Introduction to Digital Logic Design Lab

EECS 31L

Lab 5: RISK-V Single Cycle Processor

Ashley Yu

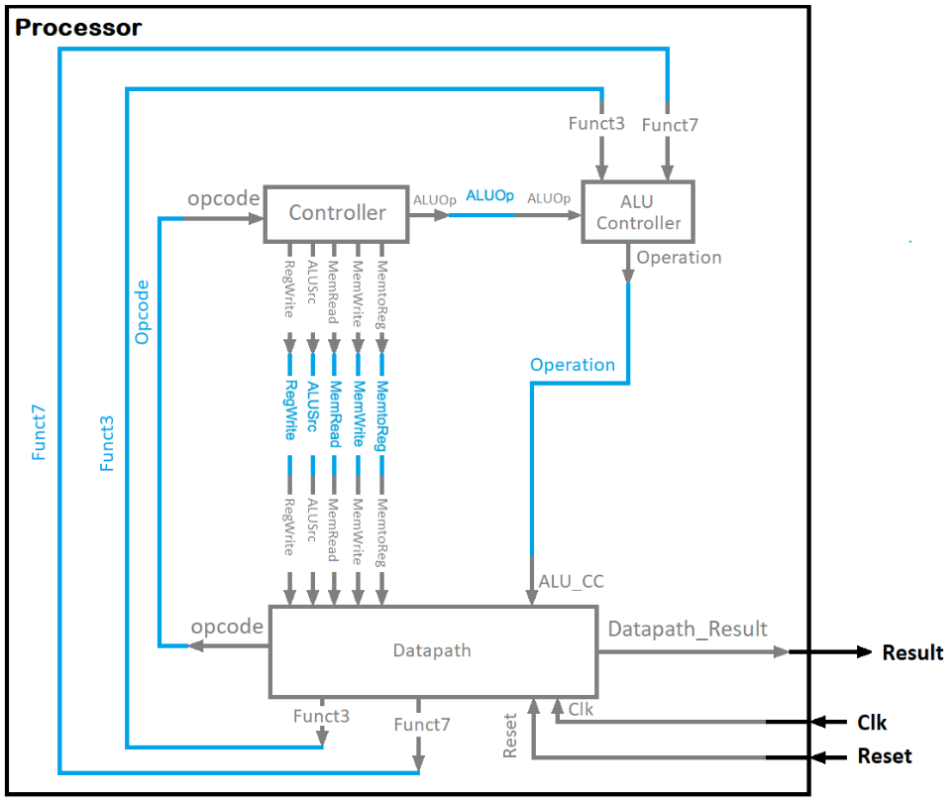
13704695

Mar 13 2022

---

1. Objective

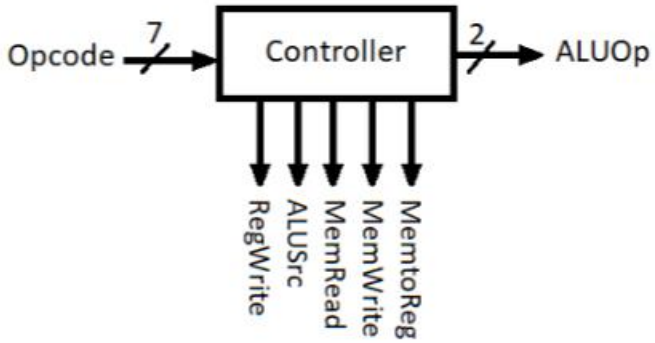
The single cycle processor takes in Clk, Reset, and outputs Result.



1.1 Low Level Module

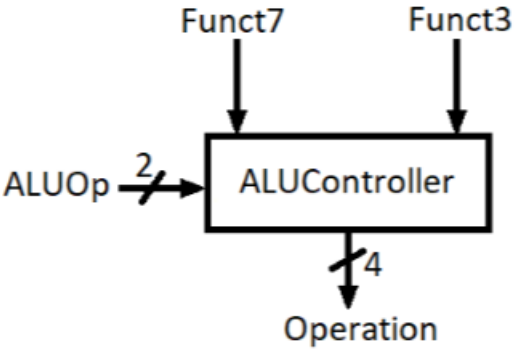
1.1.1 Controller

The controller takes a opcode as input and outputs RegWrite, ALUSrc, MemRead, MemWrite, MemtoReg, and ALUOp. If opcode is the Load instruction, MemWrite is set to 0 while everything else is 1. If it is the Store instruction, MemWrite and ALUSrc are set to 1 while the other three are 0. In both Store and Load, the ALUOp is 01. If it is the case of “0010011” then ALUSrc and RegWrite are 1 while the rest are 0 and ALUOp is 00. Lastly, when it is “0110011” RegWrite is 1 with all others being 0 and ALUOp as 10.



