Project Report: Design Your Own MIPS CPU and OS

1. Introduction

One of the course objectives is to design a complete CPU system. This starts with studying ISA, assembly language, ALU, datapath and control, and pipelining in theory. To make this learning effective, a step-by-step design of a complete CPU system was carried out using pen-and-paper, simulation tools, and simulated hardware.

The project involves designing a 32-bit single-cycle CPU with separate instruction and data memory, supported by an assembler, and tested using operating system–like programs. The CPU datapath and control unit were implemented in Verilog, and benchmark programs were used to validate execution.

2. ISA Study (Document + Program)

2.1 Overview of ISA Design

A 32-bit single-cycle CPU was designed with general-purpose instructions supporting arithmetic, logic, memory access, and control flow. Instruction formats include:

- **R-type**: Register operations (ADD, SUB, AND, OR, XOR, SLL, SRL) – implemented in `alu32.v` and `regfile32.v`.
- **I-type**: Immediate and memory operations (LW, SW, ADDI) – connected via `ram.v`, verified with `dmem.txt`.
- **J-type**: Control instructions (J, JAL, JR) – implemented in `control.v` and `datapath.v`.

The CPU has 8 general-purpose registers (R0–R7). R0 is fixed at 0; the others are used for arithmetic, logic, memory, and subroutine operations. Input/output operations are handled using `dmem.txt`, where LW and SW instructions simulate reading and writing data.

2.2 Input/Output Operations

Since external displays and keyboards cannot be connected, I/O is simulated via `dmem.txt`. Load (`LW`) and store (`SW`) instructions are used to read and write values, which represent program input and output. Results can be verified from the register file (`regfile32.v`) and memory content.

2.3 Design Requirements

- **Operands:** Register-based with memory access.

- **Instruction Set:** Arithmetic (ADD, SUB), Logical (AND, OR, XOR), Memory (LW, SW), Control (J, JAL, JR).
- **Registers:** 8 general-purpose (R0–R7).
- **Memory:** Data memory simulated in `dmem.txt`; instruction memory in `rom.v, memfile.txt`.
- **Files Used:** `alu32.v`, `regfile32.v`, `ram.v`, `control.v`, `datapath.v`, `insttest2.s`.

2.4 Benchmark Programs

Three types of programs were implemented:

1. **Arithmetic & Logic** – basic operations to test ALU.
2. **Conditional Checking** – branch instructions (BEQ) to validate decision making.
3. **Loop Programs & Subroutines** – `insttest2.s` modified to test JAL/JR and MIN, MAX, MEAN calculations.

2.5 Evaluation and Discussion

Benchmark programs executed correctly for arithmetic and logic operations. Subroutine calls using JAL/JR worked after modifying `datapath.v` and `control.v` to handle link registers and program counter updates. The AVG output remained 0, also sometimes 1 or 10 due to assembler branching issues, later identified and reported to faculty.

3. Assembler (Software)

The assembler translates `.s` assembly programs into machine code for instruction memory.

- **Language Used:** C++ (`finalassembler.cpp`)
- **Input Format:** Assembly text file (`.s`)
- **Output Format:** Binary (`no_address.bin`, `no_address.data`)

3.1 Design & Implementation

The assembler parses assembly instructions, converts them into machine code, and generates output files (`no_address.bin`, `no_address.data.bin`) compatible with Verilog RAM blocks.

Files Used: `insttest2.s`, `finalassembler.cpp`, `outp3.no_address.text.bin`, `outp3.no_address.data.bin`

3.2 Testing & Validation

Tested using `insttest2.s` to verify correct translation of JAL/JR and arithmetic instructions. Outputs were loaded into instruction memory (`rom.v`) for simulation in `MIPS_SCP.v`.

4. Operating System Based on MIPS Assembly

4.1 Program Design

- **Main Function:** Calls three subroutines (MAX, MIN, MEAN) using JAL/JR.
- **Subroutines:** Implemented using LI and SW instructions; divide subroutine provided for MEAN.
- **Input Data:** Ten integers loaded into registers and stored in `dmem.txt`.

4.2 Testing & Validation

- **Files Used:** `insttest2.s`
- **Execution:** Verified subroutine results in register file and `memfile.txt, dmem.txt`.
- **Observations:** MIN and MAX values computed correctly; MEAN remained 0, sometimes 1 or 10 due to assembler issue.

5. Full CPU (Verilog)

5.1 Datapath & Control Path Design

- **Components:** `alu32.v`, `regfile32.v`, `ram.v`, `control.v`, `datapath.v`
- **Connectivity:** Corrected for JAL/JR instructions and proper memory read/write.
- **Top-Level:** `MIPS_SCP.v` with testbench `MIPS_SCP_tb.v`.

5.2 Implementation Steps

- Write assembly code in `insttest2.s` with subroutine calls.
- Compile assembler (`finalassembler.cpp`) → generate `no_address.bin`, `outp3.no_address.text.bin`.
- Load binary into instruction memory (`rom.v`).
- Simulate in ModelSim (`MIPS_SCP_tb.v`) and verify execution.
- Check register file (`regfile32.v`) and data memory (`dmem.txt`) for results.

