Contents

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
```