Al-Azhar University
Faculty of Engineering (boys)
Systems and Computer Engineering

جامعة الأزهر
كلية الهندسة (بنين)
قسم هندسة النظم والحاسبات

Project (2)

Simple Computer Simulation

Systems and Computers Department
Faculty of Engineering
Al-AzharUniversity

By:

StudentNumber	Name

ALU Module:

```
module ALU (
  input [7:0] A,  // Operand A
  input [7:0] B,  // Operand B
  input [2:0] ALUOp,  // Operation select
  output reg [7:0] Result,  // Output
  output reg Zero,  // Zero flag
  output reg Carry  // Carry/Overflow flag
);

always @(*) begin
  Carry = 0;
  case (ALUOp)
    3'b000: {Carry, Result} = A + B;  // Addition with carry
    3'b001: {Carry, Result} = A - B;  // Subtraction with borrow (Carry=1 if borrow)
    3'b010: Result = A & B;  // AND
    3'b011: Result = A | B;  // OR
    3'b100: Result = A ^ B;  // XOR (added)
    3'b101: Result = ~A;  // NOT (moved from 100 to 101 if needed, but kept for compatibility)
    default: Result = 8'b0;
  endcase
  Zero = (Result == 8'b0);
end
endmodule
```

InstructionMemory Module:

```
module InstructionMemory (
  input [7:0] address,
  output reg [7:0] instruction
);

reg [7:0] memory [0:255];  // Increased to 256 instructions

initial begin
  // Preload example instructions (using new ISA)
  memory[0] = 8'b10100010;  // LOAD R0, 2 (Dst=R0, Imm=2)
  memory[1] = 8'b10101011;  // LOAD R1, 3 (Dst=R1, Imm=3)
  memory[2] = 8'b00001000;  // ADD R1, R0 (Dst=R1, Src=R0)
  memory[3] = 8'b11010000;  // STORE [0], R1 (Addr=0, Dst=R1 unused for store)
  memory[4] = 8'b00100001;  // SUB R0, R1 (Dst=R0, Src=R1)
  memory[5] = 8'b11100010;  // JZ 2 (if Zero, jump to addr=2)
  memory[6] = 8'b00000000;  // HLT (all zero as NOP/HLT)
  // Add more as needed
end

always @(*) begin
  instruction = memory[address];
end
endmodule
```

ControlUnit Module:

```
module ControlUnit (
  input [2:0] opcode,
  output reg [2:0] alu_op,
  output reg RegWrite,  // Write to register
  output reg MemWrite,  // Write to data memory
  output reg Branch     // Branch if Zero
);

always @(*) begin
  case (opcode)
    3'b000: begin  // ADD
      alu_op = 3'b000;
      RegWrite = 1;
      MemWrite = 0;
      Branch = 0;
    end
    3'b001: begin  // SUB
      alu_op = 3'b001;
      RegWrite = 1;
      MemWrite = 0;
      Branch = 0;
    end
    3'b010: begin  // AND
      alu_op = 3'b010;
      RegWrite = 1;
      MemWrite = 0;
      Branch = 0;
    end
    3'b011: begin  // OR
      alu_op = 3'b011;
      RegWrite = 1;
      MemWrite = 0;
      Branch = 0;
    end
    3'b100: begin  // XOR
      alu_op = 3'b100;
      RegWrite = 1;
      MemWrite = 0;
      Branch = 0;
    end
    3'b101: begin  // LOAD
      alu_op = 3'b000;  // Not used
      RegWrite = 1;
      MemWrite = 0;
      Branch = 0;
    end
    3'b110: begin  // STORE
      alu_op = 3'b000;  // Not used
      RegWrite = 0;
      MemWrite = 1;
      Branch = 0;
    end
    3'b111: begin  // JZ
      alu_op = 3'b000;
      RegWrite = 0;
      MemWrite = 0;
      Branch = 1;
    end
    default: begin
      alu_op = 3'b000;
      RegWrite = 0;
      MemWrite = 0;
      Branch = 0;
    end
  endcase
end
endmodule
```

RegisterFile Module:

```
module RegisterFile (
  input clk,
  input reset,
  input RegWrite,
  input [1:0] read_addr1,
  input [1:0] read_addr2,
  input [1:0] write_addr,
  input [7:0] write_data,
  output [7:0] read_data1,
  output [7:0] read_data2
);

reg [7:0] reg_file [0:3];

always @(posedge clk or posedge reset) begin
  if (reset) begin
    reg_file[0] <= 8'b0;
    reg_file[1] <= 8'b0;
    reg_file[2] <= 8'b0;
    reg_file[3] <= 8'b0;
  end else if (RegWrite) begin
    reg_file[write_addr] <= write_data;
  end
end

assign read_data1 = reg_file[read_addr1];
assign read_data2 = reg_file[read_addr2];
endmodule
```

