UNIVERSITY OF CALIFORNIA IRVINE

EECS 31L: Introduction to Digital Logic Laboratory (Spring 2024)

Lab 5: Single-Cycle RISC-V Processor

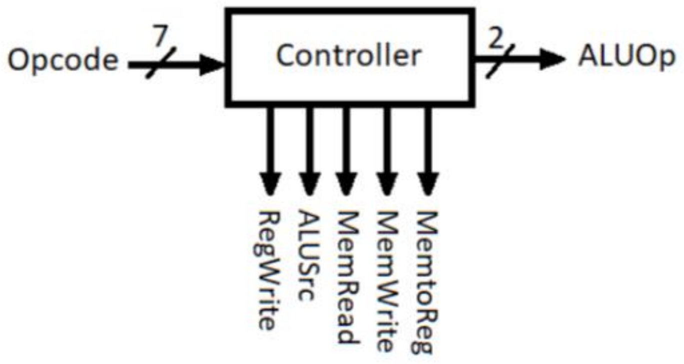
ADITYA NANDKISHOR SAWANT (sawantan@uci.edu)

Controller

1. Overview

The given design is a module for a basic control unit in a CPU. The control unit generates control signals based on the opcode of an instruction. These control signals are used to direct the operation of other parts of the CPU, such as the ALU, memory, and registers.

Block Diagram:



Inputs:

Opcode [6:0]: The opcode of the current instruction, which specifies the operation to be performed.

Outputs:

ALUOp [1:0]: Specifies the operation to be performed by the ALU.

MemtoReg: Indicates if the data should be read from memory to the register.

MemWrite: Indicates if a memory write operation should be performed.

MemRead: Indicates if a memory read operation should be performed.

ALUSrc: Specifies if the second ALU operand is from the instruction or a register.

RegWrite: Indicates if a register write operation should be performed.

2. Hardware Design

The design uses a series of combinational logic assignments to generate the control signals based on the opcode input.

Truth Table:

Opcode	ALUOp	MemtoReg	MemWrite	MemRead	ALUSrc	RegWrite
0000011	01	1	0	1	1	1
0100011	01	0	1	0	1	0
0010011	00	0	0	0	1	1
0110011	10	0	0	0	0	1
default	00	0	0	0	0	0

Verilog Code:

```
module Controller(
   input [6:0] Opcode,
   output [1:0] ALUOp,
   output MemtoReg,
   output MemWrite,
   output RegWrite,
    output ALUSrc,
    output MemRead
);


// Control Signals
assign MemtoReg = (Opcode == 7'b0000011) ? 1'b1 : 1'b0;
assign MemWrite = (Opcode == 7'b0100011) ? 1'b1 : 1'b0;
assign MemRead = (Opcode == 7'b0000011) ? 1'b1 : 1'b0;
assign ALUSrc = ((Opcode == 7'b0010011) || (Opcode == 7'b0000011) || (Opcode == 7'b0100011)) ? 1'b1 :
1'b0;
assign RegWrite = ((Opcode == 7'b0110011) || (Opcode == 7'b0010011) || (Opcode == 7'b0000011)) ? 1'b1 :
1'b0;
assign ALUOp = (Opcode == 7'b0110011) ? 2'b10 :
            (Opcode == 7'b0000011 || Opcode == 7'b0100011) ? 2'b01 : 2'b00;


endmodule
```

Explanation:

Opcode Input:

This is a 7-bit input representing the opcode of the instruction. The opcode determines which operation the control unit should signal.

Output Signals:

MemtoReg: Set to 1 when the opcode is for load instructions (0000011), indicating that data should be read from memory to the register.

MemWrite: Set to 1 when the opcode is for store instructions (0100011), indicating that a memory write operation should be performed.

MemRead: Set to 1 when the opcode is for load instructions (0000011), indicating that a memory read operation should be performed.

ALUSrc: Set to 1 for immediate-type, load, and store instructions (0010011, 0000011, 0100011), indicating that the second ALU operand comes from the instruction.

RegWrite: Set to 1 for register-type, immediate-type, and load instructions (0110011, 0010011, 0000011), indicating that a register write operation should be performed.

ALUOp: Encodes the type of ALU operation. It is 10 for register-type instructions (0110011), 01 for load/store instructions (0000011, 0100011), and 00 for others.