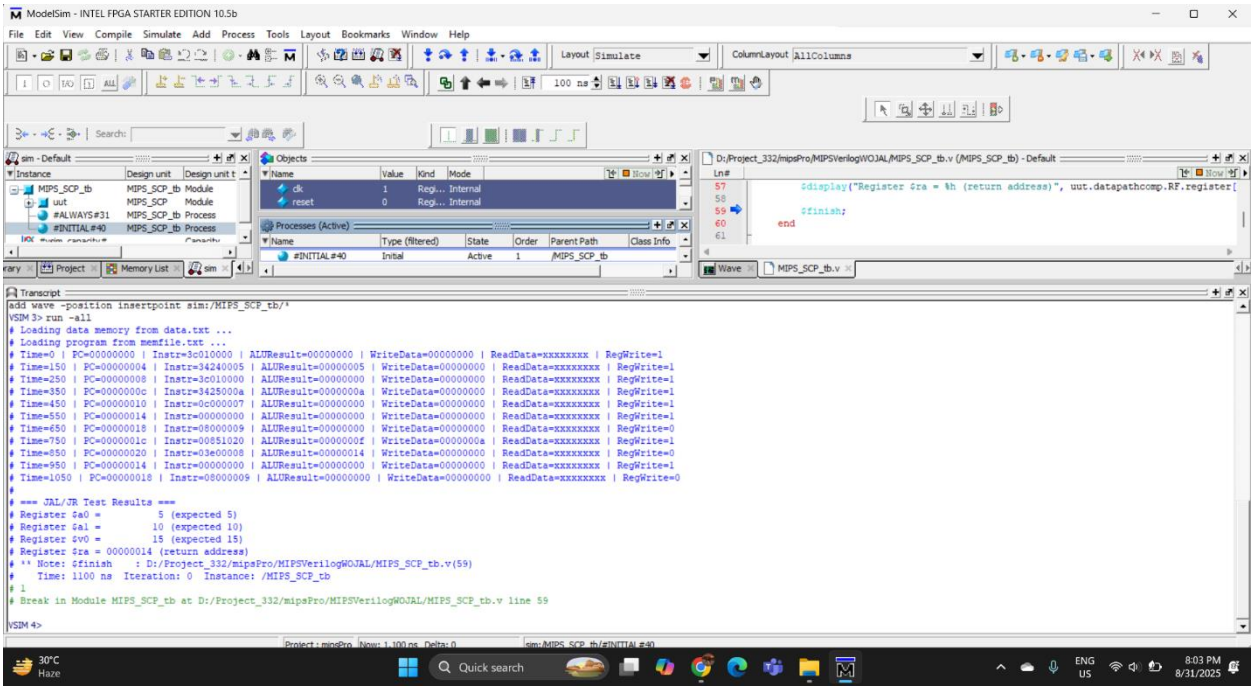
5.3 Challenges & Solutions

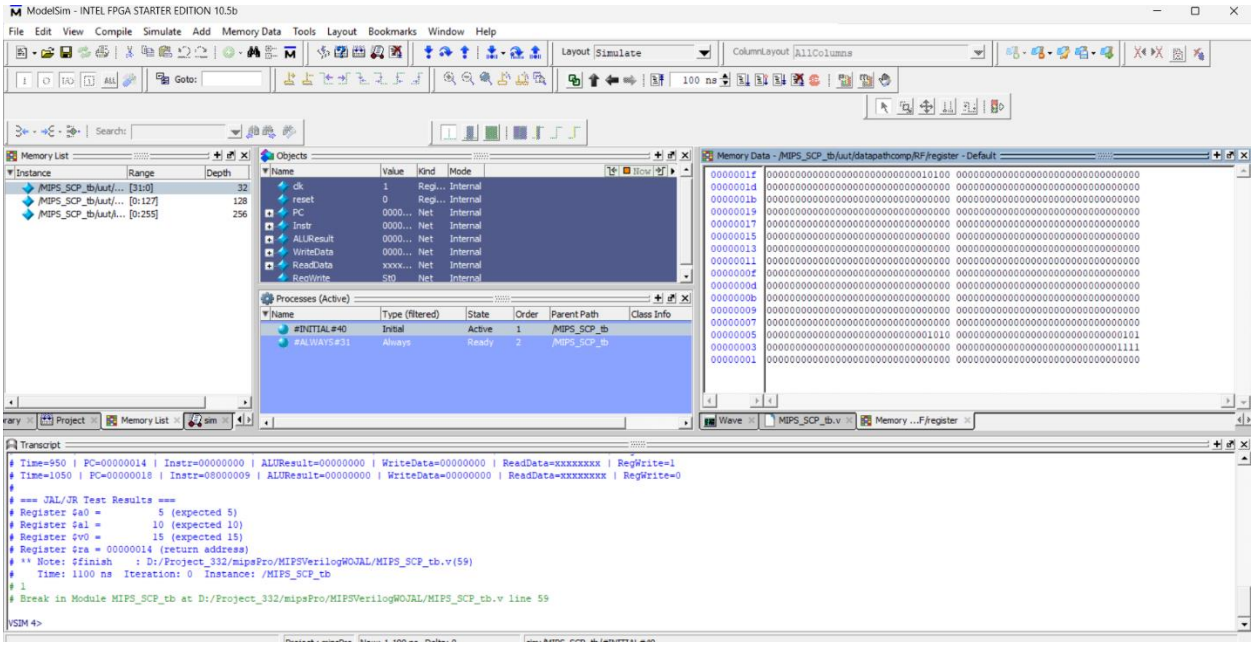
- **WSL/Windows Issues:** Resolved environment errors to run scripts.
- **ModelSim Installation Issues:** Reinstalled and configured correctly.
- **Compilation Issues:** Fixed signal connections in `datapath.v` and `ram.v`.
- **Benchmark Output Errors:** MIN/MAX correct; AVG 0/1/10 → traced to assembler branching.

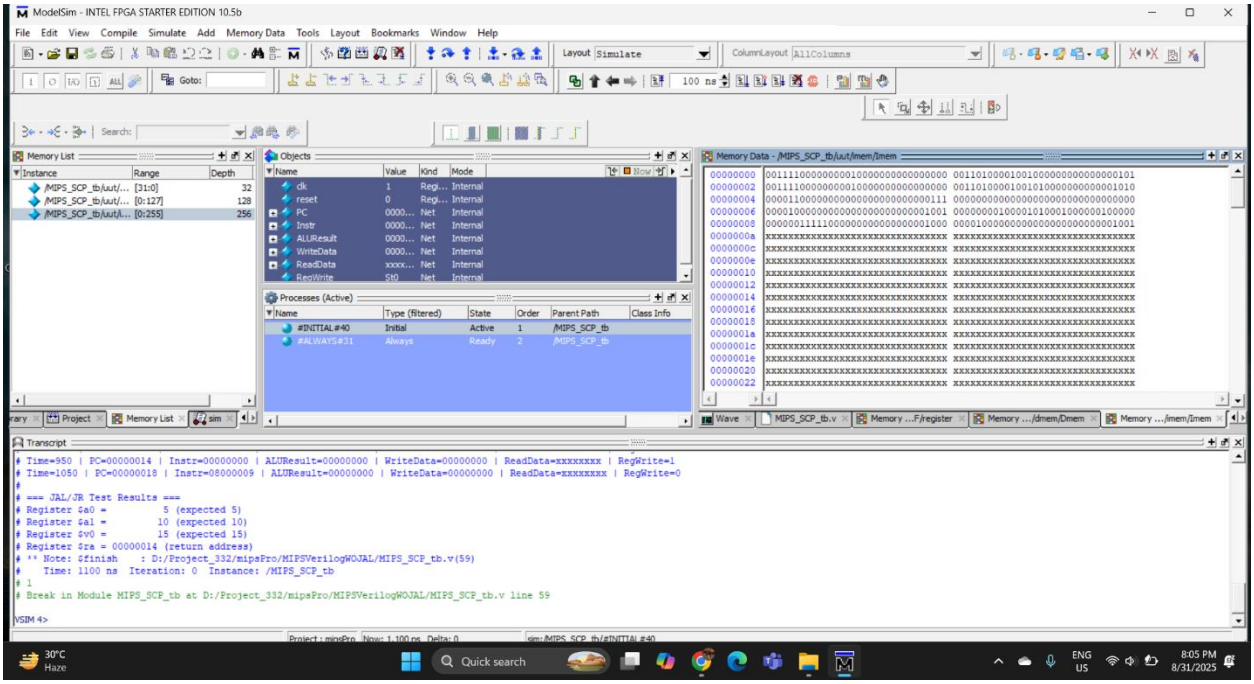
- **Debugging Steps:** Verified machine code, updated control path for JAL/JR, re-tested subroutine execution.

