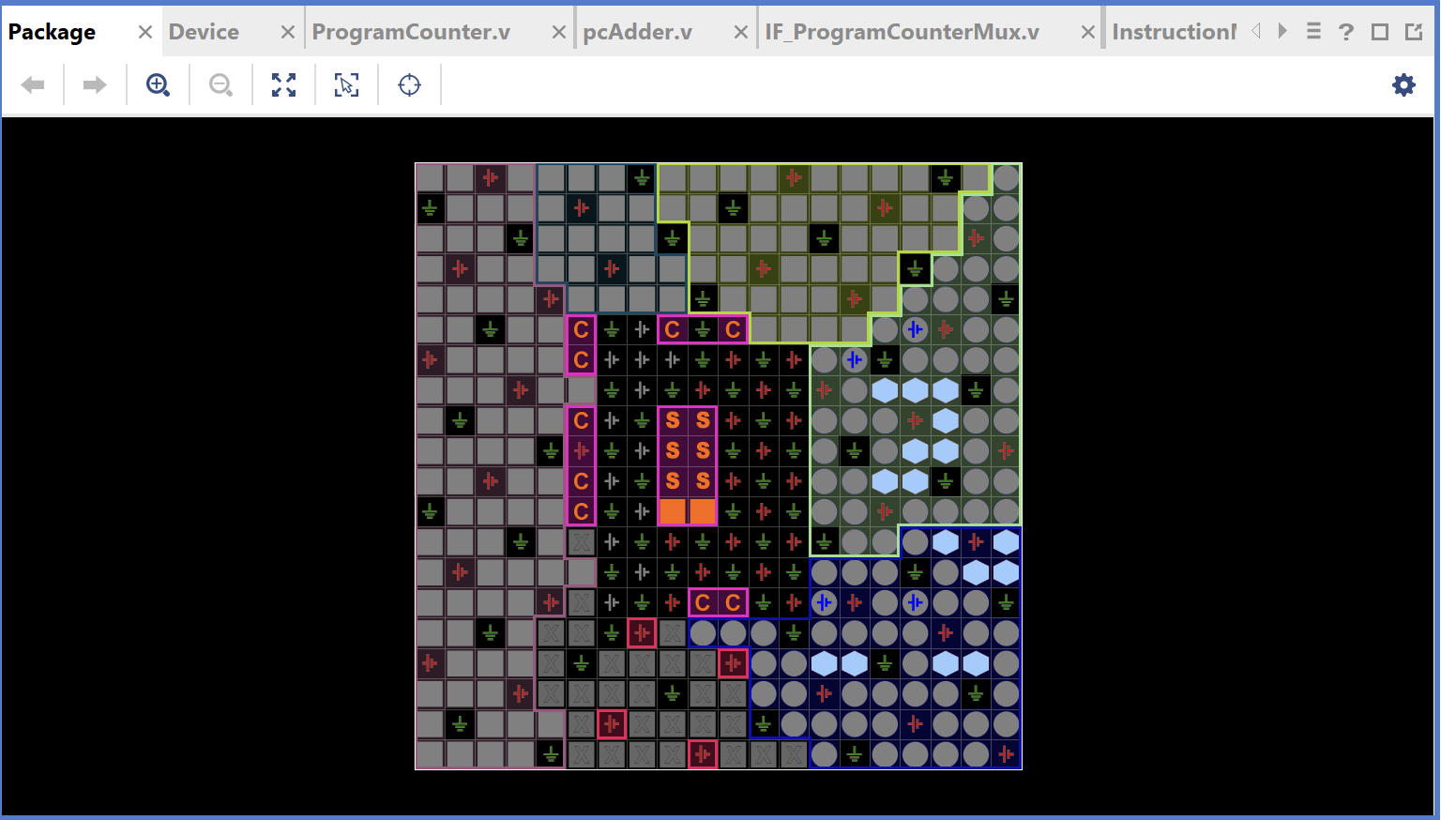
* For the extra credit, I expanded the MIPS CPU's capabilities to handle branching, jumping, load word, store word, load upper immediate, jump and link, and most r-type instructions.
* In the instruction fetch stage, a PCMux selects the next PC value based on the instruction type.
* The instruction decode stage now includes left-shift modules for jump and branch instructions, an immediate extender with sign extension, and a branch equality checker.
* In the execution stage, new modules handle program counter updates for jal instructions, mux selection for shifting operations, and processing jump and link instructions.
* No changes were made to the memory and writeback stages.

Images:

