ASSIGNMENT 5 REPORT

ES 204: Digital Systems  
Indian Institute of Technology Gandhinagar

April 19, 2025

Devansh Lodha (23110091), Laksh Jain (23110185)

# SIMPLE PROCESSOR

## DESIGN CODE

## 1] processor.v

`timescale 1ns / 1ps

module processor (

input clk,

input reset,

output [7:0] pc,

output [7:0] reg0, reg1, reg2, reg3, reg4, reg5, reg6, reg7,

output [7:0] reg8, reg9, reg10, reg11, reg12, reg13, reg14, reg15,

output reg cb\_reg

);

// We will expose the reg values to see the waveform

assign reg0 = reg\_file.regs[0];

assign reg1 = reg\_file.regs[1];

assign reg2 = reg\_file.regs[2];

assign reg3 = reg\_file.regs[3];

assign reg4 = reg\_file.regs[4];

assign reg5 = reg\_file.regs[5];

assign reg6 = reg\_file.regs[6];

assign reg7 = reg\_file.regs[7];

assign reg8 = reg\_file.regs[8];

assign reg9 = reg\_file.regs[9];

assign reg10 = reg\_file.regs[10];

assign reg11 = reg\_file.regs[11];

assign reg12 = reg\_file.regs[12];

assign reg13 = reg\_file.regs[13];

assign reg14 = reg\_file.regs[14];

assign reg15 = reg\_file.regs[15];

// Program Counter and Instruction Fetch

wire [7:0] instruction;

reg [7:0] pc\_reg;

// Instruction Memory

instruction\_memory ins\_mem (

.addr(pc\_reg),

.instruction(instruction)

);

// Other Components

wire [7:0] data\_bus;

wire [7:0] acc\_value, alu\_result, ext\_result;

wire alu\_carry;

reg [7:0] ext\_reg;

// Control Signals

wire reg\_read\_en;

wire [3:0] reg\_read\_addr;

wire reg\_write\_en;

wire [3:0] reg\_write\_addr;

wire acc\_sel, acc\_write\_en, acc\_output\_en;

wire [3:0] alu\_op;

wire ext\_write\_en, cb\_write\_en;

wire [7:0] next\_pc;

register\_file regs (

.clk(clk),

.read\_addr(reg\_read\_addr),

.read\_en(reg\_read\_en),

.write\_addr(reg\_write\_addr),

.write\_en(reg\_write\_en),

.bus(data\_bus)

);

accumulator acc (

.clk(clk),

.alu\_in(alu\_result),

.bus\_in(data\_bus),

.acc\_sel(acc\_sel),

.write\_en(acc\_write\_en),

.output\_en(acc\_output\_en),

.acc\_value(acc\_value),

.bus\_out(data\_bus)

);

alu alu (

.acc(acc\_value),

.bus(data\_bus),

.alu\_op(alu\_op),

.result(alu\_result),

.carry(alu\_carry),

.ext\_result(ext\_result)

);

control\_unit ctrl\_unt (

.clk(clk),

.reset(reset),

.instruction(instruction),

.cb\_reg(cb\_reg),

.alu\_carry(alu\_carry),

.reg\_read\_en(reg\_read\_en),

.reg\_read\_addr(reg\_read\_addr),

.reg\_write\_en(reg\_write\_en),

.reg\_write\_addr(reg\_write\_addr),

.acc\_sel(acc\_sel),

.acc\_write\_en(acc\_write\_en),

.acc\_output\_en(acc\_output\_en),

.alu\_op(alu\_op),

.ext\_write\_en(ext\_write\_en),

.cb\_write\_en(cb\_write\_en),

.next\_pc(next\_pc),

.current\_pc(pc\_reg),

.bus\_value(data\_bus)

);

always @(posedge clk) begin

if (ext\_write\_en) ext\_reg <= ext\_result;

if (reset) cb\_reg <= 0;

else if (cb\_write\_en) cb\_reg <= alu\_carry;

if (reset) pc\_reg <= 0;

else pc\_reg <= next\_pc;

end

assign pc = pc\_reg;

endmodule

## 2] instruction\_memory.v

module instruction\_memory(

input [7:0] address,

output [7:0] instruction

);

reg [7:0] mem [0:32]; // 32-byte instruction memory

integer i;

initial begin

// Load test program (Here we can add any set of instructions)

// For example:

mem[0] = 8'b1001\_0101; // MOV R5 to ACC

mem[1] = 8'b0001\_0100; // ADD R4 WITH ACC

mem[2] = 8'b1111\_1111; // HALT

// Here we fill the rest of the memory with HALT because if

// in case PC jumps to address value other than above then

// we do not want the program to run indefinitely.

