Contents

[Source Code 2](#_Toc393043649)

[ECE429\_Fetch.v: Fetch Module 2](#_Toc393043650)

[ECE429\_CPU.v: Main Pipeline 2](#_Toc393043651)

[ECE429\_CPU\_tb.v: Test Bench 8](#_Toc393043652)

[Testing 10](#_Toc393043653)

[Register File: Initial 10](#_Toc393043654)

[Register File: After Simple If 10](#_Toc393043655)

[Register File: After Simple Add 11](#_Toc393043656)

[Register File: After Bubble Sort 11](#_Toc393043657)

[Register File: After fact 12](#_Toc393043658)

[Register File: After Check Vowel 13](#_Toc393043659)

# Source Code

## ECE429\_Fetch.v: Fetch Module

module ECE429\_Fetch(clk\_in, pc\_in, pc\_out, pc\_decode\_out, rw\_out, stall\_in, access\_size\_out);

input clk\_in;

input [0:31] pc\_in;

input stall\_in;

output [0:31] pc\_out;

output [0:31] pc\_decode\_out;

output rw\_out; // 0 to read, 1 to write

output [0:1] access\_size\_out; // 11 for word, 10 for half-word, 01/00 byte

reg [0:31] program\_counter;

assign pc\_out = program\_counter;

assign pc\_decode\_out = program\_counter;

assign access\_size\_out = 2'b11;

assign rw\_out = 1'b0;

reg stallDelay1; // TODO : Remove after PD4

reg stallDelay2; // TODO : Remove after PD4

initial begin

program\_counter = 32'h80020000;

stallDelay1 = 1; // TODO : Remove after PD4

stallDelay2 = 1; // TODO : Remove after PD4

end

always @(posedge clk\_in) // TODO : Remove after PD4

begin

stallDelay1 <= stall\_in;

stallDelay2 <= stallDelay1;

end

always @ (posedge clk\_in)

begin

//if (!stall\_in) begin

if (!stallDelay2) begin // TODO : Change back after PD4

//program\_counter <= program\_counter + 4; // Changed for PD4

program\_counter <= pc\_in;

end

end

endmodule

## ECE429\_CPU.v: Main Pipeline

include "ECE429\_ControlBits.v";

include "ECE429\_RegFile.v";

module ECE429\_CPU(clock, stall, parseAddr, maxfetchAddr, parseData, parseAccessSize);

input clock;

input[0:31] parseData;

input stall;

input[0:1] parseAccessSize; // The access size to write to the memory using

input[0:31] maxfetchAddr;

input[0:31] parseAddr;

wire[0:31] ImemAddr;

wire[0:31] DmemAddr;

wire[0:31] ImemDataIn;

wire[0:31] DmemDataIn;

wire[0:1] ImemAccessSize;

wire[0:1] DmemAccessSize;

wire ImemR\_W; // Whether to read or write to memory

wire DmemR\_W;

wire fetchR\_W;

wire[0:31] fetchAddr;

wire[0:31] decAddr;

wire[0:1] fetchAccessSize;

wire[0:31] insn\_in;

wire[0:4] rsIn;

wire[0:4] rtIn;

wire[0:4] rdIn;

wire[0:31] rsOut;

wire[0:31] rtOut;

wire[0:15] immdOut;

wire[0:31] aluOut;

wire[0:25] jmpAddr;

wire aluBran;

wire[0:31] eJmpAddr;

wire[0:31] eBranAddr;

wire[0:4] shiftamt;

wire[0:31] regIn;

reg[0:31] finalRD;

reg[0:31] inputB;

reg[0:31] inputA;

reg[0:31] tmp\_insn\_in;

reg fetchStall;

reg[0:31] DX\_PC;

reg[0:31] DX\_IR;

reg[0:15] DX\_Immd;

reg[0:25] DX\_Jmp;

reg[0:4] DX\_SHAMT;

reg[0:31] DX\_Branch;

reg[0:31] DX\_Jump;

reg[0:31] DX\_Final\_PC;

reg[0:4] DX\_RD;

reg[0:31] XM\_PC;

reg[0:31] XM\_IR;

reg[0:4] XM\_RD;

reg[0:`CONTROL\_BITS\_END] DX\_Ctrl;

reg[0:`CONTROL\_BITS\_END] XM\_Ctrl;

reg[0:`CONTROL\_BITS\_END] MW\_Ctrl;

reg[0:31] XM\_O;

reg[0:31] XM\_B;

reg[0:31] MW\_O;

reg[0:31] MW\_IR;

reg[0:31] MW\_PC;

reg[0:4] MW\_RD;

reg[0:31] Ddataout\_Final;

reg[ 0:2 ] nopCount; // TODO: Remove. Only for sending 4 nops and 1 instr.