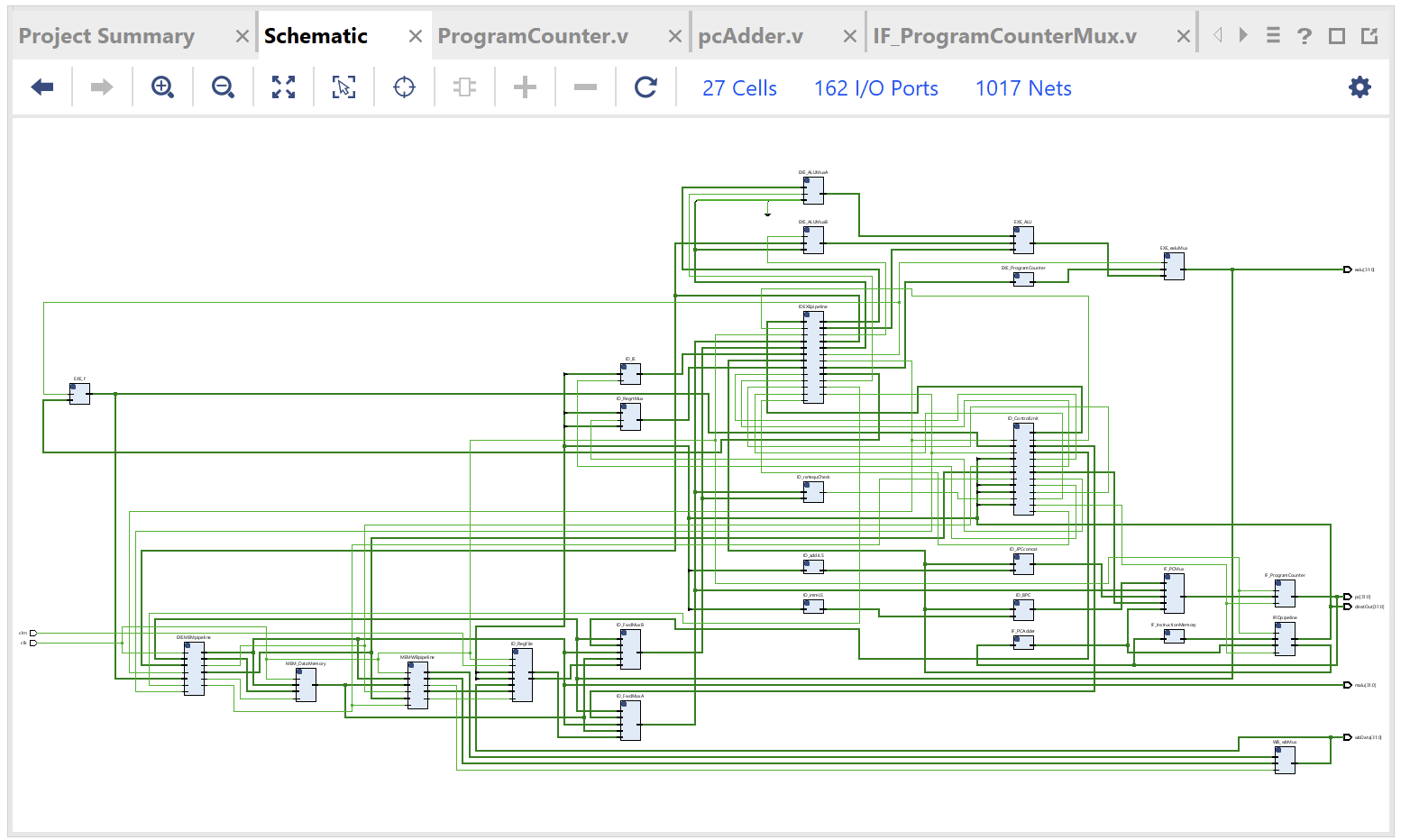
**Floorplanning:**



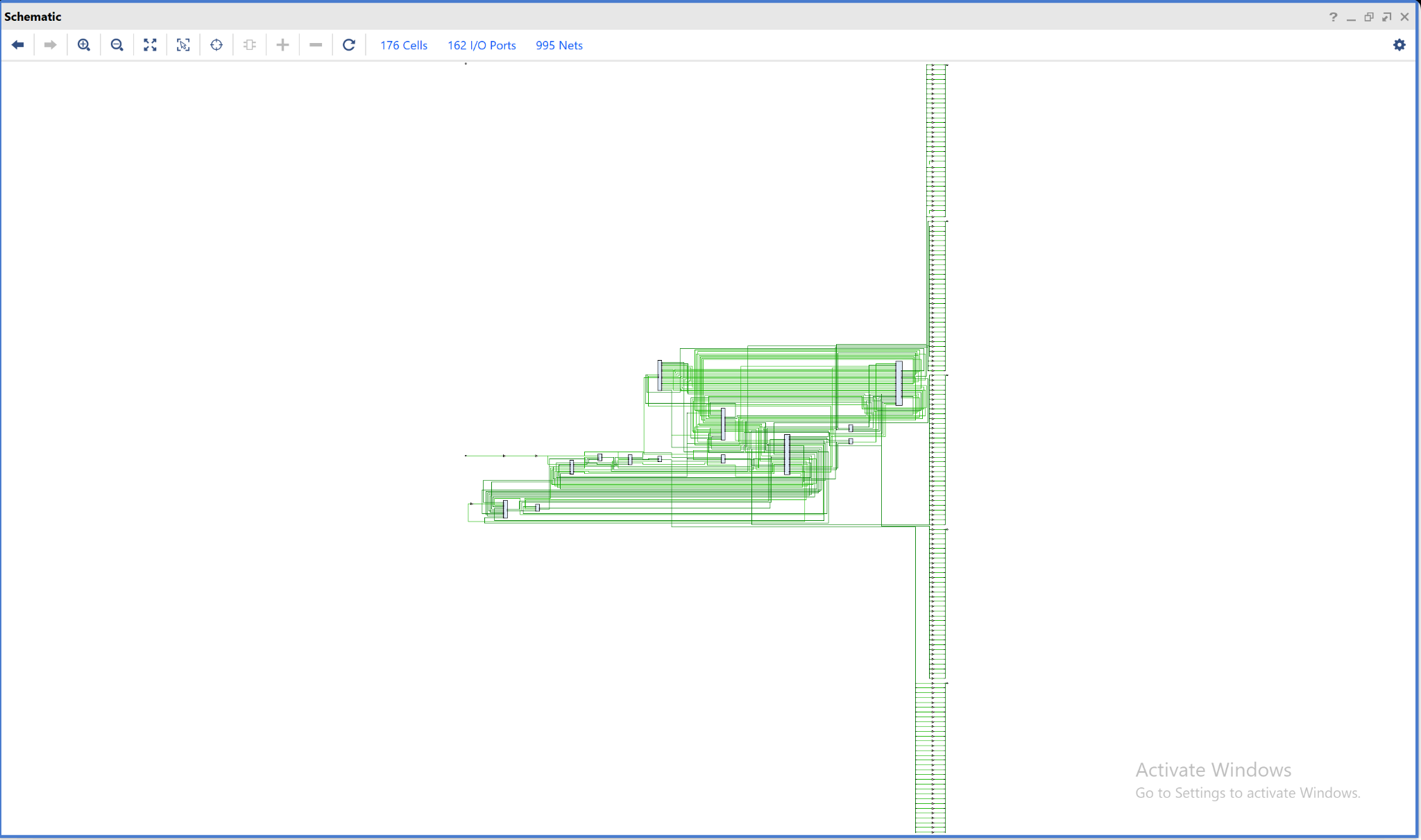
**I/O Planning:**



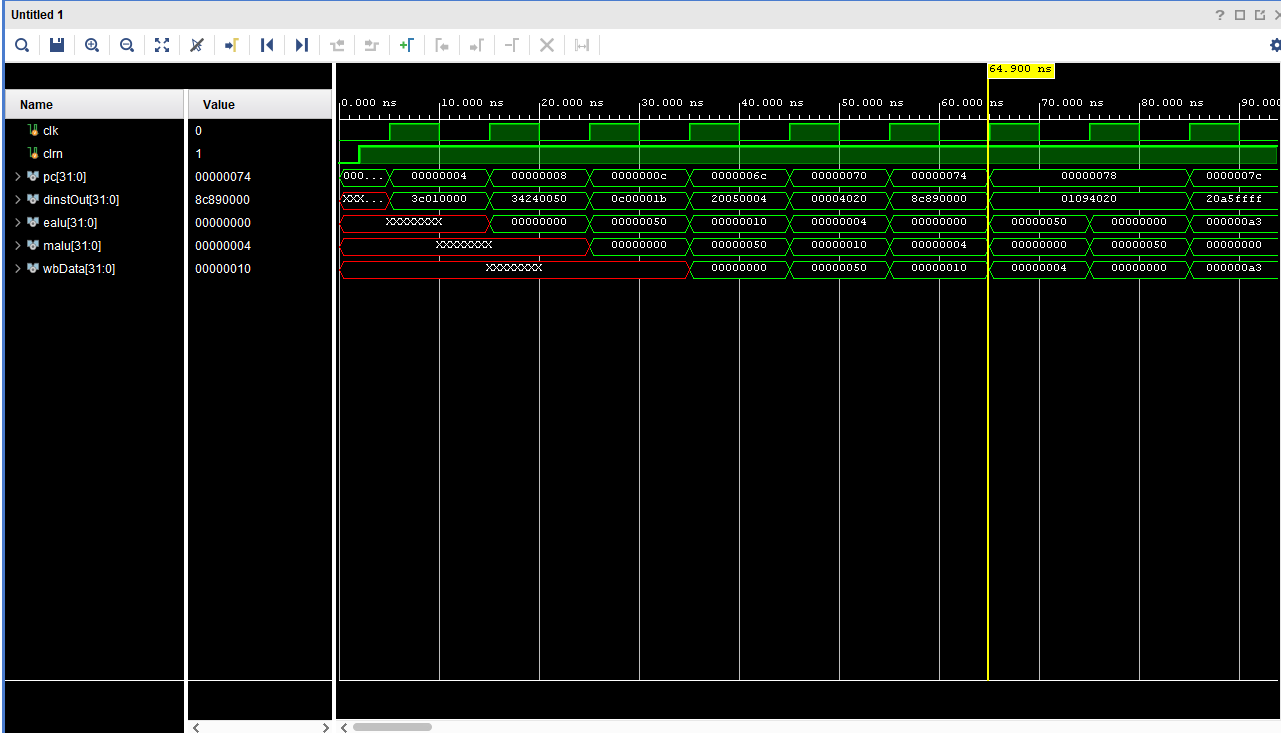
**Design Schematics**

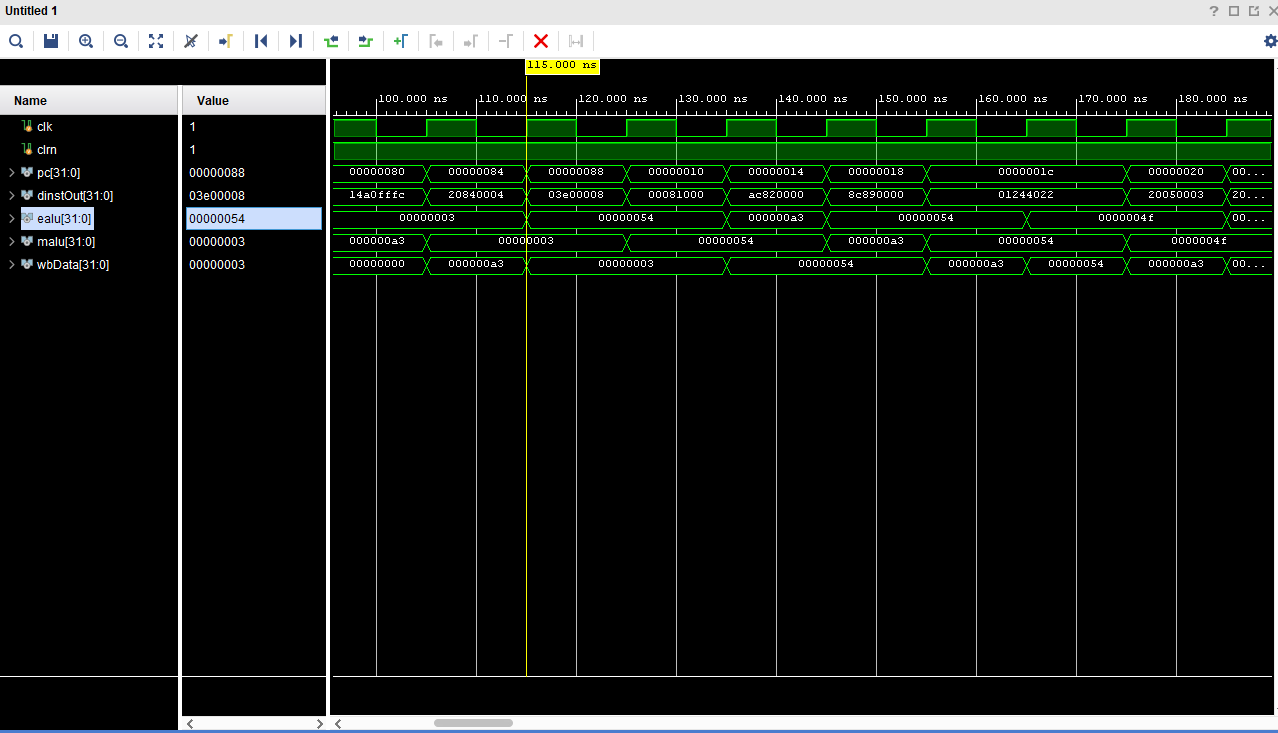


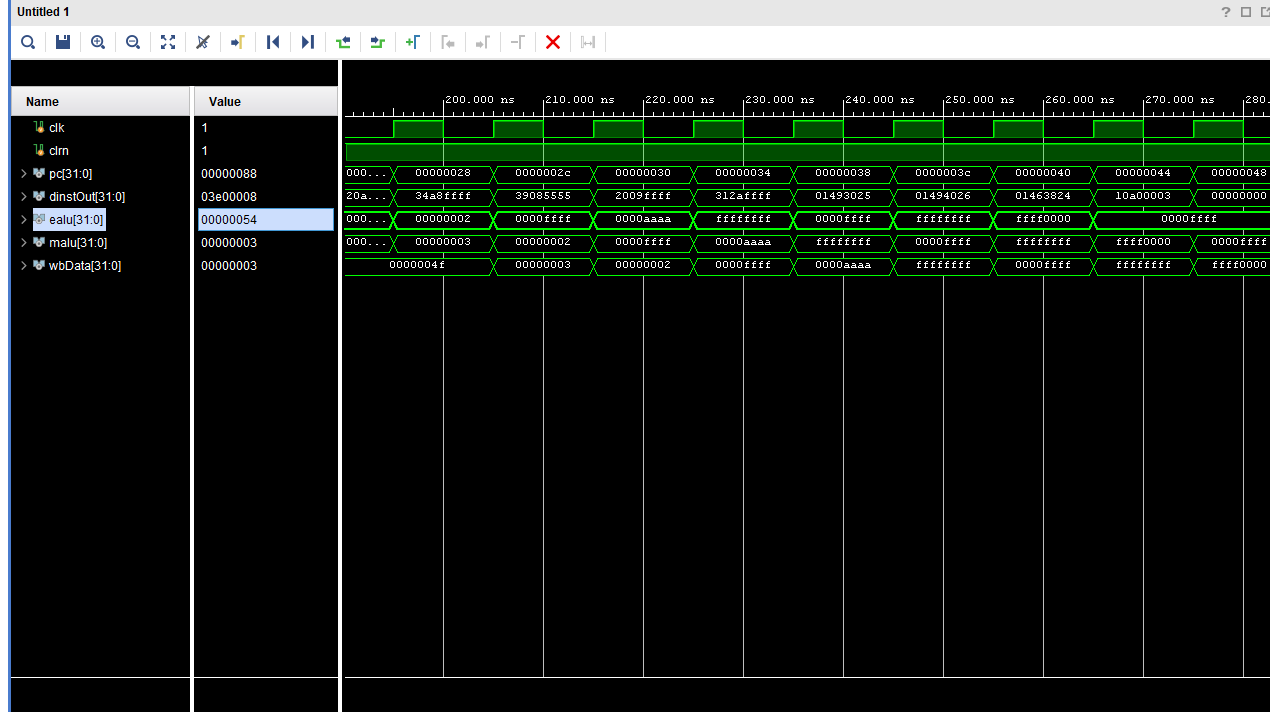
**Synthesized Design:**

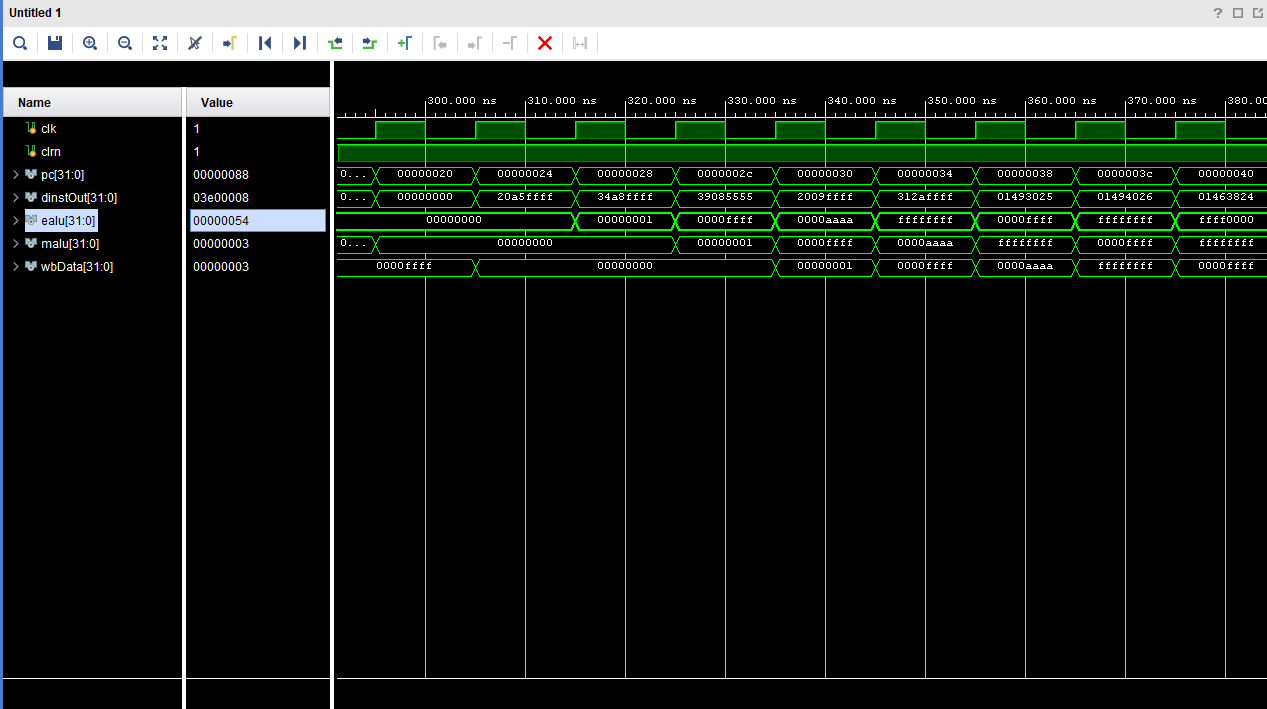


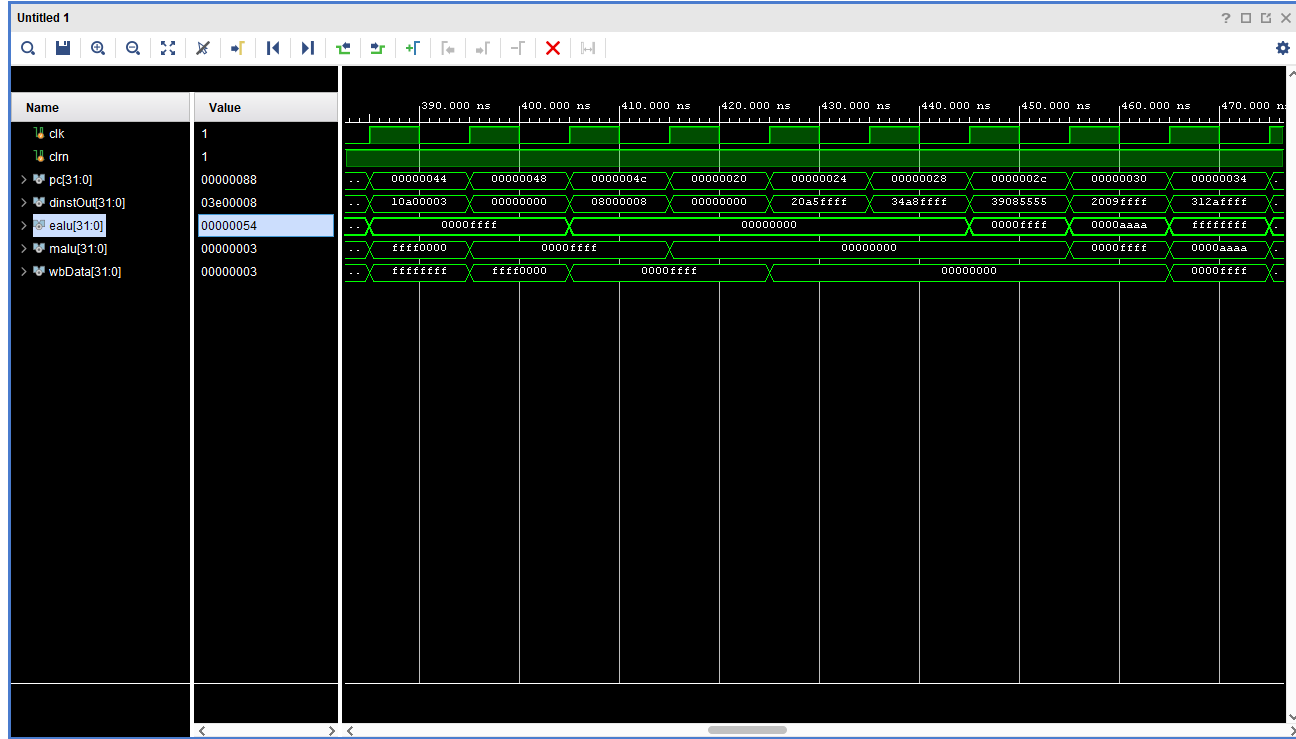
**Waveforms:**

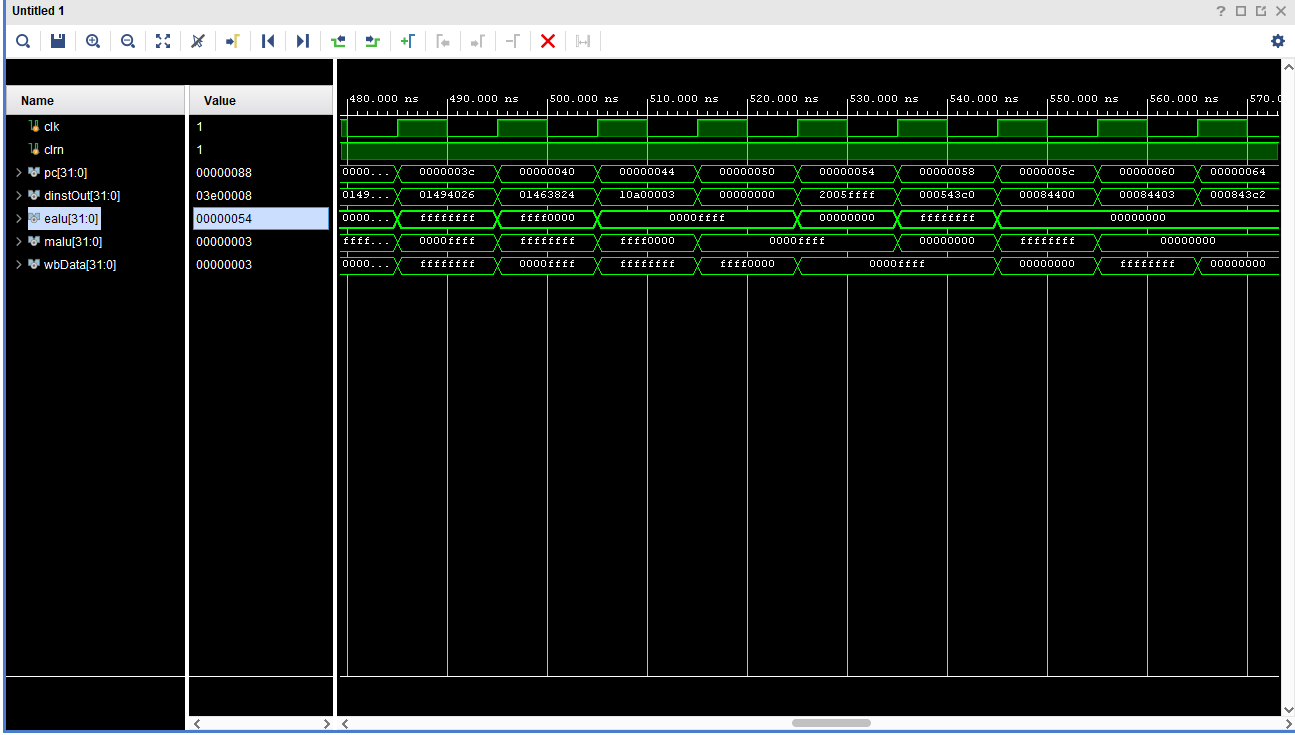


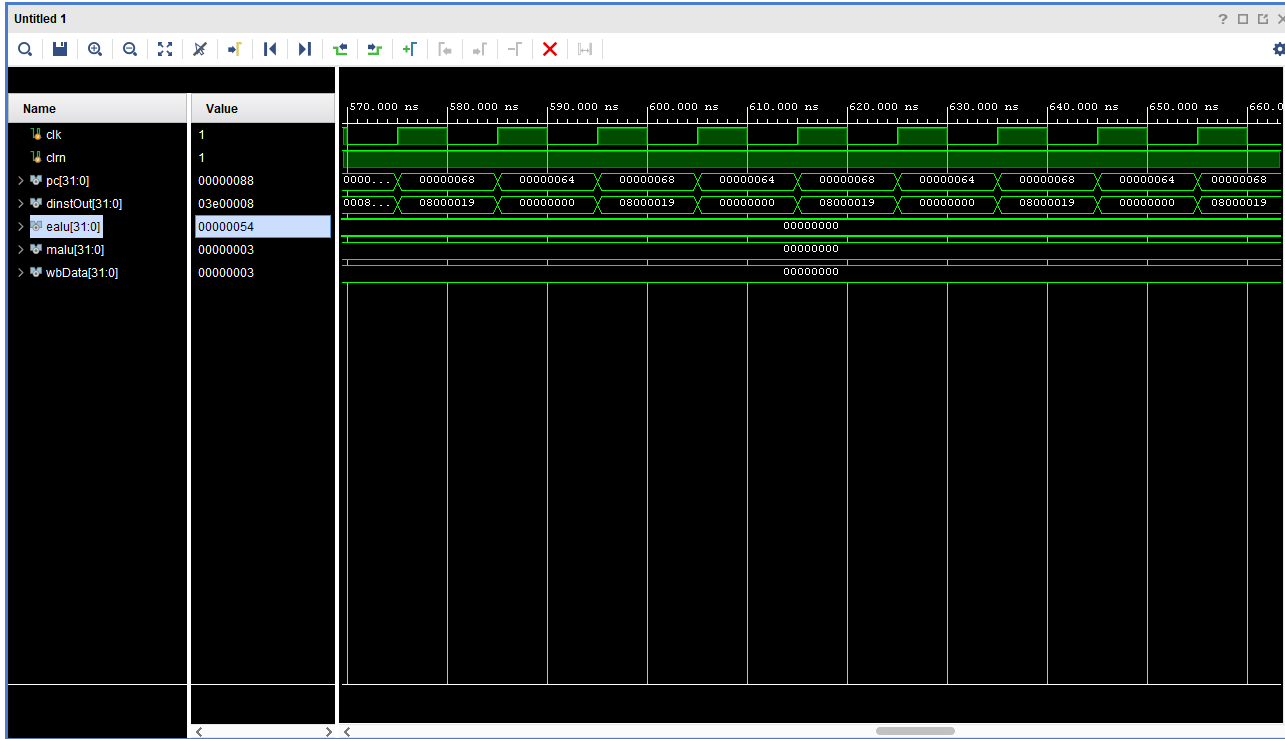


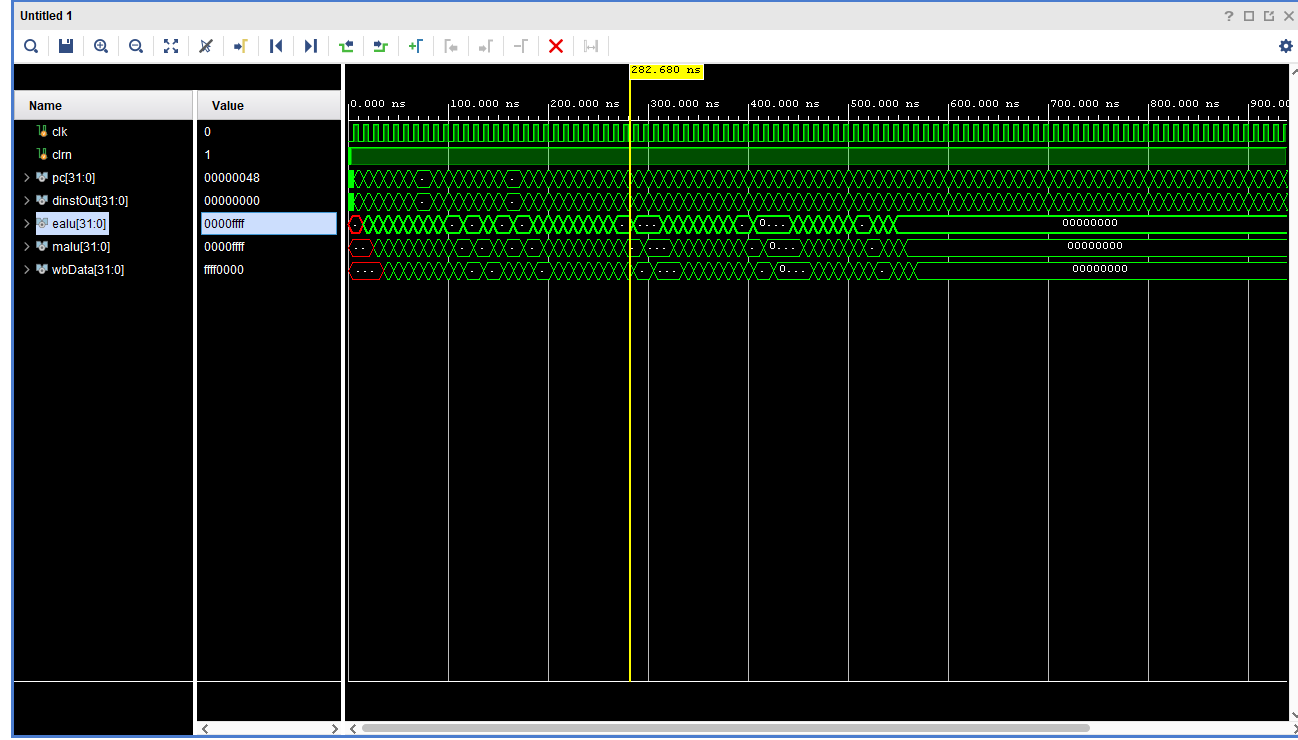












**DataPath:**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: Datapath

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module Datapath(

input clk,

input clrn,

output wire [31:0] pc,

output wire [31:0] dinstOut,

output wire [31:0] ealu,

output wire [31:0] malu,

output wire [31:0] wbData

);

wire [31:0] nextPc;

wire [31:0] pc4;

wire [31:0] bpc;

wire [31:0] da;

wire [31:0] njpc;

wire [1:0] pcsrc;

wire wpcir;

wire [31:0] dpc4;

wire [31:0] instOut;

wire [5:0] op;

wire [5:0] func;

wire [4:0] rs;

wire [4:0] rt;

wire [4:0] rd;

wire [15:0] imm;

wire [25:0] addr;

wire [31:0] qa;

wire [31:0] qb;

wire [31:0] immOut;

wire [25:0] jpc;

wire wwreg;

wire [4:0] wrn;

wire [31:0] mdo;

wire [31:0] db;

wire sext;

wire [31:0] dimm32;

wire [31:0] eimm32;

wire regrt;

wire [4:0] drn;