ProgramCounter Module:

```
module ProgramCounter (
  input clk,
  input reset,
  input Branch,         // From ControlUnit
  input Zero,           // From ALU
  input [7:0] branch_addr,  // From instruction
  output reg [7:0] pc_address
);

wire [7:0] next_pc = pc_address + 1;  // Normal increment
wire take_branch = Branch & Zero;     // Branch if enabled and Zero

always @(posedge clk or posedge reset) begin
  if (reset) pc_address <= 8'b0;
  else pc_address <= take_branch ? branch_addr : next_pc;
end
endmodule
```

SimpleCPU Module:

```
module SimpleCPU (
  input clk,
  input reset,
  output [7:0] alu_result  // For observation
);

wire [7:0] pc_address;
wire [7:0] instruction;
wire [2:0] opcode = instruction[7:5];
wire [1:0] dst_reg = instruction[4:3];
wire [2:0] src_imm = instruction[2:0];  // Src reg or imm/addr

wire [2:0] alu_op;
wire RegWrite, MemWrite, Branch;
wire Zero, Carry;

wire [7:0] reg_data1, reg_data2;
wire [7:0] alu_in2 = (opcode == 3'b101) ? {5'b0, src_imm} : reg_data2;  // Imm for LOAD

// Instruction Fetch
InstructionMemory im (
  .address(pc_address),
  .instruction(instruction)
);

// Control Unit
ControlUnit cu (
  .opcode(opcode),
  .alu_op(alu_op),
  .RegWrite(RegWrite),
  .MemWrite(MemWrite),
  .Branch(Branch)
);

// Register File
RegisterFile rf (
  .clk(clk),
  .reset(reset),
  .RegWrite(RegWrite),
  .read_addr1(dst_reg),  // Dst as read1 for ALU A
  .read_addr2(src_imm[1:0]),  // Src as read2 for ALU B (assume src is reg for arithmetic)
  .write_addr(dst_reg),
  .write_data(alu_result),  // From ALU or Mem? Here from ALU
  .read_data1(reg_data1),
  .read_data2(reg_data2)
);

// ALU
ALU alu (
  .A(reg_data1),
  .B(alu_in2),
  .ALUOp(alu_op),
  .Result(alu_result),
  .Zero(Zero),
  .Carry(Carry)
);

// Data Memory
wire [7:0] mem_read;
DataMemory dm (
  .clk(clk),
  .MemWrite(MemWrite),
  .address({5'b0, src_imm}),  // Addr from instruction
  .write_data(reg_data1),     // Write from reg_data1 (Dst value)
  .read_data(mem_read)
);

// Program Counter with branch
ProgramCounter pc (
  .clk(clk),
  .reset(reset),
  .Branch(Branch),
  .Zero(Zero),
  .branch_addr({5'b0, src_imm}),  // Addr from instruction
  .pc_address(pc_address)
);

endmodule
```

DataMemory Module:

```
module DataMemory (
  input clk,
  input MemWrite,
  input [7:0] address,
  input [7:0] write_data,
  output reg [7:0] read_data
);

reg [7:0] memory [0:255];  // 256x8 RAM

always @(posedge clk) begin
  if (MemWrite) memory[address] <= write_data;
end

always @(*) begin
  read_data = memory[address];
end
endmodule
```