5.4 Testing & Validation

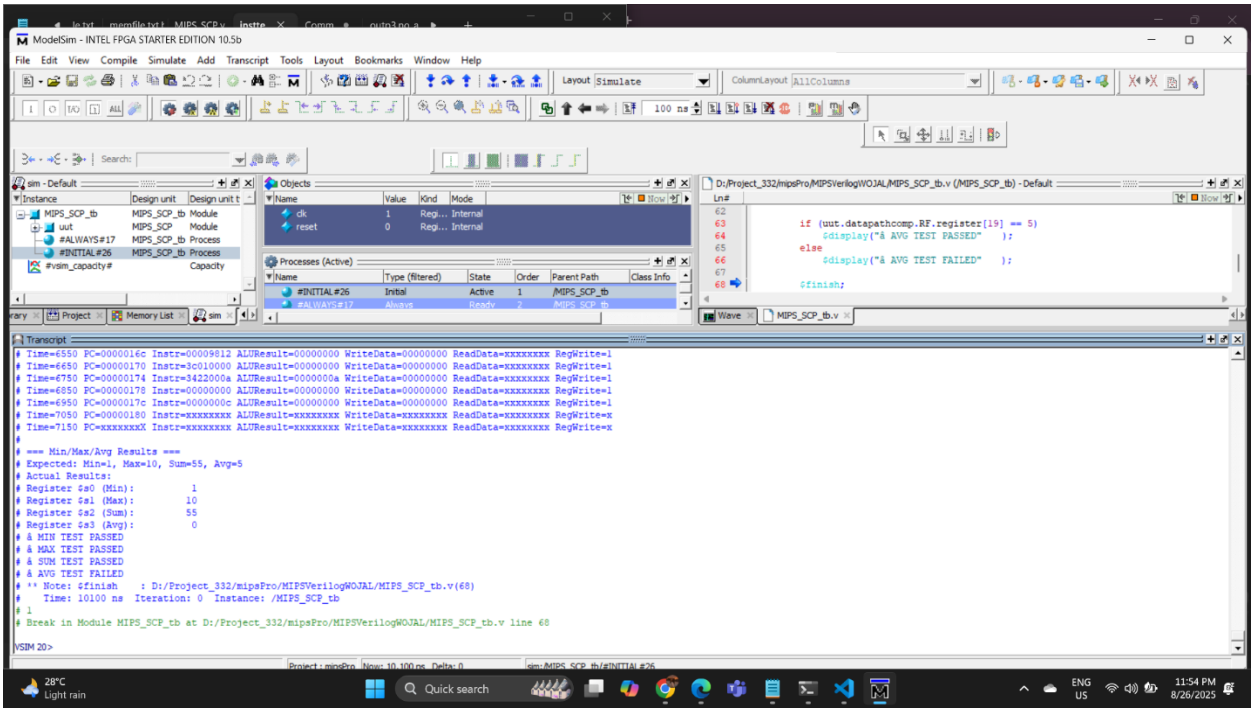
- **Verification:** Register file shows correct return addresses; data memory shows MIN and MAX outputs.
- **Simulation Outputs:** `mips_scp.vcd (For min max mean)` and `mips_scp.vcd (jal jr test)` verified instruction execution.
- **Screenshots:** Include data memory and register file content.
- **Files Submitted:** Verilog codes, assembly programs, simulation results, zipped folder.

