CSC252: Computer Organization
Spring 2016
Project #3

A. Introduction

In this project, you will create an MIPS emulator, written in C, that is capable of executing a subset of the MIPS instruction set. Starting with the test binaries provided to you, you will fetch, decode, and execute instructions to mimic an actual MIPS processor effectively creating a miniature version of MARS!

B. Getting Started

Download and unpack the project files using the command line below,

tar -xvzf Project3.tar.gz

The extracted directory will have the following file structure

1. Instruction SubSet - Contains a list of all the instructions your emulator is expected to decode and execute successfully. A format breakdown is provided for your aid.
2. MipsInstructionSetReference - Contains the ISA implementation details of all required instructions. Do read this carefully to identify nuances of instructions. You do not need to take care of integer overflow exceptions.
3. Makefile - Used this to compile you project and generate a executable. Just run the 'make' command in your command line and you should have an executable called eMIPS. The make file uses gcc to compile all the files relevant to the project. The order of files added is important so make sure when you edit the Makefile to included the .c in the right order.
4. tests/ - contains three sub directories, each corresponding to one tier of tests, containing binaries of each test program. They are all compiled for a MIPS, Big endian machine. You are required to have all these tests running. Refer to the section on grading for the point distribution.
5. src/ - This directory contains all the files that make up your emulator.
   (a) elf_reader/ - contains the base memory system and functions necessary to interpret binaries and store binary data for later retrieval.
   (b) utils/ - contains additional files necessary to make elf_reader run.

   Note: You would not have to modify any of the files present in the above 2 sub directories for your project. Curious minds are however encouraged to take a look at the content & structure of these files to further understand the emulator

   (c) PROC.c - contains the main() function and is where all parts of your project will come together. A majority of the functions to decode, execute and manipulate data will be called here.
   (d) RegFile.c/.h - contains the register file declaration and support functions.

(e) Syscall.c/.h - contains system call implementations and support functions. These files are necessary to successfully run the tire 2 asm & cpp based binaries.

C. Task

You are allowed to follow any approach to implement the emulator. The basic framework of the emulator reads binaries and extracts the instruction data for you. This data can later be retrieved from the emulators memory. It will also provide the starting address of the program in memory, stored in the PC variable found in PROC.c.

You are expected to read out the the instruction from memory using readWord() function and decode, execute & retire the instruction depending on the type of instruction.

1. Arithmetic instructions will read and write from the Register File and make use of the ALU for data modification.
2. Memory instructions will use the ALU to calculate the effective address and use the writeByte() , readByte(), writeWord() & readWord() functions defined in elf_reader.c to interact with the emulator memory. Take note that half word memory instructions are left up to you for implementation.
3. Branch and jump instruction will deal with modifying the PC variable depending on register values. Be sure to implement the branch delay slot!
4. System calls are handled by the SyscallExe() defined in Syscall.c. During the execution of a system call instruction you must pass the syscall ID, which can be found in register 2 to this function.
5. In case of an unsupported instruction arises, treat it as a NOP and proceed.

D. Running Your Code

Make sure to compile your code using the make file. To run a test program, execute the eMIPS executable, with the relative path to the test program's binary file, in the command line followed by the number of instructions you would like to execute in that binary, check the example below. Tier 2 and 3 test cases make use of the exit system call and will automatically exit when they encounter it.

```
./eMIPS tests/cpp/hello 1000
```

Note: The binaries are files that do not have any extensions. The additional files with the .s & .txt extension are present for your reference.

During the execution of the code any standard out and standard error messages will be dumped in the stdout.txt & stderr.txt files.

E. Advice

1. Start Early! Irrespective of your proficiency with C this project will take time and patience.
2. The printf() instruction is your best friend. Use it whenever in doubt.
3. The best place to start is to try and breakdown your entire process of execution and put that down on paper. This will make it easier for you to keep track of the various aspects of implementation.

F. Grading

Test Case	Points
Tier 1 **arith** **branchtest** **hilo** **linktest** **systest** **zero**	**1 pt each**
Tier 2 **MatrixMultiplication** **MergeSort** **BinarySearch**	**2 pts each**
Tier 3 **class** **hello**	**4 pts each**

G. Submission

A report is expected on the above project that would contain a brief write up on the specifics of your implementation and any other piece of information you would like to highlight about your implementation. For test cases that have a printf() or use the write to file syscall a screenshot of the stdout.txt must be included. For testcases that do not have such an output, a screenshot of the Register File output on it's final cycle must be included.

You are to compress and archive your project 3 folder using the following command & naming convention and upload the generated file, along with your report, onto blackboard.

tar -cvzf Project3_StudentName(s).tar.gz Project3