SimpleCPU_tb Module:

```
`timescale 1ns / 1ps

module SimpleCPU_tb;

reg clk;
reg reset;
wire [7:0] alu_result;

// Additional wires for verification (connect to internal signals if needed, or observe outputs)
wire [7:0] pc_address;
wire [7:0] instruction;
wire Zero;
wire Carry;
wire [7:0] mem_data;  // Assume you add output for dm.read_data in SimpleCPU

// Instantiate the unit under test (UUT)
SimpleCPU uut (
  .clk(clk),
  .reset(reset),
  .alu_result(alu_result)
  // Add more outputs if you expose them in SimpleCPU.v for testing
);

// Clock generation (10ns period)
always #5 clk = ~clk;

// Dump waves for ModelSim
initial begin
  $dumpfile("simplecpu_tb.vcd");
  $dumpvars(0, SimpleCPU_tb);
end

initial begin
  // Initialize signals
  clk = 0;
  reset = 1;

  // Test 1: Apply reset and check PC=0, all regs=0
  #10;
  reset = 0;
  #10;  // Wait one cycle after reset
  if (pc_address !== 8'b0) $display("Test Failed: PC not reset to 0");
  else $display("Test Passed: Reset successful");

  // Run simulation for multiple cycles to test all instructions
  repeat (20) begin  // Run 20 cycles (enough for the preload program)
    #10;  // Wait one full cycle

    // Example assertions based on preload program in InstructionMemory
    case (pc_address)
      8'h00: begin  // LOAD R0, 2
        if (instruction !== 8'b10100010) $display("Test Failed: Wrong instruction at PC=0");
        #10;  // Wait for execute
        if (alu_result !== 8'h02) $display("Test Failed: LOAD failed, alu_result=%h", alu_result);
        else $display("Test Passed: LOAD R0=2");
      end
      8'h01: begin  // LOAD R1, 3
        #10;
        if (alu_result !== 8'h03) $display("Test Failed: LOAD R1=3");
        else $display("Test Passed: LOAD R1=3");
      end
      8'h02: begin  // ADD R1, R0 (R1 = 3 + 2 = 5)
        #10;
        if (alu_result !== 8'h05 || Carry !== 0 || Zero !== 0) $display("Test Failed: ADD result=%h, C=%b, Z=%b", alu_result, Carry, Zero);
        else $display("Test Passed: ADD R1=5");
      end
      8'h03: begin  // STORE [0], R1 (Mem[0] = 5)
        #10;
        if (mem_data !== 8'h05) $display("Test Failed: STORE Mem[0]=%h", mem_data);
        else $display("Test Passed: STORE Mem[0]=5");
      end
      8'h04: begin  // SUB R0, R1 (R0 = 2 - 5 = FD (negative), Carry=1)
        #10;
        if (alu_result !== 8'hFD || Carry !== 1 || Zero !== 0) $display("Test Failed: SUB result=%h, C=%b, Z=%b", alu_result, Carry, Zero);
        else $display("Test Passed: SUB R0=FD");
      end
      8'h05: begin  // JZ 2 (Zero=0, so no jump, PC=6)
        #10;
        if (pc_address !== 8'h06) $display("Test Failed: JZ should not branch");
        else $display("Test Passed: No branch on JZ");
      end
      default: $display("Simulation at PC=%h, instruction=%b, alu_result=%h", pc_address, instruction, alu_result);
    endcase
  end

  // Additional test: Force a branch by setting Zero manually (for demo, in real sim use stimulus)
  // But since it's single-cycle, we can add more cycles or stimuli if needed

  $display("All tests completed.");
  $stop;  // End simulation
end

endmodule
```

How to run?

Step 1: Setup the Verilog Files

Make sure all your Verilog files are correctly placed in a folder. For the files you've mentioned, this is the folder structure:

```
<project_folder>/
├── ALU.v
├── ControlUnit.v
├── InstructionMemory.v
├── ProgramCounter.v
├── RegisterFile.v
├── SimpleCPU.v
├── SimpleCPU_tb.v
└── DataMemory.v
```

Step 2: Create a New ModelSim Project

1. Open ModelSim.
2. Create a new project in ModelSim:
   - Go to `File>New> Project` .
   - Enter a project name and location (e.g., `SimpleComputer`).
   - Select the correct library (e.g., `work`).
3. Add your Verilog files to the project:
   - Go to `Project>Add to Project > Existing File.`
   - Select all the Verilog files (`ALU.vControlUnit.v, InstructionMemory.v ,ProgramCounter.v, RegisterFile.v, SimpleCPU.v,DataMemory.v`, and `SimpleCPU_tb.v`).

Step 3: Compile the Verilog Files

1. Inthe **Project Explorer** pane, right-click on the project name (e.g., `SimpleComputer`).
2. Choose **Compile > Compile All** to compile all the Verilog files.
   - ModelSim will check for syntax errors and compile each file.
   - If there are no errors, you should see a message like: `Compilation succeeded for <file name>.`

Step 4: Simulate the Testbench

1. Inthe **Project Explorer** pane, right-click on SimpleCPU_tb and select **Simulate > Simulate**.
   - Thiswillstartthesimulation using the SimpleCPU_tb.v file as the testbench.
2. Inthe **Transcript** window, you should see messages indicating that the simulation is running.

- If everything is set up correctly, the simulation will start and run until it encounters the $stop command (at 100ns in your testbench).

Step 5: View the Waveforms

1. After running the simulation, you can view the waveforms of the signals in the simulation:
   - In the **Transcript** window, type the following command to add signals to the
   - waveform viewer:
     ```
     add wave *
     ```

     This will add all signals to the waveform viewer.

