**ALU Simulator**

Luciano Fasani Ortiz, Keshawn Triplett, Denae Douglas

May 4, 2017

copyright_symbol.png L. Fasani, K. Triplett, D. Douglas

# Revision History

**(**May 1): This is the start of our initial program structure. This version had an erroneous concept of ADDI.

(May 2): This version corrected the ADDI and implemented all test cases given in the ALU Simulator text file.

(May 3): This version has the complete ALU code written and tested.

Table of Contents

[Revision History i](#_Toc481609238)

[List of Figures 1](#_Toc481609239)

[List of Tables 2](#_Toc481609240)

[Abstract 3](#_Toc481609241)

[Principles of Operation 4](#_Toc481609242)

[Data Structures Description 5](#_Toc481609243)

[Function Descriptions 6](#_Toc481609244)

[Parameters 7](#_Toc481609245)

[Testing 8](#_Toc481609246)

[Results 9](#_Toc481609247)

[Lessons Learned 10](#_Toc481609248)

[Program Listing 11](#_Toc481609249)

# List of Figures

* Figure 1.1: ALU Simulator\_main
* Figure 1.2: ALU Simulator

# List of Tables

List of tables

# Abstract

The main objective of the ALU Simulator project was to write a program that simulates the ALU in order to have a better understanding of how the ALU functions. We had to read in instructions, decode them, then modify the contents of the register according to the values given.

We implemented our program in C. We were given source code that decoded the instructions and gave us back the correct values. The code given also had the structure for the registers themselves. The code we implemented used a switch case and if statements in order to choose the right thing to do. Once in that case or if-else block, the correct instructions were executed. The results were analyzed when the contents of the register were dumped, that is how we verified whether or not the right instructions were being executed.