1.1.2 ALUController

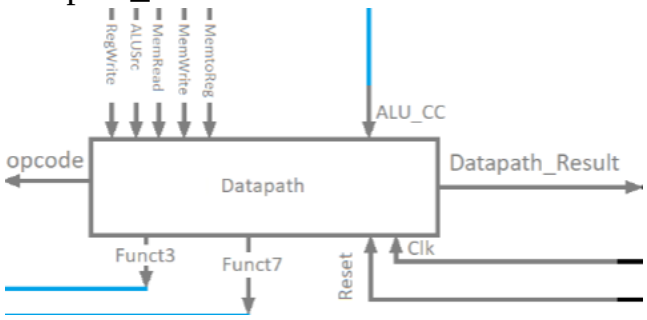
The ALU Controller takes in 2 bit ALUOp, Funct7, and Func3 as inputs when it outputs a 4 bit Operation. It decides which operation the ALU should perform based on the values of the three inputs.



1.1.3 Datapath

Datapath takes in the five outputs of MemtoReg, ALUSrc, RegWrite, MemRead, and Memread from the Controller, ALU_CC or Operation from the ALUController, and the Clk and Reset as inputs when it outputs the opcode, funct3, and

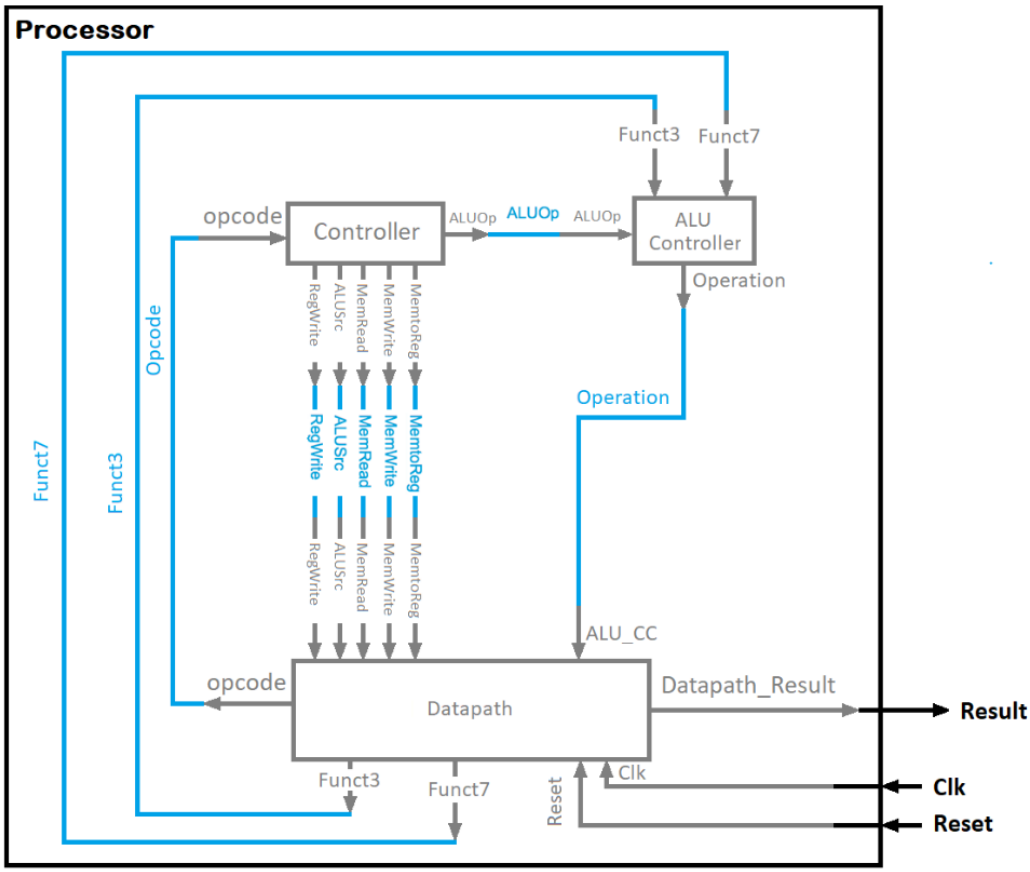
funct7 to the Controller and ALUController and the Datapath_Result out as Result.



1.2 Higher Level Module

1.2.1 Processor

The processor inputs Clk and reset while output Result.



