**Project #1**

**ALU Simulation**

**May 3,2018**

**By: Coltyn Gatton, Noah Kenn, and Blaine Edmonson**

**OUTPUT:**

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 01

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 02

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 03

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 04

RegisterFile\_Write: WrtEnb: 1; WrtAddr: 05

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

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

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

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

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

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

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

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

**SOURCE CODE:**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//

// Authors: Blaine Edmondson, Coltyn Gatton, Noah

// Kenn

// Organization: KU/EECS/EECS 645

// Date: 2018-05-02

// Last edited: 2018-05-03

// Class: EECS 645

// Description: A simulator of a MIPS ALU

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include "RegisterFile\_01.h"

#include "ALUSimulator.h"

// Set wordsize to be 32

#define WORDSIZE 32

// Create a mask that is 32 bits for use with mult

#define MASK (0xFFFFFFFF);

// Enumerate the function codes for ease of understanding

enum Function\_Code {

NOOP\_SLL = 0b000000,

SRL = 0b000010,

SRA = 0b000011,

SLLV = 0b000100,

SRLV = 0b000110,

MFHI = 0b010000,

MFLO = 0b010010,

MULT = 0b011000,

MULTU = 0b011001,

DIV = 0b011010,

DIVU = 0b011011,

ADD = 0b100000,

ADDU = 0b100001,

SUB = 0b100010,

SUBU = 0b100011,

AND = 0b100100,

OR = 0b100101,

XOR = 0b100110,

SLT = 0b101010,

SLTU = 0b101011

};

// Enumerate the OP codes for ease of understanding

enum Op\_Code {

Non\_I\_Type = 0b000000,

ADDI = 0b001000,

ADDIU = 0b001001,

SLTI = 0b001010,

SLTIU = 001011

};

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) {

// Create two integers that will be used to access the register

uint32\_t RsVal = 0;

uint32\_t RtVal = 0;

// Access the registers at the given addresses by passing pointers to the

// above integers

RegisterFile\_Read(theRegisterFile, Rs, &RsVal, Rt, &RtVal);

// Create an integer to hold the value to be written to the register

uint32\_t RdVal = 0;

switch (OpCode) {

case ADDI:

// Cast RsVal to signed int32\_t and ImmediateValue to signed int16\_t

// Add those together and save them to RdVal

RdVal = ((int32\_t)RsVal + (int16\_t)ImmediateValue);

//Write RdVal to register Rt

RegisterFile\_Write(theRegisterFile, true, Rt, RdVal);

return;

break;

case ADDIU:

// Cast ImmediateValue to signed int16\_t

// Add that to RsVal and save the result to RdVal

RdVal = (RsVal + (int16\_t)ImmediateValue);

//Write RdVal to register Rt

RegisterFile\_Write(theRegisterFile, true, Rt, RdVal);

return;

break;

case SLTI:

// Cast RsVal to signed int32\_t and ImmediateValue to signed int16\_t

// Use a ternary statement to set the value of RdVal

RdVal = ((int32\_t)RsVal < (int16\_t)ImmediateValue ? 1 : 0);

//Write RdVal to register Rt

RegisterFile\_Write(theRegisterFile, true, Rt, RdVal);

return;

break;

case SLTIU:

// Cast ImmediateValue to signed int16\_t

// Use a ternary statement to set the value of RdVal

RdVal = (RsVal < (int16\_t)ImmediateValue ? 1 : 0);

//Write RdVal to register Rt

RegisterFile\_Write(theRegisterFile, true, Rt, RdVal);

return;

break;

case Non\_I\_Type:

switch (FunctionCode) {

case NOOP\_SLL:

// Bit-Shift the value of Rt left by ShiftAmt

RdVal = RtVal << ShiftAmt;

break;

case SRL:

// Bit-Shift the value of Rt right by ShiftAmt

RdVal = RtVal >> ShiftAmt;

break;

case SRA:

// Cast RtVal to int32\_t

// Bit-Shift the value of Rt right by ShiftAmt

RdVal = (int32\_t)RtVal >> ShiftAmt;

break;

case SLLV:

// Bit-shift Rt left by RsVal

RdVal = RtVal << RsVal;

break;

case SRLV:

// Bit-shift Rt right by RsVal

RdVal = RtVal >> RsVal;

break;

case MFHI:

// Read the value in the high register (and Rt, but we ignore this)