wire [4:0] mrn;

wire mm2reg;

wire mwreg;

wire [4:0] ern;

wire em2reg;

wire ewreg;

wire wreg;

wire m2reg;

wire wmem;

wire jal;

wire [3:0] aluc;

wire aluimm;

wire shift;

wire rsrtequ;

wire [1:0] fwda;

wire [1:0] fwdb;

wire ewmem;

wire ejal;

wire [3:0] ealuc;

wire ealuimm;

wire eshift;

wire [31:0] sa;

wire [31:0] ea;

wire [31:0] eb;

wire [31:0] epc4;

wire [31:0] epc8;

wire [31:0] a;

wire [31:0] b;

wire [31:0] r;

wire [4:0] ern0;

wire mwmem;

wire [31:0] mb;

wire wm2reg;

wire [31:0] walu;

wire [31:0] wdo;

ProgramCounter IF\_ProgramCounter(.pc(pc), .clk(clk), .nextPc(nextPc), .wpcir(wpcir));

pcAdder IF\_PCAdder(.pc(pc), .pc4(pc4));

IF\_ProgramCounterMux IF\_PCMux(.pc4(pc4), .bpc(bpc), .da(da), .njpc(njpc), .pcsrc(pcsrc), .nextPc(nextPc));

InstructionMemory IF\_InstructionMemory(.pc(pc), .instOut(instOut));

IFIDpipelineReg IFIDpipeline(.wpcir(wpcir), .pc4(pc4), .instOut(instOut), .clk(clk), .dinstOut(dinstOut), .dpc4(dpc4));

ID\_addrLS ID\_addrLS(.addr(addr), .jpc(jpc));

ID\_immLS ID\_immLS(.imm(imm), .immOut(immOut));

ID\_BranchProgramCounter ID\_BPC(.dpc4(dpc4), .immOut(immOut), .bpc(bpc));

RegisterFile ID\_RegFile(.rs(rs), .rt(rt), .wwreg(wwreg), .clrn(clrn), .clk(clk), .wbData(wbData), .wrn(wrn), .qa(qa), .qb(qb));

Fwd\_FwdMuxA ID\_FwdMuxA(.qa(qa), .malu(malu), .ealu(ealu), .mdo(mdo), .fwda(fwda), .da(da));