By using these control signals, the CPU can correctly execute different types of instructions by directing the appropriate hardware components.

Lab 5: Single-Cycle RISC-V Processor

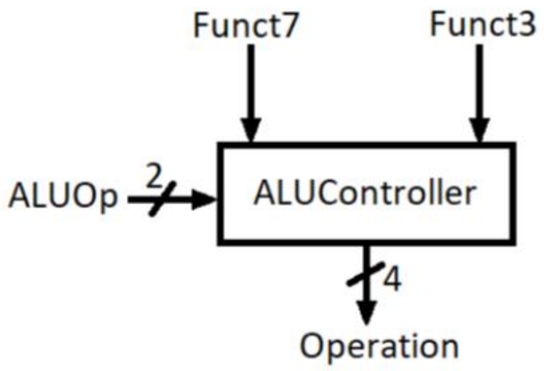
ADITYA NANDKISHOR SAWANT (sawantan@uci.edu)

ALUController

1. Overview

The provided design is a module for an ALU (Arithmetic Logic Unit) controller, which generates specific control signals to direct the ALU's operations. These signals are based on the ALUOp, Funct7, and Funct3 fields of the instruction, typically used in RISC-V or similar instruction set architectures.

Block Diagram:



Inputs:

ALUOp [1:0]: ALU operation control bits provided by the main control unit.

Funct7 [6:0]: 7-bit function field from the instruction, providing specific operation details.

Funct3 [2:0]: 3-bit function field from the instruction, providing additional operation details.

Outputs:

Operation [3:0]: 4-bit control signal to the ALU to specify the exact operation to perform.

Lab 5: Single-Cycle RISC-V Processor

ADITYA NANDKISHOR SAWANT (sawantan@uci.edu)

2. Hardware Design

The design uses a series of combinational logic assignments to generate the control signals based on the ALUOp, Funct7, and Funct3 inputs.

Truth Table:

Funct7	Funct3	ALUOp	Operation [3:0]
0000000	110	10	0001
0000000	010	10	0111

Verilog Code:

```
module ALUController(
    input [2:0] Funct3,
    input [6:0] Funct7,
    input [1:0] ALUOp,
    output [3:0] Operation
);

// Behavior
assign Operation[0] = ((Funct3 == 3'b110) || ((Funct3 == 3'b010) && (ALUOp[0] == 1'b0))) ? 1'b1 : 1'b0;
assign Operation[1] = ((Funct3 == 3'b010) || (Funct3 == 3'b000)) ? 1'b1 : 1'b0;
assign Operation[2] = ((Funct3 == 3'b100) || (Funct3 == 3'b010) && (ALUOp[0] == 1'b0)) ||
                ((Funct7 == 7'b0100000) && (Funct3 == 3'b000) && (ALUOp == 2'b10)) ? 1'b1 : 1'b0;
assign Operation[3] = (Funct3 == 3'b100) ? 1'b1 : 1'b0;

endmodule
```

Inputs:

ALUOp [1:0]: These bits specify the overall operation type (e.g., R-type, I-type).

Funct7 [6:0]: This field provides additional information for the operation, typically used to differentiate between similar operations (e.g., addition vs. subtraction).

Funct3 [2:0]: This field provides specific information about the operation to be performed.

Outputs:

Operation [3:0]: These bits specify the exact operation to be performed by the ALU. Each bit is calculated based on the input fields using combinational logic.

Operation Signal Logic:

Operation[0]: Set to 1 if Funct3 is 110 or if Funct3 is 010 and ALUOp[0] is 0.

Operation[1]: Set to 1 if Funct3 is 010 or 000.

Operation[2]: Set to 1 if Funct3 is 100, or if Funct3 is 010 and ALUOp[0] is 0, or if Funct7 is 0100000, Funct3 is 000, and ALUOp is 10.

Operation[3]: Set to 1 if Funct3 is 100.

By using these control signals, the ALU can perform the correct operation as specified by the instruction's function fields and the overall operation type provided by the control unit.