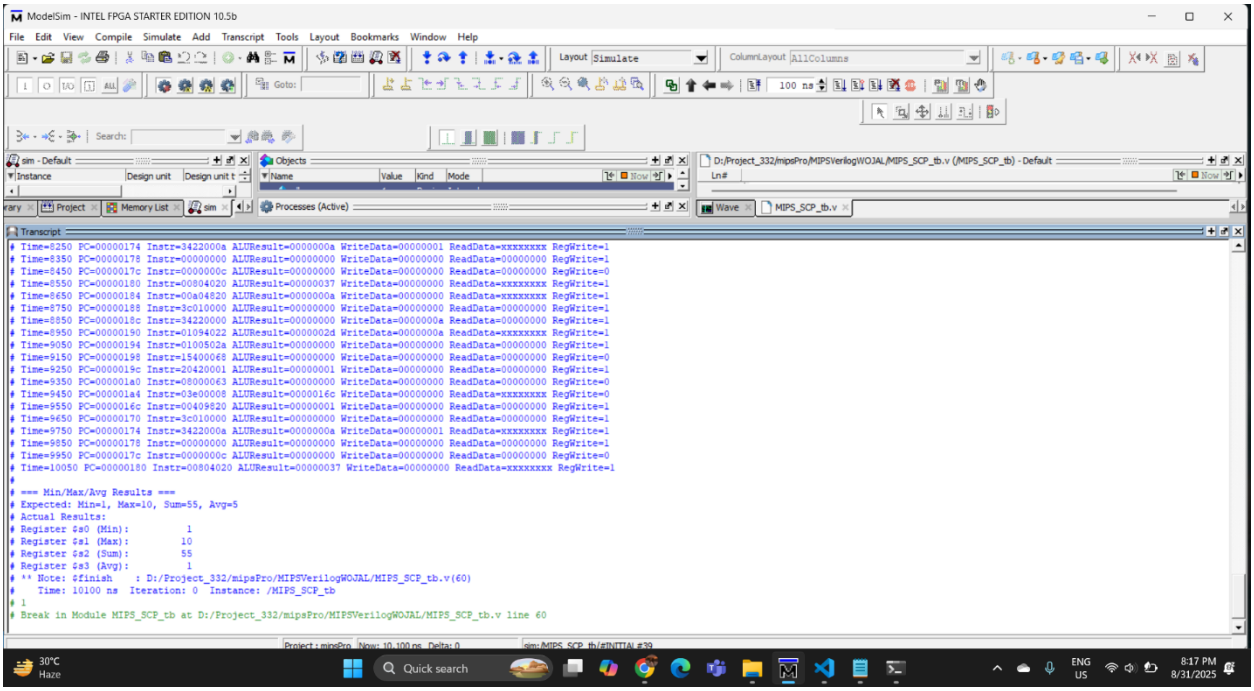


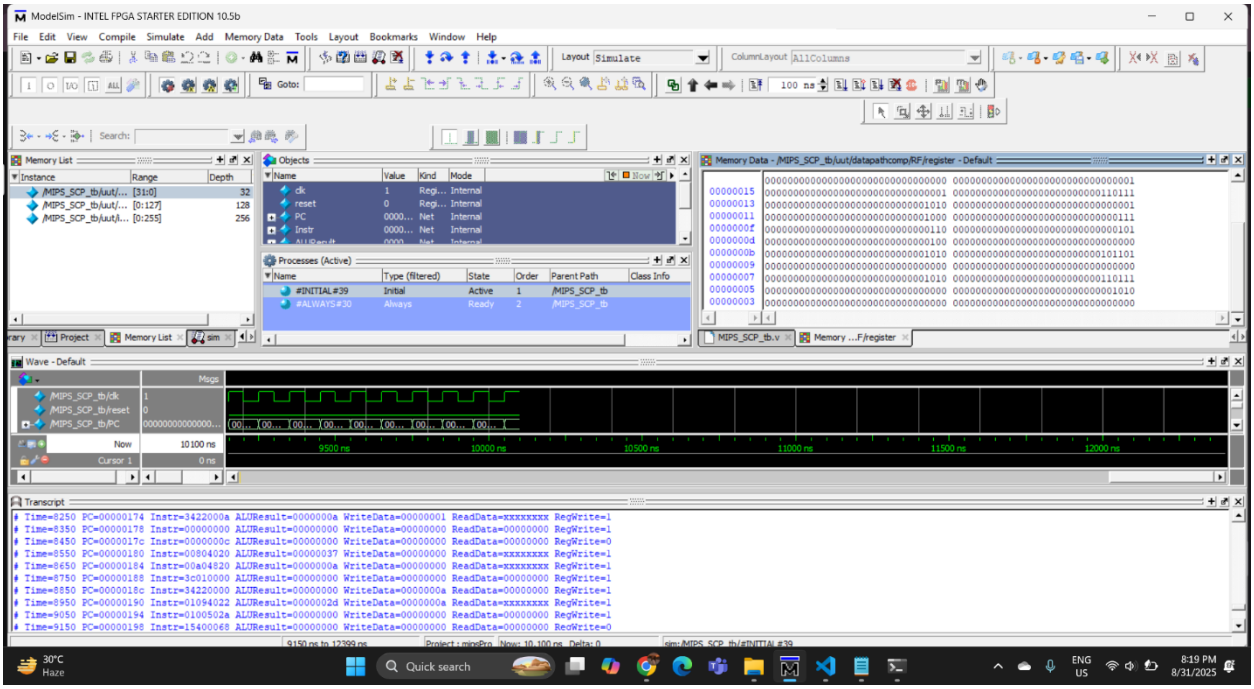


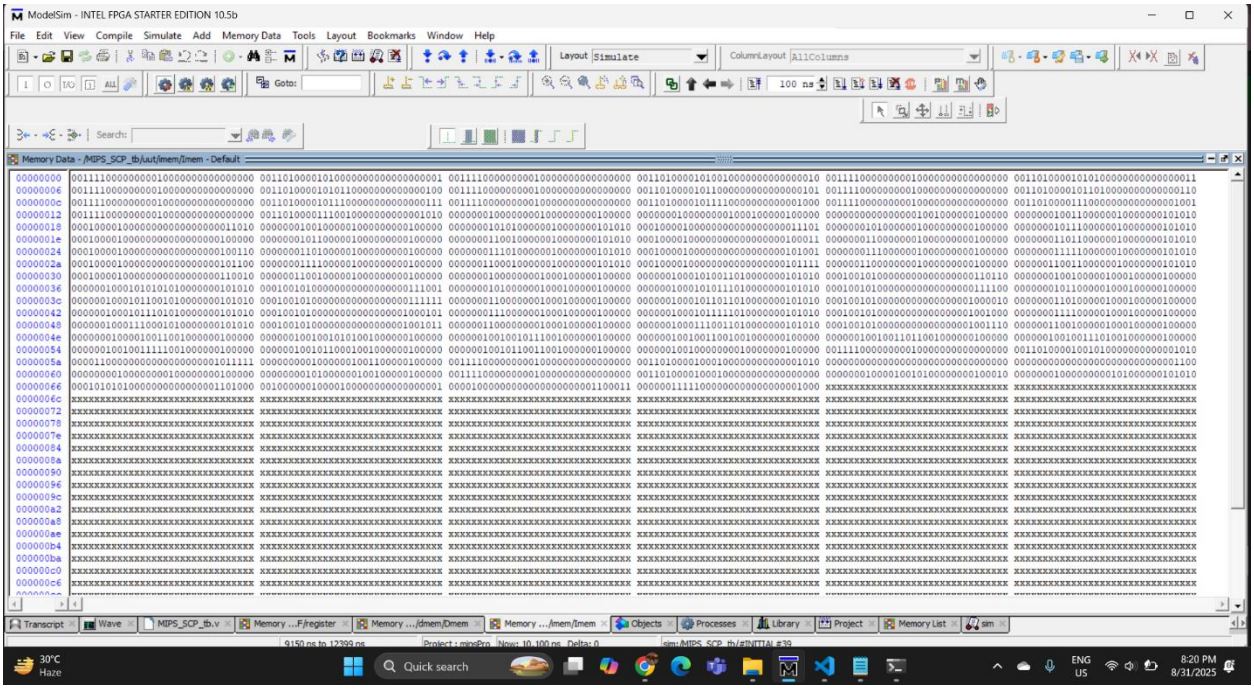


Jal Jr Work Testing With memory list


ModelSim - INTEL FPGA STARTER EDITION 10.5b
File Edit View Compile Simulate Add Transcript Tools Layout Bookmarks Window Help
Transcript
Time=6550 PC=0000016c Instr=00009812 ALUResult=00000000 WriteData=00000000 ReadData=xxxxxxxx RegWrite=1
Time=6650 PC=00000170 Instr=3c010000 ALUResult=00000000 WriteData=00000000 ReadData=xxxxxxxx RegWrite=1
Time=6750 PC=00000174 Instr=3422000a ALUResult=0000000a WriteData=00000000 ReadData=xxxxxxxx RegWrite=1
Time=6850 PC=00000178 Instr=00000000 ALUResult=00000000 WriteData=00000000 ReadData=xxxxxxxx RegWrite=1
Time=6950 PC=0000017c Instr=0000000c ALUResult=00000000 WriteData=00000000 ReadData=xxxxxxxx RegWrite=1
Time=7050 PC=00000180 Instr=xxxxxxxx ALUResult=xxxxxxxx WriteData=xxxxxxxx ReadData=xxxxxxxx RegWrite=x
Time=7150 PC=xxxxxxxX Instr=xxxxxxxx ALUResult=xxxxxxxx WriteData=xxxxxxxx ReadData=xxxxxxxx RegWrite=x
#
=== Min/Max/Avg Results ===
Expected: Min=1, Max=10, Sum=55, Avg=5
Actual Results:
Register $s0 (Min):   1
Register $s1 (Max):   10
Register $s2 (Sum):   55
Register $s3 (Avg):   0
â MIN TEST PASSED
â MAX TEST PASSED
â SUM TEST PASSED
â AVG TEST FAILED
** Note: $finish   : D:/Project_332/mipsPro/MIPSVerilogWOJAL/MIPS_SCP_tb.v(68)
Time: 10100 ns Iteration: 0 Instance: /MIPS_SCP_tb
1
Break in Module MIPS_SCP_tb at D:/Project_332/mipsPro/MIPSVerilogWOJAL/MIPS_SCP_tb.v line 68
VSIM 20>
28°C
Light rain
Quick search
ENG
US
11:54 PM
8/26/2025


ModelSim - INTEL FPGA STARTER EDITION 10.5b
File Edit View Compile Simulate Add Transcript Tools Layout Bookmarks Window Help
Transcript
Time=8250 PC=00000174 Instr=3422000a ALUResult=0000000a WriteData=00000001 ReadData=xxxxxxxx RegWrite=1
Time=8350 PC=00000178 Instr=00000000 ALUResult=00000000 WriteData=00000000 ReadData=00000000 RegWrite=1
Time=8450 PC=0000017c Instr=0000000c ALUResult=00000000 WriteData=00000000 ReadData=00000000 RegWrite=0
Time=8550 PC=00000180 Instr=00804020 ALUResult=00000037 WriteData=00000000 ReadData=xxxxxxxx RegWrite=1
Time=8650 PC=00000184 Instr=00a04820 ALUResult=0000000a WriteData=00000000 ReadData=xxxxxxxx RegWrite=1
Time=8750 PC=00000188 Instr=3c010000 ALUResult=00000000 WriteData=00000000 ReadData=00000000 RegWrite=1
Time=8850 PC=0000018c Instr=34220000 ALUResult=00000000 WriteData=0000000a ReadData=00000000 RegWrite=1