Fwd\_FwdMuxB ID\_FwdMuxB(.qb(qb), .malu(malu), .ealu(ealu), .mdo(mdo), .fwdb(fwdb), .db(db));

ImmediateExtender ID\_IE(.imm(imm), .dimm32(dimm32), .sext(sext));

RegrtMultiplexer ID\_RegrtMux(.rd(rd), .rt(rt), .regrt(regrt), .drn(drn));

ID\_rsrtequCheck ID\_rsrtequCheck(.da(da), .db(db), .rsrtequ(rsrtequ));

ID\_JumpPCconcatination ID\_JPCconcat(.jpc(jpc), .dpc4(dpc4), .njpc(njpc));

ControlUnit ID\_ControlUnit(.op(op), .rs(rs), .rt(rt), .func(func), .pcsrc(pcsrc), .wpcir(wpcir),

.mrn(mrn), .mm2reg(mm2reg), .mwreg(mwreg), .ern(ern), .em2reg(em2reg), .ewreg(ewreg), .wreg(wreg),

.m2reg(m2reg), .wmem(wmem), .jal(jal), .aluc(aluc), .aluimm(aluimm), .shift(shift), .regrt(regrt),

.rsrtequ(rsrtequ), .sext(sext), .fwda(fwda), .fwdb(fwdb));

IDEXEpipeline IDEXEpipeline(.wreg(wreg), .m2reg(m2reg), .wmem(wmem), .jal(jal), .aluc(aluc), .aluimm(aluimm),

.shift(shift), .dpc4(dpc4), .da(da), .db(db), .dimm32(dimm32), .drn(drn), .clk(clk), .ern0(ern0), .eimm32(eimm32),

.eb(eb), .ea(ea), .epc4(epc4), .eshift(eshift), .ealuimm(ealuimm), .ealuc(ealuc), .ejal(ejal), .ewmem(ewmem),

.em2reg(em2reg), .ewreg(ewreg));

EXE\_ProgramCounter EXE\_ProgramCounter(.epc4(epc4), .epc8(epc8));

EXE\_ALUMux2 EXE\_ALUMuxA(.sa(sa), .ea(ea), .eshift(eshift), .a(a));

ALUMux EXE\_ALUMuxB(.eimm32(eimm32), .eb(eb), .ealuimm(ealuimm), .b(b));