reg FD\_STALL; // TODO: Remove. Only for sending 4 nops and 1 instr.

always @\* begin // TODO: Remove. Only for sending 4 nops and 1 instr.

if (stall == 1) begin

FD\_STALL = stall;

end else begin

if ( nopCount == 0 ) begin

FD\_STALL = 0;

end else begin

FD\_STALL = 1;

end

end

end

always @(posedge clock) begin // TODO: Remove. Only for sending 4 nops and 1 instr.

if( stall == 0 ) begin

if(nopCount == 4) begin

nopCount = 0;

end else begin

nopCount = nopCount + 1;

end

end

end

wire [0:31] Idataout; // To see output of the memory

wire [0:31] Ddataout;

wire [0:`CONTROL\_BITS\_END] control\_array;

reg[0:31] SignExtendedImmed;

initial begin

tmp\_insn\_in = 32'h00000000;

fetchStall = stall;

DX\_IR <= 32'h00000000;

DX\_PC <= 32'h00000000;

DX\_Ctrl <= 18'h00000;

XM\_IR <= 32'h00000000;

XM\_PC <= 32'h00000000;

XM\_Ctrl <= 18'h00000;

nopCount <= 3'b000; // TODO: Remove after PD4

end

assign ImemR\_W = (!stall) ? fetchR\_W : stall;

assign ImemAddr = (!stall) ? fetchAddr : parseAddr;

assign ImemAccessSize = (!stall) ? fetchAccessSize : parseAccessSize;

assign ImemDataIn = parseData;

assign DmemR\_W = (!stall) ? XM\_Ctrl[`DMWE\_OFFSET] : ImemR\_W;

assign DmemAddr = (!stall) ? XM\_O : ImemAddr;

assign DmemAccessSize = (!stall) ? XM\_Ctrl[`MEMAS\_BEGIN\_OFFSET:`MEMAS\_END\_OFFSET] : ImemAccessSize;

assign DmemDataIn = (!stall) ? XM\_B : ImemDataIn;

assign decAddr = fetchAddr;

assign insn\_in = (fetchStall) ? 32'h00000000 : Idataout;

//wire[0:31] DX\_DelaySlot\_PC;

//assign DX\_DelaySlot\_PC = DX\_PC+4;

assign eJmpAddr = { DX\_PC[0:3], DX\_Jmp, 2'b00 };

assign eBranAddr = (SignExtendedImmed << 2) + DX\_PC;

assign regIn = (MW\_Ctrl[`RWD\_OFFSET]) ? Ddataout\_Final : MW\_O;

always @\* begin

if (MW\_Ctrl[`BSX\_OFFSET]) begin

Ddataout\_Final = { {24{Ddataout[24]}}, Ddataout[24:31]};

end else begin

Ddataout\_Final = Ddataout;

end

end

always @\* begin

if (control\_array[`RDST\_OFFSET]) begin

finalRD = rdIn;

end else begin

finalRD = rtIn;

end

end

always @\* begin

SignExtendedImmed = { {16{DX\_Immd[0]}}, DX\_Immd[0:15] };

end

always @\*

begin

if(DX\_Ctrl[`JALOP\_OFFSET] == 1) begin

inputA = DX\_PC;

end else begin

inputA = rsOut;

end

end

always @\*

begin

if (DX\_Ctrl[`JALOP\_OFFSET] == 1) begin

inputB = 32'h00000008;

end else begin

if (DX\_Ctrl[`ALUINB\_BEGIN\_OFFSET:`ALUINB\_END\_OFFSET] == 1) begin

inputB = SignExtendedImmed;

end else if (DX\_Ctrl[`ALUINB\_BEGIN\_OFFSET:`ALUINB\_END\_OFFSET] == 2) begin

inputB = {16'h0000, DX\_Immd[0:15]};

end else begin

inputB = rtOut;

end

end

end

always @\* begin

if (DX\_Ctrl[`BR\_OFFSET] == 1 && aluBran == 1) begin

DX\_Branch = eBranAddr;

end else begin

//DX\_Branch = DX\_PC;

DX\_Branch = fetchAddr + 4;

end

end

always @\* begin

if (DX\_Ctrl[`JP\_OFFSET] == 1) begin

DX\_Jump = eJmpAddr;

end else begin

DX\_Jump = DX\_Branch;

end

end

always @\* begin

if (DX\_Ctrl[`JROP\_OFFSET] == 1) begin

DX\_Final\_PC = rsOut;

end else begin

DX\_Final\_PC = DX\_Jump;

end

end

always @(posedge clock) begin

if (!stall) begin

$display("rising edge\n");

end

end

always @(posedge clock) begin

//fetchStall <= stall;

fetchStall <= FD\_STALL; // TODO: Change back after PD4

end

always @(posedge clock) begin

if (!stall) begin

DX\_IR <= Idataout;