Time=8950 PC=00000190 Instr=01094022 ALUResult=0000002d WriteData=0000000a ReadData=xxxxxxxx RegWrite=1
Time=9050 PC=00000194 Instr=0100502a ALUResult=00000000 WriteData=00000000 ReadData=00000000 RegWrite=1
Time=9150 PC=00000198 Instr=15400068 ALUResult=00000000 WriteData=00000000 ReadData=00000000 RegWrite=0
Time=9250 PC=0000019c Instr=20420001 ALUResult=00000001 WriteData=00000000 ReadData=00000000 RegWrite=1
Time=9350 PC=000001a0 Instr=08000063 ALUResult=00000000 WriteData=00000000 ReadData=00000000 RegWrite=0
Time=9450 PC=000001a4 Instr=03e00008 ALUResult=0000016c WriteData=00000000 ReadData=xxxxxxxx RegWrite=0
Time=9550 PC=0000016c Instr=00409820 ALUResult=00000001 WriteData=00000000 ReadData=00000000 RegWrite=1
Time=9650 PC=00000170 Instr=3c010000 ALUResult=00000000 WriteData=00000000 ReadData=00000000 RegWrite=1
Time=9750 PC=00000174 Instr=3422000a ALUResult=0000000a WriteData=00000001 ReadData=xxxxxxxx RegWrite=1
Time=9850 PC=00000178 Instr=00000000 ALUResult=00000000 WriteData=00000000 ReadData=00000000 RegWrite=1
Time=9950 PC=0000017c Instr=0000000c ALUResult=00000000 WriteData=00000000 ReadData=00000000 RegWrite=0
Time=10050 PC=00000180 Instr=00804020 ALUResult=00000037 WriteData=00000000 ReadData=xxxxxxxx RegWrite=1
#
=== Min/Max/Avg Results ===
Expected: Min=1, Max=10, Sum=55, Avg=5
Actual Results:
Register $s0 (Min):   1
Register $s1 (Max):   10
Register $s2 (Sum):   55
Register $s3 (Avg):   1
** Note: $finish   : D:/Project_332/mipsPro/MIPSVerilogWOJAL/MIPS_SCP_tb.v(60)
Time: 10100 ns Iteration: 0 Instance: /MIPS_SCP_tb
1
Break in Module MIPS_SCP_tb at D:/Project_332/mipsPro/MIPSVerilogWOJAL/MIPS_SCP_tb.v line 60
30°C
Haze
Quick search
ENG
US
8:17 PM
8/31/2025





MIN, MAX, and MEAN were tested with memory listings, showing varying average values due to assembler issues.

6. Conclusion

The project successfully designed a 32-bit single-cycle CPU, developed an assembler, implemented subroutine programs (MIN, MAX, MEAN), and simulated the full CPU in Verilog. During the project, We faced and resolved challenges related to environment setup, assembler output, and datapath connectivity through systematic debugging. Through this process, We gained hands-on experience in ISA design, assembly programming, Verilog-based CPU implementation, and debugging complex hardware-software interactions. We would like to sincerely thank **Dr. Mohammad Abdul Qayum** for his guidance, support, and valuable teaching throughout the course, which greatly helped us complete this project.

