ALU Simulator

Luciano Fasani Ortiz, Keshawn Triplett, Denae Douglas

May 4, 2017

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

List of Figures

List of Tables

No tables were used in this report.

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

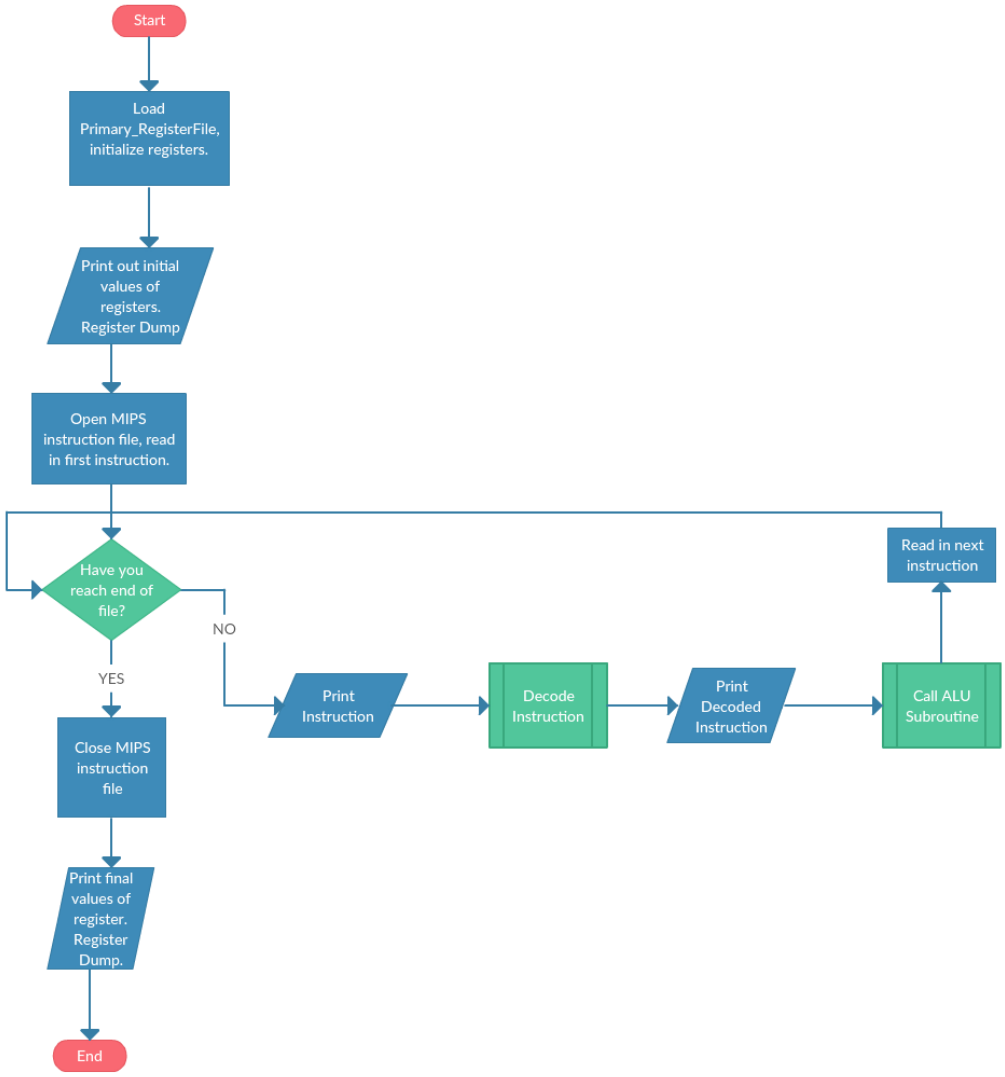

Figure 2.1: ALU Simulator_main

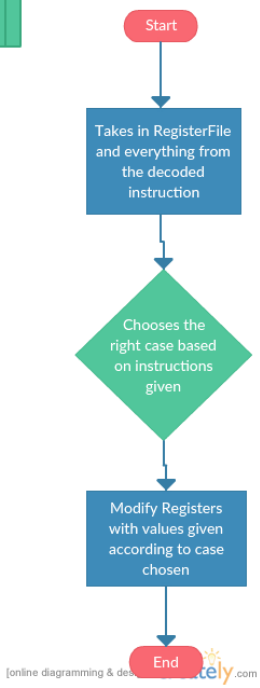

Figure 1.2: ALU Simulator

Data Structures Description

No Data Structures were used in the development of our code. However, Dr. Minden provided us with a register file modeled as an array.