ALU EXE\_ALU(.a(a), .b(b), .r(r), .ealuc(ealuc));

EXE\_ALUoutMux EXE\_ealuMux(.epc8(epc8), .r(r), .ejal(ejal), .ealu(ealu));

EXE\_f EXE\_f(.ern0(ern0), .ern(ern), .ejal(ejal));

EXEMEMpipeline EXEMEMpipeline(.ewreg(ewreg), .em2reg(em2reg), .ewmem(ewmem), .ealu(ealu), .eb(eb), .ern(ern),

.clk(clk), .mrn(mrn), .mb(mb), .malu(malu), .mwmem(mwmem), .mm2reg(mm2reg), .mwreg(mwreg));

DataMemory MEM\_DataMemory(.malu(malu), .mb(mb), .mwmem(mwmem), .mdo(mdo), .clk(clk));

MEMWBpipeline MEMWBpipeline(.mwreg(mwreg), .mm2reg(mm2reg), .mdo(mdo), .malu(malu), .mrn(mrn), .clk(clk),

.wrn(wrn), .walu(walu), .wdo(wdo), .wm2reg(wm2reg), .wwreg(wwreg));

WbMux WB\_wbMux(.wdo(wdo), .walu(walu), .wbData(wbData), .wm2reg(wm2reg));

// Assign some control signals and data values

assign op = dinstOut[31:26];

assign func = dinstOut[5:0];

assign rs = dinstOut[25:21];

assign rt = dinstOut[20:16];

assign rd = dinstOut[15:11];

assign imm = dinstOut[15:0];

assign sa = {21'b0,eimm32[10:6],6'b0};

assign addr = dinstOut[25:0];

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: ProgramCounter

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module ProgramCounter(

input clk,

input [31:0] nextPc,

input wpcir,

output reg [31:0] pc

);

initial

begin

pc = 32'd0;

end

always @(posedge clk)

begin

if (wpcir == 1)

pc = nextPc;