7. References

- UpgradedMIPS32Assembler GitHub Repository : https://github.com/RoySRC/UpgradedMIPS32Assembler.git
- ModelSim Simulator
- ChatGPT and DeepSeek – used for debugging and assisting in code analysis.
- CSE332 Course Materials, North South University

Codes

```
//ram.v
`timescale 1ns/1ns

module ram(
   input clk,
   input we,
   input [31:0] adr,
   input [31:0] din,
   output [31:0] dout
);

   parameter depth = 128;
   parameter width = 32;

   reg [width-1:0] Dmem [0:depth-1];

   // word addressing (ignore adr[1:0])
   assign dout = Dmem[adr[31:2]];

   initial begin
      $display("Loading data memory from data.txt ...");
      $readmemb("dmem.txt", Dmem);
   end

   always @(posedge clk) begin
      if (we)
         Dmem[adr[31:2]] <= din;
   end

endmodule

//rom.v
`timescale 1ns/1ns

module rom(
   input  [31:0] adr,
   output [31:0] dout
);
   parameter depth = 256;
   parameter width = 32;
   reg [width-1:0] Imem [0:depth-1];

   initial begin
      $display("Loading program from memfile.txt ...");
      $readmemb("memfile.txt",    Imem);       // binary instructions
   end

   // Word addressing (ignore lower 2 bits)
   assign dout = Imem[adr[31:2]];

endmodule

// file: Datapath.v
`include "adder.v"
`include "alu32.v"
`include "flopr_param.v"
`include "mux2.v"
`include "mux4.v"
`include "regfile32.v"
`include "signext.v"
`include "sl2.v"

`timescale 1ns/1ns

module Datapath(input clk,
                input reset,
                input RegDst,
                input RegWrite,
                input ALUSrc,
                input Jump,
                input Jal,
                input Jr,
                input MemtoReg,
                input PCSrc,
                input [3:0] ALUControl,
                input [31:0] ReadData,
                input [31:0] Instr,
                output [31:0] PC,
                output ZeroFlag,
```

```
        output [31:0] datatwo,
        output [31:0] ALUResult);

wire [31:0] PCNext, PCplus4, PCbeforeBranch, PCBranch;
wire [31:0] extendedimm, extendedimmafter, MUXresult, dataone, aluop2;
wire [4:0] writereg;
wire [31:0] PCNextFinal;

// PC Logic
flopr_param #(32) PCregister(clk, reset, PCNextFinal, PC);
adder #(32) pcadd4(PC, 32'd4, PCplus4);
slt2 shifteradd2(extendedimm, extendedimmafter);
adder #(32) pcaddsigned(extendedimmafter, PCplus4, PCbeforeBranch);
mux2 #(32) branchmux(PCplus4, PCbeforeBranch, PCSrc, PCBranch);
mux2 #(32) jumpmux(PCBranch, {PCplus4[31:28], Instr[25:0], 2'b00}, Jump, PCNext);

// JR MUX - Select between normal next PC and register value for JR
mux2 #(32) jrmux(PCNext, dataone, Jr, PCNextFinal);

// Register File
// Register File
registerfile32 RF(
   .clk(clk),
   .we(RegWrite),
   .reset(reset),
   .ra1(Instr[25:21]),
   .ra2(Instr[20:16]),
   .wa(writereg),
   .wd(MUXresult),
   .rd1(dataone),
   .rd2(datatwo)
);

// Write register selection MUX
mux4 #(5) write_reg_mux(
   .d0(Instr[20:16]),  // rt (I-type)
   .d1(Instr[15:11]),  // rd (R-type)
   .d2(5'b11111),      // $ra (for JAL)
   .d3(5'b00000),      // unused
   .s({Jal, RegDst}),
   .y(writereg)
);

// Write data selection MUX
mux4 #(32) result_mux(
   .d0(ALUResult),     // ALU result
   .d1(ReadData),      // Memory read data
   .d2(PCplus4),       // Return address (for JAL)
   .d3(32'h00000000),  // unused
   .s({Jal, MemtoReg}),
   .y(MUXresult)
);

// ALU
alu32 alucomp(
   .a(dataone),
   .b(aluop2),
   .f(ALUControl),      // match "f"
   .shamt(Instr[10:6]),
   .y(ALUResult),       // match "y"
   .zero(ZeroFlag)
);

signext immextention(Instr[15:0], extendedimm);
mux2 #(32) aluop2sel(datatwo, extendedimm, ALUSrc, aluop2);

endmodule
```

```
// Control Unit with JAL and JR support
`timescale 1ns/1ns

module ControlUnit(
    input [5:0] Opcode,
    input [5:0] Func,
    input Zero,
    output reg MemtoReg,
    output reg MemWrite,
    output reg ALUSrc,
    output reg RegDst,
    output reg RegWrite,
    output reg Jump,
    output reg Jal,
    output reg Jr,
    output PCSrc,
    output reg [3:0] ALUControl
);

reg [9:0] temp;
reg Branch, BNE;

always @(*) begin
    // Default values
    temp = 10'b0;
    ALUControl = 4'b0;
    Jal = 1'b0;
    Jr = 1'b0;
    Branch = 1'b0;
    BNE = 1'b0;

    case (Opcode)
        6'b000000: begin                    // R-type
            case (Func)
                6'b100000: begin            // ADD
                    temp = 10'b1100000000;
                    ALUControl = 4'b0000;
                end
                6'b100001: begin            // ADDU
                    temp = 10'b1100000000;
                    ALUControl = 4'b0000;
                end
                6'b100010: begin            // SUB
                    temp = 10'b1100000000;
                    ALUControl = 4'b0001;
                end
                6'b100011: begin            // SUBU
                    temp = 10'b1100000000;
                    ALUControl = 4'b0001;
                end
                6'b100100: begin            // AND
                    temp = 10'b1100000000;
                    ALUControl = 4'b0010;
                end
                6'b100101: begin            // OR
                    temp = 10'b1100000000;
                    ALUControl = 4'b0011;
                end
                6'b100110: begin            // XOR
                    temp = 10'b1100000000;