# Principles of Operation

This is the POP

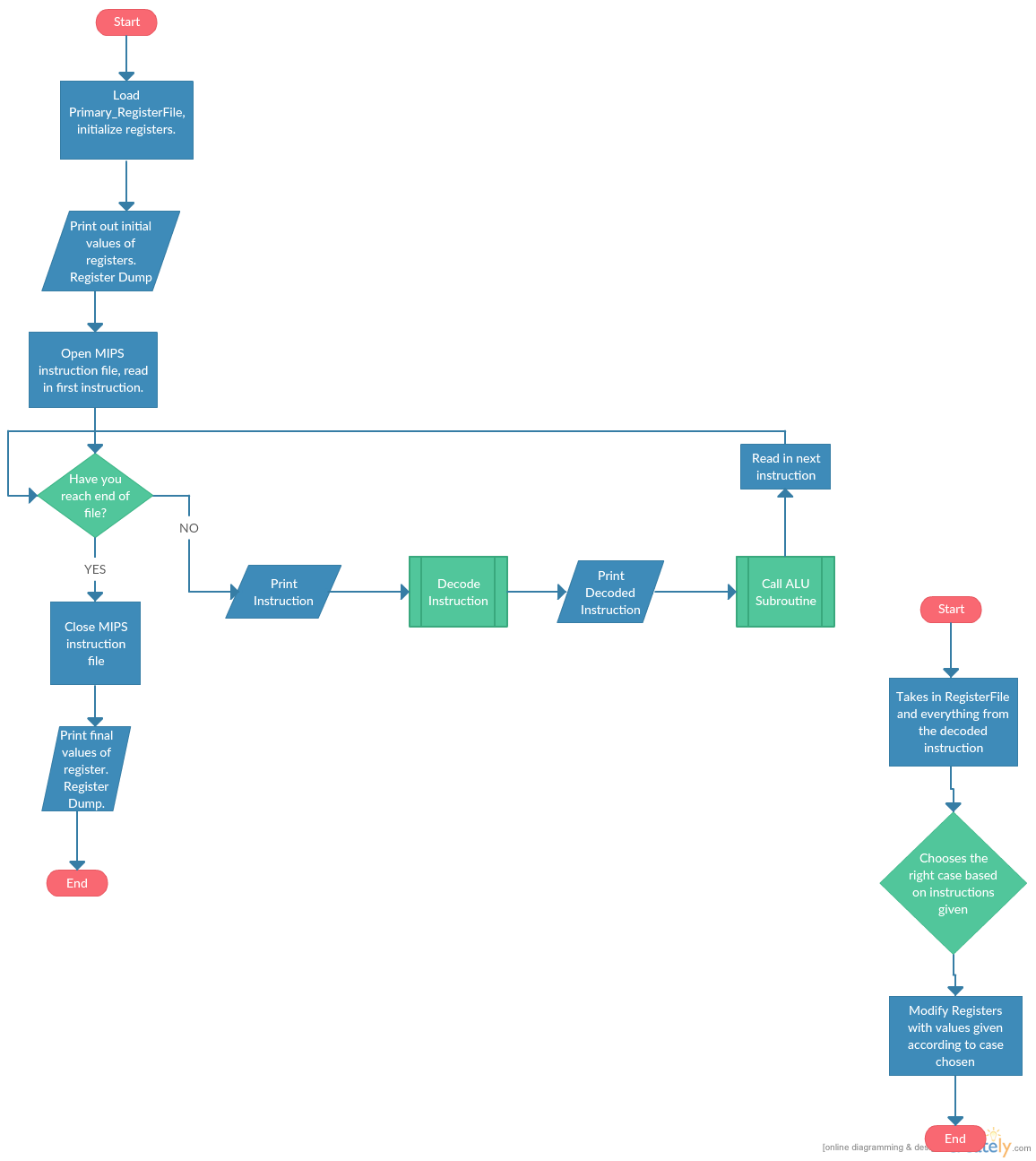


Figure 1.2: ALU Simulator

Figure .1: ALU Simulator\_main

# Data Structures Description

Description of DS

# Function Descriptions

Function Descriptions

# Parameters

Parameters

# Testing

In order to test our ALU Simulator, we used the RegisterFile\_Dump function given in the RegisterFile\_01.c file. This function prints the contents of a register file to standard output. When the program is ran, RegisterFile\_Dump prints the initial contents of the register file. After all of the instructions are simulated and written to the register file, RegisterFile\_Dump then prints the final contents of the register file. To test, each time one of the instructions in the switch case were implemented, the program was run. The final output of the RegisterFile\_Dump was then checked to ensure that the correct value was present at the corresponding register. Once all of the instructions were implemented in the switch case, a final check was performed to ensure the values in all of the registers from the final RegisterFile\_Dump were the same as the expected output.

# Results

* Below, is the output of out ALU with all of the instructions implemented. The first shows the initial writing of values to registers in the register file. RegisterFile\_Dump is then called to print the initial contents of of the registers. Next, each of the instructions from MIPS\_Instructions\_01.asm is simulated, in which the instruction is parsed out to determine which case will be called from switch case in ALUSimulator\_Main.c. The computed value from the instruction is then written to the register file in the register determined by the instruction. Once all of the instructions are simulated, RegisterFile\_Dump is again called to print the final state of the register file. From the output of our ALU simulator, it can be seen that each of our instructions from ALUSimulator\_Main.c. were implemented correctly, and that all output from the final register dump match the expected values.

ALU Simulator Output

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 01; WrtVal: 5

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 02; WrtVal: 10

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 03; WrtVal: 17

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 04; WrtVal: 390

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 05; WrtVal: 1010

Initial RegisterFile: ========================================

00000000: 00000000 00000005 00000010 00000017

00000004: 00000390 00001010 00000000 00000000

00000008: 00000000 00000000 00000000 00000000

0000000C: 00000000 00000000 00000000 00000000

00000010: 00000000 00000000 00000000 00000000

00000014: 00000000 00000000 00000000 00000000

00000018: 00000000 00000000 00000000 00000000

0000001C: 00000000 00000000 00000000 00000000

Instruction: 20490064

>>Opcode: 08; Rs: 02; Rt: 09; Rd: 00;

>>>>ShiftAmt: 01; FunctionCode: 24; ImmediateValue: 0064;

RegisterFile\_Read: RdAddr\_S: 02; RdAddr\_T: 09;

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 09; WrtVal: 74

>>ALU: Opcode: 08; Rs: 02; Rt: 09; Rd: 00;

>>>>ALU: ShiftAmt: 01; FunctionCode: 24; ImmediateValue: 0064;

Instruction: 20AAFF80

>>Opcode: 08; Rs: 05; Rt: 0A; Rd: 1F;

>>>>ShiftAmt: 1E; FunctionCode: 00; ImmediateValue: FF80;

RegisterFile\_Read: RdAddr\_S: 05; RdAddr\_T: 10;

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 10; WrtVal: F90

>>ALU: Opcode: 08; Rs: 05; Rt: 0A; Rd: 1F;

>>>>ALU: ShiftAmt: 1E; FunctionCode: 00; ImmediateValue: FF80;