2. You should now be able to see the behavior of the signals (`pc_address,` instruction, alu_result, etc.) over time.

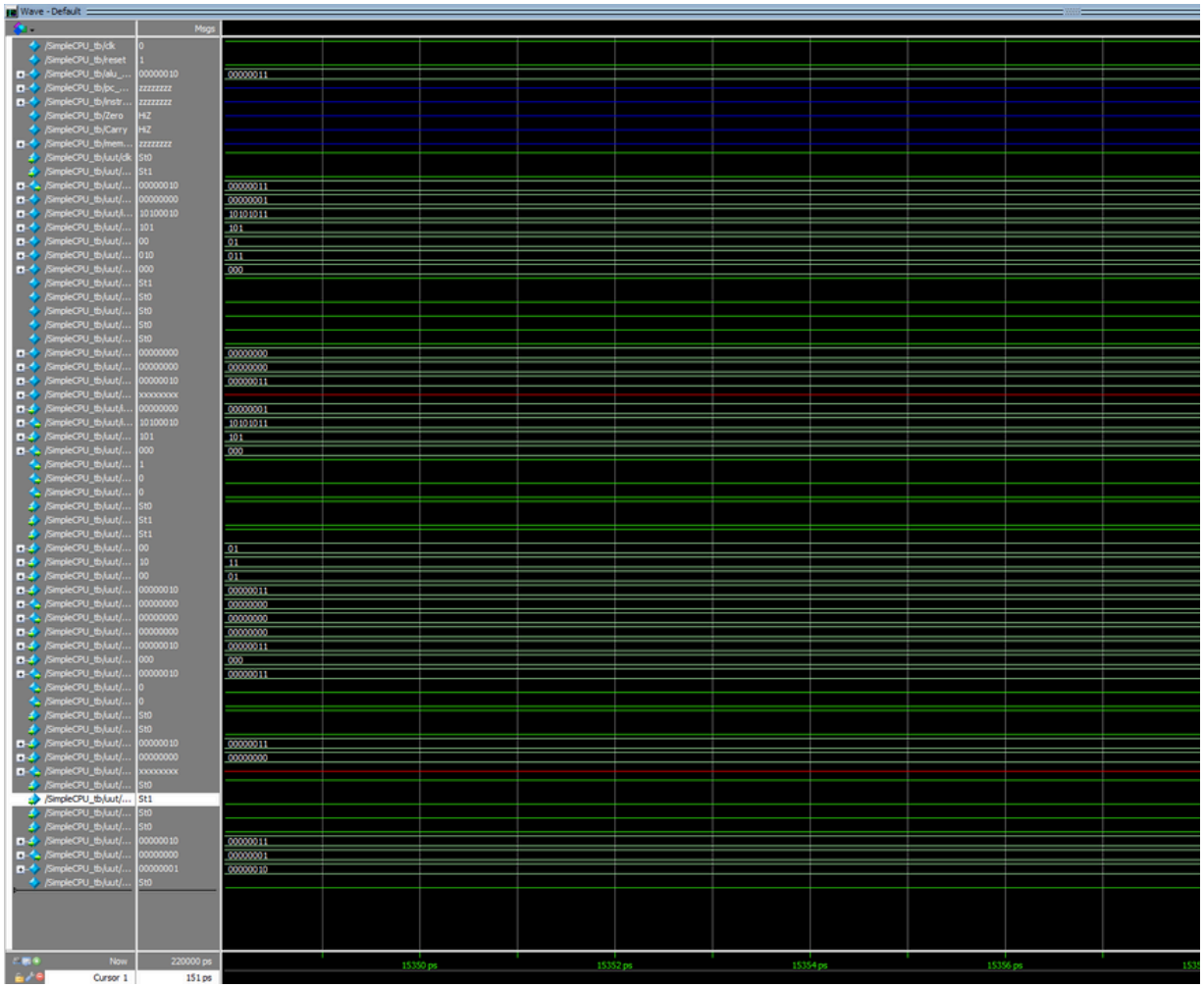
Step 6: Debugging and Adjusting the Simulation

