ICS 51: Introductory Computer Organization

Lab#2 (lab2.s)

Data file: lab2.dat

(Due:

Soft: Feb 17th, 11:59 pm

Hard: Feb 19th, 11:59 pm)

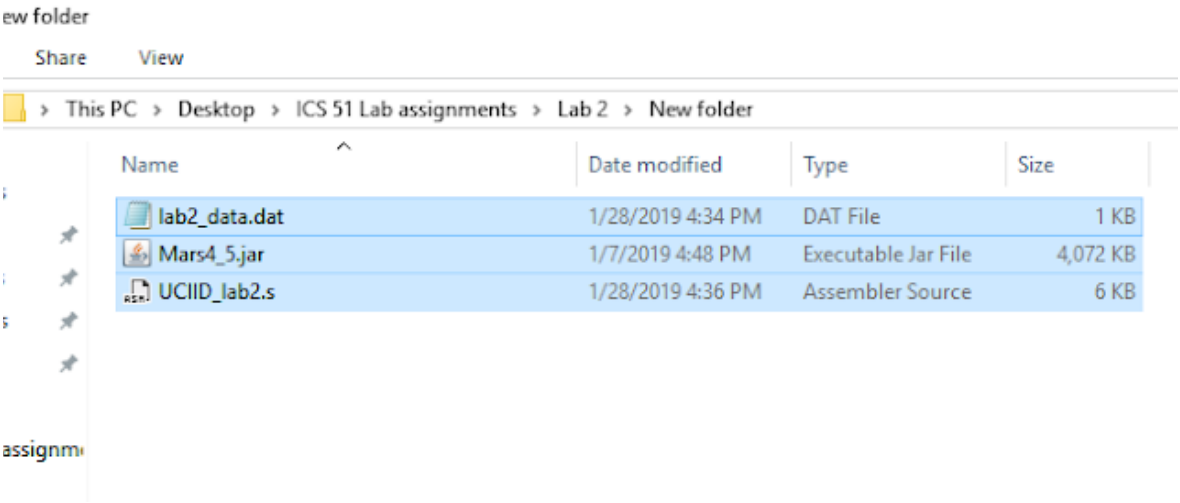
REMEMBER TO WRITE YOUR NAME AND UCI ID NUMBER IN THE LAB FILE, IF YOUR NAME IS JOHN SMITH WITH UCIID 12345 THEN REPLACE THE FIRST AND LAST NAME STRINGS WITH "JOHN SMITH" AND THE UCIID STRING WITH "12345'. SUBMITTING WOULD BE 12345_LAB2.S, DO NOT ZIP THE FILE

In this lab, you will learn how to use MIPS assembly instructions to:

1. implement FOR and WHILE loop constructs
2. read/write from/to memory
3. Read from a file

Download the lab files and implement three functions "strlen", "valid_id" and "file_read". Please take a look at comments in the provided source code for more information.

NOTE: In order for the file reading to work, your lab2 file, .dat file, and mars have to be in the same folder. Please do not modify the path inside the lab2 file as it will mess up grading. See image for example



1. Implementing loop constructs in MIPS assembly:

1.1. WHILE loop:

The most general form of loop is WHILE loop which takes the following form in C/C++:

```
while (condition) {
     <<loop body statements>>
}
```

There are two important points to note about a WHILE loop. First, the test for termination appears at the beginning of the loop. Second, as a direct consequence of the position of the termination test, the body of the loop may never execute.

A WHILE loop can be converted to assembly using the following pseudo-code template:

```
BeginningOfWhile:
If (condition) {
     <<loop body statements>>
     j BeginningOfWhile
}
EndOfWhile:
```


Therefore, you can use the technique from earlier lab to convert IF statements to assembly language along with a single JMP instruction to produce a WHILE loop. An example of such translation is shown below:

C code	MIPS Assembly code

`while ( c > 0 ) {`
`  a = b + 1;`
`  c--;`
`}`
	`lw $t1, a`
	`lw $t2, b`
	`lw $t3, c`
	
	`BeginningOfWhile:`
	`blez $t3, EndOfWhile`
	`addi $t1, $t2, 1`
	`subi $t3`
	`j BeginningOfWhile`
	
	`EndOfWhile:`

	sw $t1, a sw $t3, c

1.2. FOR loop:

A FOR loop is essentially a WHILE Loop with an initial state, a condition, and an iterative instruction. For instance, the following generic FOR Loop in C/C++:

```
For (initialization; condition; update) {
    <<loop body>>
}
```


can be converted to the following pseudo-code using WHILE loop:


```
<<initialization>>
while(condition) {
    <<loop body statements>>
    <<update>>
}
```


We already know how to translate a WHILE loop to x86 assembly, therefore we can translate any FOR loop using the above template.

2. MIPS Memory Addressing Modes:

The memory addressing mode determines, for an instruction that accesses a memory location, how the address for the memory location is specified. Two modes that we use to access data in MIPS assembly are:

1) **register addressing mode**: this is the simplest form to access memory. In this form, memory address is stored in a register.
2) **base addressing mode**: address is stored in a register, however it can be adjusted by an immediate value.

To obtain the address of data in memory we use the “la” instruction. You can access memories in byte granularity with “lb, sb” or word granularity with “sw, lw”. In this lab, since data elements are character, we use lb and sb.

Mode	Example	Effect
Register addressing	`lw $t2, ($t3)`	`$t2 = Mem[$t3]`
	`sw $t3, ($t4)`	`Mem[$t4] = $t3`
Base addressing	`lw $t2, 32($t3)`	`$t2 = Mem[$t3 + 32]`
	`Sw $t3, 16($t4)`	`Mem[$t4 + 16] = $t3`

ASCII CODE

Every character has a corresponding one-byte code in what we call the ASCII table. In this lab you are dealing with characters and checking for the validity. Go to this webpage for the full ascii reference:
http://www.asciitable.com/

File reading:

During discussion we’ll be going over how to read a file using syscalls. As in other languages, reading a file involves interaction with the operating system. In MIPS, this is done through syscalls, similar to user I/O. To read data from a file, it first needs to be opened. Opening a file returns a file descriptor, essentially an index into an array of file information maintained by the operating system. Once the file is opened, one can read information. The bit patterns stored in the file are typically read sequentially and stored in memory. Because there is a difference between the ASCII representation of a number and its two's complement representation, one must be careful on how a file is read and interpreted. For example, the number 16383 requires 5 bytes to be represented in ASCII (31 36 33 38 33 [base 16]) and 4 bytes in a 32-bit two's complement representation (00 00 3f ff [base 16]). Once all information is read from the file, one needs to close the file to free the file descriptor.

Important SYSCALLS to remember are:

1: Print the integer stored in $a0

4: Print the string using $a0 as your address
11: Print a character stored in $a0
13: Open a file
14: Read from a file

For the full reference go here:
http://courses.missouristate.edu/KenVollmar/Mars/Help/SyscallHelp.html

For a sample file on how to read from file go here: file_read.s data_file

For the assignment, you are giving the code to open and close a file. You need to use the right syscall for reading it. The file contains 4 lines, you need to print the message in those 4 line using only one line. All lowercase (a-z) should be transformed to upper (A-Z) and you should print blank space and exclamation mark. Everything else should be discarded.

NOTE: Grading will be done using a different file with 4 lines and no more than 30 bytes of characters, so make sure your code can run with all cases