// This also ensures that the memory it is pointing to is

// always defined.

for (i = 3; i < 32; i = i + 1) begin

mem[i] = 8'b1111\_1111;

end

end

assign instruction = mem[address];

endmodule

## 3] register\_file.v

`timescale 1ns / 1ps

module register\_file (

input clk,

input [3:0] read\_addr,

input read\_en,

input [3:0] write\_addr,

input write\_en,

inout [7:0] bus

);

reg [7:0] regs[0:15];

// Initialize at start of the clock

integer i;

initial begin

for (i = 0; i < 16; i = i + 1) begin

regs[i] = 8'h00 + i;

end

end

wire [7:0] reg\_out = read\_en ? regs[read\_addr] : 8'bz;

assign bus = reg\_out;

always @(posedge clk) begin

if (write\_en) regs[write\_addr] <= bus;

end

endmodule

## 4] control\_unit.v

`timescale 1ns / 1ps

module control\_unit (

input clk,

input reset,

input [7:0] instruction,

input cb\_reg,

input alu\_carry,

output reg reg\_read\_en,

output reg [3:0] reg\_read\_addr,

output reg reg\_write\_en,

output reg [3:0] reg\_write\_addr,

output reg acc\_sel,

output reg acc\_write\_en,

output reg acc\_output\_en,

output reg [3:0] alu\_op,

output reg ext\_write\_en,

output reg cb\_write\_en,

output reg [7:0] next\_pc,

input [7:0] current\_pc,

input [7:0] bus\_value

);

parameter ADD=0, SUB=1, MUL=2, AND=3, XOR=4, LS=5, RS=6, CRS=7, CLS=8, ASR=9, INC=10, DEC=11;

always @(\*) begin

reg\_read\_en = 0;

reg\_read\_addr = 0;

reg\_write\_en = 0;

reg\_write\_addr = 0;

acc\_sel = 0;

acc\_write\_en = 0;

acc\_output\_en = 0;

alu\_op = 0;

ext\_write\_en = 0;

cb\_write\_en = 0;

next\_pc = current\_pc + 1;

casez (instruction)

8'b0001\_zzzz: begin // ADD

reg\_read\_en = 1;

reg\_read\_addr = instruction[3:0];

alu\_op = ADD;

acc\_sel = 1;

acc\_write\_en = 1;

cb\_write\_en = 1;

end

8'b0010\_zzzz: begin // SUB

reg\_read\_en = 1;

reg\_read\_addr = instruction[3:0];

alu\_op = SUB;

acc\_sel = 1;

acc\_write\_en = 1;

cb\_write\_en = 1;

end

8'b0011\_zzzz: begin // MUL

reg\_read\_en = 1;

reg\_read\_addr = instruction[3:0];

alu\_op = MUL;

acc\_sel = 1;

acc\_write\_en = 1;

ext\_write\_en = 1;

end

8'b0000\_0001: begin // LS

alu\_op = LS;

acc\_sel = 1;

acc\_write\_en = 1;

end

8'b0000\_0010: begin // RS

alu\_op = RS;

acc\_sel = 1;

acc\_write\_en = 1;

end

8'b0000\_0011: begin // CRS

alu\_op = CRS;

acc\_sel = 1;

acc\_write\_en = 1;

end

8'b0000\_0100: begin // CLS

alu\_op = CLS;

acc\_sel = 1;

acc\_write\_en = 1;

end

8'b0000\_0101: begin // ARS

alu\_op = ASR;

acc\_sel = 1;

acc\_write\_en = 1;

end

8'b0101\_zzzz: begin // AND

reg\_read\_en = 1;

reg\_read\_addr = instruction[3:0];

alu\_op = AND;

acc\_sel = 1;

acc\_write\_en = 1;

end

8'b0110\_zzzz: begin // XOR

reg\_read\_en = 1;

reg\_read\_addr = instruction[3:0];

alu\_op = XOR;

acc\_sel = 1;

acc\_write\_en = 1;

end

8'b0111\_zzzz: begin // CMP

reg\_read\_en = 1;

