EECS 645 – Homework #04
ALU Simulation

Spring 2017

Gary J. Minden

1 Description

For homework #04 you will write a program that simulates a Arithmetic Logic Unit (ALU). You will be provided with a mainline and register file model written in C. C is thee very preferred programming language. Make sure your program is well commented so others can understand your approach.

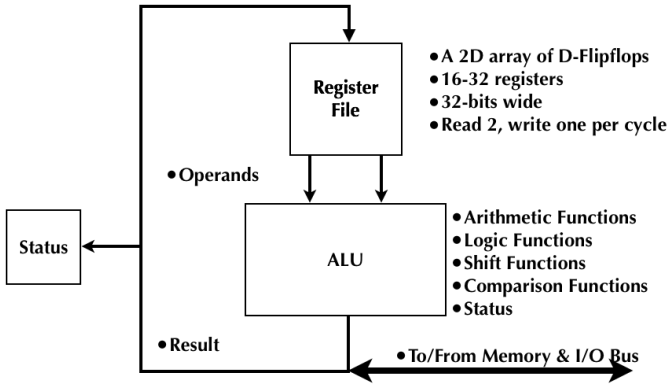
You may execute your project on any computer you have available. However, I suggest you use a computer with the gcc compiler and use an Unix-like operating system.

You may form teams of two to three people or work as an individual for this project. If you form a team, you must describe in your report the contribution of each team member.

Your results will be submitted as a "Group Submission" on Blackboard. I have set up 30 groups under the group name of "ALUSimulation_B70413." When you form a group, one member of the group should send me a list of the group members. I will select a group number and reply Each member of your group must join the group on Blackboard. If you decide to work alone, you must still join a group.

2 ALU/RegisterFile Model

The following figure illustrates the typical RISC CPU model. In this project you will simulate the ALU.



2.1 References

The following documents describing the MIPS architecture are available on the class website.
http://www.ittc.ku.edu/~gminden/Computer_Architecture/PDFs/britton-mips-tutorial.pdf

http://www.ittc.ku.edu/~gminden/Computer_Architecture/PDFs/mips-isa.pdf

http://www.ittc.ku.edu/~gminden/Computer_Architecture/PDFs/PH_ComputerOrganization_Chapter_2.pdf

Be careful of PDF viewers breaking long URLs.

2.2 Instructions to Implement

You shall implement the instructions listed in Table 1.

Table 1 Summarizing MIPS OpCodes to implement

OpCode	Function/ Register Code	Instruction Type	Mnemonic
000000	00_0000	R	NOOP
000000	00_0000	R	SLL
000000	00_0010	R	SRL
000000	00_0011	R	SRA
000000	00_0100	R	SLLV
000000	00_0110	R	SRLV
000000	01_0000	R	MFHI
000000	01_0010	R	MFLO
000000	01_1000	R	MULT
000000	01_1001	R	MULTU
000000	01_1010	R	DIV
000000	01_1011	R	DIVU
000000	10_0000	R	ADD
000000	10_0001	R	ADDU

OpCode	Function/ Register Code	Instruction Type	Mnemonic
000000	10_0010	R	SUB
000000	10_0011	R	SUBU
000000	10_0100	R	AND
000000	10_0101	R	OR
000000	10_0110	R	XOR
000000	10_1010	R	SLT
000000	10_1011	R	SLTU
001000	x_xxxx	I	ADDI
001001	x_xxxx	I	ADDIU
001010	x_xxxx	I	SLTI
001011	x_xxxx	I	SLTIU

3 Simulation Architecture

Your ALU Simulator will be part of a provided structure. The provided structure will include (a) a main program, (b) a simulation of a register file, and (c) list of instructions.

The main program will (1) initialize the contents of the RegisterFile, (2) load a list of instructions from a file, (3) for each instruction call your ALUSimulator, and (4) dump the contents of the Register.

Your ALU Simulator will be a subroutine that you write. It shall have a standard name: "ALUSimulator." The arguments to your ALUSimulator shall be (Version B): (a) the RegisterFile to use, (b) the instruction opcode, (c) register indices Rs, Rt, and Rd, (d) the shift amount, (e) the function code, (f) the immediate value, (g) status. Instructions are decoded for you in the main program.

```
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 );
```