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: pcAdder

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module pcAdder(

input [31:0] pc,

output reg [31:0] pc4

);

always @(\*) begin

pc4 <= pc + 32'b00000000000000000000000000000100;

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: IF\_ProgramCounterMux

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module IF\_ProgramCounterMux(

input [1:0] pcsrc,

input [31:0] pc4,

input [31:0] bpc,

input [31:0] njpc,

input [31:0] da,

output reg [31:0] nextPc

);

always @ (\*)

begin

case(pcsrc)

2'b00: nextPc = pc4;

2'b01: nextPc = bpc;

2'b10: nextPc = da;

2'b11: nextPc = njpc;

endcase

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: InstructionMemory

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module InstructionMemory( // instruction memory, rom

input [31:0] pc, // rom address

output reg [31:0] instOut // rom content = rom[a]

);

wire [31:0] rom [0:63]; // rom cells: 64 words \* 32 bits

// rom[word\_addr] = instruction // (pc) label instruction

assign rom[6'h00] = 32'h3c010000; // (00) main: lui $1, 0

assign rom[6'h01] = 32'h34240050; // (04) ori $4, $1, 80

assign rom[6'h02] = 32'h0c00001b; // (08) call: jal sum

assign rom[6'h03] = 32'h20050004; // (0c) dslot1: addi $5, $0, 4

assign rom[6'h04] = 32'hac820000; // (10) return: sw $2, 0($4)

assign rom[6'h05] = 32'h8c890000; // (14) lw $9, 0($4)

assign rom[6'h06] = 32'h01244022; // (18) sub $8, $9, $4

assign rom[6'h07] = 32'h20050003; // (1c) addi $5, $0, 3

assign rom[6'h08] = 32'h20a5ffff; // (20) loop2: addi $5, $5, -1

assign rom[6'h09] = 32'h34a8ffff; // (24) ori $8, $5, 0xffff

assign rom[6'h0a] = 32'h39085555; // (28) xori $8, $8, 0x5555

assign rom[6'h0b] = 32'h2009ffff; // (2c) addi $9, $0, -1

assign rom[6'h0c] = 32'h312affff; // (30) andi $10,$9,0xffff

assign rom[6'h0d] = 32'h01493025; // (34) or $6, $10, $9

assign rom[6'h0e] = 32'h01494026; // (38) xor $8, $10, $9

assign rom[6'h0f] = 32'h01463824; // (3c) and $7, $10, $6

assign rom[6'h10] = 32'h10a00003; // (40) beq $5, $0, shift

assign rom[6'h11] = 32'h00000000; // (44) dslot2: nop

assign rom[6'h12] = 32'h08000008; // (48) j loop2

assign rom[6'h13] = 32'h00000000; // (4c) dslot3: nop

assign rom[6'h14] = 32'h2005ffff; // (50) shift: addi $5, $0, -1

assign rom[6'h15] = 32'h000543c0; // (54) sll $8, $5, 15

assign rom[6'h16] = 32'h00084400; // (58) sll $8, $8, 16

assign rom[6'h17] = 32'h00084403; // (5c) sra $8, $8, 16

assign rom[6'h18] = 32'h000843c2; // (60) srl $8, $8, 15

assign rom[6'h19] = 32'h08000019; // (64) finish: j finish

assign rom[6'h1a] = 32'h00000000; // (68) dslot4: nop

assign rom[6'h1b] = 32'h00004020; // (6c) sum: add $8, $0, $0

assign rom[6'h1c] = 32'h8c890000; // (70) loop: lw $9, 0($4)

assign rom[6'h1d] = 32'h01094020; // (74) stall: add $8, $8, $9

assign rom[6'h1e] = 32'h20a5ffff; // (78) addi $5, $5, -1

assign rom[6'h1f] = 32'h14a0fffc; // (7c) bne $5, $0, loop

assign rom[6'h20] = 32'h20840004; // (80) dslot5: addi $4, $4, 4

assign rom[6'h21] = 32'h03e00008; // (84) jr $31

assign rom[6'h22] = 32'h00081000; // (88) dslot6: sll $2, $8, 0

always @(\*)

begin

instOut = rom[pc[7:2]]; // use 6-bit word address to read rom

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: IFIDpipelineReg

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module IFIDpipelineReg( //IFID pipeline

input clk, //clock input needed as dinstOut only updates on the positive edge of clock.

input [31:0] instOut, //input

//EC Input

input [31:0] pc4,

//final project input

input wpcir,

output reg [31:0] dinstOut, //output

//EC Output

output reg [31:0] dpc4

);

always @ (posedge clk) //always block that will only update dinstOut on the positive edge of the clock. dinstOut is to the instOut input of this module.

begin

if (wpcir == 1)

dinstOut <= instOut;

dpc4 <= pc4;

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: ID\_addrLS

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module ID\_addrLS(

input [25:0] addr,

output reg [25:0] jpc

);

always @ (\*)

begin

jpc = addr << 2;

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: ID\_immLS

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module ID\_immLS(

input [15:0] imm,

output reg [31:0] immOut

);

always @ (\*)

begin

immOut = {16'b0, imm} << 2;

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: ID\_BranchProgramCounter

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module ID\_BranchProgramCounter(

input [31:0] dpc4,

input [31:0] immOut,

output reg [31:0] bpc

);

always @ (\*)

begin

bpc <= dpc4 + immOut;

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: RegisterFile

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module RegisterFile(

// Inputs

input [4:0] rs, // Input for the source register (rs)

input [4:0] rt, // Input for the target register (rt)

//Lab 5 Inputs

input [4:0] wrn,

input [31:0] wbData,

input wwreg,

input clk, clrn,

// Outputs

output reg [31:0] qa, // Output for the value stored in the source register

output reg [31:0] qb // Output for the value stored in the target register

);

reg [31:0] register [0:31]; // 32x32 array for registers (register file)

// Initialize all registers to 0

integer r;

initial begin

for (r = 0; r <= 31; r = r + 1) begin

register[r] = 0; // Initialize each register to 0.

end

end

always @ (\*) // Always block to update qa and qb based on the input rs and rt values.

begin

qa = register[rs]; // Output qa is the value stored in the source register (rs).

qb = register[rt]; // Output qb is the value stored in the target register (rt).

end

always @ (negedge clk)

begin

if (wwreg == 1)

register[wrn] = wbData;

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: Fwd\_FwdMuxA

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module Fwd\_FwdMuxA(

input [1:0] fwda,

input [31:0] qa,

input [31:0] ealu,

input [31:0] malu,

input [31:0] mdo,

output reg [31:0] da

);

always @ (\*) begin

case(fwda)

2'b00: da <= qa;

2'b01: da <= ealu;

2'b10: da <= malu;

2'b11: da <= mdo;

endcase

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: Fwd\_FwdMuxB

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module Fwd\_FwdMuxB(

input [1:0] fwdb,

input [31:0] qb,

input [31:0] ealu,

input [31:0] malu,

input [31:0] mdo,

output reg [31:0] db

);

always @ (\*) begin

case(fwdb)

2'b00: db <= qb;

2'b01: db <= ealu;

2'b10: db <= malu;

2'b11: db <= mdo;

endcase

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: ImmediateExtender

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module ImmediateExtender( //immediate extender module.

input [15:0] imm,

input sext,

output reg [31:0] dimm32

);

always @ (\*) //always block to update the value of imm32.

begin

if (sext == 1)

dimm32 = {{16{imm[15]}}, imm}; //sets imm32 to be equal to imm. the last bit is concatinated to the other 16 bits based on if the sign bit is a zero or one.

else

dimm32 <= {16'b0, imm};

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: RegrtMultiplexer

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module RegrtMultiplexer(

// Inputs

input [4:0] rt, // Input register value rt

input [4:0] rd, // Input register value rd

input regrt, // Control signal to select the output (0 for rd, 1 for rt)

// Output

output reg [4:0] drn // Output register value (selected based on the control signal)

);

always @(\*)

begin

if (regrt == 0)

drn = rd; // If regrt is 0, select rd as the output.

else

drn = rt; // If regrt is 1, select rt as the output.

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: ID\_rsrtequCheck

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module ID\_rsrtequCheck(

input [31:0] da,

input [31:0] db,

output reg rsrtequ

);

always @ (\*)

begin

if (da == db)

rsrtequ <= 1;

else

rsrtequ <= 0;

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: ID\_JumpPCconcatination

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module ID\_JumpPCconcatination(

input [25:0] jpc,

input [31:0] dpc4,

output reg [31:0] njpc

);

always @ (\*)

begin

njpc <= {{dpc4[31:26]}, jpc};

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: ControlUnit

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module ControlUnit( //control unit module of the cpu

//inputs

input [5:0] op, func,

//final project inputs

input [4:0] rs, rt, mrn, ern,

input mm2reg, mwreg, em2reg, ewreg,

//Extra Credit Input

input rsrtequ,

//outputs

output reg wreg, m2reg, wmem, aluimm, regrt,

output reg [3:0] aluc,

//final project outputs

output reg [1:0] fwda, fwdb,

output reg wpcir,

//Extra Credit Outputs

output reg sext, shift, jal,

output reg [1:0] pcsrc

);

reg regUsage = 1'b1;

initial begin

wreg <= 0; //RegWrite

m2reg <= 0; //Mem2Reg

wmem <= 0; //Write Memory

aluimm <= 0; //ALU source

regrt <= 0; //Reg Destination

pcsrc <= 2'b00;

end

always @ (\*) begin //always block that will continually update

case (op) //case statement of the op code portion of dinstOut which is connected in the datapath module.