Function Descriptions

ALU Simulator(): This is the primary function used to compute values in the register file. It takes in the Register File, OpCode, Immediate Value, Rs, Rt, Rd, Function Code, Shift Amount and Status. It then uses a switch case by Opcode and if statements to determine which type of computation it needs to carry out.

PseudoCode:

```
extern void ALUSimulator( RegisterFile theRegisterFile,
                          uint32_t OpCode,
                          uint32_t Rs, uint32_t Rt, uint32_t Rd,
                          uint32_t ShiftAmt,
                          uint32_t FunctionCode,
                          uint32_t ImmediateValue,
                          uint32_t* Status ) {

    uint32_t RsValue;
    uint32_t RtValue;
    uint32_t RdValue;


    switch(OpCode){
        case 0:{
            if(FunctionCode == 32){
                Read from the register file
                Do the computation
                Write the value back to the register file in the correct register
            }
            else if(FunctionCode == 0){
                Read from the register file
                Do the computation
                Write the value back to the register file in the correct register
            }
            else if(FunctionCode == 37){
                Read from the register file
                Do the computation
                Write the value back to the register file in the correct register
            }
            else if(FunctionCode == 38){
                Read from the register file
                Do the computation
                Write the value back to the register file in the correct register
            }
            else if(FunctionCode == 34){
                Read from the register file
```

```
        Do the computation
        Write the value back to the register file in the correct register
}
else if(FunctionCode == 36){
        Read from the register file
        Do the computation
        Write the value back to the register file in the correct register
}
else if(FunctionCode == 2){
        Read from the register file
        Do the computation
        Write the value back to the register file in the correct register
}
else if(FunctionCode == 35){
        Read from the register file
        Do the computation
        Write the value back to the register file in the correct register
}
else if(FunctionCode == 4){
        Read from the register file
        Do the computation
        Write the value back to the register file in the correct register
}
else if(FunctionCode == 6){
        Read from the register file
        Do the computation
        Write the value back to the register file in the correct register
}
else if(FunctionCode == 33){
        Read from the register file
        Do the computation
        Write the value back to the register file in the correct register
}
else if(FunctionCode == 33){
        Read from the register file
        Do the computation
        Write the value back to the register file in the correct register
}
else if(FunctionCode == 42){
        Read from the register file
        Do the computation
        Write the value back to the register file in the correct register
}
else if(FunctionCode == 43){
        Read from the register file
```

```
                    Do the computation

                    Write the value back to the register file in the correct register
                }
                else if(FunctionCode == 3){
                    Read from the register file
                    Do the computation
                    Write the value back to the register file in the correct register
                }

                break;
            }
            case 8:{
                Read from the register file
                Do the computation
                Write the value back to the register file in the correct register
                break;
            }
            case 9:{
                Read from the register file
                Do the computation
                Write the value back to the register file in the correct register
                break;
            }
            case 10:{
                Read from the register file
                Do the computation
                Write the value back to the register file in the correct register
                break;
            }
            case 11:{
                Read from the register file
                Do the computation
                Write the value back to the register file in the correct register
                break;
            }
}
```

Parameters

THE_REGISTER_FILE: the register file was created by Dr. Minden and we called read and write on the register during computation.

OPCODE: opcode was given from the text file and used to determine the opcode of the instruction. There were five different opcodes to implement and we used a switch case to determine which set of computations we would choose from in the code.

RS: the second register read from the register file

RT: the first register read from the register file

RD: the third register read from the register file

SHIFTAMT: used to determine the number of shifts for functions like SLL and SRL

FUNCTION_CODE: this value specified exactly which computation was to be carried out by the ALU

IMMEDIATEVALUE: this value was used to compute instructions such as ADDI, and SLTI. This value was sometimes cast to a 32 bit int to account for negative values

RSVALUE: holds the value of the Rs register

RDVALUE: holds the value of the Rd register

RTVALUE: holds the value of the Rt value

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. Luciano also wrote the abstract and created the flow chart for the POP. Denae was responsible for writing the code for the test cases provided by Dr. Minden in the ALU_Simulator.txt. Also, she was responsible for sections in the report including: Revisions, Data Structure Descriptions, Function Descriptions, and Parameter Descriptions. Keshawn was responsible for implementing ADDIU, SLL, SRL, SLLV, SRLV, and ADDU. Keshawn also completed the results, testing, and lessons learned portions of the report.