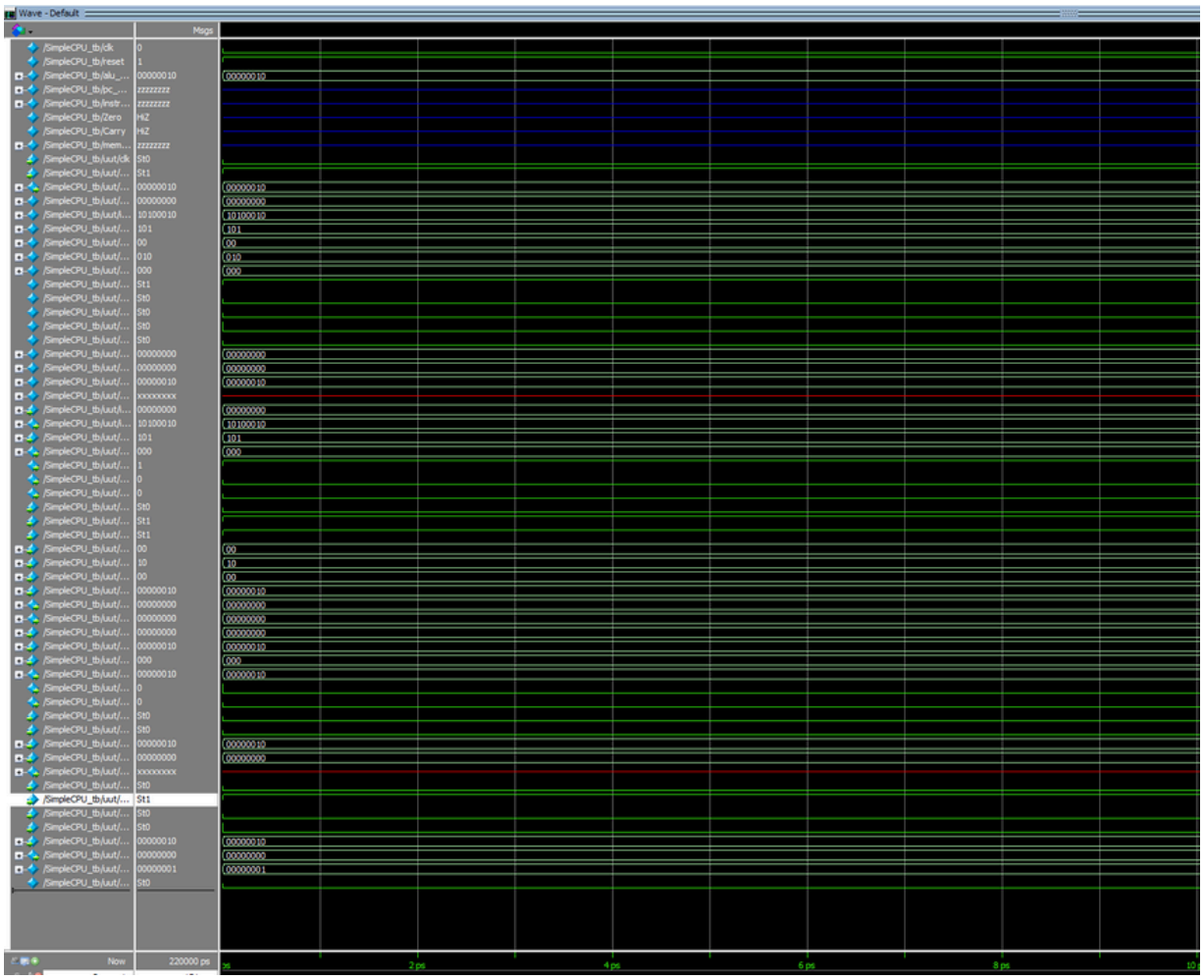
- If there are errors in your design, ModelSim will display them in the **Transcript** window.
- You can adjust your Verilog code based on the errors and rerun the simulation until everything works correctly.

Step 7: Clean Up and Close the Simulation

1. Once you're satisfied with the results, you can stop the simulation by typing $stop in the **Transcript** window.
2. If you want to clean up the workspace, use File > Close to close the project or File > Exit to close ModelSim.

Output:


Wave - Default
Msgs
/SimpleCPU_tb/clk
/SimpleCPU_tb/reset
/SimpleCPU_tb/alu_...
/SimpleCPU_tb/pc_...
/SimpleCPU_tb/instr...
/SimpleCPU_tb/Zero
/SimpleCPU_tb/Carry
/SimpleCPU_tb/mem...
/SimpleCPU_tb/uut/clk
/SimpleCPU_tb/uut/...
Now
220000 ps
Cursor 1
151 ps
15350 ps
15352 ps
15354 ps
15356 ps


Wave - Default
Msgs
/SimpleCPU_tb/clk
/SimpleCPU_tb/reset
/SimpleCPU_tb/alu_...
/SimpleCPU_tb/pc_...
/SimpleCPU_tb/instr...
/SimpleCPU_tb/Zero
/SimpleCPU_tb/Carry
/SimpleCPU_tb/mem...
/SimpleCPU_tb/uut/clk
/SimpleCPU_tb/uut/...
Now
220000 ps
2 ps
4 ps
6 ps
8 ps