                    ALUControl = 4'b0100;
                end
                6'b100111: begin            // NOR
                    temp = 10'b1100000000;
                    ALUControl = 4'b1010;
                end
                6'b101010: begin            // SLT
                    temp = 10'b1100000000;
                    ALUControl = 4'b1000;
                end
                6'b101011: begin            // SLTU
                    temp = 10'b1100000000;
                    ALUControl = 4'b1001;
                end
                6'b000000: begin            // SLL
                    temp = 10'b1100000000;
                    ALUControl = 4'b0101;
                end
                6'b000010: begin            // SRL
                    temp = 10'b1100000000;
                    ALUControl = 4'b0110;
                end
                6'b000011: begin            // SRA
                    temp = 10'b1100000000;
```

```
        ALUControl = 4'b0111;
      end
      6'b000100: begin          // SLLV
        temp = 10'b1100000000;
        ALUControl = 4'b1011;
      end
      6'b000110: begin          // SRLV
        temp = 10'b1100000000;
        ALUControl = 4'b1100;
      end
      6'b000111: begin          // SRAV
        temp = 10'b1100000000;
        ALUControl = 4'b1101;
      end
      6'b001000: begin          // JR
        temp = 10'b0000000001;
        ALUControl = 4'b0000;
        Jr = 1'b1;
      end
      default: begin
        temp = 10'b0000000000;
        ALUControl = 4'b0000;
      end
    endcase
end

6'b100011: begin                // LW
    temp = 10'b1010010000;
    ALUControl = 4'b0000;
end

6'b101011: begin                // SW
    temp = 10'b0010100000;
    ALUControl = 4'b0000;
end

6'b000100: begin                // BEQ
    temp = 10'b0001000000;
    ALUControl = 4'b0001;
    Branch = 1'b1;
end

6'b000101: begin                // BNE
    temp = 10'b0001000000;
    ALUControl = 4'b0001;
    Branch = 1'b1;
    BNE = 1'b1;
end

6'b001000: begin                // ADDI
    temp = 10'b1010000000;
    ALUControl = 4'b0000;
end

6'b001001: begin                // ADDIU
    temp = 10'b1010000000;
    ALUControl = 4'b0000;
end

6'b001100: begin                // ANDI
    temp = 10'b1010000000;
    ALUControl = 4'b0010;
end

6'b001101: begin                // ORI
    temp = 10'b1010000000;
    ALUControl = 4'b0011;
end

6'b001110: begin                // XORI
    temp = 10'b1010000000;
    ALUControl = 4'b0100;
end

6'b001010: begin                // SLTI
    temp = 10'b1010000000;
    ALUControl = 4'b1000;
end

6'b001011: begin                // SLTIU
    temp = 10'b1010000000;
    ALUControl = 4'b1001;
end
```

```
    6'b000010: begin                    // J
        temp = 10'b0000001000;
        ALUControl = 4'b0000;
        Jump = 1'b1;
    end

    6'b000011: begin                    // JAL
        temp = 10'b1000001000;
        ALUControl = 4'b0000;
        Jump = 1'b1;
        Jal = 1'b1;
    end

    6'b001111: begin                    // LUI
        temp = 10'b1010000000;
        ALUControl = 4'b1110;
    end

    default: begin                      // NOP
        temp = 10'b0000000000;
        ALUControl = 4'b0000;
    end
  endcase

  // Use blocking assignment for the concatenation
  {RegWrite, RegDst, ALUSrc, MemWrite, MemtoReg, Jump} = temp[9:4];
end

assign PCSrc = Branch & (Zero ^ BNE);

endmodule

//MIPS_SCP.v
`timescale 1ns/1ns

module MIPS_SCP(
  input clk,
  input reset,
  output [31:0] PC,
  output [31:0] Instr,
  output [31:0] ALUResult,
  output [31:0] WriteData,
  output [31:0] ReadData,
  output RegWrite
);

  wire RegDst, ALUSrc, Jump, Jal, Jr, MemtoReg, PCSrc, Zero, MemWrite;
  wire [3:0] ALUControl;

  Datapath datapathcomp(
    .clk(clk),
    .reset(reset),
    .RegDst(RegDst),
    .RegWrite(RegWrite),
    .ALUSrc(ALUSrc),
    .Jump(Jump),
    .Jal(Jal),
    .Jr(Jr),
    .MemtoReg(MemtoReg),
    .PCSrc(PCSrc),
    .ALUControl(ALUControl),
    .ReadData(ReadData),
    .Instr(Instr),
    .PC(PC),
    .ZeroFlag(Zero),
    .datatwo(WriteData),
    .ALUResult(ALUResult)
  );

  ControlUnit controller(
    .Opcode(Instr[31:26]),
    .Func(Instr[5:0]),
    .Zero(Zero),
    .MemtoReg(MemtoReg),
    .MemWrite(MemWrite),
    .ALUSrc(ALUSrc),
    .RegDst(RegDst),
    .RegWrite(RegWrite),
    .Jump(Jump),
    .Jal(Jal),
    .Jr(Jr),
```

```
    .PCSrc(PCSrc),
    .ALUControl(ALUControl)
  );

  ram dmem(
    .clk(clk),
    .we(MemWrite),
    .adr(ALUResult),
    .din(WriteData),
    .dout(ReadData)
  );

  rom imem(
    .adr(PC),
    .dout(Instr)
  );

endmodule

//MIPS_SCP_tb.v for jaljrtest
`timescale 1ns/1ns

module MIPS_SCP_tb;

  // Inputs
  reg clk;
  reg reset;

  // Outputs from UUT
  wire [31:0] PC;
  wire [31:0] Instr;
  wire [31:0] ALUResult;
  wire [31:0] WriteData;
  wire [31:0] ReadData;
  wire RegWrite;

  // Instantiate Unit Under Test (UUT)
  MIPS_SCP uut (
    .clk(clk),
    .reset(reset),
    .PC(PC),
    .Instr(Instr),
    .ALUResult(ALUResult),
    .WriteData(WriteData),
    .ReadData(ReadData),
    .RegWrite(RegWrite)
  );

  // Clock generation
  initial clk = 0;
  always #50 clk = ~clk; // 100ns period

  // Monitor to display important signals
  initial begin
    $monitor("Time=%0t | PC=%h | Instr=%h |
ALUResult=%h | WriteData=%h | ReadData=%h
| RegWrite=%b",
          $time, PC, Instr, ALUResult, WriteData,
