Tufts COMP 140 - Advanced Architecture Summer 2014

MIPS Assembly programming

Goals:

- **Become familiar with MARS**
- **Assemble and run MIPS programs**
- **Debug MIPS programs**
- **Write simple MIPS programs**

This lab is partially a tutorial. We will walk you through several simple MIPS programs and then turn you loose to write some of your own code. We will use MARS as the simulator for running your assembly code.

Getting Started

Step 1. Setup your assembly files

- Create a folder on the desktop called "Comp140".
- Download all the .asm files from the class webpage (Labs→Lab2) into your folder

Step 2. Open MARS

- Find MARS and open.
- You should see a MARS window.
- There is an Edit window and an Execute window on the left, and the register listing on the right. There is also a console window at the bottom of the screen.

MIPS Arithmetic

Step 1. Open and run your first assembly file (ALU.asm)

- Open the file that you downloaded called ALU.asm.
- Asssemble the assembly file by clicking on Run→Assemble (F3). You should see the message "Assemble: operation completed successfully" in the bottom window.
- Run the program by clicking the green Run button, or by clicking Run→Go (F5). The program should start at the first instruction in memory (0x00400000). You should then see an error: "Go: execution terminated with errors."
- Your job is to find the error.
- Re-assemble (F3) the program again, and then step through the program. You can single-step by clicking on the Green button with the "1" as a subscript. Observe how

each instruction affects the machine state (the window on the right), and then when you get to the error, you can start to debug it.

Step 2. Complete problems 1-3 below

1. What's the problem with ALU.asm (answer with 1 sentence)? Assume that all immediate values are meant to be positive, change one command to remove the problem, write down the code that you modified.
2. Add a section to the ALU.asm program that performs a subtraction that causes overflow.
3. Add code at the end of ALU.asm to *either* multiply or divide two numbers. If you multiply, make sure to select two operands that produce a result that is larger than 32-bits. Place the upper 4 bytes and lower 4 bytes of the result in the next two available memory words, respectively. If you divide, make sure to select two operands that cause a non-zero remainder. Place the 4 byte quotient and the 4 byte remainder in the next two available memory words, respectively.

Misc MIPS

In addition to the basic instructions that you have learned in class, MIPS supports several other things to help you program. This section of the lab helps you to explore *pseudo-instructions*, *assembler directives*, and *system calls*.

Pseudo-instruction means "fake instruction". These are instructions that are useful to the programmer, but they do not actually exist in the instruction set that the machine understands. Instead, the assembler converts the pseudo-instructions into actual executable instructions. This not only makes it easier to program, but can also add clarity to the program.

Assembler directives help the assembler to understand code. They start with a '.' and convey information to the assembler.

System calls are simple operating-system-like services that MIPS provide. A list of these is shown below:

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $v0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	

To run a system call, set up the appropriate registers (usually $a0-$a2), store the system call number in $v0, and use "**syscall"** in your assembly program.

Step 3. Open and run the assembly file

- Open the assembly file in MARS using the File→Open icon (an open folder).
- Assemble and run the program (F3, F5) The program should start at the first instruction in memory (0x00400000).
- Hint: for debugging, you may find it easier to single step through the program.

Step 4. Change the program to print out "red" instead of "blue."
The intention of the programmer was to print the word "red." Find out why the programmer failed. Change the 'lbu' line to fix the problem (to make the program print out 'red'). Write down the code that you modified.

1. Modify the code that *follows* label 'cx:' so that the program prints out "The color is" followed by the selected color. e.g. – "The color is red". You may use pseudo-instructions, assembler directives, and system calls as needed.

Writing Assembly

Now it is your turn to write some assembly code!

Step 5. Open max.asm

Step 6. Complete program max.asm

max.asm shows you some pseudo-code in C at the top. Your job is to implement this pseudo-code in assembly. Some of the code is done for you, but you still need to implement the loop section. You may use pseudo-instructions as needed.