Introduction to Digital Logic Design Lab
EECS 31L

Lab 5

Kyle Zyler Cayanan
80576955

03/09/2021

1 - Objective

For this lab we completed the rest of the modules necessary in order to design a RISC-V single cycle processor. The diagram is much simpler than that of DataPath, the diagram is pictured below.

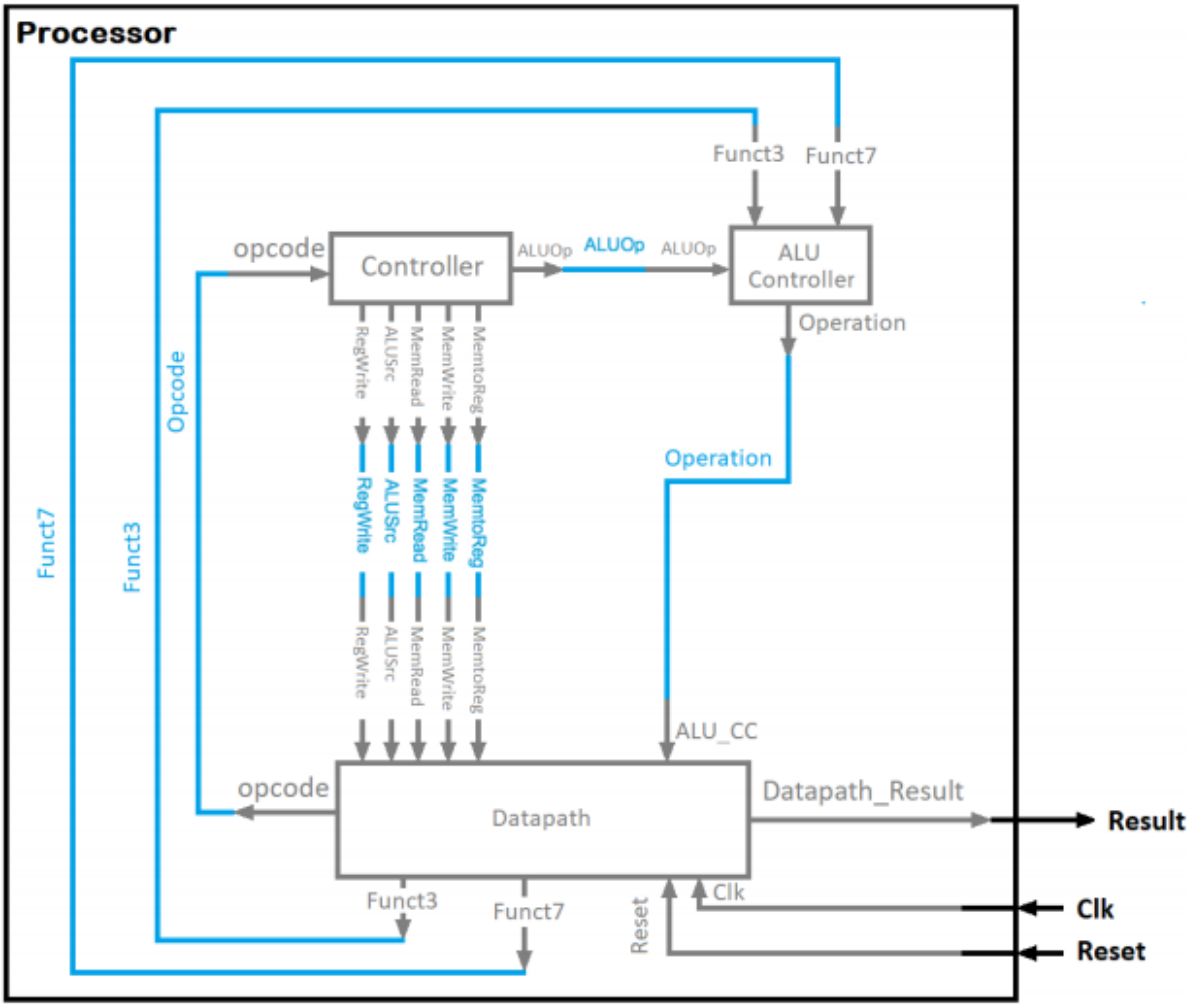

Figure 1 : Processor.

2 - Procedure

2.1 - Controller

The Controller was designed based on the table provided by the lab manual. A 7-bit opcode input would specify the signals required for that instruction as shown by the table below. The output signals are heavily reliant on the opcode. Behavioral design was used to implement this module and to closely reflect the provided table, specifically a case statement. Essentially, the opcode input enables/disables the necessary control signals needed for the operation.

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

2.2 - ALU Controller

The ALU Controller receives its input from the Controller module as well as inputs Funct7 and Funct3 which defines the operation that the Datapath module uses. This module was implemented using Behavioural design consisting mainly of ternary operators. Each individual bit of the opcode is assigned based on conditions of Funct7, Funct3, and ALUOp. The code and table are shown below to illustrate the conditions required to produce the needed output. For example, for the first bit of the Operation output, the output is only 1 whenever Funct3 is either 010 or 110 and is 0 otherwise. We use this logic to create the rest of the conditions.

```
// Module definition
module ALUController (
ALUOp , Funct7 , Funct3 , Operation
);
// Define the input and output signals
input   [1:0] ALUOp;
input   [6:0] Funct7;
input   [2:0] Funct3;
output  [3:0] Operation;

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

endmodule // 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

Table 3 : Instruction and the operation code.

operation	Operation code
AND, ANDI	0000
OR, ORI	0001
ADD, ADDI, SW, LW	0010
SUB	0110
SLT, SLTI	0111
NOR, NORI	1100

2.3 - Processor

The processor is simply a culmination of all the submodules we have designed so far. Structural design was used to implement and complete the processor. The submodules used for this design are instantiations of Controller, ALU Controller, and DataPath all by name.

3 - Simulation Results

For the result, the point output declared in the test bench was used to count the amount of instructions that had been executed. There is also the clk and reset inputs and tb_result, clk controls the clock and propagates every 10 nanoseconds, and reset will reset tb_result to zero. The tb_result output is simply the result from the processor. As long as the point output reached a value of 20 (in signed decimal), then our processor worked successfully. As pictured in the waveform below, our point output reached 20.