DX\_PC <= fetchAddr;

DX\_Ctrl <= control\_array;

DX\_Immd <= immdOut;

DX\_Jmp <= jmpAddr;

DX\_SHAMT <= shiftamt;

DX\_RD <= finalRD;

XM\_IR <= DX\_IR;

XM\_PC <= DX\_PC;

XM\_Ctrl <= DX\_Ctrl;

XM\_O <= aluOut;

XM\_RD <= DX\_RD;

XM\_B <= rtOut;

MW\_IR <= XM\_IR;

MW\_PC <= XM\_PC;

MW\_Ctrl <= XM\_Ctrl;

MW\_O <= XM\_O;

MW\_RD <= XM\_RD;

end

end

always @(negedge clock) begin

if (stall == 0) begin // TODO: Might need to change back to fetchStall after PD4

$display("read at mem[0x%h] = 0x%h\n", fetchAddr, Idataout); // TODO: Might need to change to fetchAddr-4 after PD4

/\*$display("DX\_IR: 0x%h, DX\_PC: 0x%h, DX\_Ctrl: 0x%h,

DX\_Final\_PC: 0x%h, DX\_SHAMT: 0x%h, XM\_IR: 0x%h, XM\_PC: 0x%h, XM\_Ctrl: 0x%h,

XM\_O: 0x%h, eBranAddr: 0x%h, eJmpAddr: 0x%h, aluop: 0x%h\n", DX\_IR,

DX\_PC, DX\_Ctrl, DX\_Final\_PC, DX\_SHAMT, XM\_IR, XM\_PC, XM\_Ctrl, XM\_O,

eBranAddr, eJmpAddr, DX\_Ctrl[`ALUOP\_BEGIN\_OFFSET:`ALUOP\_END\_OFFSET]);\*/

//if(fetchAddr >= (maxfetchAddr + 20)) begin

if((fetchAddr > (`REGFILE\_SPECIAL\_RA)) && (fetchAddr <= (`REGFILE\_SPECIAL\_RA+4))) begin

//if((fetchAddr < 100)) begin

$finish;

end

end

end

ECE429\_Fetch f(

.clk\_in(clock),

.pc\_in(DX\_Final\_PC),

.pc\_out(fetchAddr),

.pc\_decode\_out(decAddr),

.rw\_out(fetchR\_W),

//.stall\_in(stall), // TODO: Change back after PD4

.stall\_in(FD\_STALL),

.access\_size\_out(fetchAccessSize)

);

ECE429\_Memory m(

.clock(clock),

.address(ImemAddr),

.datain(ImemDataIn),

.access\_size(ImemAccessSize),

.r\_w(ImemR\_W),

.dataout(Idataout)

);

ECE429\_Memory dm(

.clock(clock),

.address(DmemAddr),

.datain(DmemDataIn),

.access\_size(DmemAccessSize),

.r\_w(DmemR\_W),

.dataout(Ddataout)

);

ECE429\_Decode d(

.insn\_in(insn\_in),

.pc\_in(decAddr),

.controlBitVector(control\_array),

.op\_code(),

.reg\_RS(rsIn),

.reg\_RT(rtIn),

.reg\_RD(rdIn),

.reg\_SHAMT(shiftamt),

.reg\_FUNCT(),

.jump\_ADDR(jmpAddr),

.immediate\_value(immdOut)

);

ECE429\_ALU alu(

.inputA(inputA),

.inputB(inputB),

.inputSHAMT(DX\_SHAMT),

.ALUop(DX\_Ctrl[`ALUOP\_BEGIN\_OFFSET:`ALUOP\_END\_OFFSET]),

.takeBranch(aluBran),

.ALUOutput(aluOut)

);

ECE429\_RegFile rf(

.clock(clock),

.rsIn(rsIn),

.rsOut(rsOut),

.rtIn(rtIn),

.rtOut(rtOut),

.rdIn(MW\_RD),

.rd\_dataIn(regIn),

.control(MW\_Ctrl)

);

Endmodule

## ECE429\_CPU\_tb.v: Test Bench

module ECE429\_CPU\_tb();

reg clock;

reg parseEnable;

reg[0:31] maxfetchAddr;

wire[0:31] parseAddr;

wire[0:31] parseData;

wire[0:1] parseAccessSize;

wire parseDone;

wire parseError;

wire cpuStall;

initial begin

clock = 0;

maxfetchAddr = 32'h00000000;

parseEnable = 0;

#1

parseEnable = 1;

#1

parseEnable = 0;