reg\_read\_addr = instruction[3:0];

alu\_op = SUB;

cb\_write\_en = 1;

end

8'b0000\_0110: begin // INC

alu\_op = INC;

acc\_sel = 1;

acc\_write\_en = 1;

cb\_write\_en = alu\_carry; // Using carry we can ensure it updates only at overflow

end

8'b0000\_0111: begin // DEC

alu\_op = DEC;

acc\_sel = 1;

acc\_write\_en = 1;

cb\_write\_en = alu\_carry; // Using carry we can ensure it updates only at underflow

end

8'b1000\_zzzz: begin // Conditional branch

if (cb\_reg) next\_pc = {4'b0, instruction[3:0]};

end

8'b1001\_zzzz: begin // MOV Ri to ACC

reg\_read\_en = 1;

reg\_read\_addr = instruction[3:0];

acc\_write\_en = 1;

end

8'b1010\_zzzz: begin // MOV ACC to Ri

acc\_output\_en = 1;

reg\_write\_en = 1;

reg\_write\_addr = instruction[3:0];

end

8'b1011\_zzzz: begin // RETURN

next\_pc = {4'b0, instruction[3:0]};

end

8'b1111\_1111: next\_pc = current\_pc; // HALT

default: begin

next\_pc = current\_pc + 1;

end

endcase

end

endmodule

## 5] alu.v

`timescale 1ns / 1ps

module alu (

input [7:0] acc,

input [7:0] bus,

input [3:0] alu\_op,

output reg [7:0] result,

output reg carry,

output reg [7:0] ext\_result

);

parameter ADD=0, SUB=1, MUL=2, AND=3, XRA=4, LS=5, RS=6, CRS=7, CLS=8, ASR=9, INC=10, DEC=11;

always @(\*) begin

{carry, result} = 9'b0;

ext\_result = 8'b0;

case (alu\_op)

ADD: {carry, result} = acc + bus;

SUB: {carry, result} = acc - bus;

MUL: {ext\_result, result} = acc \* bus;

AND: result = acc & bus;

XRA: result = acc ^ bus;

LS: result = {acc[6:0], 1'b0};

RS: result = {1'b0, acc[7:1]};

CRS: result = {acc[0], acc[7:1]};

CLS: result = {acc[6:0], acc[7]};

ASR: result = {acc[7], acc[7:1]};

INC: {carry, result} = {1'b0, acc} + 9'h1;

DEC: {carry, result} = {1'b0, acc} - 9'h1;

default: result = acc;

endcase

end

endmodule

## 6] accumulator.v

`timescale 1ns / 1ps

module accumulator (

input clk,

input [7:0] alu\_in,

input [7:0] bus\_in,

input acc\_sel,

input write\_en,

input output\_en,

output reg [7:0] acc\_value,

inout [7:0] bus\_out

);

wire [7:0] next\_acc = acc\_sel ? alu\_in : bus\_in;

assign bus\_out = output\_en ? acc\_value : 8'bz;

always @(posedge clk) begin

if (write\_en) acc\_value <= next\_acc;

end

endmodule

## TESTBENCH CODE (THIS REMAINS SAME FOR ALL SIMULATIONS. TO SIMULATE DIFFERENT INSTRUCTION SETS OR INDIVIDUAL INSTRUCTIONS ONLY, HARDCODE IT TO THE INSTRUCTIONS MEMORY)

`timescale 1ns / 1ps

module processor\_TB();

reg clk, reset;

wire [7:0] pc;

wire [7:0] reg0, reg1, reg2, reg3, reg4, reg5, reg6, reg7,

reg8, reg9, reg10, reg11, reg12, reg13, reg14, reg15;

wire cb\_reg;

processor uut (

.clk(clk),

.reset(reset),

.pc(pc),

.cb\_reg(cb\_reg),

.reg0(reg0), .reg1(reg1), .reg2(reg2), .reg3(reg3),

.reg4(reg4), .reg5(reg5), .reg6(reg6), .reg7(reg7),

.reg8(reg8), .reg9(reg9), .reg10(reg10),.reg11(reg11),

.reg12(reg12),.reg13(reg13),.reg14(reg14),.reg15(reg15)

);

always #5 clk = ~clk;

initial begin

clk = 0;

reset = 1;

#10 reset = 0; // Release reset after 10ns

// Run until HALT (PC stops changing)

#100 $finish;

end

endmodule

## 