Instruction: 00245820

>>Opcode: 00; Rs: 01; Rt: 04; Rd: 0B;

>>>>ShiftAmt: 00; FunctionCode: 20; ImmediateValue: 5820;

RegisterFile\_Read: RdAddr\_S: 01; RdAddr\_T: 04;

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 11; WrtVal: 395

>>ALU: Opcode: 00; Rs: 01; Rt: 04; Rd: 0B;

>>>>ALU: ShiftAmt: 00; FunctionCode: 20; ImmediateValue: 5820;

Instruction: 00056100

>>Opcode: 00; Rs: 00; Rt: 05; Rd: 0C;

>>>>ShiftAmt: 04; FunctionCode: 00; ImmediateValue: 6100;

RegisterFile\_Read: RdAddr\_S: 00; RdAddr\_T: 05;

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 12; WrtVal: 10100

>>ALU: Opcode: 00; Rs: 00; Rt: 05; Rd: 0C;

>>>>ALU: ShiftAmt: 04; FunctionCode: 00; ImmediateValue: 6100;

Instruction: 00856825

>>Opcode: 00; Rs: 04; Rt: 05; Rd: 0D;

>>>>ShiftAmt: 00; FunctionCode: 25; ImmediateValue: 6825;

RegisterFile\_Read: RdAddr\_S: 04; RdAddr\_T: 05;

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 13; WrtVal: 1390

>>ALU: Opcode: 00; Rs: 04; Rt: 05; Rd: 0D;

>>>>ALU: ShiftAmt: 00; FunctionCode: 25; ImmediateValue: 6825;

Instruction: 00437026

>>Opcode: 00; Rs: 02; Rt: 03; Rd: 0E;

>>>>ShiftAmt: 00; FunctionCode: 26; ImmediateValue: 7026;

RegisterFile\_Read: RdAddr\_S: 02; RdAddr\_T: 03;

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 14; WrtVal: 7

>>ALU: Opcode: 00; Rs: 02; Rt: 03; Rd: 0E;

>>>>ALU: ShiftAmt: 00; FunctionCode: 26; ImmediateValue: 7026;

Instruction: 00627822

>>Opcode: 00; Rs: 03; Rt: 02; Rd: 0F;

>>>>ShiftAmt: 00; FunctionCode: 22; ImmediateValue: 7822;

RegisterFile\_Read: RdAddr\_S: 03; RdAddr\_T: 02;

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 15; WrtVal: 7

>>ALU: Opcode: 00; Rs: 03; Rt: 02; Rd: 0F;

>>>>ALU: ShiftAmt: 00; FunctionCode: 22; ImmediateValue: 7822;

Final RegisterFile: ========================================

00000000:  00000000  00000005  00000010  00000017

00000004:  00000390  00001010  00000000  00000000

00000008:  00000000  00000074  00000F90  00000395

0000000C:  00010100  00001390  00000007  00000007

00000010:  00000000  00000000  00000000  00000000

00000014:  00000000  00000000  00000000  00000000

00000018:  00000000  00000000  00000000  00000000

0000001C:  00000000  00000000  00000000  00000000

Expected Final Register File

00000000:  00000000  00000005  00000010  00000017

00000004:  00000390  00001010  00000000  00000000

00000008:  00000000  00000074  00000F90  00000395

0000000C:  00010100  00001390  00000007  00000007

00000010:  00000000  00000000  00000000  00000000

00000014:  00000000  00000000  00000000  00000000

00000018:  00000000  00000000  00000000  00000000

0000001C:  00000000  00000000  00000000  00000000

# Lessons Learned

From completing this project, we learned many new things. One of the major things we gathered from this project was a better understanding of how an ALU functions. This project served as a way for us to familiarize ourselves with operations of an ALU and learn the inner workings of each computation.

If we were to complete this project again, there would be many things that we would do differently. One of the things that we would do differently is to go for simplicity from the beginning. With starting this project, inside of ALUSimulator\_Main.c, we first decided to implement each of the instructions using number of if-statements for each of the different opcodes. While this approach would have worked, we found it more efficient to use a simple switch case. The use a switch case made our code more organized, easier to work with, and easier to understand. To future students, I would advise you to try to model your ALU in a fashion that is both simple and effective. Also, I would advise students to not be afraid to ask questions about things that may not be clear.

Luciano’s contributions were implementing the SRA, AND, SLT, SLTU, SLTI, SLTIU. I also wrote the abstract and created the flow chart for the POP.

# Program Listing

Listing of source code