Following is the RegisterFile.h and RegisterFile.c code. This code is available at:

http://www.ittc.ku.edu/~gminden/Computer_Architecture/Software/

```
//  Created by Gary J. Minden on 10/20/2015.
//  Copyright 2015 Gary J. Minden. All rights reserved.
//
//  Updates:
//
//      B51020 -- initial version
//
//
//      Description:
//                This program simulates a three port register file.
//

#ifndef __RegisterFile_H_
#define __RegisterFile_H_

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

//
//      Define RegisterFile structures
//
#define Registers_Nbr 32

typedef int RegisterFile[Registers_Nbr];

//*****************************************************************************
//
// Prototypes for the APIs.
//
//*****************************************************************************

extern void RegisterFile_Cycle( RegisterFile theRegisterFile,
                                uint32_t RdAddr_S, uint32_t* RdValue_S,
                                uint32_t RdAddr_T, uint32_t* RdValue_T,
                                bool WrtEnb, uint32_t WrtAddr, uint32_t WrtValue );

extern void RegisterFile_Read( RegisterFile theRegisterFile,
                                uint32_t RdAddr_S, uint32_t* RdValue_S,
                                uint32_t RdAddr_T, uint32_t* RdValue_T );

extern void RegisterFile_Write( RegisterFile theRegisterFile,
                                bool WrtEnb, uint32_t WrtAddr, uint32_t WrtValue );

extern void RegisterFile_Dump( RegisterFile theRegisterFile );

#endif       // __RegisterFile_H_

================================================================================
================================================================================
================================================================================


//  Created by Gary J. Minden on 10/20/2015.
//  Copyright 2015 Gary J. Minden. All rights reserved.
//
//  Updates:
//
//      B51020 -- initial version
```

```
//
//
//      Description:
//                This program simulates a CPU RegisterFile.
//

#include <stdio.h>
#include <time.h>
#include <strings.h>
#include <stdlib.h>
#include <unistd.h>

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

#include "RegisterFile_01.h"

//
//      RegisterFile_Cycle performs one cycle of the register file. This involves
//           two read operations and one write operation. The write operation is
//           gated by a WrtEnb signal. The write operation is performed after
//           the read operations which better reflects the behavior of a
//           hardware register file.
//
extern void RegisterFile_Cycle( RegisterFile theRegisterFile,
                                uint32_t RdAddr_S, uint32_t* RdValue_S,
                                uint32_t RdAddr_T, uint32_t* RdValue_T,
                                bool WrtEnb, uint32_t WrtAddr, uint32_t WrtValue ) {

        printf( "RegisterFile_Cycle: RdAddr_S: %02d; RdAddr_T: %02d; WrtEnb: %01d;
WrtAddr: %02d\n",
                  RdAddr_S, RdAddr_T, WrtEnb, WrtAddr );

        //
        //   Read the operands
        //
        *RdValue_S = theRegisterFile[RdAddr_S];
        *RdValue_T = theRegisterFile[RdAddr_T];

        //
        //   If enabled, write data to the register file.
        //
        if ( WrtEnb == true ) {
             theRegisterFile[WrtAddr] = WrtValue;
        }

}

//
//      In many cases it is only necessary to read value from a register file.
//           RegisterFile_Read just does the value read operation.
//
extern void RegisterFile_Read( RegisterFile theRegisterFile,
                                uint32_t RdAddr_S, uint32_t* RdValue_S,
                                uint32_t RdAddr_T, uint32_t* RdValue_T ) {

        printf( "RegisterFile_Read: RdAddr_S: %02d; RdAddr_T: %02d;\n",
                  RdAddr_S, RdAddr_T );

        //
        //   Read the operands
        //
```



```
        *RdValue_S = theRegisterFile[RdAddr_S];
        *RdValue_T = theRegisterFile[RdAddr_T];

}

//
//      In many cases it is only necessary to write a value to the register file.
//           RegisterFile_Write just does the write operation. The WrtEnb argument
//           is included for consistency.
//
extern void RegisterFile_Write( RegisterFile theRegisterFile,
                                bool WrtEnb, uint32_t WrtAddr, uint32_t WrtValue ) {

        printf( "RegisterFile_Write:  WrtEnb: %01d; WrtAddr: %02d\n",
                  WrtEnb, WrtAddr );

        //
        //   If enabled, write data to the register file.
        //
        if ( WrtEnb == true ) {
             theRegisterFile[WrtAddr] = WrtValue;
        }

}

//
//      RegisterFile_Dump prints the contents of a register file to standard output
//           (stdout). RegisterFile_Dump is useful for debugging purposes. In the
future,
//           this subroutine should include a file descriptor indicating the output.
//
extern void RegisterFile_Dump( RegisterFile theRegisterFile ) {

        //
        //   This routine prints four values per line in hexadecimal format.
        //        The number of lines is determined by the number of registers
        //        in the register file. We assume the number of registers is
        //        a multiple of 4. If not, some registers will not be printed.
        //
        uint32_t  Lines_Nbr = ( Registers_Nbr / 4 );
        uint32_t  Line_Idx;

        for ( Line_Idx = 0; Line_Idx < Lines_Nbr; Line_Idx++ ) {
             printf( "%08X:  %08X  %08X  %08X  %08X\n",
                       (Line_Idx * 4),
                       theRegisterFile[(Line_Idx * 4) + 0],
                       theRegisterFile[(Line_Idx * 4) + 1],
                       theRegisterFile[(Line_Idx * 4) + 2],
                       theRegisterFile[(Line_Idx * 4) + 3] );
        }
}
```