RegisterFile\_Read(theRegisterFile, 0b011111, &RsVal, Rt, &RtVal);

// Set Rd to the value of Rs

RdVal = RsVal;

break;

case MFLO:

// Read the value in the low register (and Rt, but we ignore this)

RegisterFile\_Read(theRegisterFile, 0b011110, &RsVal, Rt, &RtVal);

// Set Rd to the value of Rs

RdVal = RsVal;

break;

case MULT:;

// Multiply the values together and get the full value

int64\_t fullMult = (int32\_t)RsVal \* (int32\_t)RtVal;

// Grab the first WORDSIZE digits from the multiplication

int32\_t lowVal = fullMult & MASK;

// Grab the last WORDSIZE digits from the multiplication

int32\_t highVal = (fullMult >> WORDSIZE) & MASK;

// Write lowVal and highVal to the low and high registers respectively

RegisterFile\_Write(theRegisterFile, true, 0b011110, lowVal);

RegisterFile\_Write(theRegisterFile, true, 0b011111, highVal);

return;

break;

case MULTU:;

// Multiply the values together and get the full value

uint64\_t ufullMult = RsVal \* RtVal;

// Grab the first WORDSIZE digits from the multiplication

uint32\_t ulowVal = ufullMult & MASK;

// Grab the last WORDSIZE digits from the multiplication

uint32\_t uhighVal = (ufullMult >> WORDSIZE) & MASK;

// Write lowVal and highVal to the low and high registers respectively

RegisterFile\_Write(theRegisterFile, true, 0b011110, ulowVal);

RegisterFile\_Write(theRegisterFile, true, 0b011111, uhighVal);

return;

break;

case DIV:

// Cast RsVal and RtVal to int32\_t

// Write the remainder of the division of Rs and Rt to the low Register

RegisterFile\_Write(theRegisterFile, true, 0b011110,

(int32\_t)RsVal % (int32\_t)RtVal);

// Cast RsVal and RtVal to int32\_t

// Write the quotient of the division of Rs and Rt to the high Register

RegisterFile\_Write(theRegisterFile, true, 0b011111,

(int32\_t)RsVal / (int32\_t)RtVal);

return;

break;

case DIVU:

// Write the remainder of the division of Rs and Rt to the low Register

RegisterFile\_Write(theRegisterFile, true, 0b011110, RsVal % RtVal);

// Write the quotient of the division of Rs and Rt to the high Register

RegisterFile\_Write(theRegisterFile, true, 0b011111, RsVal / RtVal);

return;

break;

case ADD:

// Cast RsVal and RtVal to int32\_t

// Add these values and save to RdVal

RdVal = (int32\_t)RsVal + (int32\_t)RtVal;

break;

case ADDU:

// Add RsVal and RtVal and save to RdVal

RdVal = RsVal + RtVal;

break;

case SUB:

// Cast RsVal and RtVal to int32\_t

// Subtract RtVal from RsVal and save to RdVal

RdVal = (int32\_t)RsVal - (int32\_t)RtVal;

break;

case SUBU:

// Subtract RtVal from RsVal and save to RdVal

RdVal = RsVal - RtVal;

break;

case AND:

// Set RdVal to the bitwise AND of RsVal and RtVal

RdVal = RsVal & RtVal;

break;

case OR:

// Set RdVal to the bitwise OR of RsVal and RtVal

RdVal = RsVal | RtVal;

break;

case XOR:

// Set RdVal to the bitwise XOR of RsVal and RtVal

RdVal = RsVal ^ RtVal;

break;

case SLT:

// Cast RsVal and RtVal to int32\_t

// Use a ternary statement to set the value of RdVal

RdVal = ((int32\_t)RsVal < (int32\_t)RtVal ? 1 : 0);

break;

case SLTU:

// Use a ternary statement to set the value of RdVal

RdVal = (RsVal < RtVal ? 1 : 0);

break;

}

break;

}

// Write the value of the RdVal to the register Rd

RegisterFile\_Write(theRegisterFile, true, Rd, RdVal);

}

**GROUP CONTRIBUTION:**

Coltyn Gatton: Translated MIPS instruction code into C code and assisted in putting the ideas into written code. Was also tasked with creating the report.

Noah Kenn: Handled interaction with the main and register files, as well as put ideas into written code.

Blaine Edmonson: Researched some questions we had regarding unsigned instructions and made sure the final implementation met the requirements.