2. Procedure

2.1 Low Level Module

1.2.2 2.1.1 Controller

I used the design block diagram from the Overview section and this truth table for the Controller.

Table 2 : Control Signals.

		Opcode			
		0110011	0010011	0000011	0100011
		AND, OR, ADD, SUB, SLT, NOR	ANDI, ORI, ADDI, SLTI, NORI	LW	SW
Control Signals	MemtoReg	0	0	1	0
	MemWrite	0	0	0	1
	MemRead	0	0	1	0
	ALUSrc	0	1	1	1
	Regwrite	1	1	1	0
	ALUOp	10	00	01	01

2.1.2 ALUController

I used this truth table and the design block diagram from the Overview section to create this ALUController.

Table 4 : The truth table for the 4 operation code.

	Funct7	Funct3	ALUop	Operation			
AND	0000000	111	10	0	0	0	0
OR	0000000	110	10	0	0	0	1
NOR	0000000	100	10	1	1	0	0
SLT	0000000	010	10	0	1	1	1
ADD	0000000	000	10	0	0	1	0
SUB	0100000	000	10	0	1	1	0
ANDI	-	111	00	0	0	0	0
ORI	-	110	00	0	0	0	1
NORI	-	100	00	1	1	0	0
SLTI	-	010	00	0	1	1	1
ADDI	-	000	00	0	0	1	0
LW	-	010	01	0	0	1	0
SW	-	010	01	0	0	1	0

2.1.3 Datapath

I used this code and the design block diagram from the Overview section to create this instruction memory.

```
`timescale 1ns / 1ps
module data_path #(
parameter PC_W = 8, // Program Counter
parameter INS_W = 32, // Instruction Width
parameter RF_ADDRESS = 5, // Register File Address
parameter DATA_W = 32, // Data WriteData
parameter DM_ADDRESS = 9, // Data Memory Address
parameter ALU_CC_W = 4 // ALU Control Code Width
)(
input clk ,
input reset ,
input reg_write ,
input mem2reg ,
input alu_src ,
input mem_write ,
input mem_read ,
input [ALU_CC_W -1:0] alu_cc ,
output [6:0] opcode ,
output [6:0] funct7 ,
output [2:0] funct3 ,
output [DATA_W -1:0] alu_result );
wire [PC_W -1:0] pc , pc_next;
wire [INS_W -1:0] instruction;
wire [DATA_W -1:0] l_alu_result;
wire [DATA_W -1:0] reg1 , reg2;
wire [DATA_W -1:0] l_read_data;
wire [DATA_W -1:0] l_reg_wr_data;
wire [DATA_W -1:0] srcb ;
wire [DATA_W -1:0] extimm;
// next pc
assign pc_next = pc + 4;
FlipFlop pcreg(clk , reset , pc_next , pc);
assign pc_next = pc + 4;
FlipFlop pcreg(clk , reset , pc_next , pc);
// instruction memory
InstMem instruction_mem (pc , instruction );
assign opcode = instruction [6:0];
assign funct7 = instruction [31:25];
assign funct3 = instruction [14:12];
// register file
RegFile rf(
.clk ( clk ),
.reset ( reset ),
.rg_wrt_en ( reg_write ),
.rg_wrt_addr ( instruction [11:7] ),
.rg_rd_addr1 ( instruction [19:15] ),
.rg_rd_addr2 ( instruction [24:20] ),
.rg_wrt_data ( l_reg_wr_data ),
.rg_rd_data1 ( reg1 ),
.rg_rd_data2 ( reg2 ));
assign l_reg_wr_data = mem2reg ? l_read_data : l_alu_result;
// sign extend
ImmGen ext_imm (instruction , extimm );
// alu
assign srcb = alu_src ? extimm : reg2;
alu_32 alu_module( .A_in(reg1), .B_in(srcb), .ALU_Sel(alu_cc),
.ALU_Out(l_alu_result), .Carry_Out (), .Zero(), .Overflow ());
assign alu_result = l_alu_result;
// data memory
DataMem data_mem (.addr(l_alu_result[DM_ADDRESS -1:0]) , .read_data(l_read_data),
.write_data(reg2), .MemWrite(mem_write), .MemRead(mem_read) );
endmodule
```

3. Simulation Results

Test:

Reset is 1 from 10ns to 30ns and 0 the rest. Clk starts a 0 and turns on every 10 ns. Result is initially X like reset for the first 10ns and 0 when reset is on. Then at every positive edge of the clock the result updates. By checking the point array I can be sure that my results are correct.