CPE 315
Spring 2019
Seng
Lab #1
Complete by 11:59pm 4/17/19 (Wednesday night)

Objectives:

- To practice MIPS programming
- Practice low-level bit manipulations

Files:

1. MIPS Sample Program - Run this yourself to see how the simulator and assembly language work
2. SPIM Simulator - If you like, you can download it for home use. Otherwise, run spim on unix1 (from /home/jseng/bin/spim).

Description:

For this lab, you will use the SPIM simulator. This simulator will simulate a MIPS processor by loading and running a file containing MIPS assembly code. SPIM is available for free download and it is available on unix1 by using '/home/jseng/bin/spim'.

Stepping through sample code

The sample code is not fully commented, but you should be able to figure out how input and output works. Please read through the following tips:

- The 'syscall' instruction calls the operating system to perform a console input/output function. Before 'syscall' is executed, a value parameter should be loaded into $v0 to denote whether an input (5) or output (4=text, 1=number) operation is desired.
- When running the code, make sure the starting PC is 0x00400000. This is where SPIM places the beginning of your code.
- To print a string, the start address of the string should be loaded into $a0 (this can be seen in the example code). When SPIM loads in strings, the first string starts at address 0x10010000. In order to find the start address for the next string, you will need to count over how many characters were in the previous string.
- 'ori' means 'OR immediate'. The instruction does a bitwise logical OR of a register with a constant.
- 'lui' means 'load upper immediate'. The instruction loads a 16-bit number into the upper 16 bits of a register.
- use can 'step' though the code an instruction at a time
- Instructions for SPIM are here:
    - spim (command line on unix1: /home/jseng/bin/spim)
- You can find a MIPS instruction reference here.

Altering the code for your programs

For this assignment, you will write 4 programs. For each program write a corresponding function in Java that does what is required, then convert your Java code to MIPS assembly. The algorithms you use in Java should be the same as the algorithms you use in assembly. Place your Java code as comments in the assembly file you turn in. Make sure each assembly program prompts the user to accept input and prints out a string stating what the answer is.

1. Write the following fast "mod" function. This function uses no modulus operator, multiplication, or division - it uses only basic arithmetic/logical operations (add, sub, and...). The function takes two integers as inputs - a number (num), and a divisor (div). You are guaranteed that div is a power of 2. You want the remainder of num / div. For example, if num=22 (00010110 in binary) and div = 4 (100) would return 2 (10). Your algorithm should *not* repeatedly subtract (or add) div from num. Name your file **mod.asm**. Program 1 only needs to work with positive numbers.

2. Write a program which prints the number that represents reverse-ordered binary of the input number. This means the your program will print the 32-bit number that is generated if the 32-bit input number's bits are written in reverse order (MSB becomes LSB and so on). Name your file **reverse.asm**. Program 2 only needs to work with positive numbers as input.

3. Write a function which divides a 64-bit unsigned number with a 31-bit unsigned number. The 31-bit divisor is guaranteed to be a power of 2. The program should take in the 64-bit as 2 32-bit numbers: first the upper 32 bits, then the lower 32 bits.

The answer should be printed out as 2 32-bit numbers. Do not worry if the output of your program is a negative number. PCSPIM prints out numbers as signed numbers, so if one of the answers happens to have a leading 1, the number will appear as negative. Here are some test cases (shown as dividend high 32, dividend low 32 / divisor = quotient high 32, quotient low 32): 1,1 / 65536 = 0,65536 and 2,10 / 65536 = 0,131072 and 42,32 / 32 = 1,1342177281 and 210,64 / 64 = 3, 1207959553. Name your file **divide.asm**. Program 3 only needs to work with positive numbers.

4. Write a function which does exponentiation: x raised to the power of y. Your function should not use multiplies, but may use repeated addition. Both x and y will be positive. Name your file **exponent.asm**.

Turning in the assignment

- Include the following header at the top of your files:

```
# Name:  include lab partner if you worked with another student
# Section:  x
# Description:  include a short description of your program
```

- Use 'handin' to turn in your 4 program files(the asm files).
- Put your Java functions as comments in the beginning of each corresponding assembly file.

At the unix prompt, type 'handin jseng' to see what assignments I am accepting. For example, in order to handin a file named 'filename.asm', type:

```
handin jseng 315_lab1 filename.asm
```