6'b000000: // R-type instructions

begin

case (func) //case statement to check which operation is performed.

6'b100000: begin

aluc = 4'b0010; //ADD Operation

wreg = 1'b1; // Write to the register file

m2reg = 1'b0; // Do not write to memory

wmem = 1'b0; // Do not write to memory

aluimm = 1'b0; // ALU source from registers

regrt = 1'b0; // Destination register address

sext = 1'bx;

jal = 1'b0;

shift = 1'b0;

pcsrc = 2'b00;

end

6'b100010: begin

aluc = 4'b0110; //SUB

wreg = 1'b1; // Write to the register file

m2reg = 1'b0; // Do not write to memory

wmem = 1'b0; // Do not write to memory

aluimm = 1'b0; // ALU source from registers

regrt = 1'b0; // Destination register address

sext = 1'bx;

jal = 1'b0;

shift = 1'b0;

pcsrc = 2'b00;

end

6'b100100: begin

aluc = 4'b0000; //AND Operation

wreg = 1'b1; // Write to the register file

m2reg = 1'b0; // Do not write to memory

wmem = 1'b0; // Do not write to memory

aluimm = 1'b0; // ALU source from registers

regrt = 1'b0; // Destination register address

sext = 1'bx;

jal = 1'b0;

shift = 1'b0;

pcsrc = 2'b00;

end

6'b100101: begin

aluc = 4'b0001; //OR

wreg = 1'b1; // Write to the register file

m2reg = 1'b0; // Do not write to memory

wmem = 1'b0; // Do not write to memory

aluimm = 1'b0; // ALU source from registers

regrt = 1'b0; // Destination register address

sext = 1'bx;

jal = 1'b0;

shift = 1'b0;

pcsrc = 2'b00;

end

6'b100110: begin

aluc = 4'b0011; //XOR Operation

wreg = 1'b1; // Write to the register file

m2reg = 1'b0; // Do not write to memory

wmem = 1'b0; // Do not write to memory

aluimm = 1'b0; // ALU source from registers

regrt = 1'b0; // Destination register address

sext = 1'bx;

jal = 1'b0;

shift = 1'b0;

pcsrc = 2'b00;

end

6'b000000: begin

aluc = 4'b0111; //sll

wreg = 1'b1; // Write to the register file

m2reg = 1'b0; // Do not write to memory

wmem = 1'b0; // Do not write to memory

aluimm = 1'b0; // ALU source from registers

regrt = 1'b0; // Destination register address

sext = 1'bx;

jal = 1'b0;

shift = 1'b1;

pcsrc = 2'b00;

end

6'b000010: begin

aluc = 4'b1110; //srl

wreg = 1'b1; // Write to the register file

m2reg = 1'b0; // Do not write to memory

wmem = 1'b0; // Do not write to memory

aluimm = 1'b0; // ALU source from registers

regrt = 1'b0; // Destination register address

sext = 1'bx;

jal = 1'b0;

shift = 1'b1;

pcsrc = 2'b00;

end

6'b000011: begin

aluc = 4'b1001; //sra

wreg = 1'b1; // Write to the register file

m2reg = 1'b0; // Do not write to memory

wmem = 1'b0; // Do not write to memory

aluimm = 1'b0; // ALU source from registers

regrt = 1'b0; // Destination register address

sext = 1'bx;

jal = 1'b0;

shift = 1'b1;

pcsrc = 2'b00;

end

6'b001000:

begin aluc = 4'bxxxx; //jr

wreg = 1'b0; // Write to the register file

// m2reg = 1'bx; // Do not write to memory

wmem = 1'b0; // Do not write to memory

aluimm = 1'bx; // ALU source from registers

regrt = 1'bx; // Destination register address

sext = 1'bx;

jal = 1'bx;

shift = 1'bx;

pcsrc = 2'b10;

end

endcase

end

6'b001000: //addi

begin

wreg <= 1'b1;

regrt <= 1'b1;

jal <= 1'b0;

m2reg <= 1'b0;

shift <= 1'b0;

aluimm <= 1'b1;

sext <= 1'b1;

aluc <= 4'b0010;

wmem <= 1'b0;

pcsrc <= 2'b00;

end

6'b00001100: //andi

begin

wreg <= 1'b1;

regrt <= 1'b1;

jal <= 1'b0;

m2reg <= 1'b0;

shift <= 1'b0;

aluimm <= 1'b1;

sext <= 1'b0;

aluc <= 4'b0000;

wmem <= 1'b0;

pcsrc <= 2'b00;

end

6'b001101: //ori

begin

wreg <= 1'b1;

regrt <= 1'b1;

jal <= 1'b0;

m2reg <= 1'b0;

shift <= 1'b0;

aluimm <= 1'b1;

sext <= 1'b0;

aluc <= 4'b0001;

wmem <= 1'b0;

pcsrc <= 2'b00;

end

6'b001110: //xori

begin

wreg <= 1'b1;

regrt <= 1'b1;

jal <= 1'b0;

m2reg <= 1'b0;

shift <= 1'b0;

aluimm <= 1'b1;

sext <= 1'b0;

aluc <= 4'b0011;

wmem <= 1'b0;

pcsrc <= 2'b00;

end

6'b100011: // LW instruction

begin

// Set control signals for LW instruction

wreg = 1'b1; // Write to the register file

m2reg = 1'b1; // Write to memory

wmem = 1'b0; // Do not write to memory

aluc = 4'b0010; // ALU operation for addition

aluimm = 1'b1; // ALU source from registers

regrt = 1'b1; // Destination register address

jal = 1'b0;

shift = 1'b0;

pcsrc = 2'b00;

sext = 1'b1;

end

6'b101011: //SW instruction

begin

wreg = 1'b0;

m2reg = 1'b0;

wmem = 1'b1;

aluc = 4'b0010;

aluimm = 1'b1;

regrt = 1'bx;

jal = 1'b0;

shift = 1'b0;

pcsrc = 2'b00;

sext = 1'b1;

end

6'b000100: //BEQ instruction

begin

wreg = 1'b0;

//m2reg = 1'bx; /\*\*\*/

wmem = 1'b0;

aluc = 4'b0011;

aluimm = 1'b0;

regrt = 1'bx;

jal = 1'bx;

shift = 1'b0;

sext = 1'b1;

if (rsrtequ == 1)

pcsrc <= 2'b01;

else

pcsrc <= 2'b00;

end

6'b000101: //BNE instruction

begin

wreg = 1'b0;

// m2reg = 1'bx; /\*\*\*/

wmem = 1'b0;

aluc = 4'b0011;

aluimm = 1'b0;

regrt = 1'bx;

jal = 1'bx;

shift = 1'b0;

sext = 1'b1;

if (rsrtequ == 1)

pcsrc <= 2'b01;

else

pcsrc <= 2'b00;

end

6'b001111: //lui

begin

wreg <= 1'b1;

regrt <= 1'b1;

jal <= 1'b0;

m2reg <= 1'b0;

shift <= 1'bx;

aluimm <= 1'b1;

sext <= 1'bx;

aluc <= 4'b0100;

wmem <= 1'b0;

pcsrc <= 2'b00;

end

6'b000010: //j

begin

wreg <= 1'b0;

regrt <= 1'bx;

jal <= 1'bx;

m2reg <= 1'bx;

shift <= 1'bx;

aluimm <= 1'bx;

sext <= 1'bx;

aluc <= 4'bxxxx;

wmem <= 1'b0;

pcsrc <= 2'b11;

end

6'b000011: //jal

begin

wreg <= 1'b1;

regrt <= 1'bx;

jal <= 1'b1;

//m2reg <= 1'bx; /\*\*\*\*/

shift <= 1'bx;

aluimm <= 1'bx;

sext <= 1'bx;

aluc <= 4'bxxxx;

wmem <= 1'b0;

pcsrc <= 2'b11;

end

endcase

end

//stall

always @ (\*)

begin

if ((ewreg == 1'b1) && (em2reg == 1'b1) && (ern != 5'b0) && (regUsage == 1'b1)