Lab 5: Single-Cycle RISC-V Processor

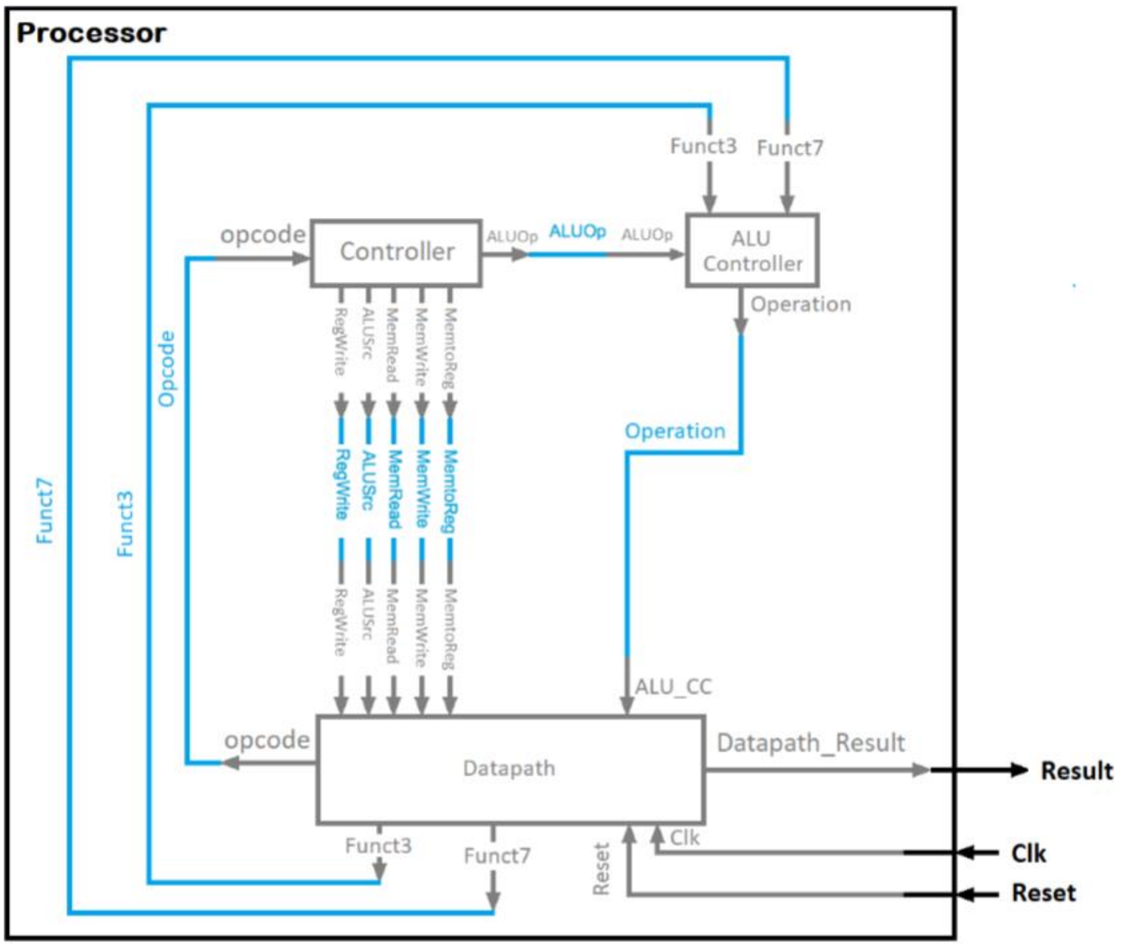
ADITYA NANDKISHOR SAWANT (sawantan@uci.edu)

Processor

1. Overview

The design provided is a basic processor module, which includes a control unit (Controller), an ALU control unit (ALUController), and a data path (data_path). The module uses these components to process instructions and produce a result.

Block Diagram:



Lab 5: Single-Cycle RISC-V Processor

ADITYA NANDKISHOR SAWANT (sawantan@uci.edu)

Inputs:

clk: Clock signal.

reset: Reset signal.

Outputs:

Result [31:0]: The output result of the processor after executing instructions.

2. Hardware Design

Processor Module:

The processor module instantiates the Controller, ALUController, and data_path modules, connecting them to orchestrate the instruction execution.

Controller Module:

Generates control signals based on the opcode input to direct the operations of the data path and ALU.

ALUController Module:

Generates ALU operation codes based on the ALUOp, Funct7, and Funct3 inputs to direct the specific operation within the ALU.

Data Path Module:

Performs the actual data operations based on control signals and generates the result.