ReadData, RegWrite);
  end

  // Test sequence
  initial begin
    // Initialize
    clk = 0;
    reset = 1;

    // Apply reset
    #100;
    reset = 0;

    // Run simulation long enough for JAL/JR
execution
    #1000;

    // Check values in registers after function
call
    $display("\n=== JAL/JR Test Results ===");
    $display("Register $a0 = %d (expected 5)",
uut.datapathcomp.RF.register[4]);
    $display("Register $a1 = %d (expected 10)",
uut.datapathcomp.RF.register[5]);
    $display("Register $v0 = %d (expected 15)",
uut.datapathcomp.RF.register[2]);
```

```
    $display("Register $ra = %h (return address)", uut.datapathcomp.RF.register[31]);

    $finish;
  end

  // VCD file for waveform viewing
  initial begin
    $dumpfile("mips_scp_jal.vcd");
    $dumpvars(0, MIPS_SCP_tb);
  end

endmodule

//MIPS_SCP_tb.v for min max mean
`timescale 1ns/1ns

module MIPS_SCP_tb;

  // Inputs
  reg clk;
  reg reset;

  // Outputs from uut
  wire [31:0] PC;
  wire [31:0] Instr;
  wire [31:0] ALUResult;
  wire [31:0] WriteData;
  wire [31:0] ReadData;
  wire RegWrite;

  // Instantiation of Unit Under Test
  MIPS_SCP uut (
    .clk(clk),
    .reset(reset),
    .PC(PC),
    .Instr(Instr),
    .ALUResult(ALUResult),
    .WriteData(WriteData),
    .ReadData(ReadData),
    .RegWrite(RegWrite)
  );

  // Clock generation
  always #50 clk = ~clk;

  // Monitor to display important signals
  initial begin
    $monitor("Time=%0t PC=%h Instr=%h ALUResult=%h WriteData=%h ReadData=%h RegWrite=%b",
             $time, PC, Instr, ALUResult, WriteData, ReadData, RegWrite);
  end

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

    // Apply reset
    #100;
    reset = 0;

    // Run for enough cycles to execute the program
    #10000;

    // Display final results
    $display("\n=== Min/Max/Avg Results ===");
    $display("Expected: Min=1, Max=10, Sum=55, Avg=5");
    $display("Actual Results:");
    $display("Register $s0 (Min): %d", uut.datapathcomp.RF.register[16]); // $s0 = reg16
    $display("Register $s1 (Max): %d", uut.datapathcomp.RF.register[17]); // $s1 = reg17
    $display("Register $s2 (Sum): %d", uut.datapathcomp.RF.register[18]); // $s2 = reg18
```

```
        $display("Register $s3 (Avg): %d", uut.datapathcomp.RF.register[19]); // $s3 = reg19

        $finish;
    end
    // VCD file generation for waveform viewing
    initial begin
        $dumpfile("mips_scp.vcd");
        $dumpvars(0, MIPS_SCP_tb);
    end

endmodule
```

```
//insttest2.s (assembly code for min max mean)
.data
newline: .asciiz "\n"

.text
main:
    # Load numbers into $t0-$t9
    li $t0, 1
    li $t1, 2
    li $t2, 3
    li $t3, 4
    li $t4, 5
    li $t5, 6
    li $t6, 7
    li $t7, 8
    li $t8, 9
    li $t9, 10

    # ---------------- Min ----------------
    move $s0, $t0       # min = first value
    move $s1, $t0       # max = first value
    move $s2, $zero     # sum = 0

    slt $a0, $t1, $s0
    beq $a0, $zero, check2
    move $s0, $t1
check2:
    slt $a0, $t2, $s0
    beq $a0, $zero, check3
    move $s0, $t2
check3:
    slt $a0, $t3, $s0
    beq $a0, $zero, check4
    move $s0, $t3
check4:
    slt $a0, $t4, $s0
    beq $a0, $zero, check5
    move $s0, $t4
check5:
    slt $a0, $t5, $s0
    beq $a0, $zero, check6
    move $s0, $t5
check6:
    slt $a0, $t6, $s0
    beq $a0, $zero, check7
    move $s0, $t6
check7:
    slt $a0, $t7, $s0
    beq $a0, $zero, check8
    move $s0, $t7
check8:
    slt $a0, $t8, $s0
    beq $a0, $zero, check9
    move $s0, $t8
check9:
    slt $a0, $t9, $s0
    beq $a0, $zero, max_calc
    move $s0, $t9

# ---------------- Max ----------------
max_calc:
    move $s1, $t0       # max = first value
    slt  $s4, $s1, $t1
    beq  $s4, $zero, max2
    move $s1, $t1
max2:
    slt  $s4, $s1, $t2
    beq  $s4, $zero, max3
    move $s1, $t2
max3:
    slt  $s4, $s1, $t3
```

```
   beq  $s4, $zero, max4
   move $s1, $t3
max4:
   slt  $s4, $s1, $t4
   beq  $s4, $zero, max5
   move $s1, $t4
max5:
   slt  $s4, $s1, $t5
   beq  $s4, $zero, max6
   move $s1, $t5
max6:
   slt  $s4, $s1, $t6
   beq  $s4, $zero, max7
   move $s1, $t6
max7:
   slt  $s4, $s1, $t7
   beq  $s4, $zero, max8
   move $s1, $t7
max8:
   slt  $s4, $s1, $t8
   beq  $s4, $zero, max9
   move $s1, $t8
max9:
   slt  $s4, $s1, $t9
   beq  $s4, $zero, sum_calc
   move $s1, $t9

# ---------------- Sum ----------------
sum_calc:
   add  $s2, $t0, $t1
   add  $s2, $s2, $t2
   add  $s2, $s2, $t3
   add  $s2, $s2, $t4
   add  $s2, $s2, $t5
   add  $s2, $s2, $t6
   add  $s2, $s2, $t7
   add  $s2, $s2, $t8
   add  $s2, $s2, $t9

move $a0, $s2    # dividend = sum (55)
li $a1, 10       # divisor = 10
jal division_sub
nop
move $s3, $v0
li $v0, 10
nop


division_sub:
   move $v0, $zero    # quotient = 0
   move $v1, $a0      # remainder = dividend
division_loop:
   slt $t1, $v1, $a1      # use $t1 instead of $t0
   bne $t1, $zero, division_done
   sub $v1, $v1, $a1
   addiu $v0, $v0, 1
   j division_loop
division_done:
   jr $ra
   nop
```

//insttest2.s (assembly code for Jal Jr test)

```
.text
main:
   li $a0, 5       # Load 5 into $a0
   li $a1, 10      # Load 10 into $a1
   jal add_numbers  # Call add_numbers
   nop
   j end          # Infinite loop

# Function to add $a0 + $a1 and store in $v0
add_numbers:
   add $v0, $a0, $a1
   jr $ra         # Return to main

end:
   j end          # Infinite loop
```