&& ((ern == rs) || (ern == rt))) begin

wreg = 1'b0;

wmem = 1'b0;

wpcir = 1'b0;

end

else begin

wpcir = 1'b1;

end

// forwarding

if ((ewreg == 1'b1) && (ern != 5'b0) && (ern == rs) && (em2reg == 1'b0)) begin

fwda = 2'b01;

end

else if ((mwreg == 1'b1) && (mrn != 5'b0) && (mrn == rs) && (mm2reg == 1'b0)) begin

fwda = 2'b10;

end

else if ((mwreg == 1'b1) && (mrn != 5'b0) && (mrn == rs) && (mm2reg == 1'b1)) begin

fwda = 2'b11;

end

else begin

fwda = 2'b00;

end

if ((ewreg == 1'b1) && (ern != 5'b0) && (ern == rt) && (em2reg == 1'b0)) begin

fwdb = 2'b01;

end

else if ((mwreg == 1'b1) && (mrn != 5'b0) && (mrn == rt) && (mm2reg == 1'b0)) begin

fwdb = 2'b10;

end

else if ((mwreg == 1'b1) && (mrn != 5'b0) && (mrn == rt) && (mm2reg == 1'b1)) begin

fwdb = 2'b11;

end

else begin

fwdb = 2'b00;

end

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: IDEXEpipeline

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module IDEXEpipeline(

// Inputs

input wreg, // Control signal for writing to the register file

input m2reg, // Control signal for writing to the register file (M2 stage)

input wmem, // Control signal for writing to memory

input [3:0] aluc, // ALU control signal

input aluimm, // ALU immediate value

input [4:0] drn, // Destination register address

input [31:0] da, // Value from source register A

input [31:0] db, // Value from source register B

input [31:0] dimm32, // 32-bit immediate value

input clk, // Clock signal

//EC Input

input [31:0] dpc4,

input shift, jal,

// Outputs

output reg ewreg, // Output for write enable signal

output reg em2reg, // Output for write enable signal (M2 stage)

output reg ewmem, // Output for memory write enable signal

output reg [3:0] ealuc, // Output for ALU control signal

output reg ealuimm, // Output for ALU immediate value

output reg [4:0] ern0, // Output for destination register address

output reg [31:0] ea, // Output for source register A value

output reg [31:0] eb, // Output for source register B value

output reg [31:0] eimm32, // Output for 32-bit immediate value

//EC Output

output reg [31:0] epc4,

output reg ejal, eshift

);

// On the positive edge of the clock, update the output signals with the input values.

always @ (posedge clk)

begin

ewreg = wreg;

em2reg = m2reg;

ewmem = wmem;

ealuc = aluc;

ealuimm = aluimm;

ern0 = drn;

ea = da;

eb = db;

eimm32 = dimm32;

epc4 = dpc4;

ejal = jal;

eshift = shift;

end

endmodule // End of the module

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: EXE\_ProgramCounter

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module EXE\_ProgramCounter(

input [31:0] epc4,

output reg [31:0] epc8

);

always @ (\*)

begin

epc8 <= epc4 + 32'b00000000000000000000000000000100;

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: EXE\_ALUMux2

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module EXE\_ALUMux2(

input [31:0] sa,

input [31:0] ea,

input eshift,

output reg [31:0] a

);

always @ (\*)

begin

case(eshift)

2'b0: a = ea;

2'b1: a = sa;

endcase

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: ALUMux

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module ALUMux(

input [31:0] eb, // Input B data from the ALU

input [31:0] eimm32, // Immediate value from the pipeline

input ealuimm, // Mux control signal

output reg [31:0] b // Output data selected by the Mux

);

always @(\*) begin

case(ealuimm)

1'b0: b <= eb;

1'b1: b <= eimm32;

endcase

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: ALU

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module ALU(

input [31:0] a, // Input A for the ALU

input [31:0] b, // Input B for the ALU

input [3:0] ealuc, // ALU control signal

output reg [31:0] r // Output of the ALU

);

// ALU operation codes

// 0000 - AND

// 0001 - OR

// 0010 - ADD

// 0110 - SUBTRACT

// 0111 - SET LESS THAN

// 1100 - NOR

// 0011 - XOR

always @ (\*)

begin

case(ealuc)

4'b0000: r = a & b; // AND operation

4'b0001: r = a | b; // OR operation

4'b0010: r = a + b; // ADD operation

4'b0110: r = a - b; // SUBTRACT operation

4'b1100: r = ~(a | b); // NOR operation

4'b0011: r = a ^ b; // XOR operation

4'b0111: r = b << a;//sll

4'b1110: r = b >> a; //srl

4'b1001: r = $signed(b) >>> a; //sra

4'b0100: r = b << 16; //lui

endcase

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: EXE\_ALUoutMux

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module EXE\_ALUoutMux(

input [31:0] epc8,

input [31:0] r,

input ejal,

output reg [31:0] ealu

);

always @ (\*)

begin

case(ejal)

1'b0: ealu = r;

1'b1: ealu = epc8;

endcase

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: EXE\_f

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module EXE\_f(

input [4:0] ern0,

input ejal,

output reg [4:0] ern

);

always @ (\*)

begin

case(ejal)

1'b0: ern = ern0;

1'b1: ern = 5'b11111;

endcase

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: EXEMEMpipeline

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module EXEMEMpipeline(

input ewreg, // Control signal for writing to the register file

input em2reg, // Control signal for writing to the register file (Memory stage)

input ewmem, // Control signal for writing to memory

input [4:0] ern, // Destination register address

input [31:0] ealu, // Result from the ALU

input [31:0] eb, // Value from source register B

input clk, // Clock signal

output reg mwreg, // Output for write enable signal

output reg mm2reg, // Output for write enable signal (M2 stage)

output reg mwmem, // Output for memory write enable signal

output reg [4:0] mrn, // Output for destination register address

output reg [31:0] malu, // Output for result from the ALU

output reg [31:0] mb // Output for value from source register B

);

always @ (posedge clk)

begin

mwreg = ewreg;

mm2reg = em2reg;

mwmem = ewmem;

mrn = ern;

malu = ealu;

mb = eb;

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: DataMemory

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module DataMemory( // data memory, ram

input clk, // clock

input [31:0] malu, // ram address (addr)

input [31:0] mb, // data in (to memory) (datain)

input mwmem, // write enable (we)

output [31:0] mdo // data out (from memory) (dataout)

);

reg [31:0] ram [0:31]; // ram cells: 32 words \* 32 bits

assign mdo = ram[malu[6:2]]; // use 5-bit word address

always @ (posedge clk) begin

if (mwmem) ram[malu[6:2]] = mb; // write ram

end

integer i;

initial begin // ram initialization

for (i = 0; i < 32; i = i + 1)

ram[i] = 0;

// ram[word\_addr] = data // (byte\_addr) item in data array

ram[5'h14] = 32'h000000a3; // (50) data[0] 0 + a3 = a3

ram[5'h15] = 32'h00000027; // (54) data[1] a3 + 27 = ca

ram[5'h16] = 32'h00000079; // (58) data[2] ca + 79 = 143

ram[5'h17] = 32'h00000115; // (5c) data[3] 143 + 115 = 258

// ram[5'h18] should be 0x00000258, the sum stored by sw instruction

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: MEMWBpipeline

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module MEMWBpipeline(

input mwreg, // Control signal for writing to the register file

input mm2reg, // Control signal for writing to the register file (Memory Stage)

input [4:0] mrn, // Destination register address

input [31:0] malu, // Result from the data memory

input [31:0] mdo, // Data read from the data memory

input clk, // Clock signal

output reg wwreg, // Output for write enable signal

output reg wm2reg, // Output for write enable signal (Memory stage)

output reg [4:0] wrn, // Output for destination register address

output reg [31:0] walu, // Output for result from the data memory

output reg [31:0] wdo // Output for data read from the data memory

);

always @ (posedge clk)

begin

wwreg = mwreg;

wm2reg = mm2reg;

wrn = mrn;

walu = malu;

wdo = mdo;

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: WbMux

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module WbMux(

input [31:0] walu,

input [31:0] wdo,

input wm2reg,

output reg [31:0] wbData

);

always @ (\*)

begin

if (wm2reg == 0)

wbData = walu;

else

wbData = wdo;

end

endmodule

**TestBench:**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/27/2024 02:48:56 AM

// Design Name:

// Module Name: TestBench

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module TestBench;

reg clk;

reg clrn;

wire [31:0] pc;

wire [31:0] dinstOut;

wire [31:0] ealu;

wire [31:0] malu;

wire [31:0] wbData;

initial begin

clk <= 1'b0;

clrn <=1'b0;

end

Datapath Datapath(.clrn(clrn), .clk(clk), .pc(pc), .dinstOut(dinstOut), .ealu(ealu), .malu(malu), .wbData(wbData));

// Clock generation

always begin

#5;

clk = ~clk;

end

always begin

#2; clrn = 1'b1;

end

endmodule