Lab 5: Single-Cycle RISC-V Processor

ADITYA NANDKISHOR SAWANT (sawantan@uci.edu)

Verilog Code:

```
// Processor module
module processor (
    input clk, reset,
    output [31:0] Result
);

    // Instantiate signals
    wire [6:0] opcode;
    wire [2:0] funct3;
    wire [6:0] funct7;
    wire [3:0] operation;
    wire ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite;
    wire [1:0] ALUOp;

    data_path datapath_inst (
        .clk(clk),
        .reset(reset),
        .reg_write(RegWrite),
        .mem2reg(MemtoReg),
        .alu_src(ALUSrc),
        .mem_write(MemWrite),
        .mem_read(MemRead),
        .alu_cc(operation),
        .opcode(opcode),
        .funct7(funct7),
        .funct3(funct3),
        .alu_result(Result)
    );

    Controller controller_inst (
        .Opcode(opcode),
        .ALUSrc(ALUSrc),
        .MemtoReg(MemtoReg),
        .RegWrite(RegWrite),
        .MemRead(MemRead),
        .MemWrite(MemWrite),
        .ALUOp(ALUOp)
    );

    ALUController alucontroller_inst (
        .ALUOp(ALUOp),
        .Funct7(funct7),
        .Funct3(funct3),
        .Operation(operation)
    );

endmodule
```

3. Simulation Results

To validate the design, a testbench tb_processor is provided. It initializes the processor, generates clock pulses, and tests various instructions to verify the correctness of the Result output.

Testbench Code:

```
`timescale 1ns / 1ps

module tb_processor ();

  reg clk, rst;
  wire [31:0] tb_Result;

  processor processor_inst (
    .clk(clk),
    .reset(rst),
    .Result(tb_Result)
  );

  always begin
    #10;
    clk = ~clk;
  end
```

```
initial begin
  clk = 0;
  @(posedge clk);
  rst = 1;
  @(posedge clk);
  rst = 0;
end

integer point = 0;

always @(*) begin
  #20;
  if (tb_Result == 32'h00000000) // and
    point = point + 1;

  #20;
  if (tb_Result == 32'h00000001) // addi
    point = point + 1;

  // Continue for all other test cases

  #20;
  if (tb_Result == 32'h00000030) // sw
    point = point + 1;
    #20;
    if (tb_Result == 32'h00000030) // lw
      point = point + 1;


    $display("%s%d", "The number of correct test cases is:", point);
  end


  initial begin
    #430;
    $finish;
  end


endmodule
```

Clock Generation:

A clock signal is generated using an always block that toggles every 10 time units.

Reset Signal:

The reset signal is asserted and then de-asserted to initialize the processor.

Lab 5: Single-Cycle RISC-V Processor

ADITYA NANDKISHOR SAWANT (sawantan@uci.edu)

Test Cases:

The testbench includes multiple test cases to check different instructions by comparing tb_Result to expected values.

Points Calculation:

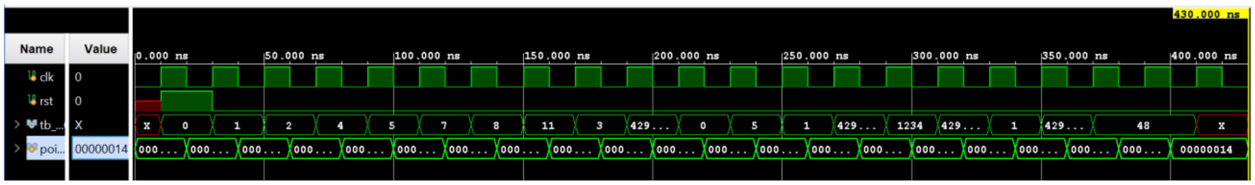
The point variable increments for each correct test case, providing a summary of successful tests.

Simulation Results:

The simulation results should include the tb_Result values for each test case, verifying that the processor correctly executes the instructions.

A display message will show the number of correct test cases.

Simulation Screenshots:


430.000 ns
Name
Value
0.000 ns
50.000 ns
100.000 ns
150.000 ns
200.000 ns
250.000 ns
300.000 ns
350.000 ns
400.000 ns
clk 0
rst 0
tb_... X
X 0 1 2 4 5 7 8 11 3 429... 0 5 1 429... 1234 429... 1 429... 48 X
poi... 00000014
000... 000... 000... 000... 000... 000... 000... 000... 000... 000... 000... 000... 000... 000... 000... 000... 000... 000... 000... 000... 00000014