Program Listing

```
//*****************************************************************************
*
//--ALUSimulator.c
//
//              Author:             Gary J. Minden
//              Organization:  KU/EECS/EECS 645
//              Date:               2017-04-22 (B70422)
//              Version:            1.0
//              Description:    This is the test standin for a simple ALU simulator
//              Notes:
//
//*****************************************************************************
*
//

#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdarg.h>

#include <stdio.h>

#include "RegisterFile_01.h"
#include "ALUSimulator.h"
#include <math.h>
extern void ALUSimulator( RegisterFile theRegisterFile,
                                        uint32_t OpCode,
                                        uint32_t Rs, uint32_t Rt, uint32_t Rd,
                                        uint32_t ShiftAmt,
                                        uint32_t FunctionCode,
                                        uint32_t ImmediateValue,
                                        uint32_t* Status ) {

        uint32_t RsValue;
        uint32_t RtValue;
        uint32_t RdValue;


        switch(OpCode){
                case 0:{
                        if(FunctionCode == 32){
                                RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
                                RdValue = RsValue + RtValue;
                                RegisterFile_Write(theRegisterFile, 1, Rd, RdValue);
```

```
}
else if(FunctionCode == 0){
        RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
        RdValue = RtValue << ShiftAmt;
        RegisterFile_Write(theRegisterFile, 1, Rd, RdValue);
}
else if(FunctionCode == 37){
        RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
        RdValue = RsValue | RtValue;
        RegisterFile_Write(theRegisterFile, 1, Rd, RdValue);
}
else if(FunctionCode == 38){
        RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
        RdValue = RsValue ^ RtValue;
        RegisterFile_Write(theRegisterFile, 1, Rd, RdValue);
}
else if(FunctionCode == 34){
        RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
        RdValue = RsValue - RtValue;
        RegisterFile_Write(theRegisterFile, 1, Rd, RdValue);
}
else if(FunctionCode == 36){
        RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
        RdValue = RsValue & RtValue;
        RegisterFile_Write(theRegisterFile, 1, Rd, RdValue);
}
else if(FunctionCode == 2){
        RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
        RdValue = RtValue >> ShiftAmt;
        RegisterFile_Write(theRegisterFile, 1, Rd, RdValue);
}
else if(FunctionCode == 35){
        RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
        RdValue = RsValue - RtValue;
        RegisterFile_Write(theRegisterFile, 1, Rd, RdValue);
}
else if(FunctionCode == 4){
        RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
        RdValue = RtValue << RsValue;
        RegisterFile_Write(theRegisterFile, 1, Rd, RdValue);
}
else if(FunctionCode == 6){
        RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
        RdValue = RtValue >> RsValue;
        RegisterFile_Write(theRegisterFile, 1, Rd, RdValue);
```

```
        }
        else if(FunctionCode == 33){
                RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
                RdValue = RtValue + RsValue;
                RegisterFile_Write(theRegisterFile, 1, Rd, RdValue);
        }
        else if(FunctionCode == 42){
                RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
                if(RsValue < RtValue){
                        RdValue = 1;
                } else {
                        RdValue = 0;
                }
                RegisterFile_Write(theRegisterFile, 1, Rd, RdValue);
        }
        else if(FunctionCode == 43){
                RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
                if(RsValue < RtValue){
                        RdValue = 1;
                } else {
                        RdValue = 0;
                }
                RegisterFile_Write(theRegisterFile, 1, Rd, RdValue);
        }
        else if(FunctionCode == 3){
                RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
                for(int i = 0; i < ShiftAmt; i++){
                        RdValue = floor(RtValue/2);
                }
                RegisterFile_Write(theRegisterFile, 1, Rd, RdValue);
        }

        break;
}
case 8:{
        RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
        RtValue = RsValue + (int32_t)(int16_t)ImmediateValue;
        RegisterFile_Write(theRegisterFile, 1, Rt, RtValue);
        break;
}
case 9:{
        RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
        RtValue = RsValue + ImmediateValue;
        RegisterFile_Write(theRegisterFile, 1, Rt, RtValue);
        break;
```

```
		}
		case 10:{
			RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
			if(RsValue < (int32_t)(int16_t)ImmediateValue){
				RtValue = 1;
			} else {
				RtValue = 0;
			}
			RegisterFile_Write(theRegisterFile, 1, Rt, RtValue);
			break;
		}
		case 11:{
			RegisterFile_Read(theRegisterFile, Rs, &RsValue, Rt, &RtValue);
			if(RsValue < (int32_t)(int16_t)ImmediateValue){
				RtValue = 1;
			} else {
				RtValue = 0;
			}
			RegisterFile_Write(theRegisterFile, 1, Rt, RtValue);
			break;
		}
	}


	printf( ">>ALU: Opcode: %02X; Rs: %02X; Rt: %02X; Rd: %02X;\n",
				OpCode,
				Rs,
				Rt,
				Rd );

	printf( ">>>>ALU: ShiftAmt: %02X; FunctionCode: %02X; ImmediateValue:
%04X;\n",
				ShiftAmt,
				FunctionCode,
				ImmediateValue );

}
```