@(posedge clock);

end

always begin

#10 clock = ~clock;

end

assign cpuStall = ~parseDone;

always @(parseAddr) begin

if(parseAddr > maxfetchAddr) begin

maxfetchAddr = parseAddr;

end

end

always @(parseError) begin

if(parseError == 1) begin

$finish;

end

end

ECE429\_CPU c(

.clock(clock),

.stall(cpuStall),

.parseAddr(parseAddr),

.maxfetchAddr(maxfetchAddr),

.parseData(parseData),

.parseAccessSize(parseAccessSize)

);

ECE429\_SRECParser #("CheckVowel.srec") s(

.clock(clock),

.parseEnable(parseEnable),

.parseAddr(parseAddr),

.memData(parseData),

.parseAccessSize(parseAccessSize),

.parseDone(parseDone),

.parseError(parseError)

);

endmodule

# Testing

## Register File: Initial

# r 0: 0

# r 1: 1

# r 2: 2

# r 3: 3

# r 4: 4

# r 5: 5

# r 6: 6

# r 7: 7

# r 8: 8

# r 9: 9

# r 10: 10

# r 11: 11

# r 12: 12

# r 13: 13

# r 14: 14

# r 15: 15

# r 16: 16

# r 17: 17

# r 18: 18

# r 19: 19

# r 20: 20

# r 21: 21

# r 22: 22

# r 23: 23

# r 24: 24

# r 25: 25

# r 26: 26

# r 27: 27

# r 28: 28

# r 29: 2148663296

# r 30: 30

# r 31: 0

## Register File: After Simple If

# r 0: 0

# r 1: 1

# r 2: 7

# r 3: 5

# r 4: 4

# r 5: 5

# r 6: 6

# r 7: 7

# r 8: 8

# r 9: 9

# r 10: 10

# r 11: 11

# r 12: 12

# r 13: 13

# r 14: 14

# r 15: 15

# r 16: 16

# r 17: 17

# r 18: 18

# r 19: 19

# r 20: 20

# r 21: 21

# r 22: 22

# r 23: 23

# r 24: 24

# r 25: 25

# r 26: 26

# r 27: 27

# r 28: 28

# r 29: 2148663296

# r 30: 30

# r 31: 0

## Register File: After Simple Add

# r 0: 0

# r 1: 1

# r 2: 5

# r 3: 3

# r 4: 4

# r 5: 5

# r 6: 6

# r 7: 7

# r 8: 8

# r 9: 9

# r 10: 10

# r 11: 11

# r 12: 12

# r 13: 13

# r 14: 14

# r 15: 15

# r 16: 16

# r 17: 17

# r 18: 18

# r 19: 19

# r 20: 20

# r 21: 21

# r 22: 22

# r 23: 23

# r 24: 24

# r 25: 25

# r 26: 26

# r 27: 27

# r 28: 28

# r 29: 2148663296

# r 30: 30

# r 31: 0

## Register File: After Bubble Sort

# r 0: 0

# r 1: 1

# r 2: 0

# r 3: 8

# r 4: 2148663256

# r 5: 8

# r 6: 6

# r 7: 7

# r 8: 8

# r 9: 9

# r 10: 10

# r 11: 11

# r 12: 12

# r 13: 13

# r 14: 14

# r 15: 15

# r 16: 16

# r 17: 17

# r 18: 18

# r 19: 19

# r 20: 20

# r 21: 21

# r 22: 22

# r 23: 23

# r 24: 24

# r 25: 25

# r 26: 26

# r 27: 27

# r 28: 28

# r 29: 2148663296

# r 30: 30

# r 31: 0

## Register File: After fact

# r 0: 0

# r 1: 1

# r 2: 0

# r 3: 40320

# r 4: 0

# r 5: 5

# r 6: 6

# r 7: 7

# r 8: 8

# r 9: 9

# r 10: 10

# r 11: 11

# r 12: 12

# r 13: 13

# r 14: 14

# r 15: 15

# r 16: 16

# r 17: 17

# r 18: 18

# r 19: 19

# r 20: 20

# r 21: 21

# r 22: 22

# r 23: 23

# r 24: 24

# r 25: 25

# r 26: 26

# r 27: 27

# r 28: 28

# r 29: 2148663296

# r 30: 30

# r 31: 0

## Register File: After Check Vowel

# r 0: 0

# r 1: 1

# r 2: 0

# r 3: 0

# r 4: 1800826743

# r 5: 1130915171

# r 6: 6

# r 7: 7

# r 8: 8

# r 9: 9

# r 10: 10

# r 11: 11

# r 12: 12

# r 13: 13

# r 14: 14

# r 15: 15

# r 16: 16

# r 17: 17

# r 18: 18

# r 19: 19

# r 20: 20

# r 21: 21

# r 22: 22

# r 23: 23

# r 24: 24

# r 25: 25

# r 26: 26

# r 27: 27

# r 28: 28

# r 29: 2148663296

# r 30: 30

# r 31: 0